Installing and Operating 2.9BSD
17 March 1998

Michael J. Karels
Carl F. Smith

University of California
Berkeley, California 94720

ABSTRACT

This document contains instructions for installation and operation of the Second Berkeley Software Distribution's 2.9BSD release of the PDP-11† UNIX‡ system. It is adapted from the paper *Installing and Operating 4.1bsd* by Bill Joy.

This document explains the procedures for installation of Berkeley UNIX on a PDP-11 or to upgrade an existing Berkeley PDP-11 UNIX system to the new release. It then explains how to configure the kernel for the available devices and user load, lay out file systems on the available disks, set up terminal lines and user accounts, and do system specific tailoring. It also explains system operations procedures: shutdown and startup, hardware error reporting and diagnosis, file system backup procedures, resource control, performance monitoring, and procedures for recompiling and reinstalling system software. Technical details on the kernel changes are presented in the accompanying paper, "Changes in the Kernel in 2.9BSD."

The 2.9BSD release, unlike previous versions of the Second Berkeley Software Distribution, is a complete Version 7 UNIX system with all of the standard UNIX tools and utilities, with or without Berkeley modifications. Therefore, it does not need to be layered onto an existing Version 7 system; because of the many changes and additions throughout the system, it would require a substantial effort to merge into most earlier systems.

---

† DEC, MASSBUS, PDP, and UNIBUS are trademarks of Digital Equipment Corporation.
‡ UNIX is a trademark of Bell Laboratories.

1. INTRODUCTION

This document explains how to install the 2.9BSD release of the Berkeley version of UNIX for the PDP-11 on your system. If you are running the July 1981 release of the system, which was called 2.8BSD, you can avoid a full bootstrap from the new tape by extracting only the software that has changed. Be warned, however, that there are a large number of changes. Unless you have many local modifications it will probably be easier to bring up an intact 2.9BSD system and merge your local changes into it. If you are running any other version of UNIX on your PDP-11, you will have to do a full bootstrap. This means dumping all file systems which are to be retained onto tape in a format that can be read in again later (*tar* format is best, or V7 *dump* if the file system configuration will be the same). A new root file system can be made and read in using standalone utilites on the tape. The system sources and the rest of the /usr file system can then be extracted. Finally, old file systems can be reloaded from tape.

To get an overview of the process and an idea of some of the alternative strategies that are available, it is wise to look through all of these instructions before beginning.

0.1. Hardware supported

This distribution can be booted on a PDP-11/23, 24, 34, 34A, 40, 44, 45, 55, 60, or 70 CPU with at least 192 Kbytes of memory and any of the following disks†:

DEC MASSBUS:	RM03, RM05, RP04, RP05, RP06
DEC UNIBUS:	RK05, RK06, RK07, RL01, RL02, RM02, RP03, RP04, RP05, RP06
AED 8000 UNIBUS:	AMPEX DM980 (emulating RP03)
AED STORM-II	AMPEX DM980 (emulating RM02)
DIVA COMP V MASSBUS:	AMPEX 9300
EMULEX SC-21 UNIBUS:	AMPEX 9300, CDC 9766 (emulating RM05)
EMULEX SC-11 or SC-21 UNIBUS:	CDC 9762, AMPEX DM980

The tape drives† supported by this distribution are:

DEC MASSBUS:	TE16, TU45, TU77
DEC UNIBUS:	TE10, TE16, TS11, TU45, TU77
DATUM 15X20 UNIBUS:	KENNEDY 9100 (emulating TE10)
EMULEX TC-11 UNIBUS:	KENNEDY 9100, 9300 (emulating TE10)

0.2. Distribution format

The distribution format is two 9-track 800bpi 2400' magnetic tapes. The tapes are also available at 1600bpi. The format for 1600bpi tapes is the same. If you are able to do so, it is a good idea to immediately copy the tapes in the distribution kit to guard against disaster. The first tape contains some 512-byte records, some 1024-byte records, followed by many 10240-byte records. There are interspersed tape marks; end-of-tape is signaled by a double end-of-file. The second tape contains only 10240-byte records with no interspersed tape marks.

The boot tape contains several standalone utility programs, a *dump* image of a root file system, and a *tar* image of part of the /usr file system. The files on this tape are:

† Other controllers and drives may be easily usable with the system, but might require minor modifications to the system to allow bootstrapping. The controllers and the drives shown here are known to work as bootstrap devices.

File	Contents	Record Size
0	boot block	512
	(EOR)	
	boot block	512
	(EOR)	
	Standalone **Boot**	512
	(EOF)	
1	Standalone **cat**	1024
	(EOF)	
2	This index	1024
	(use **cat** to read)	
	(EOF)	
3	Standalone **mkfs**	1024
	(see *mkfs* (8)†)	
	(EOF)	
4	Standalone **restor**	1024
	(see *restor* (8))	
	(EOF)	
5	Standalone **icheck**	1024
	(see *icheck* (8))	
	(EOF)	
6	Dump of small "root" file system	10240
	(213 10K-byte blocks; see *dump* (8))	
	(EOF)	
7	Tar archive of /usr files	10240
	(most of the tape; see *tar* (1))	
	(EOF)	
	(EOF)	

The last file on the tape is a *tar* image of most of the /usr file system except for sources (relative to /usr; see *tar* (1)). It contains most of the binaries and other material which is normally kept on-line, with the following directories: **70 adm bin contrib dict doc games include lib local man msgs preserve public spool sys tmp ucb**. It contains 1594 10K byte blocks. The second tape contains one file in *tar* format, again relative to /usr, consisting of 1958 10K byte blocks and containing the source tree with all command and kernel sources. It contains the directories **net**, **src**, and **ingres** and includes the Berkeley additions (which are in /usr/src/ucb and /usr/ingres). The **net** directory contains sources for the TCP/IP system.

0.3. UNIX device naming

UNIX has a set of names for devices that are different from the DEC names for the devices. The disk and tape names used by the bootstrap and the system are:

RK05 disks	**rk**
RK06, RK07 disks	**hk**
RL01, RL02 disks	**rl**
RP02, RP03 disks	**rp**
TE16, TU45, TU77/TM02, 3 tapes	**ht**
TE10/TM11 tapes	**tm**
TS11 tapes	**ts**

† References of the form *X*(*Y*) mean the subsection named *X* in section *Y* of the Berkeley PDP-11 UNIX Programmer's manual.

There is also a generic disk driver, **xp**, that will handle most types of SMD disks on one or more controllers (even different types on the same controller). The **xp** driver handles RM02, RM03, RM05, RP04, RP05 and RP06 disks on DEC, Emulex, and Diva UNIBUS or MASSBUS controllers.

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

xx(*y*,*z*)

where *xx* is one of **hk**, **ht**, **rk**, **rl**, **rp**, **tm**, **ts**, or **xp**. The value *y* specifies the device or drive unit to use. The *z* value is interpreted differently for tapes and disks: for disks it is a block offset for a file system and for tapes it is a file number on the tape.

Large UNIX physical disks (**hk**, **rp**, **xp**) are divided into 8 logical disk partitions, each of which may occupy any consecutive cylinder range on the physical device. The cylinders occupied by the 8 partitions for each drive type are specified in section 4 of the Berkeley PDP-11 UNIX Programmer's manual.† Each partition may be used for either a raw data area such as a swapping area or to store a UNIX file system. It is conventional for the first partition on a disk to be used to store a root file system, from which UNIX may be bootstrapped. The second partition is traditionally used as a swapping area, and the rest of the disk is divided into spaces for additional "mounted file systems" by use of one or more additional partitions.

The disk partitions have names in the standalone system of the form "*xx*(*y*,*z*)" as described above. Thus partition 1 of an RK07 at drive 0 would be "hk(0,5940)". When not running standalone, this partition would normally be available as "/dev/hk0b". Here the prefix "/dev" is the name of the directory where all "special files" normally live, the "hk" serves an obvious purpose, the "0" identifies this as a partition of hk drive number "0" and the "b" identifies this as partition 1 (where we number from 0, the 0th partition being "hk0a"). Finally, "5940" is the sector offset to partition 1, as determined from the manual page *hk*(4).

Returning to the discussion of the standalone system, we recall that tapes also took two integer parameters. In the case of a TE16/TU tape formatter on 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. The distribution tapes have data structures in the tape files and though the first tape contains only 8 tape files, it contains several thousand UNIX files.

0.4. UNIX devices: block and raw

UNIX makes a distinction between "block" and "character" (raw) 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/*xx*0a", etc. are block devices and thus use the system's normal buffering mechanism. There are also raw devices available, which do physical I/O operations directly from the program's data area. These have names like "/dev/r*xx*0a", the "r" here standing for "raw." In the bootstrap procedures we will often suggest using the raw devices, because these tend to work faster. In general, however, the block devices are used. They are where file systems are "mounted." The UNIX name space is increased by logically associating (*mount*ing) a UNIX file system residing on a given block device with a directory in the current name space. See *mount*(2) and *mount*(8). This association is severed by *umount*.

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

0.5. Reporting problems or questions

Problems with the software of this distribution, or errors or omissions in the documentation, should be reported to the 2BSD group. Whenever possible, submit such reports by electronic mail; the address is:

† It is possible to change the partitions by changing the values in the disk's sizes array in ioconf.c.

2bsd@berkeley	(by ARPAnet)
or	
ucbvax!2bsd	(by UUCP)

There is a mail drop for bug reports and fixes:

2bsd-bugs@berkeley	(by ARPAnet)
or	
ucbvax!2bsd-bugs	(by UUCP)

These reports or fixes are expected to be in the format generated by the *sendbug* (1) program. A redistribution list of users who have indicated that they would like to receive bug reports is also maintained:

2bsd-people@berkeley	(by ARPAnet)
or	
ucbvax!2bsd-people	(by UUCP)

This list may also be used as a general forum for help requests, sharing common experiences, etc. Requests to be added to (or deleted from) this list should be made to the 2bsd address above. If it is not possible to use electronic mail, then call or write the 2BSD office. Although there is seldom someone there to take your call, there is an answering machine, and your request will be forwarded to the appropriate person. The phone number and mailing address are:

Berkeley PDP-11 Software Distribution – 2BSD
Computer Science Division, Department of EECS
573 Evans Hall
University of California, Berkeley
Berkeley, California 94720
(415) 642-6258

2. BOOTSTRAP PROCEDURES

This section explains the bootstrap procedures that can be used to get one of the kernels supplied with this tape running on your machine. If you are not yet running UNIX or are running a version of UNIX other than 2.8BSD, you will have to do a full bootstrap.

If you are running 2.8BSD you can use the update procedure described in section 4.2 instead of a full bootstrap. This will affect modifications to the local system less than a full bootstrap. Note, however, that a full bootstrap will probably require less effort unless you have made major local modifications which you must carry over to the new system.

If you are already running UNIX and need to do a full bootstrap you should first save your existing files on magnetic tape. The 2.9BSD file system uses 1K-byte blocks by clustering disk blocks (as did the 2.8BSD system); file systems in other formats cannot be mounted. **Those upgrading from 2.8 should note that 2.9BSD uses generally different file system partition sizes than 2.8BSD, and that a few of the major device numbers have changed (in particular, that for the hk).** The easiest way to save the current files on tape is by doing a full dump and then restoring in the new system. This works also in converting V7, System-III, or System-V 512-byte file systems. Although the dump format is different on V7, System-III, and System-V, *512restor*(8) can restore old format V7 *dump* image tapes into the file system format used by 2.9BSD. *Tar*(1) can also be used to exchange files from different file system formats, and has the additional advantage that directory trees can be placed on different file systems than on the old configuration. Note that 2.9BSD does not support *cpio* tape format.

The tape bootstrap procedure involves three steps: loading the tape bootstrap monitor, creating and initializing a UNIX ‘‘root’’ file system system on the disk, and booting the system.

2.1. Booting from tape

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. Then use the normal bootstrap ROM, console monitor or other bootstrap to boot from the tape. 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 $172526, r0)
172526
010040    (mov r0, -(r0))
012740    (mov $60003, -(r0))
060003
000777    (br .)
```

When this is executed, the first block of the tape will be read into memory. Halt the CPU and restart at location 0.

The console should type

*nn***Boot**
:

where *nn* is the CPU class on which it believes it is running. The value will be one of 24, 40, 45 or 70, depending on whether separate instruction and data (separate I/D) and/or a UNIBUS map are detected. The CPUs in each class are:

Class	PDP11s	Separate I/D	UNIBUS map
24	24	-	+
40	23, 34, 34A, 40, 60	-	-
45	45, 55	+	-
70	44, 70	+	+

The bootstrap can be forced to set up the machine as for a different class of PDP11 by placing an appropriate value in the console switch register (if there is one) while booting it. The value to use is the PDP11 class, interpreted as an *octal* number (use, for example, 070 for an 11/70). **Warning:** some old DEC bootstraps use the switch register to indicate where to boot from. On such machines, if the value in the switch register indicates an incorrect CPU, be sure to reset the switches immediately after initiating the tape bootstrap.

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.

To first check that everything is working properly, you can use the *cat* program on the tape to print the list of utilities on the tape. Through the rest of this section, substitute the correct disk type for *dk* and the tape type for *tp*. In response to the prompt of the bootstrap which is now running, type

tp (0,1)   (load file 1 from tape 0)

Cat will respond

Cat
File?

The table of contents is in file 2 on the tape, therefore answer

tp (0,2)

The tape will move, then a short list of files will print on the console, followed by:

exit called
*nn***Boot**
:

After *cat* is finished, it returns to the bootstrap for the next operation.

2.2. Creating an empty UNIX file system

Now create the root file system using the following procedures. First determine the size of your root file system from the following table:

Disk	Root File System Size (1K-byte blocks)
hk	2970
rk†	2000
rl01†	4000
rl02†	8500
rp	5200
xp	4807 (RP04/RP05/RP06)
	2400 (RM02/RM03)
	4560 (RM05)
	4702 (DIVA)

If the disk on which you are creating a root file system is an **xp** disk, you should check the drive type register at this time to make sure it holds a value that will be recognized correctly by the driver. There are

†These sizes are for full disks less some space used for swapping.

numbering conflicts; the following numbers are used internally:

Drive Type Register Low Byte (standard address: 0776726)	Drive Assumed
022	RP04/05/06
025	RM02/RM03
027	RM05
076	Emulex SC-21/300 Mb RM05 emulation (815 cylinders)
077	Diva Comp-V/300 Mb SMD

Check the drive type number in your controller manual, or halt the CPU and examine this register. If the value does not correspond to the actual drive type, you must place the correct value in the switch register after the tape bootstrap is running and before any attempt is made to access the drive. This will override the drive type register. This value must be present at the time each program (including the bootstrap itself) first tries to access the disk. On machines without a switch register, the *xptype* variable can be patched in memory. After starting each utility but before accessing the disk, halt the CPU, place the new drive type number at the proper memory location with the console switches or monitor, and then continue. The location of *xptype* in each utility is mkfs: 032700, restor: 031570, icheck: 030150 and boot: 0427754 (the location for boot is higher because it relocates itself). Once UNIX itself is booted (see below) you must patch it also.

Finally, determine the proper interleaving factors *m* and *n* for your disk and CPU combination from the following table. These numbers determine the layout of the free list that will be constructed; the proper interleaving will help increase the speed of the file system. If you have a non-DEC disk that emulates one of the disks listed, you may be able to use these numbers as well, but check that the actual disk geometry is the same as the emulated disk (rather than the controller mapping onto a different physical disk). Also, the rotational speed must be the same as the DEC disk for these numbers to apply.

Disk Interleaving Factors for Disk/CPU Combinations (*m*/*n*)

CPU	RK05	RK06/7	RL01/2	RM02	RM03	RM05	RP03	RP04/5/6
11/23	X/12	X/33	X/10	X/80	-	-	X/100	X/209
11/24	X/12	7/33	X/10	10/80	-	-	X/100	10/209
11/34	X/12	6/33	X/10	8/80	-	-	3/100	8/209
11/40	2/12	6/33	X/10	8/80	-	-	3/100	8/209
11/44	X/12	4/33	X/10	6/80	-	-	2/100	6/209
11/45	2/12	5/33	X/10	7/80	-	-	3/100	7/209
11/55	X/12	5/33	X/10	7/80	-	-	3/100	7/209
11/60	X/12	5/33	X/10	7/80	-	-	3/100	7/209
11/70	X/12	3/33	X/10	5/80	7/80	7/304	X/100	5/209

For example, for an RP06 on an 11/70, *m* is 5 and *n* is 209. See *mkfs* (8) for more explanation of the values of *m* and *n*. An X entry means that we do not know the correct number for this combination of CPU and disk. If you do, please let us know. If *m* is unspecified or you have a disk which emulates a DEC disk, use the number for the most similar disk/CPU pair. **If *n* is unspecified, use the cylinder size divided by 2.**

Then run a standalone version of the *mkfs* (8) program. In the following procedure, substitute the correct types for *tp* and *dk* and the size determined above for *size*:

: *tp* (0,3)	
Mkfs	
file system: *dk* (0,0)	(root is the first file system on drive 0)
file system size: *size*	(count of 1024 byte blocks in root)
interleaving factor (m, 5 default): *m*	(interleaving, see above)
interleaving modulus (n, 10 default): *n*	(interleaving, see above)
isize = XX	(count of inodes in root file system)
m/n = *m n*	(interleave parameters)
Exit called	
*nn***Boot**	
:	(back at tape boot level)

You now have an empty UNIX root file system.

2.3. Restoring the root file system

To restore a small root file system onto it, type

: *tp* (0,4)	
Restor	
Tape? *tp* (0,6)	(unit 0, seventh tape file)
Disk? *dk* (0,0)	(into root file system)
Last chance before scribbling on disk.	(just hit return)
(30 second pause then tape should move)	
(tape moves for a few minutes)	
end of tape	
Exit called	
*nn***Boot**	
:	(back at tape boot level)

If you wish, you may use the *icheck* program on the tape, *tp* (0,5), to check the consistency of the file system you have just installed.

2.4. Booting UNIX

You are now ready to boot from disk. It is best to read the rest of this section first, since some systems must be patched while booting. Then type:

:*dk* (0,0)*dk*unix	(bring in *dk*unix off root system)

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

Berkeley UNIX (Rev. 2.9.5) Mon Aug 2 18:44:30 PDT 1983
mem = xxx

CONFIGURE SYSTEM:
(Information about various devices will print;
most of them will probably not be found until
the addresses are set below.)
erase=ˆ?, kill=ˆU, intr=ˆC
#

If you are booting from an *xp* with a drive type that is not recognized, it will be necessary to patch the system before it first accesses the root file system. Halt the processor after it has begun printing the version string but before it has finished printing the ‘‘mem = xxx’’ string. Place the drive type number corresponding to your drive at location 061472; the addresses for drives 1, 2 and 3 are 061506, 061522 and 061536 respectively. If you plan to use any drives other than 0 before you recompile the system, you should patch these locations. Make the patches and continue the CPU. The value before patching must be zero. If it is not, you have halted too late and should try again.

UNIX begins by printing out a banner identifying the version of the system that is in use and the date it was compiled. Note that this version is different from the system release number, and applies only to the operating system kernel.

Next the *mem* message gives the amount of memory (in bytes) available to user programs. On an 11/23 with no clock control register, a message "No clock???" will print next; this is a reminder to turn on the clock switch if it is not already on, since UNIX cannot enable the clock itself. 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 can be edited to correspond to your hardware. However, 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. The file /.profile contained commands to set the UNIX erase, line kill and interrupt characters to be what is standard on DEC systems so that it is consistent with the DEC console interface characters. This is not normal for UNIX, but is convenient when working on a hardcopy console; change it if you like.

UNIX is now running, and the Berkeley PDP-11 UNIX Programmer's manual applies. The '#' is the prompt from the Shell, and lets you know that you are the super-user, whose login name is "root."

There are a number of copies of *unix* on the root file system, one for each possible type of root file system device. All but one of them (*xpunix*) has had its symbol table removed (i.e. they have been "stripped"; see *strip* (1)). The unstripped copy is linked (see *ln* (1)) to */unix* to provide a system namelist for programs like *ps* (1) and *autoconfig* (8). All of the systems were created from */unix* by the C shell script */genallsys.sh*. If you had to patch the *xp* type as you booted, you may want to use *adb* (see *adb* (1)) to make the same patch in a copy of *xpunix*. If you are short of space, you can patch a copy of */unix* instead (setting the rootdev, etc.) and install it as */unix* after verifying that it works. See */genallsys.sh* for examples of using *adb* to patch the system. The system load images for other disk types can be removed. **Do not remove or replace the copy of */unix*, however, unless you have made a working copy of it that is patched for your file system configuration and still has a symbol table.** Many programs use the symbol table of */unix* in order to determine the locations of things in memory, therefore */unix* should always be an unstripped file corresponding to the current system. If at all possible, you should save the original UNIX binaries for your disk configuration (*dk*unix and unix) for use in an emergency.

There are a few minor details that should be attended to now. The system date is initially set from the root file system, and should be reset. The root password should also be set:

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

2.5. Installing the disk bootstrap

The disk with the new root file system on it will not be bootable directly until the block 0 bootstrap program for the disk has been installed. There are copies of the bootstraps in /mdec. This is not the usual location for the bootstraps (that is /usr/src/sys/mdec), but it is convenient to be able to install the boot block now. Use *dd* (1) to copy the right boot block onto the disk; the first form of the command is for small disks (**rk**, **rl**) and the second form for disks with multiple partitions (**hk**, **rp**, **xp**), substituting as usual for *dk*:

dd if=*dk*uboot of=/dev/r*dk*0 count=1

or

dd if=*dk*uboot of=/dev/r*dk*0a count=1

will install the bootstrap in block 0. Once this is done, booting from this disk will load and execute the block 0 bootstrap, which will in turn load /boot (actually, the boot program on the first file system, which is root). The console will print

>boot	(printed by the block 0 boot)
*nn***Boot**	(printed by /boot)
:	

The '>' is the prompt from the first bootstrap. It automatically boots */boot* for you; if */boot* is not found, it will prompt again and allow another name to be tried. It is a very small and simple program, however, and can only boot the second-stage boot from the first file system. Once /boot is running and prints its ": " prompt, boot unix as above, using *dk*unix or unix as appropriate.

2.6. Checking the root file system

Before continuing, check the integrity of the root file system by giving the command

fsck /dev/r*dk*0a

(omit the **a** for an RK05 or RL). The output from *fsck* should look something like:

/dev/r*xx*0a
File System: /

**** Checking /dev/r*xx*0a**
**** Phase 1 - Check Blocks and Sizes**
**** Phase 2 - Check Pathnames**
**** Phase 3 - Check Connectivity**
**** Phase 4 - Check Reference Counts**
**** Phase 5 - Check Free List**
236 files 1881 blocks xxxxx free

If there are inconsistencies in the file system, you may be prompted to apply corrective action; see the document describing *fsck* for information. The number of free blocks will vary depending on the disk you are using for your root file system.

3. DEVICE AND FILE SYSTEM CONFIGURATION

This section will describe ways in which the file systems can be set up for the disks available. It will then describe the files and directories that will be set up for the local configuration. These are the */dev* directory, with special files for each peripheral device, and the tables in */etc* that contain configuration-dependent data. Some of these files should be edited after reading this section, and others can wait until later if you choose. The disk configuration should be chosen before the rest of the distribution tape is read onto disk to minimize the work of reconfiguration.

3.1. Disk configuration

This section describes how to lay out file systems to make use of the available space and to balance disk load for better system performance. The steps described in this section (3.1) are optional.

3.1.1. Disk naming and divisions

Each large physical disk drive can be divided into up to 8 partitions; UNIX typically uses only 3 to 5 partitions. For instance, on an RM03 the first partition, rm0a, is used for a root file system, a backup thereof, or a small file system like /tmp; the second partition, rm0b, is used for swapping or a small file system; and the third partition, rm0c, holds a user file system. Many disks can be divided in different ways; for example, the third section (**c**) of the RM03 could instead be divided into two file systems, using the rm0d and rm0e partitions instead, perhaps holding /usr and the user's files. The disk partition tables are specified in the *ioconf.c* file for each system, and may be changed if necessary. The last partition (**h**) always describes the entire disk, and can be used for disk-to-disk copies.

Warning: for disks on which DEC standard 144 bad sector forwarding is supported, the last track and up to 126 preceeding 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**, **hp**, **rm**, and **xp** drivers. It has been only lightly tested in the latter three cases.

3.1.2. Space available

The space available on a disk varies per device. The amount of space available on the common disk partitions for /usr is listed in the following table. Not shown in the table are the partitions of each drive devoted to the root file system and the swapping area.

Type	Name	Size
RK06	hk?d	9.2 Mb
RK07	hk?c	22.4 Mb
RM02, RM03	rm?c	60.2 Mb
RM02, RM03	rm?d	30.9 Mb
RP03	rp?c	33.3 Mb
RP04, RP05, RP06	hp?c	74.9 Mb
RP06	hp?d	158.9 Mb
RM05	xp?c	115.4 Mb
RM05	xp?e	80.9 Mb

Each disk also has a swapping area and a root file system. The distributed system binaries and sources occupy about 38 megabytes.

The sizes and offsets of all of the disk partitions are in the manual pages for the disks; see section 4 of the Berkeley PDP-11 UNIX Programmer's manual. Be aware that the disks have their sizes measured in "sectors" of 512 bytes each, while the UNIX file system blocks are 1024 bytes each. Thus if a disk partition has 10000 sectors (disk blocks), it will have only 5000 UNIX file system blocks, and you **must** divide by 2 to use 5000 when specifying the size to the *mkfs* command. The sizes and offsets in the kernel (ioconf.c) and the manual pages are in 512-byte blocks. If bad sector forwarding is supported for your disk, be sure to leave sufficient room to contain the bad sector information when making new file systems.

3.1.3. Layout considerations

There are several considerations in deciding how to adjust the arrangement of things on your disks: the most important is making sure there is adequate space for what is required; secondarily, throughput should be maximized. Swapping space is an important parameter. Since running out of swap space often causes the system to panic, it must be large enough that this does not happen.

Many common system programs (the C compiler, the editor, the assembler etc.) create intermediate files in the /tmp directory, so the file system where this is stored also should be made large enough to accommodate most high-water marks; if you have several disks, it makes sense to mount this in a ‘‘root’’ or ‘‘swap’’ (i.e. first or second partition) file system on another disk. On RK06 and RK07 systems, where there is little space in the hk?c or hk?d file systems to store the system source, it is normal to mount /tmp on /dev/hk1a.

The efficiency with which UNIX is able to use the CPU is often strongly affected by the configuration of disks. For general time-sharing applications, the best strategy is to try to split the most actively-used sections among several disk arms. There are at least five components of the disk load that you can divide between the available disks:

1. The root file system.
2. The swap area.
3. The /tmp file system.
4. The /usr file system.
5. The user files.

Here are several possibilities for utilizing 2, 3 and 4 disks:

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

The most important consideration is to even out the disk load as much as possible, and to do this by decoupling file systems (on separate arms) between which heavy copying occurs. Note that a long term average balanced load is not important; it is much more important to have instantaneously balanced load when the system is busy. When placing several busy file systems on the same disk, it is helpful to group them together to minimize arm movement, with less active file systems off to the side.

Intelligent experimentation with a few file system arrangements can pay off in much improved performance. It is particularly easy to move the root, the /tmp file system and the swapping areas. Note, though, that the disks containing the root and swapping area can never be removed while UNIX is running. Place the user files and the /usr directory as space needs dictate and experiment with the other, more easily moved file systems.

As an example, consider a system with RM03s. On the first RM03, **rm0**, we will put the root file system in **rm0a**, and the **/usr** file system in **rm0c**, which has enough space to hold it and then some. If we had only one RM03, we would put user files in the **rm0c** partition with the system source and binaries, or split them between **rm0d** and **rm0e**. The /tmp directory will be part of the root file system, as no file system will be mounted on /tmp.

If we had a second RM03, we would create a file system in **rm1c** and put user files there, calling the file system /mnt. We would keep a backup copy of the root file system in the **rm1a** disk partition, a file system for /tmp on **rm0b**, and swap on **rm1b**.

3.1.4. Implementing a layout

Once a disk layout has been chosen, the appropriate special files for the disk partitions must be created (see Setting up the /dev directory, below). Empty file systems will then be created in the appropriate partitions with *mkfs* (8), and the files belonging in the file system can then be restored from tape. The section on setting up the /usr file system contains detailed information on this process. The swap device is specified when the kernel is configured, which is also discussed later. At that time, you may also want to consider whether to use the root device or another file system (e.g. /tmp) for the pipe device (the pipe device is a file system where the kernel keeps temporary files related to pipe I/O; it should be *mount*ed before any I/O through pipes is attempted).

3.2. Setting up the /dev directory

Devices are accessed through special files in the file system, made by the *mknod* (8) program and normally kept in the /dev directory. Devices to be supported by UNIX are implemented in the kernel by drivers; the proper driver is selected by the major device number and type specified to *mknod* . All devices supported by the distribution system already have nodes in /dev. They were created by the /dev/MAKE shell script. It is easiest to rebuild this directory from the beginning with the correct devices for your configuration. First, determine the UNIX names of the devices on your system (e.g. *dh*, *lp*, *xp*). Some will be the same as the names of devices on the generic system. Others need not be. See section 4 of the UNIX Programmer's Manual. Next create a new directory /newdev, copy /dev/MAKE into it, edit MAKE to provide an entry for local needs, replacing the case LOCAL, and run it to generate the desired devices in the /newdev directory. The LOCAL entry can be used for any unusual devices, and to rename standard devices as desired. It should also move the node for the disk partition being used as the swap area to *swap* (or, if swap is after a file system as on RK05 or RL disks, link the other node to *swap*). Different devices are specified to MAKE in various ways. Terminal multiplexors (DZ and DH) are specified by boards, and 8 or 16 nodes will be made, as appropriate. Disks are made by partition, for example xp0c, so that you may make the nodes corresponding to the file systems that you intend to use. Note that *hp*, *rm* and *xp* are actually synonyms, but you should use the name corresponding to the driver you plan to use. The kernel configuration section (section 5.4.1) has more information. For tape drives, there are different invocations for different types of controllers, although the nodes produced will have the same names. The different types are *ht*, *tm* and *ts*, as above, and also *ut*, which is used for the Emulex TC-11 and other TM-11 emulations that are also capable of selecting 1600 or 800 bpi under software control. Making *ht0* or *ut0* will result in nodes *mt0* and *mt1* (800 and 1600 bpi, respectively) and parallel nodes for other options; *ht1* uses the names *mt2* and *mt3*. See *ht* (4) and *tm* (4). In contrast, the MAKE script makes only one set of nodes for *tm* or *ts*, without changing the unit number specified. Different sites use different naming conventions for tapes; you could use the LOCAL entry in MAKE to move the tape files to your favorite names.

As an example, if your machine had a single DZ-11, two DH-11s, an RP03 disk, two RP06 disks, and a TM03 tape formatter you would do:

```
# cd /
# mkdir newdev
# cp /dev/MAKE /newdev/MAKE
# cd newdev
# ./MAKE dz0 dh1 ht0 std LOCAL
# ./MAKE rp0a rp0b rp0c hp0a hp0b hp0c hp1a hp1b hp1d hp1e
```

Note the ‘‘std’’ argument here that causes standard devices such as *console*, the console terminal, to be created.

You can then do

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

to install the new device directory. Once you are confident that the new directory is set up properly, you can remove /genericdev.

3.3. Editing system-dependent configuration files

There are a number of small files in */etc* that are used by various programs to determine things about the local configuration. At this point, several of these should be edited to describe the local configuration. You may have old versions of some of them which you may want to use, or you may edit the files that are provided as examples. Some of this may be done later at your convenience, but is presented here for organization. Both **/etc/dtab** and **/etc/fstab** should be edited now.

3.3.1. /etc/dtab

This file contains the list of devices which will be checked at boot time by *autoconfig* (8). The devices that are listed are tested to see whether they exist and have the correct register addresses and interrupt vectors. If they do, and the kernel has a corresponding driver routine, *autoconfig* notifies the driver that the device exists at that address. In this way, the addresses and vectors of most devices do not need to be compiled into the operating system. The exception is that disks must be preconfigured if they are to be used as root file systems.

This file should be edited to include all of the devices on the system with the exception of the clock and console device. Other device entries can be deleted or commented out with a '#' at the beginning of the line. The format of the entries is defined in *dtab* (5). *Autoconfig* (8) describes the autoconfiguration process. One word of caution: if a device fails to interrupt as expected, and if its unit number is specified (not a '?' wildcard), *autoconfig* will notify the driver that the device is **not** present, and preconfigured devices (like root disks) could be disconnected. Thus, it is probably best to use a '?' instead of a unit number for your root disks until you are confident that the probe always finds that disk, especially if your disk controller is an emulation of another disk type. Disks that are not used as boot devices for UNIX can be properly listed with unit numbers.

3.3.2. /etc/fstab

This file contains the list of file systems normally mounted on the system. Its format is defined in *fstab* (5). Programs like *df* (1) and *fsck* (8) use this list to control their actions. Each disk partition that has been assigned a function should be listed here. See the manual pages for specifics on how to configure this file.

3.3.3. /etc/ident

The banner printed by *getty* (8) is read from /etc/ident. Edit this file to the banner you wish to use. It may contain special characters to clear terminal screens, etc., but note that the same file is used for all terminals.

3.3.4. /etc/motd

The contents of /etc/motd, the ‘‘message of the day,’’ is displayed at the terminal when a user is logged in by *login* (1).

3.3.5. /etc/passwd, /etc/group

These files obviously need local modifications. See the section on adding new users. Entries for pseudo-users (user IDs that are not used for logins) have password fields containing ‘‘***’’, since encrypted passwords never not contain asterisks.

3.3.6. /etc/rc

As the system begins multiuser operations, it executes the commands in /etc/rc (see *init* (8)). Most of the commands in this file are standard and should not be changed, including the section for checking file systems after a reboot. These commands will be ignored if autoreboot is not enabled. You should edit /etc/rc to set your machine's name. Look for the line

```
/etc/hostname hostnameunknown
```

and change *hostnameunknown* to the name of your machine. This name will be used by *Mail* (1) and *uucp* (1) (among others) and should correspond to the name by which your machine is known to external networks (if any). At this time you may wish to add additional commands to this file if you need to start additional daemons, remove old lock files, or perform any other cleanup as the system comes up.

3.3.7. Configuring terminals

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

Terminals connected via DZ interfaces are conventionally named **tty***dd* where *dd* is a decimal number, the ‘‘minor device’’ number. The lines on dz0 are named /dev/tty00, /dev/tty01, ... /dev/tty07. Lines on DH interfaces are conventionally named **ttyh***x*, where *x* is a hexadecimal digit. If more than one DH interface is present in a configuration, successive terminals would be named **ttyi***x*, **ttyj***x*, etc.

To add a new terminal be sure the device is configured into the system, that the special file for the device has been made by /dev/MAKE, and the special file exists. Then set the first character of the appropriate line of /etc/ttys to 1 (or add a new line). The first character may also be 3 if the line is also to be used in maintenance mode (see *init* (8)).

The second character of each line in the /etc/ttys file lists the speed and initial parameter settings for the terminal. The most common choices, from *getty* (8), are:

0	300-1200-150-110
3	1200-300
4	300 (e.g. console)
5	300-1200
6	1200
7	2400
8	4800
9	9600
B	autobaud

Here the first speed is the speed a terminal starts at, and ‘‘break’’ switches speeds. Thus a newly added terminal /dev/tty00 could be added as

```
19tty00
```

if it was wired to run at 9600 baud. The ‘‘B’’ indicates that *getty* should attempt to guess a line’s speed when the user types a carriage return or control-C. Note that this requires kernel support. See section 5.3.6 below.

Dialup terminals should be wired so that the 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 always appears to be present, or (for lines on a DH-11 or DZ-11) add 0200 to the minor device number to indicate that carrier is to be ignored. See *dh* (4) and *dz* (4) for details.

You should also edit the file /etc/ttytype placing the type of each terminal there (see *ttytype* (5)).

When the system starts running multi-user, all terminals that are listed in /etc/ttys having a 1 or 3 as the first character of their line are enabled. If, during normal operations, it is desired to disable a terminal line, the super-user can edit the file /etc/ttys, change the first character of the corresponding line to 0 and then send a hangup signal to the *init* process, by typing (see *kill* (1))

```
# kill −1 1
  or
# kill −HUP 1
```

Terminals can similarly be enabled by changing the first character of a line from a 0 to a 1 and sending a hangup to *init*.

Note that if a special file is inaccessible when *init* tries to create a process for it, init will print a message on the console and try to reopen the terminal every minute, reprinting the warning message every 10 minutes.

Finally note that you should change the names of any dialup terminals to ttyd? where ? is in [0-9a-f] since some programs use this property of the names to decide whether a terminal is a dialup. Shell commands to do this should be put in the /dev/MAKE script under case LOCAL.

4. SETTING UP THE /usr FILE SYSTEM

The next step in bringing up the 2.9BSD distribution is to read in the binaries and sources on the /usr file system. This will also demonstrate how to add new file systems in general, and the overall procedure can be repeated to set up additional file systems. There are two portions of the /usr file system, one on each tape. The first tape contains the binary directories, manual pages and documentation, as well as skeletal directories such as spool and msgs. If you have room, it is easiest to extract everything. The size of the entire /usr file system image on the distribution tapes is 38 megabytes. It will not fit on a single RK05, RK06/7 or RL01/2. In these cases, the /usr file system will have to be extracted in sections or split across multiple disks. The *bin*, *include*, *lib*, and *ucb* subdirectories are essential. The system sources will also be needed to reconfigure the kernel; they are in /usr/src/sys. The *adm*, *dict*, *msgs*, *preserve*, *spool*, *sys* and *tmp* directories may also be extracted to provide a skeletal system. The first part of this section describes how to extract /usr as part of a full bootstrap; the second part explains how to install 2.9BSD as an upgrade to a 2.8BSD system if you decide not to perform a full bootstrap.

4.1. Full bootstrap procedure

This procedure will create a new file system and extract the /usr directory into it. First determine the name of the disk on which you plan to place the new file system, for example rm0c, and substitute it for *disk* throughout this section. You may want to create a small ‘‘prototype’’ file to describe the file system (see *mkfs* (8)) in order to change the size of the inode list. This is the same as the maximum number of files that can be created on the file system. The default is to allow 16 inodes (occupying one block) per 24 file system blocks, allowing the file system to be completely filled with small files (1-2 blocks). This is more than required for /usr and other file systems which have larger average file size. If you decide to set up a prototype file for *mkfs*, use its name for *proto* below. The prototype file needs to contain only the name of the bootstrap, the sizes, and the line for the root directory (don’t forget the ’$’ to terminate). Look up the correct size for this file system in the manual section for the disk. Note that the size given to *mkfs* is in file system blocks of 1024 bytes, and thus the sizes in the manual page will have to be divided by 2. If not using a prototype file, substitute the size for *proto* in the mkfs command below. Finally, recall the interleaving parameters *m* and *n* that you used in making the root file system. They are in the table in section 2.2. Comments are enclosed in ( ); don’t type these. Then execute the following commands (substituting *rmt1* and *nrmt1* for *rmt0* and *nrmt0* respectively if you have a 1600 bpi tape on an ht or tm controller):

# /etc/mkfs /dev/r*disk proto m n*	(create empty user file system)
isize = *nnnnn*	(the count of available inodes)
m/n = *m n*	(free list interleave parameters)
	(this takes a few minutes)
# /etc/mount /dev/*disk* /usr	(mount the usr file system)
# cd /usr	(make /usr the current directory)
	(make sure that the first tape is mounted)
# mt -t /dev/nrmt0 fsf 7	(skip first seven tape files)
# tar xpf /dev/rmt0	(extract the /usr file system binaries)
	(this takes about 20 minutes)
	(now mount the second tape)
# tar xpf /dev/rmt0	(extract the /usr file system sources)
	(this takes another 20 minutes)

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

# cd /	(back to root)
# /etc/umount /dev/*disk*	(unmount /usr)
# fsck /dev/r*disk*	

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

/etc/mount /dev/*disk* /usr

If you are installing the distribution on a PDP11/44, 11/45, or 11/70 (machines with separate instruction and data space) you should test and install the separate I/D versions of csh, ex, etc. in /usr/70. Note, however, that these binaries assume the existence of hardware floating point support.

4.2. Bootstrap path 2: upgrading 2.8BSD

Begin by reading the other parts of this document to see what has changed since the last time you bootstrapped the system. Also look at the new manual sections provided to you. If you have local system modifications to the kernel to install, look at the document "Changes in the Kernel in 2.9BSD" to get an idea of how the system changes will affect your local mods. Disclaimer: there are a very large number of changes from 2.8BSD to 2.9. This section may not be complete, and if a new program fails to work after being recompiled, you may find that additional libraries or other components may also need to be updated.

There are 6 major areas of changes that you will need to incorporate to convert to the new system:

1. The new kernel and the associated programs that implement job control or read kernel memory: autoconfig, csh, the jobs library, login, ps, pstat, w, etc.
2. The programs related to system reboots and shutdowns.
3. The programs directly related to user text overlays: adb and ld.
4 The C compiler driver, C preprocessor, and assembler.
5 The new version of the standard I/O library.
6. Other programs with significant bug fixes, significant improvements, or which were previously unavailable because they had not been overlaid.

Here is a step-by-step guide to converting. Before you begin you should do a full backup of your root and /usr file systems as a precaution against irreversible mistakes.

1. Set the shell variable "nbsd" to the name of a directory where an empty file system can be mounted and a quantity of material from the tape (you should allow for about 38 megabytes) can be extracted. Choose a disk of sufficient size to hold this quantity of material, make a file system, and mount $nbsd on this disk. Next, restore (see *restor*(8)) the root file system dump image to this disk. Finally, change directory to "$nbsd/usr", and extract the eighth file from the first distribution tape and all of the second tape using *tar* (see *tar*(1)).
2. Install the new include files by copying $nbsd/usr/include/*.h to /usr/include and $nbsd/usr/include/sys/*.h to /usr/include/sys. Install the C compiler driver from the new system by copying $nbsd/bin/cc to /bin/cc. Install the assembler from the new system by copying $nbsd/bin/as to /bin/as and $nbsd/lib/as2 to /lib/as2. Install the new C preprocessor by copying $nbsd/lib/cpp to /lib/cpp. Install the new versions of adb and ld by copying $nbsd/bin/adb and $nbsd/bin/ld to /bin.
3. Reconfigure the system in $nbsd/usr/src/sys to correspond to your configuration according to the instructions in section 5.
4. Put in the new versions of the following programs:

   /bin: csh, kill, login, iostat, ps, pstat, vmstat

   /etc: autoconfig, fsck, init, mount, reboot, savecore, shutdown, umount

   /usr/ucb: ex, w

   Merge any local changes to /etc/rc into $nbsd/etc/rc. Put the resulting file in /etc/rc. Create the directory /usr/sys and perhaps some files in this directory (read *savecore*(8)). Make a device description file for *autoconfig*. See *dtab*(5) and *autoconfig*(8).
5. Try bootstrapping the new system; it should now work. Make sure to write new instructions to your operators.
6. Incorporate some other important bug fixes or enhancements:
   a) Replace the file tmac.an in the directory /usr/lib/tmac with the version from $nbsd/usr/lib/tmac. Replace the file /usr/lib/me/local.me with the version from $nbsd/usr/lib/me; copy

$nbsd/usr/lib/me/refs.me to /usr/lib/me.

b) Install the new C library source, /usr/src/lib/c, rebuild and reinstall /lib/libc.a and /usr/lib/libovc.a.

c) Install the jobs library, /usr/src/lib/jobs and build and install /usr/lib/libjobs.a and /usr/lib/libovjobs.a.

d) Replace the directory /usr/src/cmd/refer. Then rebuild and reinstall the programs.

e) Install the new Mail source, /usr/src/ucb/Mail and reinstall /usr/ucb/Mail.

f) If the target machine is a nonseparate I/D CPU, install the new *lex* and *yacc* directories, compile and install the programs.

g) Install the new version of *tar* from $nbsd/usr/src/cmd/tar.c and also the program *mt* from $nbsd/usr/src/ucb/mt.c.

h) Merge your changes to /usr/src/ucb/termcap/reorder and reinstall the terminal data base, /etc/termcap. Install the new terminal library, /usr/src/ucb/termlib, remake and reinstall /usr/lib/libtermcap.a and /usr/lib/libovtermcap.a. Then make and install the new version of *ex.*

i) If you want the new version of the Pascal system incorporating overlays (for nonseparate I/D CPUs), remake the directories *pi* and *px* in $nbsd/usr/src/cmd and install the programs.

j) Install the new F77 compiler, /usr/src/cmd/f77, and the new libraries, /usr/src/lib/lib*77. Then remake and reinstall them.

k) Install the new library sources, /usr/src/lib/{ape,curses,m,mp,plot} and remake and reinstall the new libraries.

l) Install new versions of as many of the following programs as you choose: 512dumpdir, 512restor, atrun, cat, catman, ccat, compact, checkobj, ctags, df, diff, du, egrep, error, expand, fgrep, find, from, grep, hostname, jove, l11, lint, ln, lock, login, lpr, ls, m11, make, man, mkfs, more, msgs, mv, ncheck, printenv, pq, ranm, rewind, rm, rmdir, sed, setquota, size, sort, split, sq, strings, strip, stty, sysline, tail, tbl, tset, ul, uncompact, unexpand, vsh, wc.

m) Install the modified or new administrative programs: ac, getty, last.

n) Install some security fixes in the mail systems by installing new sources for berknet (/usr/src/ucb/berknet), delivermail (/usr/src/ucb/delivermail), mail (/usr/src/cmd/mail.c), and secret mail (/usr/src/cmd/xsend), and remaking and reinstalling the new binaries.

o) Install the new version of uucp (/usr/src/cmd/uucp).

p) Install the news (/usr/contrib/news) or notes (/usr/contrib/notes) bulletin board system if you wish.

q) Install the new *eqn* (1) symbol macros, /usr/public/eqnSyms.

r) Install manual pages corresponding to the new and changed programs.

s) Remove the old programs /bin/ovas, /bin/ovld, /lib/ovas2, and /bin/ovadb. Remove the libucb-path library. Remove the old version of reset and link the new version of tset to reset.

5. CONFIGURING AND COMPILING THE KERNEL

This section describes procedures used to set up a PDP-11 UNIX kernel (operating system). It explains the layout of the kernel code, compile time options, how files for devices are made and drivers for the devices are configured into the system and how the kernel is rebuilt to include the needed drivers. Procedures described here are used when a system is first installed or when the system configuration changes. Procedures for normal system operation are described in the next section. We also suggest ways to organize local changes to the kernel.

5.1. Kernel organization

The kernel source is kept in the subdirectories of /usr/src/sys. The directory /usr/src/sys/sys contains the mainline kernel code, implementing system calls, the file system, memory management, etc. The directory /usr/src/sys/dev contains device drivers and other low-level routines. The header files and scripts used to compile the kernel are kept in /usr/src/sys/conf, and are copied from there into a separate directory for each machine configuration. It is in this directory, /usr/src/sys/*machine*, that the kernel is compiled.

5.2. Configuring a System

The kernel configuration of each PDP-11 UNIX system is described by a set of header files (one for each device driver) and one file of magic numbers (ioconf.c) stored in a subdirectory of /usr/src/sys for each configuration. Pick a name for your machine (call it PICKLE). Then in the /usr/src/sys/conf directory, create a configuration file PICKLE describing the system you wish to build, using the format in *config* (8). This is most easily done by making a copy of the GENERIC file used for the distributed UNIX binary. Many of the fields in the configuration file correspond to parameters listed in the remainder of this section, which should be scanned before proceeding. See especially section 5.4.3 on how to set up automatic reboots and dumps. Then use *config* to create a system directory ../PICKLE with "config PICKLE." Note the difference between *config* and *autoconfig*. *Config* sets up a directory in which the kernel will be compiled, with all of the system-specific files used in compilation, and specifies what devices will potentially be supported. *Autoconfig* adapts the running kernel to the hardware actually present, by testing and setting the register addresses and interrupt vectors.

Config does most of the work of configuration, but local needs will dictate some changes in the options and parameters in the header files. All of the options are listed in the next section. Examine whoami.h, localopts.h, param.h, and param.c and make any changes required; it might also be wise to look through the header files for the devices that you have configured, to check any options specific to the device drivers that are listed there. After you have finished configuring a kernel and tested it, you should install whoami.h in /usr/include, and copy localopts.h and param.h into /usr/include/sys. This will allow user-level programs to stay in sync with the running kernel.

If you wish to change any disk partition tables or device control status register addresses (other than those configured at boot time by *autoconfig* (8)), edit ioconf.c and change the appropriate line (s). The file l.s contains the interrupt vectors and interface code and may also be edited if necessary, but usually will require no change. Both c.c and l.s include support for all normal devices according to the header files per device, and with autoconfiguration, the actual vectors need not be specified in advance. Finally, examine the Makefile, especially the options near the top and the load rules. If you have placed the include files in the standard directories, you shouldn't have to make any changes to the options there.

The following sections give short descriptions of the various compile-time options for the kernel, and more extensive information on the autoreboot and disk monitoring setup. After verifying that those features are configured correctly for your system, you can proceed to kernel compilation.

5.3. Compile Time Options

The 2.9BSD kernel is highly tunable. This section gives a brief description of the many compile-time options available, and references to sections of the Berkeley PDP-11 UNIX Programmer's manual where more information can be found. Options fall into four categories; the letters following each will be used to mark the options throughout the rest of this section.

Standard (S)	These include options which we consider necessary for reasonable system performance or resiliency.
Desirable (D)	These include many other features that are convenient but which may be turned off if system size is critical. The user programs and libraries distributed with 2.9BSD generally assume that these are turned on, so turning them off may necessitate recompiling libraries or programs. These options, along with those designated ‘‘standard’’, have received the most thorough testing.
Configuration Dependent (C)	Options that depend on such things as the physical configuration or speed issues fall into this category.
Experimental (X)	New features that have not been well tested, options that have known problems, or ones that we do not normally use are listed as experimental. You should not use such options unless the problems listed are not considerations for your system, or you are willing to watch things closely and possibly do some debugging.

The following sections list the parameters and options used in the kernel. The parameters (section 5.3.2) have numeric values, usually table sizes, and most of them are in param.h or param.c. Those that are in param.h are typically not changed, with the possible exception of **MAXMEM**, as their values are set by convention. The option flags are either defined or undefined to enable or disable the corresponding feature, with the exception of **UCB_NKB**, which is unlikely to change. Each option is marked with a letter to indicate into which of the four categories above it falls.

5.3.1. Hardware

ENABLE34	**X**	Automatically detect and support Able Computer’s ENABLE/34† memory management board. This option implies **UNIBUS_MAP**.
NONFP	**C**	Do not compile in code to automatically detect and support an FP11 floating point processor. Also, include a fast illegal-instruction trap handler and modify the signal routines to make it possible to run programs using the floating-point interpreter under trace.
NONSEPARATE	**C**	Do not attempt to support separate I/D user programs.
PARITY	**C**	Recognize and deal with cache and memory parity traps.
PDP11	**C**	This should be set to the CPU type of the target machine (23, 24, 34, 40, 44, 45, 60, 70, or GENERIC). You should use 34 for an 11/34A and 45 for an 11/55. GENERIC should be used to build a system which runs on a variety of CPUs. It was used to make the distributed kernels. **MENLO_KOV** and **NONSEPARATE** are defined if **PDP11** is 23, 24, 34, 40, or 60. **MENLO_KOV** is also defined if **PDP11** is GENERIC. **UNIBUS_MAP** is defined if **PDP11** is 44, 70, or GENERIC.
SMALL	**C**	Use smaller (by about a factor of 8) queues and hash tables.
UNIBUS_MAP	**C**	Compile in code to detect (and support if present) a UNIBUS map.

5.3.2. Parameters

5.3.2.1. Global configuration

MAXUSERS	This is the maximum number of users the system should normally expect to support. *Config* sets this from the corresponding field in the description file; the definition is copied into the system Makefile rather than a header file. It is not intended to be a hard limit. It is used in sizing other parameters

†ENABLE/34 is a trademark of Able Computer, Inc.

(**CMAPSIZ**, **NFILE**, **NINODE**, **NPROC**, **NTEXT**, and **SMAPSIZ**). The formulae are found in *param.c*. Reasonable values for **MAXUSERS** might be 3 or 4 on a small system (11/34, 11/40), 15 for an 11/44 with a reasonable amount of memory, and 15-30 for an 11/70 system.

TIMEZONE The number of minutes westward from Greenwich. *Config* sets this from the corresponding field in the description file. Examples: for Pacific Standard time, 8 (* 60); for EST, 5.

DSTFLAG Should be 1 if daylight savings time applies in your locality and 0 otherwise. *Config* sets this from the field in the description file.

HZ This is the line clock frequency (e.g. 50 for a 50 Hz. clock).

5.3.2.2. Tunable parameters

CMAPSIZ This is the number of fragments into which memory can be broken. If this number is too low, the kernel's memory allocator may be forced to throw away a section of memory being freed because there is no room in the map to hold it. In this case, a diagnostic message is printed on the console. Normally scaled automatically according to **MAXUSERS**.

MAXMEM This sets an administrative limit on the amount of memory a process may have. It is specified as (*nn**16), where the first number is the desired value in kilobytes (the product is in clicks). This number is usually considerably lower than the theoretical maximum (304 Kb for a nonseparate I/D CPU, 464 Kb for a separate I/D CPU, assuming **MENLO_OVLY** is defined). Normal values are 128 Kb if there is no UNIBUS map (maximum physical memory 248 Kb), otherwise 200 Kb.

NBUF This sets the size of the system buffer cache. It can be no greater than 248. If **UCB_NKB** is defined, these are 1024 byte buffers. Otherwise, they are 512 byte buffers. The buffers are not in kernel data space, but are allocated at boot time. Normally scaled automatically according to **MAXUSERS**, but should be examined in the light of the disk load and amount of memory. For a small to medium system, around 20 buffers should be sufficient; a large system with many disks might use 40 to 60 or more.

NCALL This is the maximum number of simultaneous callouts (kernel event timers). Callouts are used to time events such as tab or carriage return delays. Normally scaled automatically according to **MAXUSERS**.

NCLIST This is the maximum number of clist segments. Clists are small buffer areas, used to hold tty characters while they are being processed. If **UCB_CLIST** is defined, they are not in kernel data space, and this number must be less than 512 if you are using 14 character clists (the default), or 256 for 30 character clists. (The clist size, **CBSIZE**, is in param.h.)

NDISK This is the maximum number of disks and controllers for which I/O statistics can be gathered. See *iostat*(8). Care must be taken that this is large enough for the parameters for each disk (*XX*_DKN and number of disks; see the section on disk monitoring).

NFILE This sets the maximum number of open files. An entry is made in this table each time a file is "opened" (see *creat*(2)), *open*(2)). Processes share these table entries across forks (see *fork*(2), *vfork*(2)). Normally scaled automatically according to **MAXUSERS**.

NINODE This sets the size of the inode table. There is one entry in the inode table for each open file or device, current working or root directory, saved text segment, active quota node (if **UCB_QUOTAS** is defined), and mounted file system. Normally scaled automatically according to **MAXUSERS**.

NMOUNT		This indicates the maximum number of mountable file systems. It should be large enough that you don't run out at inconvenient times.
NPROC		This sets the maximum number of active processes. Normally scaled automatically according to **MAXUSERS**.
NTEXT		This sets the maximum number of active shared text images (including inactive saved text segments). Normally scaled automatically according to **MAXUSERS**.
SMAPSIZ		This is the analogy of **CMAPSIZ** for secondary memory (swap space). Normally scaled automatically according to **MAXUSERS**.

5.3.2.3. Parameters that are set by convention

CANBSIZ		This sets the maximum size of a terminal line input buffer. If using the old tty line discipline, exceeding this bound causes *all* characters to be lost. In the new tty line discipline, no more characters are accepted until there is room. Normally 256.
MAXSLP		This is the maximum time a process can sleep before it is no longer considered a "short term sleeper." It is used only if **UCB_METER** is defined. Normally 20.
MAXUPRC		This sets the maximum number of processes each user is allowed. Normally 20, but can be lower on heavily loaded systems.
MSGBUFS		This is the number of characters saved from system error messages. It is actually the size of circular buffer into which messages are temporarily saved. It is expected that *dmesg* (8) will be run by *cron* (8) frequently enough that no message is overwritten before it can be saved in the system error log. Normally 128.
NCARGS		This is the maximum size of an *exec* (2) argument list (in bytes). Normally 5120.
NOFILE		This sets the maximum number of open files each process is allowed. Normally 20.
SINCR		The increment (in clicks) by which a process's stack is expanded when a stack overflow segmentation fault occurs. Normally 20.
SSIZE		The initial size (in clicks) of a process's stack. This should be made larger if commonly run processes have large data areas on their stacks. Normally 20.

5.3.3. General Options

ACCT	**D**	Enable code which (optionally) writes an accounting record for each process at exit. See *lastcomm* (1), *sa* (1), *acct* (2), *accton* (8).
CGL_RTP	**C**	Support a system call which marks a process as a "real time" process, giving it higher priority than all others. See *rtp* (2).
DIAGNOSTIC	**C**	Turn on more stringent error checking. This enables various kernel consistency checks which are considered extremely unlikely to fail. It is useful when the system is inexplicably crashing.
INSECURE	**C**	Do not turn off the set-user-id or set-group-id permissions on a file when it is written.
MENLO_JCL	**D**	Support reliable signal handling and enhanced process control features. See *sigsys* (2j), *jobs* (3j), *sigset* (3j). This option requires **UCB_NTTY**.
MENLO_KOV	**C**	Support automatic kernel text overlays. This is required for nonseparate I/D systems and is defined automatically if **PDP11** is defined to be 23, 24, 34, 40, 60, or GENERIC.

MENLO_OVLY	**D**	Support automatic user text overlays. This is required in order to run certain programs (e.g. *ex* version 3.7 or, on nonseparate I/D systems, the process control C shell).
OLDTTY	**C**	Support the standard V7 tty line discipline (see *tty*(4)). This must be defined if **UCB_NTTY** is not defined.
UCB_AUTOBOOT	**D**	Allows the kernel to automatically reboot itself, either on demand (see *reboot*(2) and *reboot*(8)) or after *panic*s. This option requires a little planning; see section 5.4.3. **This option requires UCB_FSFIX.**
UCB_CLIST	**C**	Map clists out of kernel virtual data space. If there is sufficient space in kernel data for an adequate number of clists, this option should not used. Mostly used on large systems, or on systems where kernel data space is tight.
UCB_GRPMAST	**C**	Allow one user to be designated a ‘‘group super-user,’’ able to perform various functions previously restricted to root or the file’s owner alone. In the kernel, users whose group and user ids are the same are granted the same permissions with respect to files in the same group as is the owner. User level software implements other permissions, allowing the group super-user to change the password of a user in the same group. The most common use for this is in allowing teaching assistants to oversee students.
UCB_NET	**X**	Enable code implementing a PDP-11 port of Berkeley’s version of TCP/IP. The code is experimental and the implementation is incomplete.
UCB_NTTY	**S**	Support the Berkeley tty line discipline (see *tty*(4) and *newtty*(4)). This must be defined if **OLDTTY** is not defined.
UCB_PGRP	**C**	Fix a bug in the way standard V7 counts a user’s processes. This should be enabled only if **MENLO_JCL** is undefined, since the notion of process groups is completely different in the two cases. If **UCB_PGRP** and **MENLO_JCL** are both defined, the limit on the number of processes allowed per user (**MAXUPRC**) is effectively eliminated.
UCB_SCRIPT	**X**	Allow scripts to specify their own interpreters. For example, executing a script beginning with ‘‘#! /bin/sh’’ causes /bin/sh to be executed to interpret the script. This is not (yet) the same as the facility on 4.1BSD VMUNIX, and probably needs a little work. The Bourne shell, /bin/sh, would need modification also.
UCB_UPRINTF	**D**	Write error messages directly on a user’s terminal when the user causes a file system to run out of inodes or free blocks, or on certain mag tape errors.
UCB_VHANGUP	**D**	Support a system call which allows *init*(8) to revoke access to a user’s terminal when the user has logged out. This is used to give new users ‘‘clean’’ terminals on login.
VIRUS_VFORK	**D**	Implement a much more efficient version of fork in which parent and child share resources until the child *exec*s. See *vfork*(2). Note that this changes the way processes appear in memory. It makes swap operations slower, and thus might not be desirable on systems which swap heavily.

5.3.4. File system

INTRLVE	**X**	Allows interleaving of file systems across devices. See *intrlve*(4).
MPX_FILS	**X**	Include code for the V7 multiplexer. The code is buggy and unsupported.
UCB_FSFIX	**S**	Ensure that file system updates are done in the correct order, thus making damaged file systems less likely and more easily repairable. **This option is required by UCB_AUTOBOOT (actually, by the −p option of *fsck*(8), which makes certain assumptions about the state of the file systems).**

UCB_SYMLINKS	**C**	Add a new inode type to the file system: the symbolic link. Symbolic links cause string substitution during the pathname interpretation process. See *ln* (1), *readlink* (2), and *symlink* (2).
UCB_NKB	**S**	Use file system blocks of *N* KB, normally 1. Changes the fundamental file system unit from 512 byte blocks to 1024 byte blocks (with a corresponding reduction in the size of in-core inodes). This increases file system bandwidth by 100%. Note that **UCB_NKB** is not boolean, but is defined as 1 for 1KB blocks. Other values are possible, but require additional macro definitions. All file systems would have to be remade with new versions of *mkfs* and *restor*. **All supplied software expects this option to be enabled.**
UCB_QUOTAS	**C**	Support a simplistic (and easily defeated) dynamic disk quota scheme. See *ls* (1), *pq* (1), *quota* (2), and *setquota* (8).

5.3.5. Performance Monitoring

DISKMON	**C**	Keep statistics on the buffer cache. They are printed by the −*b* option of *iostat* (8).
UCB_LOAD	**D**	Enable code that computes a Tenex style load average. See *la* (1), *gldav* (2), *loadav* (3).
UCB_METER	**D**	Keep statistics on memory, queue sizes, process states, interrupts, traps, and many other (possibly useful) things. See *vmstat* (1) and section 7.5 of this paper.

5.3.6. Device Drivers

In this section, an **XX_** prefix refers to the UNIX name of the device for which the option is intended to be enabled. For example, **TM_IOCTL** refers to mag tape *ioctl*s in tm.c. Most of these definitions go in the header file *xx.h* for the device. The exceptions are **BADSECT**, **MAXBAD**, **UCB_DEVERR**, and **UCB_ECC**.

BADSECT	**C**	Enable bad-sector forwarding. Sectors marked bad by the disk formatter are transparently replaced when read or written. Currently, only the hk driver's code has been thoroughly tested.
DDMT	**C**	Currently used only by the tm driver. Should be defined if you have a TM-11 emulator which supports 800/1600 bpi dual density drives with software selection.
DZ_PDMA	**C**	Configure the dz driver to do pseudo-dma.
MAXBAD	**C**	This sets the maximum number of replacement sectors available on a disk supporting DEC standard bad sector forwarding. It can be no larger than 126 but may be smaller to reduce the size of kernel data space. See the include file */usr/include/sys/dkbad.h*.
TEXAS_AUTOBAUD	**C**	Support an *ioctl* which defeats detection of framing or parity errors. This is used by *getty* (8) to accurately guess a line's speed when a carriage return is typed.
UCB_DEVERR	**D**	Print device error messages in a human readable (mnemonic) format.
UCB_ECC	**C**	Recognize and correct soft ecc disk transfer errors.
VP_TWOSCOMPL	**C**	Used in the Versatec (vp) driver. If defined, the byte count register will be loaded with the twos-complement of the byte count, rather than the byte count itself. Check your controller manual to see whether your controller requires this.
XX_IOCTL	**D**	Turn on optional *ioctl*s for the corresponding device. See section 4 of the Berkeley PDP-11 UNIX Programmer's manual for details.

XX_SILO **D** Used in the dh and dz drivers. If defined, the drivers will use silo interrupts to avoid taking an interrupt for each character received.

XX_SOFTCAR **C** Currently used only by the dh and dz drivers. Should be defined if not all of the lines on a DH-11 or DZ-11 use modem control. It allows one to select lines on which modem control will be disabled. See *dh*(4) and *dz*(4). It can also be used with escape-code autodialers to allow modem control to be ignored while talking to the dialer.

XX_TIMEOUT **D** Enable a watchdog timer. This is used to kick devices prone to losing interrupts. It is currently available only for the tm driver.

5.3.7. Miscellaneous System Calls

UCB_LOGIN **C** Support a system call which can mark a process as a ‘‘login process’’ and set its recharge number (for accounting purposes). This is usually done by *login*(1). See *login*(2).

UCB_RENICE **D** Support a system call which allows a user to dynamically change a process’s ‘‘nice’’ value over the entire range (-127 to 127) of values. See *renice*(1) and *renice*(2).

UCB_SUBM **C** Support a system call to mark a process as having been ‘‘submitted,’’ permitting it to run after the user has logged out and enabling special accounting for its CPU use. See *submit*(1) and *submit*(2). If this option is enabled, *init*(8) sends a SIGKILL signal to a user’s unsubmitted processes when that user logs out. It is ineffective if **MENLO_JCL** is defined.

5.3.8. Performance Tuning

NOKA5 **C** Simplify the code for kernel remapping by assuming that KDSA5 will not be used for normal kernel data. Kernel data space must end before 0120000 if this option is enabled. It is unfortunate but unavoidable that one must first make a kernel and size it to determine whether this option may be safely defined. It is usually possible on all but the largest separate I/D kernels, and on the small-to-medium nonseparate, overlaid kernels. The *checksys* utility will print a warning message if the data limit is exceeded when a new kernel is loaded.

PROFIL **C** Turn on system profiling. This requires a separate I/D cpu equipped with a KW11-P clock. It cannot be used on machines with ENABLE/34 boards since they have no spare page address registers. If profiling is enabled, you should change the definition of SPLFIX in the corresponding machine Makefile to *:splfix.profil*. The directory */usr/contrib/getsyspr* contains a program for extracting the profiling information from the kernel.

UCB_BHASH **D** Compile in code to hash buffer headers (and cut the time required by the *getblk* routine by 50% or more on large systems).

UCB_FRCSWAP **C** Force swaps on all forks and expands (but not vforks). This is used to transfer some of the load from a compute-bound CPU to an idle disk controller. This is probably not a good idea with **VIRUS_VFORK** defined, but then the load is better reduced by using vfork instead of fork.

UCB_IHASH **D** Compile in code to hash in-core inodes (and cut the time required by the *iget* routine by 50% or more on large systems).

UNFAST **C** Do not use inline macro expansions designed to speed up file system accesses at the cost of a larger text segment.

5.4. Additional configuration details

A few of the parameters and options require a little care to set up; those considerations are discussed here.

5.4.1. Alternate disk drivers

There are several disk drivers provided for SMD disks. The **hp** driver supports RP04/05/06 disks; **rm** supports RM02/03 disks, and **dvhp** supports 300 Mbyte drives on Diva controllers. In addition, there is an **xp** driver which handles any of the above, plus RM05 disks, multiple controllers, and disks which are similar to those listed but with different geometry (e.g. Fujitsu 160 Mbyte drives). It can be used with UNIBUS or MASSBUS controllers or both. In general, if you have only one type of disk and one controller, the **hp**, **rm** or **dvhp** drivers are the best choices, since they are smaller and simpler. If you use the **xp** driver, it can be set up in one of two ways. If **XP_PROBE** is defined in xp.h, the driver will attempt to determine the type of each disk and controller by probing and using the drive type register. To save the space occupied by this routine, or to specify different drive parameters, the drive and controller structures can be initialized in ioconf.c if **XP_PROBE** is not defined. The controller addresses will have to be initialized in either case (at least the first, if it is a boot device). The file /usr/include/sys/hpreg.h provides the definitions for the flags and sizes. Ioconf.c has an example of initialized structures. *Xp* (4) gives more information about drive numbering, etc.

5.4.2. Disk monitoring parameters

The kernel is capable of maintaining statistics about disk activity for specified disks; this information can be printed by *iostat* (8). This involves some setup, however, and if parameters are set incorrectly can cause the kernel monitoring routines to overrun their array bounds. To set this up correctly, choose the disks to be monitored. *Iostat* is configured for a maximum of 4 disks, but that could be changed by editing the headers. The drivers that do overlapped seeks (hk, hp, rm and xp) use one field for each drive (N*XX*) plus one for the controller; the others use only one field, for the controller. When both drives and controllers are monitored, the drives come first, starting at *DK*_DKN, followed by the controller (or controllers, in the case of xp). Then set **NDISK** in param.c to the desired number. The number of the first slot to use for each driver is defined as *DK*_DKN in the device's header file, or is undefined if that driver is not using monitoring. *Iostat* currently expects that if overlapped seeks are being metered, those disks are first in the array (i.e., DKN for that driver is 0). As an example, for 3 RP06 disks using the hp driver plus 1 RL02, HP_DKN should be 0, RL_DKN should be 4, and **NDISK** should be 5 (3 hp disks + 1 hp controller + 1 rl). The complete correspondence for *iostat* would then be:

0 (HP_DKN + 0)	hp0 seeks
1 (HP_DKN + 1)	hp1 seeks
2 (HP_DKN + 2)	hp2 seeks
3 (HP_DKN + NHP)	hp controller transfers
4 (RL_DKN + 0)	rl transfers

It is very important that NDISK be large enough, since the drivers do not check for overflow.

After the kernel disk monitoring is set up, *iostat* itself needs to be edited to reflect the numbers and types of the disks. The source is in /usr/src/cmd.

5.4.3. Automatic reboot

The automatic reboot facility (**UCB_AUTOBOOT**) includes a number of components, several of which must know details of the boot configuration. The kernel has an integral boot routine, found in boot.s in the configuration directory for the machine, which reads in a block 0 bootstrap from the normal boot device and executes it. The block 0 bootstrap normally loads **boot** from the first file system on drive 0 of the disk; this can be changed if necessary. The second-stage bootstrap, /boot, needs to know where to find unix.

The first step is to determine which kernel boot to use. Currently, there are boot modules supplied for the following disk types: hk, rl, rm, rp, dvhp, sc11 and sc21 (the last two are for Emulex SC11 and SC21 controllers, using the boot command). If one of these will work with your boot disk, place that entry in the **bootdev** field in the device configuration file before running *config*, or simply copy ../conf/*dk*boot.s to boot.s in the machine configuration directory. If no boot module supplied will work, it is not too difficult to create one for your machine. The easiest way to do this is to copy one of the other boot modules, and modify the last section which actually reads the boot block. If you have a bootstrap ROM, you can simply jump to the correct entry with any necessary addresses placed in registers first. Or, you can write a small routine to read in the first disk block. If you don't have a boot module, **bootdev** in the configuration file should be specified as **none**, and noboot.s will be installed. This is a dummy file that keeps the load rules from changing. The **UCB_AUTOBOOT** option should not be defined until a boot module is obtained.

The other change that is normally required is to specify where /unix will be found. This is done by changing the definition of **RB_DEFNAME** in /usr/include/sys/reboot.h. The definition is a string in the same format as the manual input to boot, for example "xp(0,0)unix". After making this change, boot will need to be recompiled (in /usr/src/sys/stand/bootstrap) and installed. It can be installed initially as /newboot, and the original boot can be used to load it for testing:

>boot

*nn***Boot**
: *dk*(0,0)newboot

*nn***Boot**
: *dk*(0,0)unix

If you want to have core dumps made after crashes, this must be specified in the configuration file as well. Dumps are normally taken on the end of the swap device before rebooting, and after the system is back up and the file systems are checked, the dump will be copied into /usr/sys by *savecore*(8). Dump routines are available for the hk, hp, rm and xp drivers. To install, change the **dumpdev** entry to the same value as the swap device. Then set **dumplo** to a value that will allow as much as possible of memory to be saved. The dump routine will start the dump at dumplo and continue to the end of memory or the end of the swap device partition, whichever comes first. Dumplo should be larger than swplo so that any early swaps will not overwrite the dump, but if possible, should be low enough that there is room for all of memory. The **dumproutine** entry in the configuration file is then set to *dk*dump, where *dk* is the disk type. Finally, after running *config*, edit the header file *dk*.h in the new configuration directory to define *DK*_DUMP, so that that dump routine will be included when the driver is compiled.

5.4.4. Considerations on a PDP-11/23

If setting up a kernel on a PDP-11/23, it is necessary to consider the interrupt structure of the hardware. If there are any single-priority boards on the bus, they must be behind all multiple-priority devices. Otherwise, they may accept interrupts meant for another, higher-priority device farther from the processor, at a time when the system has set the processor priority to block the single-level device. The alternative is to use spl6 uniformly for any high processor priority (spl4, spl5, spl6). This may be accomplished by changing the _spl routines in mch.s, the definitions of br4 and br5 in l.s, and by changing the script :splfix.mtps (in the *conf* directory).

Berkeley UNIX does not support more than 256K bytes of memory on the 11/23. If you have extra memory and a way to use it (e.g. a disk driver capable of 22-bit addressing) you will want to change this.

5.5. Compiling the kernel

Once you have made any local changes, you are ready to compile the kernel. If you have made any changes which will affect the dependency rules in the Makefile, run "make depend" (the output of this command is best appreciated on a crt). Then, "make unix." Note: although several shortcuts have been built into the makefile, the nonseparate I/D *make* occasionally runs out of space while recompiling the kernel. If this happens, just restart it and it will generally make it through the second time. The split I/D version of *make* in /usr/70 should have no problem. Also note, it is imperative that overlaid kernels be compiled with the 2.9BSD versions of *cc*, *as* (and *as2*) and *ld*. Use of older C preprocessors or assemblers will result in compile-time errors or (worse) systems that will almost run, but crash after a short time.

After the unix binary is loaded, the makefile runs a small program called *checksys* which checks for size overflows. If you are building an overlaid system, check the size of the object file (see *size*(1)) and overlay layout. The overlay structure may be changed by editing the makefile. For a non-separate I/D system, the base segment size must be between 8194 and 16382 bytes and each overlay must be at most 8192 bytes. The final object file ''unix'' should be copied to the root, and then booted to try it out. It is best to name it /newunix so as not to destroy the working system until you're sure it does work:

```
# cp unix /newunix
# sync
```

It is also a good idea to keep the old system around under some other name. In particular, we recommend that you save the generic distribution version of the system permanently as /genericunix for use in emergencies.

To boot the new version of the system you should follow the bootstrap procedures outlined in section 2.4 above. A systematic scheme for numbering and saving old versions of the system is best.

You can repeat these steps whenever it is necessary to change the system configuration.

5.6. Making changes to the kernel

If you wish to make local mods to the kernel you should bracket them with

```
#ifdef PICKLE
...
#endif
```

perhaps saving old code between

```
#ifndef PICKLE
...
#endif
```

This will allow you to find changed code easily.

To add a device not supported by the distribution system you will have to place the driver for the device in the directory /usr/src/sys/dev, edit a line into the block and/or character device table in /usr/src/sys/PICKLE/c.c, add the name of the device to the OPTIONAL line of the file Depend, and to the makefile load rules. Place the device's address and interrupt vector in the files ioconf.c and l.s respectively if it is not going to be configured by *autoconfig*(8); otherwise, l.s will only need the normal interface to the C interrupt routine. If you use autoconfiguration, you will need an attach routine in the driver, and a probe routine in the driver or in *autoconfig*. Use the entries for a similar device as an example. If the device driver uses the UNIBUS map or system buffers, it will probably need modifications. Check ''Changes in the Kernel in 2.9BSD'' for more technical information regarding driver interfacing. You can then rebuild the system (be sure to make *depend* first). After rebooting the resulting kernel and making appropriate entries in the /dev directory, you can test out the new device and driver. Section 7.1 explains shutdown and reboot procedures.

6. RECOMPILING SYSTEM SOFTWARE

We now describe how to recompile system programs and install them. Some programs must be modified for the local system at this time, and other local changes may be desirable now or later. Before any of these procedures are begun, be certain that the include files <whoami.h>, <sys/localopts.h> and <sys/param.h> are correct for the kernel that has been installed. This is important for commands that wish to know the name of the local machine or that size their data areas appropriately for the type of CPU. The general procedures are given first, followed by more detailed information about some of the major systems that require some setup.

6.1. Recompiling and reinstalling system software

It is easy to regenerate the system, and it is a good idea to try rebuilding pieces of the system to build confidence in the procedures. The system consists of three major parts: the kernel itself, along with the bootstrap and standalone utilities (/usr/src/sys), the user programs (/usr/src/cmd, /usr/src/ucb, and subdirectories), and the libraries (/usr/src/lib). The major part of this is /usr/src/cmd.

We have already seen how to recompile the system itself. The commands and libraries can be recompiled in their respective source directories using the Makefile (or Ovmakefile if there are both overlaid and non-overlaid versions). However, it is generally easier to use one of the MAKE scripts set up for /usr/src/lib, /usr/src/cmd, and /usr/src/ucb. These are used in a similar fashion, such as

    # ./MAKE −40 [−cp] [−f] file ...

The first, required flag sets the CPU class for which to compile. Three classes are used to used to set requirements for separate instruction and data and for floating point. ‘‘MAKE −40’’ makes nonseparate I/D versions that load the floating point interpreter as required. ‘‘MAKE −34’’ is similar but assumes a hardware floating point unit. ‘‘MAKE −70’’ is used for separate I/D machines and also assumes floating point hardware. ‘‘MAKE −70 −f’’ is used for separate I/D machines without floating point hardware. The use of these MAKE scripts automates the selection of CPU-dependent options and makes the optimal configuration of each program for the target computer. The optional argument −cp causes each program to be installed as it is made. They are installed in the normal directories, unless the environment variable DESTDIR is set, in which case the normal path is prepended by DESTDIR. This can be used to compile and create a new set of binary directories, e.g. /nbsd/bin, /nbsd/lib, etc. Running the command ‘‘MAKE −70 −cp *’’ in /usr/src/lib, /usr/src/cmd and /usr/src/ucb would thus create a whole new tree of system binaries. The six major libraries are the C library in /usr/src/lib/c, the jobs library, /usr/src/lib/jobs, the FORTRAN libraries /usr/src/lib/libF77, /usr/src/lib/libI77, and /usr/src/lib/libU77, and the math library /usr/src/lib/m. Most libraries are made in two versions, one each for use with and without process overlays. In each case the library is remade by changing into /usr/src/lib and doing

    # ./MAKE −*cpu* *libname*

or made and installed by

    # ./MAKE −*cpu* −cp *libname*

Similar to the system,

    # make clean

cleans up in each subdirectory.

To recompile individual commands, change to /usr/src/cmd or /usr/src/ucb, as appropriate, and use the MAKE script in the same way. Thus to compile adb, do

    # ./MAKE −*cpu* adb

where cpu is 34, 40, or 70. To recompile everything, use

    # ./MAKE −*cpu* *

After installing new binaries, you can use the script in /usr/src to link files together as necessary and to set all the right set-user-id bits.

```
# cd /usr/src
# ./MAKE aliases
# ./MAKE modes
```

6.2. Making local modifications

To keep track of changes to system source we migrate changed versions of commands in /usr/src/cmd in through the directory /usr/src/new and out of /usr/src/cmd into /usr/src/old for a time before removing them. Locally written commands that aren't distributed are kept in /usr/src/local and their binaries are kept in /usr/local. This allows /usr/bin, /usr/ucb, and /bin to correspond to the distribution tape (and to the manuals that people can buy). People wishing to use /usr/local commands are made aware that they aren't in the base manual. As manual updates incorporate these commands they are moved to /usr/ucb.

A directory /usr/junk to throw garbage into, as well as binary directories /usr/old and /usr/new are useful. The *man* (1) command supports manual directories such as /usr/man/mann for new and /usr/man/manl for local to make this or something similar practical.

6.3. Setting up the mail system

The mail system can be set up in at least two ways. One strategy uses the *delivermail* (8) program to sort out network addresses according to the local network topology. It is not perfect, especially in the light of changing ARPAnet conventions. However, if you use the Berkeley network or are connected directly or indirectly to the ARPAnet, it is probably the method of choice for the time being. On the other hand, if you use only local mail and UUCP mail, /bin/mail (*mail* (1)) will suffice as a mail deliverer. In that case, you will only need to recompile *mail* (1) and *Mail* (1).

The entire mail system consists of the following commands:

/bin/mail	old standard mail program (from V7 or System III)
/usr/ucb/Mail	UCB mail program, described in Mail(1)
/usr/lib/Mail.rc	aliases and defaults for Mail(1)
/etc/delivermail	mail routing program
/usr/net/bin/v6mail	local mailman for berknet
/usr/spool/mail	mail spooling directory
/usr/spool/secretmail	secure mail directory
/usr/bin/xsend	secure mail sender
/usr/bin/xget	secure mail receiver
/usr/lib/aliases	mail forwarding information for delivermail
/usr/ucb/newaliases	command to rebuild binary forwarding database

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 *delivermail* (8) or *mail* (1) for routing and/or delivery.

Mail is normally accessible in the directory /usr/spool/mail and is readable by all users.† To send mail which is secure against any possible perusal (except by a code-breaker) you should use the secret mail facility, which encrypts the mail so that no one can read it.

6.3.1. Setting up mail and Mail

Both /bin/mail and /usr/ucb/Mail should be recompiled to make local versions. Remake mail in /usr/src/cmd with the command

† You can make your mail unreadable by others by changing the mode of the file /usr/spool/mail/*yourname* to 600 and putting the line "set keep" in your .mailrc file. The directory /usr/spool/mail must not be writable (mode 755) for this to work.

./MAKE −*cpu* mail

Install the new binary in /bin after testing; it must be setuserid root. Section 6.1 gives more details on the use of the MAKE scripts. To configure *Mail*, change directories to /usr/src/ucb/Mail. Edit the file v7.local.h to assign a letter to your machine with the definition of LOCAL; if you do not have a local area network, the choice is arbitrary as long as you pick an unused letter. If you wish to use *delivermail*, the definition of SENDMAIL should be uncommented. Then add your machine to the table in config.c; configdefs.h gives some information on this. The network field should specify which networks (if any) you are connected to (note: the Schmidt net, SN, is Berknet). After the changes are made, move to /usr/src/ucb and

./MAKE −40 Mail (on a nonseparate I/D machine)
or
./MAKE −70 Mail (on a separate I/D machine)

Install *Mail* in /usr/ucb; it should **not** be setuserid. The Mail.rc file in /usr/lib can be used to set up limited distribution lists or aliases if you are not using *delivermail*.

6.3.2. Setting up delivermail

To set up the *delivermail* facility you should read the instructions in the file READ_ME in the directory /usr/src/ucb/delivermail and then adjust and recompile the *delivermail* program, installing it as /etc/delivermail. The routing algorithm uses knowledge of network name syntax built into its tables and aliasing and forwarding information built into the file /usr/lib/aliases to process each piece of mail. Local mail is delivered by giving it to the program /usr/net/bin/v6mail which adds it to the mailboxes in the directory /usr/spool/mail/*username*, using a locking protocol to avoid problems with simultaneous updates. You should also set up the file /usr/lib/aliases for your installation, creating mail groups as appropriate.

6.4. Setting up a uucp connection

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

You should first read the paper in volume 2B of the UNIX Programmers Manual: ‘‘Uucp Implementation Description.’’ It describes in detail the file formats and conventions, and will give you a little context. For any configuration, you must recompile all system dependent programs.

Change directory to /usr/src/cmd/uucp and examine uucp.h, making any necessary changes. Recompile uucp with ‘‘make’’ and su to ‘‘make install.’’

You should ensure that the directories /usr/spool/uucp and /usr/spool/uucppublic exist. The former should be owned by uucp, mode 755 (or 777 is OK) and the latter should be mode 777 (and the home directory for login uucp).

Periodically you should clean out /usr/spool/uucp and /usr/spool/uucppublic, as they can accumulate junk, especially if you don’t have a dialer. Run ‘‘uulog’’ once a day, and ‘‘/usr/lib/uucp/uuclean’’ periodically with appropriate options to get rid of old stuff.† You can also just remove some of the files in /usr/spool/uucp, but if you do this blindly you will cause some error messages to be generated when uucp tries to access a file another file claims is there. (For instance, each mail transaction creates three files.) The /usr/spool/uucppublic directory is a place for people at other sites to send to when sending files to users on your machine. You should clean it out by hand when it gets excessive.

If both sites have both a dialer and dialup: follow the directions in the volume 2B paper − this is the intended mode of operation and the directions fit well. You have to configure the following files in /usr/lib/uucp:

L .sys	setup all fields − this lists the other sites
L-devices	your dialer
USERFILE	permissions − this can be left alone

† The *cron* (8) program can arrange to execute these commands periodically.

You must also establish a login ‘‘uucp’’ in /etc/passwd with shell /usr/lib/uucp/uucico. Each site must know the other site’s phone number, login, and password.

If you have only a dialup: you can be a second-class citizen on the uucp net. You must find another site that has a dialer, and have them poll you regularly. (Once a day is about the minimum that is reasonable.) When you send mail to another site, you must wait for them to call you. You must set up /usr/lib/uucp/USERFILE and /usr/lib/uucp/L.sys. Only the first 4 fields of L.sys are necessary, and in practice only the first field (site name) is looked at. A typical L.sys for a passive node might be:

```
ucbvax     Any ACU 300
research   Any ACU 300
```

where the first field on each line is a site that will poll you and *ACU* is either ‘‘ACU’’ or ‘‘DIR.’’ You need to put a password on the uucp login and let the other site know your phone number, uucp login name (which is usually uucp), and password. It doesn’t matter whether they call you at 300 or 1200 baud.

If you have a dialer and want to poll another site: normally, uucp will call the other site when it has anything to send it, and while it’s at it will check to see if anything should come back. The command

/usr/lib/uucp/uucico −r1 −sucbvax

will force *uucp* to poll ucbvax, even if there is nothing waiting. This command can be conveniently put in /usr/lib/crontab to run early each morning. If you are having trouble with the connection, invoke uucico by hand:

/usr/lib/uucp/uucico −r1 −sucbvax −x7

where the **-x** option turns on debugging output. The higher the number, the more debugging output you get; 1, 4, and 7 are reasonable choices.

6.5. Miscellaneous software

The directory /usr/contrib contains programs and packages that you may wish to install on your system. Also, some programs or libraries in the *ucb* directory are sufficiently unique to be noteworthy. Here is a brief summary.

6.5.1. Ape

Ape (*A*rbitrary *P*recision *E*xtended) is a replacement for the multiple precision arithmetic routines (*mp* (3)). It is much faster and contains numerous bug fixes.

6.5.2. L11, M11

M11 is a Macro-11 assembler. It recognizes and emulates almost all of the directives of standard DEC Macro-11 assemblers. *L11* is its loader.

6.5.3. Jove

Jove (*J*onathan’s *O*wn *V*ersion of *E*MACS) is an EMACS style editor developed at Lincoln Sudbury Regional High School.

6.5.4. News

The network bulletin board system developed at Duke University and the University of North Carolina and since heavily modified at Berkeley.

6.5.5. Notes

The network bulletin board system developed at the University of Illinois. This version contains many enhancements and clean *news* interfaces.

6.5.6. Ranm

Ranm is a fast uniform pseudorandom number generator package developed at Berkeley.

7. SYSTEM OPERATION

This section describes procedures used to operate a PDP-11 UNIX system. Procedures described here are used periodically, to reboot the system, analyze error messages from devices, do disk backups, monitor system performance, recompile system software and control local changes.

7.1. Bootstrap and shutdown procedures

The system boot procedure varies with the hardware configuration, but generally uses the console emulator or a ROM routine to boot one of the disks. /boot comes up and prompts (with ``: '') for the name of the system to load. Simply hitting a carriage return will load the default system. The system will come up with a single-user shell on the console. To bring the system up to a multi-user configuration from the single-user status, all you have to do is hit ^D on the console (you should check and, if necessary, set the date before going multiuser; see *date* (1)). The system will then execute /etc/rc, a multi-user restart script, and come up on the terminals listed as active in the file /etc/ttys. See *init* (8) and *ttys* (5). Note, however, that this does not cause a file system check to be performed. Unless the system was taken down cleanly, you should run ``fsck –p'' or force a reboot with *reboot* (8) to have the disks checked.

In an automatic reboot, the system checks the disks and comes up multi-user without intervention at the console. If the file system check fails, or is interrupted (after it prints the date) from the console when a delete/rubout is hit, it will leave the system in special-session mode, allowing root to log in on one of a limited number of terminals (generally including a dialup) to repair file systems, etc. The system is then brought to normal multiuser operations by signaling init with a SIGINT signal (with ``kill -INT 1'').

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

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

The system can also be halted or rebooted with *reboot* (8) if automatic reboots are enabled. Otherwise, the system is halted by switching to single-user mode to kill all processes, updating the disks with a ``sync'' command, and then halting.

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

7.2. Device errors and diagnostics

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

Error messages printed by the devices in the system are described with the drivers for the devices in section 4 of the Berkeley PDP-11 UNIX Programmer's manual. If errors occur indicating hardware problems, you should contact your hardware support group or field service. It is a good idea to examine the error log file regularly (e.g. with ``tail –r /usr/adm/messages'').

If you have DEC field service, they should know how to interpret these messages. If they do not, tell them to contact the DEC UNIX Engineering Group.

7.3. File system checks, backups and disaster recovery

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

Dumping of the file systems should be done regularly, since once the system is going it is easy to become complacent. Complete and incremental dumps are easily done with *dump*(8). You should arrange to do a towers-of-Hanoi dump sequence; we tune ours so that almost all files are dumped on two tapes and kept for at least a week in almost every case. We take full dumps every month (and keep these indefinitely).

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 block size.

It is desirable that full dumps of the root file system are made regularly. This is especially true when only one disk is available. Then, if the root file system is damaged by a hardware or software failure, you can rebuild a workable disk using a standalone restore in the same way that *restor* was used to build the initial root file system.

Exhaustion of user-file space is certain to occur now and then; the only mechanisms for controlling this phenomenon are occasional use of *df*(1), *du*(1), *quot*(8), threatening messages of the day, personal letters, and (probably as a last resort) quotas (see *setquota*(8)).

7.4. Moving file system data

If you have the equipment, the best way to move a file system is to dump it to magtape using *dump*(8), to use *mkfs*(8) to create the new file system, and restore, using *restor*(8), the tape. If for some reason you don't want to use magtape, dump accepts an argument telling where to put the dump; you might use another disk. Sometimes a file system has to be increased in logical size without copying. The super-block of the device has a word giving the highest address that can be allocated. For small increases, this word can be patched using the debugger *adb*(1) and the free list reconstructed using *fsck*(8). The size should not be increased greatly by this technique, since the file system will then be short of inode slots. Read and understand the description given in *filsys*(5) before playing around in this way.

If you have to merge a file system into another, existing one, the best bet is to use *tar*(1). If you must shrink a file system, the best bet is to dump the original and restor it onto the new file system. However, this will not work if the i-list on the smaller file system is smaller than the maximum allocated inode on the larger. If this is the case, reconstruct the file system from scratch on another file system (perhaps using *tar*(1)) and then dump it. If you are playing with the root file system and only have one drive the procedure is more complicated. What you do is the following:

1. GET A SECOND PACK!!!!
2. Dump the root file system to tape using *dump*(8).
3. Bring the system down and mount the new pack.
4. Load the standalone versions of *mkfs*(8) and *restor*(8) as in sections 2.1-2.3 above.
5. Boot normally using the newly created disk file system.

Note that if you add new disk drivers they should also be added to the standalone system in /usr/src/sys/stand.

7.5. Monitoring System Performance

The *iostat*(8) and *vmstat*(8) programs provided with the system are designed to aid in monitoring systemwide activity. By running them when the system is active you can judge the system activity in several dimensions: job distribution, virtual memory load, swapping activity, disk and CPU utilization. Ideally, there should be few blocked (DW) jobs, there should be little swapping activity, there should be available bandwidth on the disk devices (most single arms peak out at 30-35 tps in practice), and the user CPU utilization (US) should be high (above 60%).

If the system is busy, then the count of active jobs may be large, and several of these jobs may often be blocked (DW).

If you run *vmstat* when the system is busy (a ‘‘vmstat 5’’ gives all the numbers computed by the system), you can find imbalances by noting abnormal job distributions. If many processes are blocked (DW), then the disk subsystem is overloaded or imbalanced. If you have several non-DMA devices or open teletype lines that are ‘‘ringing’’, or user programs that are doing high-speed non-buffered input/output, then the system time may go high (60-70% or higher). It is often possible to pin down the cause of high system time by looking to see if there is excessive context switching (CS), interrupt activity (IN) or system call activity (SY).

If the system is heavily loaded, or if you have little memory for your load (248K is little in almost any case), then the system will be forced to swap. This is likely to be accompanied by a noticeable reduction in system performance and pregnant pauses when interactive jobs such as editors swap out. If you expect to be in a memory-poor environment for an extended period you might consider administratively limiting system load.

7.6. Adding users

New users can be added to the system by adding a line to the password file /etc/passwd. You should add accounts for the initial user community, giving each a directory and a password, and putting users who will wish to share software in the same group. User id’s should be assigned starting with 16 or higher, as lower id’s are treated specially by the system. Default startup files should probably provided for new users and can be copied from /usr/public. Initial passwords should be set also.

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

7.7. Accounting

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

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

7.8. Resource control

Resource control in the current version of UNIX is rather primitive. Disk space usage can be monitored by *du* (1) or *quot* (8) as was previously mentioned. Disk quotas can be set and changed with *setquota* (8) if the kernel has been configured for quotas. Our quota mechanism is simplistic and easily defeated but does make users more aware of the amount of space they use.

7.9. Files which need periodic attention

We conclude the discussion of system operations by listing the files and directories that continue to grow and thus require periodic truncation, along with references to relevant manual pages. *Cron* (8) can be used to run scripts to truncate these periodically, possibly summarizing first or saving recent entries. Some of these can be disabled if you don’t need to collect the information.

/usr/adm/acct	sa(8)	raw process account data
/usr/adm/messages	dmesg(8)	system error log
/usr/adm/shutdownlog	shutdown(8)	log of system reboots
/usr/adm/wtmp	ac(8)	login session accounting

/usr/spool/uucp/LOGFILE	uulog(1)	uucp log file
/usr/spool/uucp/SYSLOG	uulog(1)	more uucp logging
/usr/dict/spellhist	spell(1)	spell log
/usr/lib/learn/log	learn(1)	learn lesson logging
/usr/sys	savecore(8)	system core images

8. KERNEL MAGIC NUMBERS

This sections contains a collection of magic numbers for use in patching core or an executable unix binary. Some of them have also been mentioned earlier in this paper. With the exception of the *xp_type[i]* variables (which hold bytes) and *swplo* (which is a long) all locations given contain short integers. N.B.: in the case of paired interrupt vectors (for DHs and DZs) the address of the second vector of the pair is four more than the address of the first vector.

Interrupt Vectors

Vector	Handler	Contents	Block device	Character device
0160	rlio	01202	8	18
0210	hkio	01142	4	19
0220	rkio	01172	0	9
0224	tmio	01222	3	12
0224	htio	01152	7	15
0224	tsio	01232	9	20
0254	xpio	01242	6	14
0260	rpio	01212	1	11
†	dzin	01132	-	21
†	dzdma	02202	-	21
†	dhin	01112	-	4
†	dhou	01122	-	4
†	lpio	01162	-	2

† Set by *autoconfig* (8).

Other Variables

Name	Address	Contents
xp_addr	061464	0176700
xp_type[0]	061472	‡
xp_type[1]	061506	‡
xp_type[2]	061522	‡
xp_type[3]	061536	‡
HKADDR	061006	0177440
HTADDR	0114236	†
RKADDR	061152	0177400
RLADDR	061154	0174400
RPADDR	061236	0176710
TMADDR	0113330	†
TSADDR	0113622	†
dz_addr	0113324	†
dh_addr	0114146	†
lp_addr	0113462	†
rootdev	060772	*
pipedev	060776	*
swapdev	060774	*
swplo	061000	*
nswap	061004	*

† Set by *autoconfig* (8).

‡ Set by reading the corresponding drive type register.

* System dependent.