Installing and Operating 2.11BSD on the PDP-11
June 13, 1995

Steven Schultz

GTE Government Systems
112 Lakeview Canyon
Thousand Oaks CA 91362
sms@wlv.iipo.gtegsc.com

ABSTRACT

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

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

The "bugs" address supplied with this release will work for some unknown period of time; make sure the "Index:" line of the bug report indicates that the release is "2.11BSD". See the *sendbug*(8) program for more details. All fixes that I make, or that are sent to me, will be posted on *USENET*, in the news group "comp.bugs.2bsd".

---

† DEC, PDP-11, VAX, IDC, SBI, UNIBUS and MASSBUS are trademarks of Digital Equipment Corporation.

‡ UNIX is a Trademark of Bell Laboratories.

1. INTRODUCTION

This document explains how to install 2.11BSD UNIX for the PDP-11 on your system. This document has been revised several times since the first release of 2.11BSD, most recently in July 1995 to reflect the addition of disk labels to the system. The format of the bootable tape has changed. There is now a standalone **disklabel** program present. While the system call interface is the same as that of 2.10.1BSD, a full bootstrap from the distribution tape is required because the filesystem has changed to allow file names longer than 14 characters. Also, the 3 byte block number packing scheme used by earlier versions of UNIX for the PDP-11 has been eliminated. Block numbers are always 4 byte **longs** now.

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

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

Binaries from 2.10.1BSD which do not read directories or inode structures may be used but should be recompiled to pick up changes in the standard libraries. Note too, that the portable ASCII format of *ar*(1) archives is now in place - any local archive files will have to be converted using */usr/old/arcv*.

1. Hardware supported

This distribution can be booted on a PDP-11 with 1Mb of memory or more↑, separate I&D, and with any of the following disks:

RK06, RK07
Any MSCP disk, including but not limited to: RD53, RD54, RA81, RZ2x
RM03, RM05
RP04, RP05, RP06
Many other SMD disks, for example: CDC 9766, Fuji 160, Fuji Eagle

Other disks are supported (RX23, RX33, RX50, RD51) but are not large enough to hold a root filesystem plus a swap partition. The old restriction of using RL02 drives in pairs has been lifted. It is now possible to define a root ('a') partition and a swap partition ('b') and load at least the root filesystem to a single RL02. Discs which are too small to hold even a root filesystem (floppies for example) may be used as data disks or as standalone boot media, but are not useable for loading the distribution. Others, while listed above, are not very well suited to loading the distribution. The RK06/07 drives are hard pressed to even hold the system binaries, much less the sources.

The tape drives supported by this distribution are:

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

Although 2.11BSD contains a kernel level floating point simulator, it has never been tested. In fact it would not even compile/assemble without errors! That problem has been fixed but it is still not know if the simulator works, KDJ-11 based systems have builtin floating point so the simulator can not be tested. At the release of 2.10BSD some thought was given to the possibility of lifting the separate I&D restriction, but

↑ 2.11BSD would probably only require a moderate amount of squeezing to fit on machines with less memory, but it would also be very unhappy about the prospect.

that thought has languished. The work will never be done. As time passes more and more programs have become almost too large even with separate I&D.

2. Distribution format

The basic distribution contains the following items:

- (2) 1600bpi 2400’ magnetic tapes, or
- (2) TK25 tape cartridges, or
- (1) TK50 tape cartridge, and
- (1) Hardcopy of this document,
- (1) Hardcopy of the *Changes in 2.11BSD* document,
- (1) Hardcopy of the 2.11BSD /README and /VERSION files, and
- (1) Hardcopy of manual pages from sections 4, and 8.

Installation on any machine requires a tape unit. Since certain standard PDP-11 packages do not include a tape drive, this means one must either borrow one from another PDP-11 system or one must be purchased separately.

The distribution does not fit on several standard PDP-11 configurations that contain only small disks. If your hardware configuration does not provide at **least 75** Megabytes of disk space you can still install the distribution, but you will probably have to operate without source for the user level commands and, possibly, the source for the operating system.

The root file system now occupies **a minimum of 4Mb**. If at all possible a larger, 6 or 7Mb, root partition should be defined when using the standalone **disklabel** program.

If you have the facilities, it is a good idea to copy the magnetic tape(s) in the distribution kit to guard against disaster. The tapes are 9-track 1600 BPI, TK50 or TK25 cartridges and contain some 512-byte records, followed by 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 basic bootstrap material is present in six short files at the beginning of the first tape. The first file on the tape contains preliminary bootstrapping programs. This is followed by several standalone utilities (*disklabel*, *mkfs* (8), *restor* (8), and *icheck* (8)↑) followed by a full dump of a root file system (see *dump* (8)). Following the root file system dump is a tape archive image of **/usr** except for **/usr/src** (see *tar* (1)). Finally, a tape archive of source for include files and kernel source ends the first tape. The second tape contains a tape archive image, also in *tar* format, of all the remaining source that comes with the system.

The entire distribution (barely) fits on a single TK50 cartridge, references to the second tape should be treated as being the 9th file on the TK50. Many of the programs in /usr/src/new have been tar+compress’d in order to keep the distribution to a single tape.

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

TAPE 1:

Tape file	Record size	Records↑	Contents
0	512	1	primary tape boot block
	512	1	boot block (some tape boot ROMs go for this copy)
	512	69	standalone **boot** program
1	1024	37	standalone **disklabel**
2	1024	33	standalone **mkfs**(8)
3	1024	35	standalone **restor**(8)
4	1024	32	standalone **icheck**(8)
5	10240	285	*dump* of “root” file system
6	10240	3368	*tar* dump of /usr, excepting /usr/src
7	10240	519	*tar* dump of /usr/src/include and /usr/src/sys

TAPE 2:

Tape file	Record size	Records↑	Contents
0	10240	4092	*tar* dump of /usr/src, excepting include and sys

3. UNIX device naming

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

The disk and tape names used by the bootstrap and the system are:

RK06, RK07 disks	hk
RL01, RL02 disks	rl
RK05	rk
MSCP disks	ra
RM02/03/05	xp
RP04/05/06	xp
SMD disks	xp
TM02/03, TE16, TU45, TU77 tapes	ht
TE10/TM11 tapes	tm
TS11 tapes	ts
TMSCP tapes	tms

Additionally, the following non-DEC devices are also supported:

SI 9500, CDC 9766	si
SI, CDC 9775	xp
SI6100, Fujitsu Eagle 2351A	xp
Emulex SC01B or SI9400, Fujitsu 160	xp
Emulex SC-21, xp	

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

MSCP disks and TMSCP tapes include SCSI drives attached to the RQZX1 controller on the PDP-11/93. MSCP disks and TMSCP tapes also include SCSI drives attached to the Emulex UC07 or UC08 Q-BUS controllers on Q-bus systems as well as the UC17 and UC18 controllers on UNIBUS systems.

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

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

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

where *xx* is one of **hk**, **ht**, **rk**, **rl**, **tm**, **ts**, **tms**, or **xp**. The value *c* specifies the controller number (0-3). This value is usually not explicitly given. The default is 0 if booting from the standard (first) CSR of a device.

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

The 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 partition number (0 thru 7) corresponding to partitions ´a´ thru ´h´ respectively. This should always be zero unless you **really** know what you are doing. The ability to load a kernel from the swap area is planned for the future but does not presently exist. For tapes *z* is a file number on the tape.↑

In all simple cases, a drive with unit number 0 (determined either by a unit plug on the front of the drive, or jumper settings on the drive or controller) will be called unit 0 in its UNIX file name. file name. If there are multiple controllers, the drive unit numbers will normally be counted within each controller. Thus drives on the the first controller are numbered 0 thru 7 and drives on the second controller are numbered 0 thru 7 on controller 1. Returning to the discussion of the standalone system, 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 7 tape files, it contains several thousand UNIX files.

Each UNIX physical disk is divided into 8 logical disk partitions, each of which may occupy any consecutive cylinder range on the physical device. While overlapping partitions are allowed they are not a good idea, being an accident waiting to happen (making one filesystem will destroy the other overlapping filesystems). The cylinders occupied by the 8 partitions for each drive type are specified by the disk label read from the disk.

If no label exists the disk will not be bootable and while the kernel attempts not to damage unlabeled disks (by swapping to or doing a crash dump on a live filesystem) there is a chance that filesystem damage will result if a kernel is loaded from an unlabeled disk.

The standalone **disklabel** program is used to define the partition tables. Each partition may be used either as a raw data area (such as a swapping area) or to store a UNIX file system. It is mandatory for the first partition on a disk to start at sector offset 0 because the ’a’ partition is used to read and write the label (which is at the beginning of the disk). If the drive is to be used to bootstrap a UNIX system then the ’a’ partition must be of type **2.11BSD** (FS_V71K in *disklabel.h*) and at least 4Mb is size. A ’b’ partition of at least 2-3Mb (4Mb is a good choice if space is available) for swapping is also needed. If a drive is being used solely for data then that drive need not have a ’b’ (swap) partition but partition ’a’ must still span the first part of the disk. The second partition is 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.

↑ **Note:** that while a tape file consists of a single data stream, the distribution tape(s) have data structures in these files. Although the first tape contains only 8 tape files, they comprise several thousand UNIX files.

Note: The standalone tape drive unit number is specially encoded to specify both unit number and tape density (BPI). Most tape subsystems either automatically adjust to tape density or have switches on the drives to force the density to a particular setting, but for those which don’t the following density select mechanisms may be necessary. The **ts** only operates at 1600BPI, so there is no special unit density encoding. The **ht** will operate at either 800BPI or 1600BPI. Units 0 through 3 corresponding to 800BPI, and Units 4 through 7 corresponding to 1600BPI on drives 0 through 3 respectively. The standard DEC **tm** only supports 800BPI (and hence can’t be used with the 2.11BSD distribution tape), but several widely used **tm** emulators support 1600BPI and even 6250BPI. Units 0 through 3 corresponding to 800BPI, Units 4 through 7 corresponding to 1600BPI, and Units 8 through 11 corresponding to 6250BPI on drives 0 through 3 respectively.

The third ('c') logical partition of each physical disk also has a conventional usage: it allows access to the entire physical device, including the bad sector forwarding information recorded at the end of the disk (one track plus 126 sectors). It is occasionally used to store a single large file system or to access the entire pack when making a copy of it on another. Care must be taken when using this partition not to overwrite the last few tracks and thereby destroying the bad sector information.

Unfortunately while the drivers can follow the rules above the entries in */etc/disktab* (*disktab* (5)) do not. The entries in */etc/disktab* are translations of the old partition tables which used to be embedded in the device drivers and are thus probably not suitable for use without editing. In some cases it may be that the 8th ('h') partition is used for access to the entire disk rather than the third ('c') partition. Caution should be observed when using the *newfs* (8) and *disklabel* (8) commands.

4. UNIX devices: block and raw

UNIX makes a distinction between "block" and "raw" (character) devices. Each disk has a block device interface where the system makes the device byte addressable and you can write a single byte in the middle of the disk. The system will read out the data from the disk sector, insert the byte you gave it and put the modified data back. The disks with the names "/dev/xx0a", etc are block devices. There are also raw devices available. These have names like "/dev/rxx0a", the "r" here standing for "raw". Raw devices bypass the buffer cache and use DMA directly to/from the program's I/O buffers; they are normally restricted to full-sector transfers. In the bootstrap procedures we will often suggest using the raw devices, because these tend to work faster. Raw devices are used when making new filesystems, when checking unmounted filesystems, or for copying quiescent filesystems. The block devices are used to mount file systems, or when operating on a mounted filesystem such as the root.

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

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

2. BOOTSTRAP PROCEDURE

This section explains the bootstrap procedure that can be used to get the kernel supplied with this distribution running on your machine. It is mandatory to do a full bootstrap since the filesystem has changed from 2.10.1BSD to 2.11BSD.

The safest route is to use *tar*(1) to dump all of your current file systems, do a full bootstrap of 2.11BSD and then restore user files from the backups. There is also an untested version of *512restor*(8) available for V7 sites that need to read old dump tapes.

It is also desirable to make a convenient copy of system configuration files for use as guides when setting up the new system; the list of files to save from earlier PDP-11 UNIX systems, found in chapter 3, may be used as a guideline.

2.11BSD *restor*(8) is able to read and automatically convert to the new on disk directory format *dump*(8) tapes made under 2.9BSD, 2.10BSD and 2.10.1BSD.

2.1. Booting from tape

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

1) Load the tape bootstrap monitor.
2) Create the partition tables on the disk using *disklabel*.
3) Create a UNIX ‘‘root’’ file system system on disk using *mkfs*(8).
4) Restore the full root file system using *restor*(8).
5) Boot the UNIX system on the new root file system and copy the appropriate *sector 0 boot block* to your boot device.
6) Build and restore the /usr file system from tape with *tar*(1).
7) Restore the include and kernel sources from tape.
8) Extract the remaining source from the second tape.
9) Tailor a version of UNIX to your specific hardware (see section 4.2).

Certain of these steps are dependent on your hardware configuration. If your disks require formatting, standard DEC diagnostic utilities will have to be used, they are not supplied on the 2.11BSD distribution tape.

2.1.1. Step 1: loading the tape bootstrap monitor

To load the tape bootstrap monitor, first mount the magnetic tape on drive 0 at load point, making sure that the write ring is not inserted. Then use the normal bootstrap ROM, console monitor or other bootstrap to boot from the tape.

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

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

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

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

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

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

The console should type

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

where *nn* is the CPU type on which it believes it is running. The value will be one of 23, 24, 40, 44, 45, 53, 60, 70, 73, 83, 84, 93 or 94 depending whether separate instruction and data (separate I/D) and/or a UNIBUS map are detected. For KDJ-11 systems the System Maintenance Register is examined to determine the cpu type. At present 2.11BSD runs on the 44, 53, 70, 73, 83, 84, 93 and 94 **only**. It must be emphasized that 2.11BSD requires separate I/D.

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

drive is the drive unit number.

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

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

You are now talking to the tape bootstrap monitor. At any point in the following procedure you can return to this section, reload the tape bootstrap, and restart. Through the rest of this section, substitute the correct disk type for *dk* and the tape type for *tp*.

2.1.2. Step 2: creating the disk label

The standalone *disklabel* program is then run:

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

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

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

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

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

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

2.1.3. Step 3: creating a UNIX "root" file system

Now create the root file system using the following procedure.↑

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

↑ **Note:** These instructions have changed quite a bit during the evolution of the system from 2.10.1BSD. Previously, if the disk on which you are creating a root file system was an **xp** disk you would have been asked to check the drive type register and possibly halt the processor to patch a location (hopefully before the driver accessed the drive). **This is no longer needed**. All geometry and partition information is obtained from the disklabel created in step 2. We also used to give tables of **m** and **n** values for various disks, which are now purposely omitted.

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

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

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

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

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

You now have an empty UNIX root file system.

2.1.4. Step 4: restoring the root file system

To restore the root file system onto it, type

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

This takes about 8 minutes with a TZ30 on a 11/93 and about 15 minutes using a TK50 on a 11/73.

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

2.1.5. Step 5: booting UNIX

You are now ready to boot from disk. Type:

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

The standalone boot program will then load unix from the root file system you just created, and the system should boot:

2.11BSD BSD UNIX #1: Sat Jul 4 01:33:03 PDT 1992
root@wlonex.iipo.gtegsc.com:/usr/src/sys/GENERIC
phys mem = *???*
avail mem = *???*
user mem = *???*

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

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

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

The information about different devices being attached or not being found is produced by the *autoconfig* (8) program. Most of this is not important for the moment, but later the device table, */etc/dtab*, can be edited to correspond to your hardware. 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. This message is present to remind you that the character erase, line erase, and interrupt characters are set to what is standard for DEC systems; this insures that things are consistent with the DEC console interface characters.

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

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

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

Block zero bootstraps and the devices they support are:

boot	driver	devices
hkuboot	hk	RK06/07
rauboot	ra	All RA, RD, RZ, RX (except RX01,02) and RC25 drives
rkuboot	rk	RK05
rluboot	rl	RL01/02
si95uboot	si	SI 9500, CDC 9766
dvhpuboot	xp	Diva Comp V, Ampex 9300
hpuboot	xp	RP04/05/06
rm03uboot	xp	RM03
rm05uboot	xp	RM05 or SI 9500, CDC 9766
si51uboot	xp	SI 6100, Fujitsu Eagle 2351A
si94uboot	xp	Emulex SC01B/SC03B or SI 9400, Fujitsu 160

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

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

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

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

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

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

2.1.6. Step 6: setting up the /usr file system

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

disk=hk (if you have RK06's or RK07's)
disk=rl (if you have RL01's or RL02's)
disk=ra (if you have an MSCP drive)
disk=xp (if you have an RP06, RM03, RM05, or other SMD drive)

The next thing to do is to extract the rest of the data from the tape. You might wish to review the disk configuration information in section 4.3 before continuing; you will have to select a partition to restore the /usr file system into which is at least **25** Megabytes in size (this is just barely enough for the system binaries and such and leaves no room for the system source.)↑

In the command below *part* is the partition name (a-h) for the partition which will hold /usr.

name=${disk}0${part}

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

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

Then execute the following commands:

# date *yymmddhhmm*	(set date, see *date* (1))
....	
# passwd root	(set password for super-user)
New password:	(password will not echo)
Retype new password:	
# hostname *mysitename*	(set your hostname)
# newfs ${name}	(create empty user file system)
(this takes a minute)	
# mount /dev/${name} /usr	(mount the usr file system)
# cd /usr	(make /usr the current directory)
# mt rew	
# mt fsf 6	
# tar xpbf 20 /dev/rmt12	(extract all of usr except usr/src)
(this takes about 15-20 minutes except for the TK50 and TZ30 which are **much** slower)	

The data on the seventh tape file has now been extracted. All that remains on the first tape is a small archive containing source for the kernel and include files.

If you have an existing/old password file to be merged back into 2.11BSD, special steps are necessary to convert the old password file to the shadow password file format (shadow password file and password aging were ported from 4.3BSD and are standard in 2.11BSD).

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

```
# mt -f /dev/rmt12 fsf                 (position tape at beginning of next tape file)
# mkdir src                            (make directory for source)
# cd src                               (make /usr/src the current directory)
# tar xpbf 20 /dev/rmt12               (extract the system and include source)
(this takes about 5-10 minutes)
# cd /                                 (back to root)
# chmod 755  /  /usr  /usr/src /usr/src/sys
# rm −f sys
# ln −s usr/src/sys sys                (make a symbolic link to the system source)
# umount /dev/${name}                  (unmount /usr)
```

The first tape has been been completely loaded. You can check the consistency of the /usr file system by doing

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

The output from *fsck* should look something like:

**** /dev/r*xx*0g**
File System: /usr

NEED SCRATCH FILE (179 BLKS)
ENTER FILENAME: /tmp/xxx
**** Last Mounted on /usr**
**** Phase 1 - Check Blocks and Sizes**
**** Phase 2 - Check Pathnames**
**** Phase 3 - Check Connectivity**
**** Phase 4 - Check Reference Counts**
**** Phase 5 - Check Free List**
671 files, 3497 used, 137067 free

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

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

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

2.1.7. Step 7: extracting remaining source from the second tape

You can then extract the source code for the commands from the second distribution tape↑ (with the exception of RK07's, RM03's, and RD52's and other small disks this will fit in the /usr file system):

```
# cd /usr/src
# tar xpb 20
```

If you get an error at this point, most likely it was a problem with tape positioning. Rewind the tape and use the **mt** command to skip files, then retry the **tar** command.

2.2. Additional conversion information

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

↑ On the TK50 the remaining source is the 9th file on the cartridge.

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

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

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

3. UPGRADING AN EXISTING SYSTEM

Begin by reading the document "Changes to the System in 2.11BSD" to get an idea of how the system changes will affect your local modifications. If you have local device drivers, see the file */sys/OTHERS/README* for hints on how to integrate your drivers into 2.11BSD.

The only upgrade path to 2.11BSD is to do a full bootstrap as described in Chapter 2. As always, full backups of the existing system should be made to guard against errors or failures. **NOTE:** The old filesystems can not be mounted by the new kernel. If you must access old discs or filesystems, there is a version of *dump* (8) in /usr/src/old/dump which can be used with the **raw** disc to dump old filesystems.

The archive file format has changed, the 4.3BSD *ar*(5) format is now used. Local archives will have to be converted by the */usr/old/arcv* program.

3.1. Files to save

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

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

/.cshrc	↑	root csh startup script
/.login	↑	root csh login script
/.profile	↑	root sh startup script
/.rhosts	↑	for trusted machines and users
/dev/MAKEDEV	‡	in case you added anything here
/dev/MAKEDEV.local	*	for making local devices
/etc/disktab	*	in case you changed disk partition sizes
/etc/dtab	‡	table of devices to attach at boot time
/etc/fstab	↑	disk configuration data
/etc/ftpusers	↑	for local additions
/etc/gateways	↑	routing daemon database
/etc/gettytab	↑	getty database
/etc/group	↑	group data base
/etc/hosts	↑	for local host information
/etc/hosts.dir	*	must be rebuilt with mkhosts
/etc/hosts.pag	*	must be rebuilt with mkhosts
/etc/hosts.equiv	↑	for local host equivalence information
/etc/networks	↑	for local network information
/etc/netstart	*	site dependent network startup script
/etc/passwd	*	must be converted to shadow password file format
/etc/passwd.dir	*	must be rebuilt with mkpasswd
/etc/passwd.pag	*	must be rebuilt with mkpasswd
/etc/printcap	↑	line printer database
/etc/protocols	‡	in case you added any local protocols
/etc/rc	*	for any local additions
/etc/rc.local	*	site specific system startup commands
/etc/remote	↑	auto-dialer configuration
/etc/services	‡	for local additions
/etc/syslog.conf	↑	system logger configuration
/etc/securettys	*	for restricted list of ttys where root can log in
/etc/ttys	↑	terminal line configuration data
/etc/ttytype	*	terminal line to terminal type mapping data
/etc/termcap	‡	for any local entries that may have been added
/lib	‡	for any locally developed language processors
/usr/dict/*	‡	for local additions to words and papers
/usr/hosts/MAKEHOSTS	↑	for local changes
/usr/include/*	‡	for local additions
/etc/aliases	↑	mail forwarding data base
/etc/crontab	↑	cron daemon data base
/usr/share/font/*	‡	for locally developed font libraries
/usr/lib/lib*.a	↑	for local libraries
/usr/share/lint/*	‡	for locally developed lint libraries
/etc/sendmail.cf	↑	sendmail configuration
/usr/share/tabset/*	‡	for locally developed tab setting files
/usr/share/term/*	‡	for locally developed nroff drive tables
/usr/share/tmac/*	‡	for locally developed troff/nroff macros
/etc/uucp/*	↑	for local uucp configuration files
/usr/man/manl	*	for manual pages for locally developed programs
/usr/msgs	↑	for current msgs
/usr/spool/*	↑	for current mail, news, uucp files, etc.
/usr/src/local	↑	for source for locally developed programs
/sys/conf/HOST	↑	configuration file for your machine
/sys/conf/files.HOST	↑	list of special files in your kernel
/*/quotas	*	file system quota files

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

3.1.1. Installing 2.11BSD

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

Once you have extracted the 2.11BSD system and booted from it, you will have to build a kernel customized for your configuration. If you have any local device drivers, they will have to be incorporated into the new kernel. See section 4.2.3 and “Building 2.11BSD UNIX Systems.”

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

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

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

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

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

For sites running 2.10.1BSD, converting local configuration files should be very simple. In general very little has changed between 2.10.1BSD and 2.11BSD with regard to these files.

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

If you have any home grown device drivers that use major device numbers reserved by the system you will have to modify the commands used to create the devices or alter the system device configuration tables in /sys/pdp/conf.c. Note that almost all 2.11BSD major device numbers are different from those in previous PDP-11 UNIX systems except 2.10.1BSD. A couple more device numbers were added since the release of 2.10.1BSD for the kernel logging facility (/dev/klog) and a (new) TK50/TU81 driver.

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

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

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

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

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

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

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

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

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

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

The C library and system binaries on the distribution tape are compiled with versions of *gethostbyname* and *gethostbyaddr* which use ndbm host table lookup routines instead of the name server. You must run *mkhosts*(8) to create the *ndbm* host table database from */etc/hosts*. For 2.11BSD the *mkhosts* program has been enhanced to support multiple addresses per host with order being preserved (the order in which the multiple addresses appear in */etc/hosts* for the same host is the same order the addresses will be returned to the caller of *gethostbyname*).

There is a version of the nameserver which runs under 2.11BSD. However in addition to having a voracious appetite for memory there are memory leaks which cause *named*(8) to crash after running for an extended period. Restarting *named*(8) nightly from *cron* is the only work around solution at present.

If you want to compile your system to use the name server resolver routines instead of the ndbm host table, you will need to modify /usr/src/lib/libc/Makefile according to the instructions there and then recompile all of the system and local programs (see section 6.5).↑

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

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

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

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

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

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

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

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

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

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

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

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

↑ Note: The resolver routines add about 5kb of text and 1kb of data to each program. Also, the resolver routines use more stack space which may cause large programs to crash due to failure to extend the stack area.

Daylight Savings Time problems once and for all.

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

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

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

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

3.4. Hints on possible problems upgrading from the 2.10.1BSD

3.4.1. New utmp UT_NAMESIZE.

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

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

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

3.4.2. man system

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

3.4.3. NMOUNT lowered

The value of **NMOUNT** in */sys/h/param.h* is set to 5 in the distribution system. This will be too small for many sites. Since each mount table entry costs about 440 bytes of valuable kernel dataspace this number should be chosen with care. See Appendix A for an explanation of how to reconfigure **NMOUNT.**

3.4.4. Shadow passwords

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

3.4.5. New /etc/rc startup scripts

/etc/rc and */etc/rc.local* have changed fairly significantly, and

/etc/netstart has been added to configure site specific network features (much of this was pulled from the old rc.local). */etc/netstart* uses the tiny program *testnet* which attempts to create a socket and prints NO on stdout if an error is returned by the kernel, YES if no error was returned.

3.4.6. mkfs, mkproto, mklost+found

mkfs(8) no longer can populate a filesystem with files. The 4.3BSD versions of *mkfs*(8) and *mkproto*(8) were ported to 2.11BSD. There is a limit on the size of the file which *mkproto*(8) can place on a newly created filesystem. Only files up to single indirect (about 260kb) may be copied at this time.

mklost+found (8) is a ported version from 4.3BSD, the only change being to use 63 character file names (MAXNAMLEN is 63 at this time in 2.11BSD) instead of 255. *mklost+found* (8) is really not needed, *fsck* (8) is now capable of automatically extending lost+found by up to the number of direct blocks in an inode.

3.4.7. /etc/disktab

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

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

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

The new /etc/disktab file looks like this:

```
:ty=removable:ns#32:nt#5:nc#823:sf:
:b0=/mdec/rm03uboot:
:pa#9600:oa#0:ba#1024:fa#1024:ta=2.11BSD:
:pb#9600:ob#9600:bb#1024:fb#1024:tb=swap:
:pc#131520:oc#0:bc#1024:fc#1024:
:pf#121920:of#9600:bf#1024:ff#1024:tf=2.11BSD:
:pg#112320:og#19200:bg#1024:fg#1024:tg=2.11BSD:
:ph#131520:oh#0:bh#1024:fh#1024:th=2.11BSD
```

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

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

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

4. SYSTEM SETUP

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

4.1. Creating a UNIX boot

/boot uses the device information passed to it from the bootstrap in determining the device, unit and file to load. If an autoreboot is being done the kernel will have passed the device information to the bootstrap as well as setting the autoreboot flag.

/boot does not require recompilation to adapt to a new autoreboot device.

4.2. Kernel configuration

This section briefly describes the layout of the kernel code and how files for devices are made.

4.2.1. Kernel organization

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

The directory /sys/sys contains the mainline machine independent operating system code. Files within this directory are conventionally named with the following prefixes:

init_	system initialization
kern_	kernel (authentication, process management, etc.)
quota_	kernel portion of disk quota system
subr_	misc. subroutines used throughout the kernel
sys_	system calls and the like
tty_	terminal handling
ufs_	file system
uipc_	interprocess communication
vm_	memory management

The remaining directories are organized as follows:

/sys/h	machine independent include files
/sys/conf	site configuration files and basic templates
/sys/net	network independent, but network related code
/sys/netinet	DARPA Internet code
/sys/netimp	IMP support code
/sys/netns	Xerox NS support code
/sys/pdp	PDP-11 specific mainline code
/sys/pdpif	PDP-11 network interface code
/sys/pdpmba	PDP-11 MASSBUS device drivers and related code
/sys/pdpuba	PDP-11 UNIBUS device drivers and related code

Many of these directories are referenced through /usr/include with symbolic links. For example, /usr/include/sys is a symbolic link to /sys/h. The system code, as distributed, is mostly independent of the include files in /usr/include. Unfortunately not all references to /usr/include have been eradicated, so compiling the system requires the /usr file system to be mounted.

4.2.2. Devices and device drivers

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

Determine the set of devices that you have and create a new /dev directory by running the MAKEDEV script. First create a new directory /newdev, copy MAKEDEV into it, edit the file MAKEDEV.local to provide an entry for local needs, and run it to generate a /newdev directory. For instance, if your machine has a single DZ11, a single DH11, an RM03 disk, an EMULEX UNIBUS SMD disk controller, an AMPEX 9300 disk, and a TE16 tape drive you would do:

```
# cd /
# mkdir newdev
# cp dev/MAKEDEV newdev/MAKEDEV
# cd newdev
# MAKEDEV dz0 dh0 xp0 xp1 ht0 std LOCAL
```

Note the ‘‘std’’ argument causes standard devices such as */dev/console*, the machine console, */dev/null*, */dev/tty*, */dev/klog*, etc. to be created.

You can then do

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

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

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

4.2.3. Building new system images

The kernel configuration of each UNIX system is described by a single configuration file, stored in the */sys/conf* directory. The format of this file is very simple consisting of lines starting with an *identifier* followed by a *value*. Blank lines and anything past a ‘‘#’’ (including the #) are comments. This file is processed by the shell script *config* in the same directory. The manual pages in section 4 of the UNIX manual specify the configuration lines necessary for various devices. A comprehensive list of system options with descriptions of their meanings and effects can be found in appendix A.

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

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

↑ note that non-separate systems are not currently supported

```
# cp GENERIC SYSTEM
# TERM=terminal_type; export TERM
# vi SYSTEM
# ./config SYSTEM
# cd ../SYSTEM
# make
```

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

The configured system image ''unix''‡ should be copied to the root, and then booted to try it out. It is best to save the old kernel to a known name so as not to destroy the working system until you're sure the new one does work. It is an **better** idea to have a non network kernel (/emergencyunix) always kept on the system:

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

To boot the new version of the system you should follow the bootstrap procedures outlined in section titled ''**Bootstrap and shutdown procedures**'' A systematic scheme for numbering and saving old versions of the system may be useful.

4.3. Disk configuration

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

4.3.1. Disk naming and divisions

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

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

The space available on a disk varies, not surprisingly, per device. Disklabels make a table giving sizes meaningless since there are no predefined partition sizes embedded in the kernel any longer. The root filesystem (**a**) must be at least 4Mb, preferably 6 to 7Mb if possible. The swap area (almost always the **b**

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

partition) should be about 3Mb or so. If your system has a small amount (less than 2Mb) of memory you will need more swap space, perhaps 4 or 5Mb. It is a rare case where more than 5 or 6Mb of swap space is required. The system will run out of other resources by the time enough activity is generated to need that much swap space.

The system (boot) disk has a swapping area and a root file system. Other drives may use those partitions for data. **Remember:** the **a** partition must start at sector 0 or *disklabel*(8) or else the kernel will not be able to read/write the label.

The distributed system binaries occupy about 34 Megabytes while the major sources occupy another 36 Megabytes. Adding in the miscellaneous sources, a few locate works of art bring the total for a complete system to about 90 Megabytes. This overflows RK07, RL02 and RM03 systems, but fits easily on most other hardware configurations. 2.11BSD is quite happy on RD54 or larger. Simply fitting the distribution isn't enough, there must still be space left for user files, objects when compiling programs, spooling directories, usw.

Be aware that the disks have their sizes measured in disk sectors (512 bytes), 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 for instance. The *newfs*(8) program performs this calculation automatically. You should **never** need to run *mkfs* manually. All user programs report disk space in kilobytes and, where needed, disk sizes are always specified in units of sectors. The /etc/disktab file used in making file systems specifies disk partition sizes in sectors; the default sector size may be overridden with the ''se'' attribute. **Note** that the only sector size currently supported is NBPG as defined in */sys/pdp/machparam.h*. This restriction is enforced in several places in the disklabeling process as a safeguard against specifying a sector size other than NBPG (512). Any other sector size would produce strange results and almost certainly curdled filesystems.

4.3.2. Layout considerations

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

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

The efficiency with which UNIX is able to use the CPU is often strongly affected by the configuration of disk controllers. For general time-sharing applications, the best strategy is to try to split the most actively-used sections among several disk arms.

It is critical for good performance to balance disk load. There are at least five components of the disk load that you can divide between the available disks:

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

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

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

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

4.3.3. Implementing a layout

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

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

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

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

To make the /mnt file system we would do:

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

↑ Note also, that only a single swapping area is supported. The *swapon* (2) system call and multiple swapping areas have **not** been implemented under 2.11BSD (yet. no real need since enough other resources are exhausted by the time a 4mb 11/73 needs additional swap space).

4.4. Configuring terminals

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

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

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

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

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

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

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

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

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

```
ttyp0  none  network
```

(Note the fourth field is not needed here.)

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

```
# kill −1 1
```

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

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

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

4.5. Adding users

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

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

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

4.6. Site tailoring

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

/etc/netstart

At or about line 25 in /etc/netstart you should find a line similar to:

/bin/hostname *myname.my.domain*

defines the value returned by the *gethostname* (2) system call. Your hostname should be your fully qualified domain name. Programs such as *getty* (8), *mail* (1), *wall* (1), *uucp* (1), and *who* (1) use this system call so that the binary images are site independent.

4.7. Setting up the mail system

The mail system consists of the following commands:

/bin/mail	old standard mail program, *binmail* (1)
/usr/ucb/mail	UCB mail program, described in *mail* (1)
/usr/sbin/sendmail	mail routing program
/usr/spool/mail	mail spooling directory
/etc/aliases	mail forwarding information
/usr/bin/newaliases	command to rebuild binary forwarding database
/usr/ucb/biff	mail notification enabler↑
/usr/libexec/comsat	mail notification daemon↑

Mail is normally sent and received using the *mail* (1) command, which provides a front-end to edit the messages sent and received, and passes the messages to *sendmail* (8) for routing. The routing algorithm uses knowledge of the network name syntax, aliasing and forwarding information, and network topology, as defined in the configuration file /etc/sendmail.cf, to process each piece of mail. Local mail is delivered by giving it to the program /bin/mail that adds it to the mailboxes in the directory /usr/spool/mail/*username*, using a locking protocol to avoid problems with simultaneous updates. After the mail is delivered, the local mail delivery daemon /usr/libexec/comsat is notified, which in turn notifies users who have issued a "*biff* y'' command that mail has arrived↑.

To set up the mail facility you should read the instructions in the file READ_ME in the directory /usr/src/usr.lib/sendmail and then adjust the necessary configuration files. You should also set up the file /etc/aliases for your installation, creating mail groups as appropriate. Documents describing *sendmail*'s operation and installation are also included on the distribution tape.

4.7.1. Setting up a UUCP connection

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
ny     1-315-
nj     1-201-
bostn  1-617-
```

while our L-devices file is:

```
ACU cul0 unused 1200 ventel
```

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

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

5. NETWORK SETUP

The following section has been lightly edited to correspond to the current 2.11BSD networking. Several parts of it do not really apply to 2.11BSD, for example, it is unlikely that anyone will connect a PDP-11 to an IMP but it is possible as the LH/DH-11 networking interface and the IMP modules have been ported and lightly tested, or that anyone will run the nameserver. The ‘‘correct’’ use of the networking in 2.11BSD is probably with a list of the local net addresses in the */etc/hosts* file and with one default gateway for all network traffic. In particular, do not run *routed* (8) unless you’re extremely sure that you know what you’re doing. This is doubly true if SL/IP is being used as the primary connection to the outside world. The IMP and PRONET drivers are known to work, but long term robustness is unknown. Sites that wish to hook 2.11BSD into more than a simple local ethernet may have some work ahead of them. If any additional drivers are ported, I would really like a copy.

The networking in 2.11BSD, runs in supervisor mode, separate from the mainstream kernel. There is room without overlaying to hold both a SL/IP and ethernet driver. This is a major win, as it allows the networking to maintain its mbufs in normal data space, among other things. The networking portion of the kernel resides in ‘‘/netnix’’, and is loaded after the kernel is running. Since the kernel only looks for the file ‘‘/netnix’’, it will not run if it is unable to load ‘‘/netnix’’ , sites should build and keep a non-networking kernel in ‘‘/’’ at all times, as a backup. **NOTE**: The ‘‘/unix’’ and ‘‘/netnix’’ imagines must have been created at the same time, do not attempt to use mismatched images. The ability to have **boot** tell the kernel which network image to load is on the wish list (had to have something take the place of wishing for disklabels ;-)).

2.11BSD provides support for the DARPA standard Internet protocols IP, ICMP, TCP, and UDP. These protocols may be used on top of a variety of hardware devices ranging from the IMP’s (PSN’s) used in the Internet to local area network controllers for the Ethernet. Network services are split between the kernel (communication protocols) and user programs (user services such as TELNET and FTP). This section describes how to configure your system to use the Internet networking support. 2.11BSD also includes code to support the Xerox Network Systems (NS) protocols; the basic porting work has been done, but it is completely untested.

5.1. System configuration

To configure the kernel to include the Internet communication protocols, define the INET option. This automatically defines the NLOOP option. TCP_COMPAT_42 is always defined. Xerox NS support is enabled with the NS option. In either case, include the pseudo-device ‘‘pty’’ in your machine’s configuration file, using the NPTY options. The ‘‘pty’’ pseudo-device forces the pseudo terminal device driver to be configured into the system, see *pty* (4). The NLOOP option forces inclusion of the software loopback interface driver. The loop driver is used in network testing as well as for the system talking to itself rather than transmitting the data over the wire.

If you are planning to use the Internet network facilities on a 10Mb/s Ethernet, the pseudo-device ‘‘ether’’ should also be included in the configuration using the NETHER option; this forces inclusion of the Address Resolution Protocol module used in mapping between 48-bit Ethernet and 32-bit Internet addresses. Also, if you have an IMP connection, you will need to include the pseudo-device ‘‘imp’’, using the option NIMP. The IMP software is ported and is in use at at least one site.

Before configuring the appropriate networking hardware, you should consult the manual pages in section 4 of the Programmer’s Manual. The following table lists the devices for which software support exists. Again, much of this software is unported and untested; only the basic networking has been stressed at all. Many other devices are available, but unported. Porting should simply be a matter of making the hardware device work. The directories ‘‘/sys/pdpif’’ and ‘‘/sys/vaxif’’ contain many drivers. The ones in ‘‘pdpif’’ are either the current, working drivers, or drivers that, at some time, worked on PDP-11’s. The ones in ‘‘vaxif’’ are the current VAX drivers, and, as such, will have to have their memory usage changed, but serve as an excellent example of how the hardware works.

Device name	Manufacturer and product
de	DEC DEUNA/DELUA 10Mb/s Ethernet
qe	DEC DEQNA 10Mb/s Ethernet
qt	DEC DELQA-YM 10Mb/s Ethernet
ec	3Com 10Mb/s Ethernet
il	Interlan 1010 and 10101A 10Mb/s Ethernet interfaces
vv	Proteon ProNET - Token Ring Interface
acc	LH/DH-11 1822 IMP/PSN Interface

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

All network interface drivers including the loopback interface, require that their host address(es) be defined at boot time. This is done with *ifconfig* (8) commands included in the */etc/rc.local* file. Interfaces that are able to dynamically deduce the host part of an address may check that the host part of the address is correct. The manual page for each network interface describes the method used to establish a host's address. *Ifconfig* (8) can also be used to set options for the interface at boot time. Options are set independently for each interface, and apply to all packets sent using that interface. These options include disabling the use of the Address Resolution Protocol; this may be useful if a network is shared with hosts running software that does not yet provide this function. Alternatively, translations for such hosts may be set in advance or "published" by a 2.11BSD host by use of the *arp* (8) command. Note that the use of trailer link-level is now negotiated between 2.11BSD hosts using ARP, and it is thus no longer necessary to disable the use of trailers with *ifconfig*. It is **STRONGLY** recommended, however, that 2.11BSD networking be run without trailers, as the trailer code in most of the drivers has either been removed, commented out, is untested or is **known** not to work. This is a problem with certain releases of *Ultrix*, which has to be explicitly configured not to send trailers if it and 2.11BSD are to coexist.

To use the pseudo terminals just configured, device entries must be created in the "/dev" directory. To create 32 pseudo terminals (plenty, you can probably get by with many less) execute the following commands.

```
# cd /dev
# MAKEDEV pty0 pty1
```

More pseudo terminals may be made by specifying *pty2*, *pty3*, etc. The kernel normally includes support for 16 pseudo terminals unless the configuration file specifies a different number. Each pseudo terminal really consists of two files in /dev: a master and a slave. The master pseudo terminal file is named /dev/ptyp?, while the slave side is /dev/ttyp?. Pseudo terminals are also used by several programs not related to the network. **NOTE**: the terminal structures are 78 bytes each, declaring more than 16 pseudo terminals is potentially wasteful of kernel D space. See the comment in the kernel config files. In addition to creating the pseudo terminals, be sure to install them in the */etc/ttys* file (with a 'none' in the second column so no *getty* is started).

5.2. Local subnetworks

In 2.11BSD the DARPA Internet support includes the notion of "subnetworks". This is a mechanism by which multiple local networks may appears as a single Internet network to off-site hosts. Subnetworks are useful because they allow a site to hide their local topology, requiring only a single route in external gateways; it also means that local network numbers may be locally administered. The standard describing this change in Internet addressing is RFC-950.

To set up local subnetworks one must first decide how the available address space (the Internet "host part" of the 32-bit address) is to be partitioned. Sites with a class A network number have a 24-bit address space with which to work, sites with a class B network number have a 16-bit address space, while sites with a class C network number have an 8-bit address space↑. To define local subnets you must steal some bits

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

from the local host address space for use in extending the network portion of the Internet address. This reinterpretation of Internet addresses is done only for local networks; i.e. it is not visible to hosts off-site. For example, if your site has a class B network number, hosts on this network have an Internet address that contains the network number, 16 bits, and the host number, another 16 bits. To define 254 local subnets, each possessing at most 255 hosts, 8 bits may be taken from the local part. (The use of subnets 0 and all-1's, 255 in this example, is discouraged to avoid confusion about broadcast addresses.) These new network numbers are then constructed by concatenating the original 16-bit network number with the extra 8 bits containing the local subnetwork number.

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

```
ifconfig en0 netmask 0xffffff00 128.32.1.7
```

This specifies that for interface ‘‘en0’’, the upper 24 bits of the Internet address should be used in calculating network numbers (netmask 0xffffff00), and the interface's Internet address is ‘‘128.32.1.7’’ (host 7 on network 128.32.1). Hosts *m* on sub-network *n* of this network would then have addresses of the form ‘‘128.32.*n.m*’’; for example, host 99 on network 129 would have an address ‘‘128.32.129.99’’. For hosts with multiple interfaces, the network mask should be set for each interface, although in practice only the mask of the first interface on each network is actually used.

5.3. Internet broadcast addresses

The address defined as the broadcast address for Internet networks according to RFC-919 is the address with a host part of all 1's. The address used by 4.2BSD was the address with a host part of 0. 2.11BSD uses the standard broadcast address (all 1's) by default, but allows the broadcast address to be set (with *ifconfig*) for each interface. This allows networks consisting of both 4.2BSD and 2.11BSD hosts to coexist. In the presence of subnets, the broadcast address uses the subnet field as for normal host addresses, with the remaining host part set to 1's (or 0's, on a network that has not yet been converted). 2.11BSD hosts recognize and accept packets sent to the logical-network broadcast address as well as those sent to the subnet broadcast address, and when using an all-1's broadcast, also recognize and receive packets sent to host 0 as a broadcast.

5.4. Routing

If your environment allows access to networks not directly attached to your host you will need to set up routing information to allow packets to be properly routed. Two schemes are supported by the system. The first scheme employs the routing table management daemon *routed* to maintain the system routing tables. The routing daemon uses a variant of the Xerox Routing Information Protocol to maintain up to date routing tables in a cluster of local area networks. By using the */etc/gateways* file created by *htable* (8), the routing daemon can also be used to initialize static routes to distant networks (see the next section for further discussion). When the routing daemon is started up (usually from */etc/rc.local*) it reads */etc/gateways* if it exists and installs those routes defined there, then broadcasts on each local network to which the host is attached to find other instances of the routing daemon. If any responses are received, the routing daemons cooperate in maintaining a globally consistent view of routing in the local environment. This view can be extended to include remote sites also running the routing daemon by setting up suitable entries in */etc/gateways*; consult *routed* (8) for a more thorough discussion.

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

route add default *smart-gateway* 1

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

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

```
# netstat –r
```

will display the contents of the routing tables, while

```
# netstat –r –s
```

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

5.5. Use of 2.11BSD machines as gateways

Only sheer insanity could prompt the use of 2.11BSD machines as gateways. If you **really** want to do this then the best recourse is to prowl the sources and see what has to be done. The code is all there, and the "ipforwarding" variable is present.

Local area routing within a group of interconnected Ethernets and other such networks may be handled by *routed* (8). Gateways between the Internet and one or more local networks require an additional routing protocol, the Exterior Gateway Protocol (EGP), to inform the core gateways of their presence and to acquire routing information from the core.

5.6. Network servers

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

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

or if you are using the *tcp_wrapper* program as

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

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

5.7. Network data bases

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

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

The files distributed are set up for Internet hosts. Local networks and hosts should be added to describe the local configuration. Network numbers will have to be chosen for each Ethernet. For sites not connected to the Internet, these can be chosen more or less arbitrarily, otherwise the normal channels should be used for allocation of network numbers.

5.7.1. Regenerating /etc/hosts and /etc/networks

When using the host address routines that use the Internet name server, the file */etc/hosts* is only used for setting interface addresses and at other times that the server is not running, and therefore it need only contain addresses for local hosts. There is no equivalent service for network names yet. The days of retrieving a host file containing all systems on the Internet are over. Besides, you would grow very old and run out of disk space while waiting for *mkhosts* (8) to process a hosts file containing the several million entries. Therefore the details of retrieving a master hosts file using *htable* (8) and *gettable* (8) have been removed from this document. However if you do use local hosts files you will still need to run *mkhosts* (8) and this is described below.

If you are using the host table for host name and address mapping, you should run *mkhosts* (8) after installing */etc/hosts*. The *mkhosts* (8) program has been enhanced for 2.11BSD to allow multiple addresses per host. The order in which the addresses are given in */etc/hosts* is preserved, so the entries for a given host should be in order of importance. If you are using the name server for the host name and address mapping, you only need to install *networks* and a small copy of *hosts* describing your local machines. The full host table in this case might be placed somewhere else for reference by users. The gateways file may be installed in */etc/gateways* if you use *routed* (8) for local routing and wish to have static external routes installed when *routed* is started. This procedure is essentially obsolete, however, except for individual hosts that are on the Milnet and do not forward packets from a local network. Other situations require the use of **gated**. That program can never be made to run on a PDP-11 due to address space considerations. Also, the networking code could not even begin to handle the number of routes which would be received.

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

5.7.2. /etc/hosts.equiv

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

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

```
wlv
wlonex
wlonex0
wlbr
```

5.7.3. /etc/rc.local

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

Program	Server	Started by
rshd	shell server	inetd
rexecd	exec server	inetd
rlogind	login server	inetd
telnetd	TELNET server	inetd
ftpd	FTP server	inetd
fingerd	Finger server	inetd
tftpd	TFTP server	inetd
rwhod	system status daemon	/etc/rc
syslogd	error logging server	/etc/rc
sendmail	SMTP server	/etc/rc
routed	routing table management daemon	/etc/rc

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

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

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

5.7.4. /etc/ftpusers

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

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

```
# cd ~ftp
# chmod 555 .; chown ftp .; chgrp ftp .
# mkdir bin etc pub
# chown root bin etc
# chmod 555 bin etc
# chown ftp pub
# chmod 777 pub
# cd bin
# cp /bin/sh /bin/ls .
# chmod 111 sh ls
# cd ../etc
# cp /etc/passwd /etc/group .
# chmod 444 passwd group
```

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

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

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

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

```
uucp
root
```

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

6. 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.

6.1. Bootstrap and shutdown procedures

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

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

```
>>> B
```

will boot from the default device and ask for the name of the system to be booted. Other systems such as the 11/44 require a device name to be given:

```
>>> B DU
```

to boot from a MSCP/UDA device. Typing a carriage return will cause the default system (as compiled in in section 4.1), to be booted. In any case, the system selected will come up in single-user mode.

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

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

```
# kill 1
```

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

```
# cd /
# umount -a
# ˆD
```

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

6.2. Device errors and diagnostics

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

Error messages printed by the devices in the system are described with the drivers for the devices in section 4 of the programmer’s manual. Some drivers have been modified to use the kernel logger, others still simply do printf statements. If errors occur suggesting hardware problems, you should contact your hardware support group or field service. It is a good idea to examine the error log files regularly (e.g. with ‘‘tail –r */usr/adm/messages*’’).

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

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

Thus a typical dump sequence would be:

tape name	level number	date	opr	size
FULL	0	Nov 24, 1979	jkf	137MB
D1	3	Nov 28, 1979	jkf	29MB
D2	2	Nov 29, 1979	rrh	34MB
D3	5	Nov 30, 1979	rrh	19MB
D4	4	Dec 1, 1979	rrh	22MB
W1	1	Dec 2, 1979	etc	40MB
D5	3	Dec 4, 1979	rrh	15MB
D6	2	Dec 5, 1979	jkf	25MB
D7	5	Dec 6, 1979	jkf	15MB
D8	4	Dec 7, 1979	rrh	19MB
W2	1	Dec 9, 1979	etc	118MB
D9	3	Dec 11, 1979	rrh	15MB
D10	2	Dec 12, 1979	rrh	26MB
D1	5	Dec 15, 1979	rrh	14MB
W3	1	Dec 17, 1979	etc	71MB
D2	3	Dec 18, 1979	etc	13MB
FULL	0	Dec 22, 1979	etc	135MB

Weekly dumps are done often enough that daily dumps always fit on one tape.

Dumping of files by name is best done by *tar* (1) but the amount of data that can be moved in this way is limited to a single tape. Finally if there are enough drives entire disks can be copied with *dd* (1) using the raw special files and an appropriate blocking factor; the number of sectors per track is usually a good value to use, consult */etc/disktab*.

It is desirable that full dumps of the root file system be made regularly. These dumps should be made in "bootable" format, including the standalone programs mentioned back in chapter 2 (boot, mkfs, restor and icheck). This can easily be done by going to /sys/pdpstand and doing:

```
make all
./maketape /dev/nrmtXX maketape.data
dump 0u /
```

This is especially true when only one disk is available. Then, if the root file system is damaged by a hardware or software failure, you can rebuild a workable disk doing a restore in the same way that the initial root file system was created.

Exhaustion of user-file space is certain to occur now and then and may be managed with a combination of disc quotas (the 4.3BSD disc quota system is available as a kernel configuration option), threatening messages of the day, and personal letters.

6.4. Moving file system data

If you have the equipment, the best way to move a file system is to dump it to magtape using *dump* (8), use *newfs* (8) to create the new file system, and restore the tape, using *restor* (8). If for some reason you don't want to use magtape, dump accepts an argument telling where to put the dump; you might use another disk. Filesystems may also be moved by piping the output of a *tar* (1) to another *tar*. The *restor* program accesses the raw device, laying down inodes and blocks in the same place they came from as recorded by dump. Care must therefore be taken when restoring a dump into a file system smaller than the original file system.

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

1. GET A SECOND PACK!!!!
2. Dump the root file system to tape using *dump* (8).
3. Bring the system down and mount the new pack.
4. Load the distribution tape and install the new root file system as you did when first installing the system.
5. Boot normally using the newly created disk file system.

Note that if you add new disk drivers they should also be added to the standalone system in */sys/pdpstand*. If you change the disk partition tables the default disk partition tables in */etc/disktab* should be modified.

6.5. Recompiling and reinstalling system software

It is easy to regenerate the system, and it is a good idea to try rebuilding pieces of the system to build confidence in the procedures. The system consists of two major parts: the kernel itself (/sys) and the user programs (/usr/src and subdirectories). The major part of this is /usr/src.

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

```
# make
```

and then installed by

```
# make install
```

Similar to the system,

```
# make clean
```

cleans up.

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

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

```
# whereis date
date: /usr/src/bin/date.c /bin/date /usr/man/man1/date.1
# cd /usr/src/bin
# make date
```

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

```
# install −s date /bin/date
```

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

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

make
make DESTDIR=*pathname* install

To regenerate all the system source you can do

```
# cd /usr/src
# make
```

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

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

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

6.6. Making local modifications

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

6.7. Accounting

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

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

6.8. Resource control

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

6.9. Files that need periodic attention

The discussion of system operations is concluded by listing the files that require periodic attention or are system specific

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

APPENDIX A – KERNEL CONFIGURATION OPTIONS

7.1. Kernel configuration options

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

Prototypes for all the following options can be found in the generic kernel configuration file */sys/conf/GENERIC*. The process of configuring a new kernel consists simply of copying the generic configuration file to a new file, *SYSTEM* and then editing the options in *SYSTEM* to reflect your needs. You can treat the items copied from GENERIC as a ''grocery list'', checking off those options you want, crossing out those you don't and setting numeric parameters to reasonable values.

7.2. Configuring the number of mountable file systems (NMOUNT)

Because of time constraints the **NMOUNT** constant was not moved into the kernel configuration file where it belongs. **NMOUNT** is used to configure the number of mountable file systems in 2.11BSD. Since each slot in the kernel mount table takes up close to a half Kb of valuable kernel data space, the distribution kernel comes configured with **NMOUNT** set to 5. This is almost certainly too small for most sites and should be increased to the number of file systems you expect to mount.

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

7.3. GENERIC kernel configuration

All of the generic kernels support the following devices:

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

The generic kernel adapts automatically to the booted device. The 'a' partition on the booted device is automatically made the root filesystem and the 'b' partition the swap area (except for the RL02 which uses the second drive). The size of the swap partition is determined at run time, the kernel queries the driver for the number of block in the 'b' partition. **NOTE:** If the swap partition is not labeled as being of type *swap* the kernel will panic.

7.3.1. GENERIC kernel configuration file

```
# Machine configuration file for 2.11BSD distributed kernel.
#
# Format:
#       name    value           comments
# An item's value may be either numerical, boolean or a string; if it's
# boolean, use "YES" or "NO" to set it or unset it, respectively.  Use
```

```
# the default value and the comments field as indicators of the type of
# field it is.

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

# Machine type
# Split I/D and hardware floating point are required.
#
# Including UNIBUS map support for machines without a UNIBUS will not cause
# a kernel to die.  It simply includes code to support UNIBUS mapping if
# present.
#
# The define UNIBUS_MAP implements kernel support for UNIBUS mapped
# machines.  However, a kernel compiled with UNIBUS_MAP does *not* have to
# be run on a UNIBUS machine.  The define simply includes support for UNIBUS
# mapping if the kernel finds itself on a machine with UNIBUS mapping.
UNIBUS_MAP  YES                 # include support for UNIBUS mapping

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

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

# PDP-11 machine type; allowable values are GENERIC, 44, 70, 73.  GENERIC
# should only be used to build a distribution kernel.  The only use of this
# option is to select the proper in-line PS instructions (references to the
# PSW use 'spl', 'mfps/mtps' or 'movb' instructions depending on the cpu type).
PDP11       GENERIC             # distribution kernel
#PDP11      44                  # PDP-11/44
#PDP11      70                  # PDP-11/70,45,50,55
#PDP11      73                  # PDP-11/73,53,83,93,84,94

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

IDENT       GENERIC                     # machine name
MAXUSERS    4                   # maxusers on machine

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

```
#BOOTDEV     sc11              # Emulex SC11/B boot device
#BOOTDEV     sc21              # Emulex SC21 boot device
#BOOTDEV     si                # si 9500 boot device

# Timezone, in minutes west of GMT
#TIMEZONE    300               # EST
#TIMEZONE    360               # CST
#TIMEZONE    420               # WST
TIMEZONE     480               # PST
DST          1                 # Daylight Savings Time (1 or 0)

# Filesystem configuration
# Rootdev, swapdev etc. should be in terms of makedev.  For example,
# if you have an SMD drive using the xp driver, rootdev would be xp0a,
# or "makedev(10,0)".  Swapdev would be the b partition, xp0b, or
# "makedev(10,1)".  The 10 is the major number of the device (the offset
# in the bdevsw table in conf.c) and the 0 and 1 are the minor numbers
# which correspond to the partitions as described in the section 4 manual
# pages.  You can also get the major numbers from the MAKEDEV script in
# /dev.
PIPEDEV         makedev(10,0)          # makedev(10,0) xp0a
ROOTDEV         makedev(10,0)          # makedev(10,0) xp0a
SWAPDEV         makedev(10,1)          # makedev(10,1) xp0b

# DUMPROUTINE indicates which dump routine should be used.  DUMPDEV
# should be in terms of makedev.  If DUMPDEV is NODEV no automatic
# dumps will be taken, and DUMPROUTINE needs to be set to "nulldev" to
# resolve the reference.  See param.h and ioconf.c for more information.
# DUMPLO should leave room for the kernel to start swapping without
# overwriting the dump.
DUMPLO       512               # dump start address
DUMPDEV         NODEV                  # makedev(10,1) xp0b
DUMPROUTINE nulldev                    # no dump routine.
#DUMPROUTINE hkdump            # hk driver dump routine
#DUMPROUTINE hpdump            # hp driver dump routine
#DUMPROUTINE radump            # ra driver dump routine
#DUMPROUTINE rldump            # rl driver dump routine
#DUMPROUTINE rmdump            # rm driver dump routine
#DUMPROUTINE brdump            # br driver dump routine
#DUMPROUTINE sidump            # si driver dump routine
#DUMPROUTINE xpdump            # xp driver dump routine
#DUMPROUTINE tmsdump                   # tms driver dump routine

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

BADSECT         NO             # bad-sector forwarding
EXTERNALITIMES  YES            # map out inode time values
UCB_CLIST    NO                # clists moved from kernel data space
NOKA5        NO                # KA5 not used except for buffers
                               # and clists (_end < 0120000);
QUOTA        NO                # dynamic file system quotas
                               # NOTE -- *very* expensive
```

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

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

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

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

NBR          0             # EATON BR1537/BR1711, BR1538A, B, C, D

NHK          2             # RK611, RK06/07

NRAC         1             # NRAD controllers
NRAD         2             # RX50, RC25, RD51/52/53, RA60/80/81

NRK          0             # RK05

NRL          2             # RL01/02

NRX          0             # RX02

NSI          0             # SI 9500 driver for CDC 9766 disks

# Because the disk drive type registers conflict with other DEC
# controllers, you cannot use XP_PROBE for the Ampex 9300 and
# Diva drives.  Read through /sys/pdpuba/hpreg.h and /sys/pdpuba/xp.c
# for information on how to initialize for these drives.
NXPC         1             # NXPD controllers (RH70/RH11 style)
NXPD         2             # RM02/03/05, RP04/05/06, CDC 9766,
                           # Ampex 9300, Diva, Fuji 160, SI Eagle.
XP_PROBE     YES           # check drive types at boot

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

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

```
#########################################

NHT             2               # TE16, TU45, TU77

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

NTS             2               # TS11

NTMSCP          2               # TMSCP controllers
NTMS            2               # TMSCP drives
TMSCP_DEBUG     NO              # debugging code in TMSCP drive (EXPENSIVE)

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

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

#########################################
# PERIPHERALS: OTHER                    #
#########################################
NDN             0               # DN11 dialer
NLP             0               # Line Printer
LP_MAXCOL       132             # Maximum number of columns on line printers
NDR             0               # DR11-W

#########################################
# PSEUDO DEVICES, PROTOCOLS, NETWORKING #
#########################################
# Networking only works with split I/D and SUPERVISOR space, i.e. with the
# 11/44/45/50/53/55/70/73/83/84.  NETHER should be non-zero for networking
# systems using any ethernet.  CHECKSTACK makes sure the networking stack
# pointer and the kernel stack pointer don't collide; it's fairly expensive
# at 4 extra instructions for EVERY function call AND return, always left
# NO unless doing serious debugging.
INET            NO              # TCP/IP
CHECKSTACK      NO              # Kernel & Supervisor stack pointer checking
NETHER          0               # ether pseudo-device

# Note, PTY's and the select(2) system call do not require the kernel to
# be configured for networking (INET).  Note that you can allocate PTY's
```

```
# in any number (multiples of 8, of 16, even, odd, prime, whatever).  Nothing
# in the kernel cares.  PTY's cost 78 bytes apiece in kernel data space.  You
# should probably have at least 8-10 since several applications use them:
# script, jove, window, rlogin, ...
NPTY        0           # pseudo-terminals - GENERIC sys needs NONE

# To make the 3Com Ethernet board work correctly, splimp has to be promoted
# to spl6; splfix files that do this are in conf/3Com; the config script
# does the right thing.
NEC         0           # 3Com Ethernet
NDE         0           # DEUNA/DELUA
NIL         0           # Interlan Ethernet
NSL         0           # Serial Line IP
NQE         0           # DEQNA
NQT         0           # DEQTA (DELQA-YM, DELQA-PLUS)
NVV         0           # V2LNI (Pronet)
NACC        0           # ACC LH/DH ARPAnet IMP interface
PLI         NO          # LH/DH is connected to a PLI
NIMP        0           # ARPAnet IMP 1822 interface

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

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

# Defining FPSIM to YES compiles a floating point simulator into the kernel
# which will catch floating point instruction traps from user space.  This
# doesn't work at present.
FPSIM       NO          # floating point simulator

# To enable profiling, the :splfix script must be changed to use spl6 instead
# of spl7 (see conf/:splfix.profile), also, you have to have a machine with a
# supervisor PAR/PDR pair, i.e. an 11/44/45/50/53/55/70/73/83/84, as well
# as both a KW11-L and a KW11-P.
#
# Note that profiling is not currently working.  We don't have any plans on
# fixing it, so this is essentially a non-supported feature.
PROFILE             NO          # system profiling with KW11P clock

INGRES      NO          # include the Ingres lock driver
```

APPENDIX B – STANDALONE DISKLABEL PROGRAM

8.1. Standalone disklabel example

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

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

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

BOOT> **MU 0**

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

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

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

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

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

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

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

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

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

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

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

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

7 partitions:

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

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

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

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

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

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

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

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

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

8.2. Standalone disklabel program

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

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

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

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

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

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

8.3. Disklabel – tour of the levels.

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

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

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

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

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

> Type is one of: SMD, MSCP, old DEC, SCSI, ESDI, ST506, floppy.
>
> Name is a string up to 16 characters in length. It is typically something like **rd54** or **rm03** but may be any meaningful string.
>
> Label is an arbitrary string up to 16 characters in length – nothing in the system checks for or depends on the contents of the pack label string.
>
> Rpm is the rotational speed of the drive. Nothing in the system uses or depends on this at the present time. Default is 3600.
>
> Drivedata consists of 5 longwords of arbitrary data. Reserved for future use.

modify misc flags

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

> Ecc says that the controller/driver can correct errors.
>
> Badsect indicates that the controller/driver supports bad sector replacement.
>
> Removable indicates that the drive uses removable media (floppy, RL02, RA60 for example).

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

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

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

> Zero clears the size and offset fields of a partition entry and sets the filesystem type to **unused**.
>
> Type is the filesystem type and of the possible choices only **2.11BSD**, **swap** and **unused** make any sense to specify.
>
> Offset is the number of sectors from the beginning of the disk at which the partition starts.
>
> Size is the number of sectors which the partition occupies.
>
> Frag is the number of fragments a filesystem block can be broken into. It is not presently used and should be left at the default of 1.
>
> Fsize is the filesystem blocksize and should be left at the default of 1024.
>
> modify partitions
> d(isplay) n(umber) s(elect) q(uit)? **q**
> modify
> d(isplay) g(eometry) m(isc) p(artitions) q(uit)? **q**
> d(isplay) D(efault) m(odify) w(write) q(uit)? **q**
> Label changed. Discard changes [y/n]? **y**
>
> 73Boot from tms(0,0,1) at 0174500

8.4. Disklabel − helpful hints and tips.

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

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

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

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

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