1194

1/80

Setting Up PWB/UNIX

R. C. Haight

Bell Laboratories
Murray Hill, New Jersey 07974

M. J. Petrella

Bell Laboratories
Piscataway, New Jersey 08854

L. A. Wehr

Bell Laboratories
Murray Hill, New Jersey 07974

1. INTRODUCTION

1.1 Prerequisites

Before attempting to generate a PWB/UNIX* system, you should understand that a considerable knowledge of the attendant documentation is required and assumed. In particular, you should be very familiar with the following documents:

- *The UNIX Time-Sharing System*
- *PWB/UNIX User's Manual*
- *C Reference Manual*
- *Administrative Advice for UNIX/TS*
- *PWB/UNIX Operations Manual*

A complete list of pertinent documentation is contained in *Documents for PWB/UNIX*. Throughout this document, each reference of the form *name*(N), where N is a number, refers to entry *name* in Section N of the *PWB/UNIX User's Manual*.

You must have a basic understanding of the operation of the hardware. This includes the console panel, the tape drives, and the disk drives, all of which are assumed to have standard addresses and interrupt vectors. It is also assumed that the hardware works and has been completely installed. The DEC® diagnostics should have been run to test the configuration, and you must have a detailed description of the hardware, including device addresses, interrupt vectors, and bus levels. This information is very important to generate the PWB/UNIX system.

Older versions of PWB/UNIX cannot be correctly *updated* with a PWB/UNIX Release 2.0 system; therefore, the installation of PWB/UNIX Release 2.0 must be done as an initial load.

1.2 Procedure

PWB/UNIX is distributed on two magnetic tapes, recorded in 9-track format at 800-bpi. Tape 1 is essential to the entire procedure, because it contains the initial load program and a copy of the *root* file system, as well as a copy of the */usr* file system (which contains source and supplemental commands). The initial load program will copy the *root* file system from tape (either a TU10 or a TU16) to disk (either an RP03 or an RP06). Note that RP04 and RP05 drives are considered to be equivalent to RP06 drives; any differences will be noted explicitly. Once the *root* file system has been successfully loaded to disk, PWB/UNIX may be booted and the available utility programs may then be used to complete the installation. The */usr* file system contains information essential to generating a new system that will match your particular hardware and software environment.

Tape 2 contains the machine-readable manual pages, as well as the source for selectable subsystems of PWB/UNIX.

---

* UNIX is a Trademark of Bell Laboratories.

During updates, the *cpio*(1) program will not replace any file if its replacement has a modification time that is less than (i.e., earlier than) the modification time of the original file. This can be due to local modifications. In addition, special files are *never* updated. Furthermore, certain administrative files (e.g., */etc/passwd*, */usr/lib/crontab*) are sent with a modification time of Jan 1, 1970 to ensure that they do not replace their counterparts during updates. Any file not copied will cause *cpio*(1) to print a message to that effect. These messages should always be investigated to ensure that any files not copied were of that type. However, note that, depending on respective modification times, a locally-modified file may get updated, thus destroying the local modifications.

There are several difficulties that can arise when installing a PWB/UNIX system. One of the most common problems is running out of disk space when performing an update. Should this occur, the original contents of the file system should be restored from a backup copy and the contents of the update tape should be read into a spare file system using the *cpio*(1) program. Unwanted material can then be removed and the original file system can be updated from this new file system using the −p option of *cpio*(1). Modification times of files should also be preserved using the −m option of *cpio*(1).

2. LOAD PROCEDURES

2.1 Tape 1 Format

Tape 1 contains five files: a loader, a physical copy of the *root* file system, the *cpio*(1) program, a *cpio* structured copy of the *root* file system, and a *cpio* structured copy of the */usr* file system. *Root* refers to the directory "/", which is the root of all the directory trees. The format of this tape is as follows:

file 1:	
record 0:	Tape boot loader—512-bytes;
record 1:	Tape boot loader—512-bytes (same as record 0);
remainder of file 1:	Initial load program—several 512-byte records;
end-of-file	
file 2:	*root* file system (physical)—5,120-byte records (blocking factor 10);
end-of-file	
file 3:	*cpio* program—512-byte records;
end-of-file	
file 4:	*root* file system (structured in *cpio* format)—5,120-byte records;
end-of-file	
file 5:	*/usr* file system (same format as file 4).
end-of-file	

The *root* (/) file system contains the following directories:

bck:	Directory used to mount a backup file system for file restoring.
bin:	Public commands; most of what's described in Section 1 of the *PWB/UNIX User's Manual*.
dev:	Special files, all the devices on the system.
etc:	Administrative programs and tables.
lib:	Public libraries, parts of the assembler, C compiler.
mnt:	Directory used to mount a file system.
stand:	Stand-alone boot programs.
tmp:	Directory used for temporary files; should be cleaned at reboot.
usr:	Directory used to mount the */usr* file system; user directories often kept here also.

2.2 Initial Load of root

Mount Tape 1 on drive 0 and position it at the load point. Next, bootstrap the tape by reading either record 0 or record 1 into memory starting at address 0 and start execution at address 0. This may be accomplished by using a standard DEC ROM bootstrap loader, a special ROM, or some manual procedure. See *romboot*(8), *tapeboot*(8), and *70boot*(8).

The tape boot loader will then type "UNIX tape boot loader" on the console terminal and read in and execute the initial load program. The program will then type detailed instructions about the operation of the program on the console terminal. Next, it will ask what type of disk drive you have and which drive you plan to use for the copy. The disk controller used must be at the standard DEC address indicated by the program. However, other disk controllers on your system may be at non-standard addresses. You must mount a formatted, error-free pack on the drive you have indicated. If necessary, use the appropriate DEC diagnostic program to format the pack. Note that the pack will be written on. Next, the program will ask what type of tape drive you have and which drive contains Tape 1. Normally, this will be drive 0, but the program will work with other drives. Note that the tape is currently positioned correctly after the end-of-file between the initial load program and the *root* file system. When everything is ready, the program will copy the *root* file system from the tape to the disk and give instructions for booting PWB/UNIX. After the copy is complete and you have booted the basic version of PWB/UNIX, check (using *fsck*(1M)) the *root* file system and browse through it.

The file */stand/mmtest* is a stand-alone memory mapping diagnostic program. If you are not *absolutely* sure that DEC FCO (field change order) M8140-R002 has been applied to your PDP 11/70 CPU, *stand/mmtest* should be booted and allowed to run at least 20 minutes. To boot this program, go through the boot procedure, but specify:

```
0= stand/mmtest
```

PWB/UNIX Release 2.0 comes with optional power-fail recovery. This feature *requires* that the *power-up* and *power-down* interrupt vectors be 024.

2.3 Update of root

It is very important that the system be running in *single-user* mode during the update phase. To update an already existing *root* file system, files three and four on Tape 1 will be used. It is necessary to first make a copy of your *root* file system using *volcopy*(1M) and then update this copy. The copy should be made on a separate disk pack using the same section number as your *root* file system (always section 0). Also, after the update is completed, check if any of your local administrative files in the directory */etc* need modification. Most of these are mentioned in Section 4 below.

Mount Tape 1 on drive 0 and position it at the load point. We assume that disk drive 1 is available for making the copy, and that the *root* file system is on */dev/rp0*. The following procedure will then make a copy of the *root* file system, and then update this copy. Note that */dev/mt4* refers to tape drive 0 but has the side effect of spacing forward to the next end-of-file (no rewind option). The −B option of *cpio* specifies that the input is in 5,120-byte records.

```
volcopy root /dev/rrp0 pkname1 /dev/rrp10 pkname2
mount /dev/rp10 /bck
# The two echoes will position the tape at file 3
echo </dev/mt4; echo </dev/mt4
cp /dev/mt4 /bck/bin/cpio
chmod 755 /bck/bin/cpio
chown bin /bck/bin/cpio
cd /bck
/bck/bin/cpio -idmB </dev/rmt0
cd /
umount /dev/rp10
```

Pkname1 and *pkname2* are the volume names of the source and destination disk packs, respectively. If the new copy is satisfactory, shut down and halt the system, change disk packs, and reboot the system using the new *root*.

2.4 File 5 (/usr) Format

File 5 contains the */usr* file system in *cpio*(1) format (5,120-byte records). The */usr* file system contains commands and files that must be available (mounted) when the system is in *multi-user* mode. The file contains the following directories:

adm:	Miscellaneous administrative command and data files, including the connect-time accounting file *pacct*.
bin:	Public commands; an overflow for */bin*.
dict:	Dictionaries for word processing programs.
games:	Various demonstration and instructional programs.
include:	Public C language *#include* files.
lib:	Archive libraries, including the text processing macros; also contains data files for various programs, such as *spell*(1) and *cron*(1M).
mdec:	Hardware bootstrap loaders and programs.
news:	Place for all the various system news.
pub:	Handy public information, e.g., table of ASCII characters.
spool:	Spool directory for daemons.
src:	Source for commands, libraries, the operating system, etc.
tmp:	Directory for temporary files; should be cleaned at reboot.

2.5 Initial Load of /usr

Mount a file system (device) as */usr*. The ultimate size and location of this file system on a device is an administrative decision; initially, the following procedure will suffice:

```
# The four echoes will position the tape at file 5
echo </dev/mt4
echo </dev/mt4
echo </dev/mt4
echo </dev/mt4
cd /
mkfs /dev/rrp1 35000 7 418
# For RP03 disks, the last argument above should be 200
labelit /dev/rrp1 usr pkname
fsck /dev/rrp1
mount /dev/rp1 /usr
chmod 755 /usr
cd /usr
cpio -idmB </dev/rmt0
```

Pkname is the volume name of the pack (e.g., "p0001"). After the copy is complete, check (using *fsck*(1M)) the */usr* file system and browse through it.

Because */usr* must be mounted when the system is in *multi-user* mode, the file */etc/rc* must be changed to include the command lines:

```
mount /dev/rp1 /usr
```

and

```
umount /dev/rp1
```

These lines must be inserted at the appropriate places in */etc/rc*, as indicated by the comments in the prototype file. Next, the file */etc/checklist* should be changed to include */dev/rrp1*; see also *fsck*(1M), *labelit*(1M), *mkfs*(1M), *mount*(1M).

2.6 Update of /usr

It is advisable that the system be running in *single-user* mode during the update phase. It is also wise to first make a copy of your */usr* file system for backup purposes. Next, mount Tape 1 on drive 0 and position it at file 5. The */usr* file system *must* be mounted. The following procedure will perform the update:

```
cd /usr
cpio -idmB </dev/rmt0
```

2.7 Tape 2 (Selectable Items) Format

Tape 2 contains four selectable subsystems for PWB/UNIX. Additional subsystems may be added in the future. Any item not desired can simply be skipped. The format of this tape is as follows:

file 1:	manual pages (*cpio* format) – 5,120-byte records:
end-of-file	
file 2:	rje software (same format as file 1);
end-of-file	
file 3:	graphics software (same format as file 1);
end-of-file	
file 4:	M2 software (same format as file 1).
end-of-file	

A table of contents of this tape may be obtained by using the –t option of *cpio*(1); after installation, files and directories deemed useless by the local administrator may be easily removed. Alternately, only parts of the tape may be extracted using the pattern matching capabilities of *cpio*(1).

2.8 Initial Load or Update of Selectable Items

The initial load and update procedures are essentially the same; the only exception being the creation of the selectable item directory on the initial load.

Mount Tape 2 on drive 0 and position it at the load point. Make sure that the */usr* file system is mounted. The following procedure will read in the source for each of the selectable subsystems. If a particular subsystem is not desired, simply skip that file on the tape by executing the following command:

```
echo </dev/mt4
```

The tape can be rewound after any subsystem by specifying */dev/rmt0* instead of */dev/rmt4*.

```
cd /usr
mkdir man; chown bin man; chmod 755 man
cd man
cpio -idmB </dev/rmt4


cd /usr/src/cmd
mkdir rje; chown bin rje; chmod 755 rje
cd rje
cpio -idmB </dev/rmt4
```

```
cd /usr/src/cmd
mkdir graf; chown bin graf; chmod 755 graf
cd graf
cpio -idmB </dev/rmt4
```

```
cd /usr/src/cmd
mkdir M2; chown bin M2; chmod 755 M2
cd M2
cpio -idmB </dev/rmt0
```

After installing the source for the *rje*, *graphics*, and *M2* subsystems, the software must be *built* and *installed*. (Only execute the following commands for the subsystems that you have elected to take.)

To build and install graphics:

```
cd /usr/src
./:mkcmd graf
```

To build and install *rje*, select the appropriate *./:mkcmd* lines:

```
cd /usr/src
./:mkcmd rje hasp    (makes a single hasp system, including send and rjestat)
./:mkcmd rje hasp2   (makes hasp and hasp2)
./:mkcmd rje hasp3   (makes hasp, hasp2, and hasp3)
./:mkcmd rje uvac    (makes a single uvac system, including send and rjestat)
./:mkcmd rje uvac2   (makes uvac and uvac2)
./:mkcmd rje uvac3   (makes uvac, uvac2, and uvac3)
```

The *M2* file supplied is only a place holder at this time.

2.8.1 UNIVAC Remote Job Entry

UNIVAC *rje* requires an additional tape to be used on the UNIVAC side of the system. The UNIVAC *rje* program is duplicated in the first and second files of its release tape in @COPY.G format. One should be read into a file called *RJEFIL*. The documentation on the configuration skeleton can be obtained by executing:

```
@DOC RJEFILE.DOCELM
```

Examples of *rje* configurations are in the file elements *CONFIGRTN* (for level 35 or higher), and *CONFIGRTN/L32* and *CONFIGRTN/L33* (for levels 32 and 33 respectively). A start stream that will build the *rje* program, and cull it, is in *START/RJEMAK* (the account and project fields should be modified). A sample *rje* start stream where the absolute is placed in *RJE=RJE.RJECTL/PDP* is in *START/RJECTL*. Other sample runstreams using the file *SYSS=ABS.RJECTL/PDP* are in *RJEU1*, *RJEU2*, and *RJEU3* for using the first, second, or third configured line.

The LINEDATA cards in *CONFIGRTN* assume the EXEC is configured with 3 lines to a UNIX system whose CTM and LINE cards specify GB17, GB18, and GA01 respectively, and whose STATION cards specify a LOCAL station of UNIX01, UNIX02, and UNIX03 respectively. The LINEDATA cards in *CONFIGRTN/L32* and *CONFIGRTN/L33* assume the EXEC is configured with one line to a UNIX system whose LTG ID is 'UNXRJE' and whose REMOTE TERMINAL card specifies BPDP01.

3. CONFIGURATION PLANNING

3.1 PWB/UNIX Configuration

The basic PWB/UNIX operating systems supplied on Tape 1 support only the console, a disk controller (disk drive 0), and a tape controller (tape drive 0). The actual configuration of your system must be described by you. All of the PWB/UNIX operating system source code and object libraries are in */usr/src/uts*. All of the configuration information is kept in the directory */usr/src/uts/cf*. There are only two files that must be changed to reflect your system configuration, *low.s* and *conf.c*; the program *config*(1M) makes this task relatively simple.

Config requires a *system description file* and produces the two needed files. The first part of the system description file lists all of the hardware devices on your system. Next, various system information is listed. A brief explanation of this information follows (for more details of syntax and structure, see *config*(1M), and *master*(5)):

- *root*— Specifies the device where the *root* file system is to be found. The device must be a block device with read/write capability because this device will be mounted read/write as "/". Thus, a tape can not be mounted as the *root*, but can be mounted as some read-only file system. Normally, *root* is disk drive 0, section 0.
- *swap*— Specifies the device and blocks that will be used for *swapping*. *Swplo* is the first block number used and *nswap* indicates how many blocks, starting at *swplo*, to use. *Swplo* can not be zero. Typically, systems require approximately 2,000 blocks. Care must be taken that the swap area specified does not overlap any file system. For example, if section 0 is 8,000 blocks long, the *root* file system could occupy the first 6,000 blocks and *swap* the remaining 2,000 by specifying:

```
root rp04 0
swap rp04 0 6000 2000
```

- *pipe*— Specifies where pipes are to be allocated (must be a *mounted* file system— the *root* file system is normally used).
- *dump*— Specifies the device to be used to dump memory after a system crash. Currently only the TU10 and TU16 tape drives are supported for this purpose.
- *buffers*— Specifies how many *system buffers* to allocate. In general, you will want as many buffers as possible without exceeding the system data-space limitations. Normally, *buffers* is in the range of 24-50. Each entry requires 512 bytes *outside* the system address space, and 26 bytes *inside* the system.
- *sabufs*— Specifies how many *system addressable* buffers to allocate. One buffer is needed for every mounted file system. Certain I/O drivers need such buffers. Normally, *sabufs* is in the range 10-15; the default value is 8. Each entry requires 540 bytes.
- *inodes*— Specifies how many *inode table* entries to allocate. Each entry represents a unique open inode. When the table overflows, the warning message "Inode table overflow" will be printed on the console. The table size should be increased if this happens regularly. The number of entries used depends on the number of active processes, texts, and mounts. Normally, *inodes* is in the range of 100-150. Each entry requires 74 bytes.
- *files*— Specifies how many *open-file table* entries to allocate. Each entry represents an open file. When the table overflows, the warning message "no file" will be printed on the console. The table size should be increased if this happens regularly. Normally, *files* is in the same range as the number of inodes. Each entry requires 8 bytes.
- *mounts*— Specifies how many *mount table* entries to allocate. Each entry represents a mounted file system. The *root* (/) will always be the first entry. When full, the *mount*(2) syscall will return the error EBUSY. Normally, *mounts* is in the range of 8-16. Each entry requires 10 bytes.
- *maxproc*— Specifies the maximum number of active processes a non-super-user may spawn. The default value is 25.
- *power*— Specifies whether to attempt restart after a power failure. A value of 0 indicates no restart, while a value of 1 attempts power-fail restart. On restart, device drivers are called, and process 1 (*init*) is sent a hangup signal; see *init*(8). The default value is 0.

- *coremap*— Specifies how many entries to allocate to the *list of free memory*. Each entry represents a contiguous group of 64-byte blocks of free memory. When the list overflows, due to excessive fragmentation, the system will undoubtedly crash in an unpredictable manner. The number of entries used depends on the number of processes active, their sizes, and the amount of memory available. Normally, *coremap* is in the range of 50-100. Each entry requires 4 bytes.
- *swapmap*— Specifies how many entries to allocate to the *list of free swap blocks*. Exactly like the coremap, except it represents free blocks in the swap area, in 512-byte units. Each entry requires 4 bytes.
- *calls*— Specifies how many *callout table* entries to allocate. Each entry represents a function to be invoked at a later time by the clock handler. The time unit is 1/60 of a second. The callout table is used by the terminal handlers to provide terminal delays and by various other I/O handlers. When the table overflows, the system will crash and print the panic message "Timeout table overflow" on the console. Normally, *calls* is in the range of 30-60. Each entry requires 6 bytes.
- *procs*— Specifies how many *process table* entries to allocate. Each entry represents an active process. The scheduler is always the first entry and *init*(8) is always the second entry. The number of entries depends on the number of terminal lines available and the number of processes spawned by each user. The average number of processes per user is in the range of 2-5. When full, the *fork*(2) syscall will return the error EAGAIN. Normally, *procs* is in the range of 50-200. Each entry requires 28 bytes.
- *texts*— Specifies how many *text table* entries to allocate. Each entry represents an active read-only text segment. Such programs are created by using the −i or −n option of the loader *ld*(1). When the table overflows, the warning message "out of text" is printed on the console. Normally, *texts* is in the range of 25-50. Each entry requires 12 bytes.
- *clists*— Specifies how many *character list buffers* to allocate. Each buffer contains up to 24 bytes. The buffers are dynamically linked together to form input and output queues for the terminal lines and various other slow-speed devices. The average number of buffers needed per terminal line is in the range of 5-10. When full, input characters from terminals will be lost and not echoed. Normally, *clists* is in the range of 100-300. Each entry requires 28 bytes.

3.2 PWB/UNIX Generation

Before generating your *first* PWB/UNIX system, you must modify the file called *.Makefile* in the */usr/src/uts/cf* directory. This file contains four symbols that are used for system identification; they initialize the internal *utsname* structure (see *uname*(1), *uname*(2), and *utsname*(5)). The four symbols (which are 8 characters maximum) are as follows:

SYS	your system name (e.g., *pwba*);
NODE	the name by which your system is known on the *uucp*(1) network (e.g., *pwba*);
REL	the operating system release (e.g., *PWB2.0*);
VER	The current version of the system; this is usually four characters indicating when the system was made (e.g., *0420* for April 20).

Only the first three symbols need to be modified locally. The VER symbol will be defined when you *make*(1) the system. The name of the executable file produced by the *make* procedure will be the concatenation of the SYS and VER symbols. Thus, if SYS is *pwba*, and you specify VER as *0420*, the executable file will be called *pwba0420*.

To generate a new PWB/UNIX operating system, follow this general procedure:

```
cd /usr/src/uts/cf
ed dfile
a
   [description file as described above]
.
w
q
config dfile
make VER= version
```

The system has a finite address space, so that if table sizes or the number of device types are too large, various error messages will result and the above procedure will only create an *a.out* file. In particular, the maximum available data space is 49,152 bytes. The actual data space requested can be found by using *size*(1) on *a.out* and adding the *data* and *bss* segment sizes. One then reduces the specified values for the various system entries (normally, the number of buffers) until it all fits. The amount of space in the *bss* segment used for each entry is indicated in Section 3.1 above.

When you are satisfied with the new system, you can test it by the following procedure:

```
cd /usr/src/uts
cp name /
cd /
rm /unix
ln /name /unix
sync
```

Note that *name* is the name of the executable file produced by the *make* procedure. Halt the processor and reboot *unix*. Note that this procedure results in two names for the operating system object, the generic */unix*, and the actual name, say */pwba0420*. An old system may be booted by referring to the actual name, but remember that many programs use the generic name */unix* to obtain the *name-list* of the system.

If the new system does not work, verify that the correct device addresses and interrupt vectors have been specified. If the *wrong* interrupt vector and the *correct* device address have been specified for a device, the operating system will print the error message "stray interrupt at XXX" when the device is accessed, where XXX is the correct interrupt vector. If the *wrong* device address is specified, the system will crash with a panic trap of type 0 (indicating a UNIBUS[5] timeout) when the device is accessed.

3.3 Special Files

A special file must be made for every device on your system. Normally, all special files are located in the directory */dev*. Initially, this directory will contain:

console	console terminal
error	see *err*(4)
mem, kmem, null	see *mem*(4)
tty	see *tty*(4)
rp[0–7], rrp[0–7]	disk drive 0, sections 0–7
mt0, rmt0	tape drive 0 (800 bpi)
mt4, rmt4	tape drive 0 (800 bpi, no rewind).

There are two types of special files, block and character. This is indicated by the character *b* or *c* in the listing produced by *ls*(1) with the −l flag.

In addition, each special file has a major device number and a minor device number. The major device number refers to the device type and is used as an index into either the *bdevsw* or *cdevsw* table in the configuration file *conf.c*. The minor device number refers to a particular unit of the device type and is used only by the driver for that type.

For example, using the following sample portion of a configuration file, a block special file with major device number 1 and minor device number 0 would refer to the TU10 magtape, drive 0, while a character special file with major device number 1 and minor device number 4 would refer to the DH11 asynchronous multiplexor, line 4.

```
int     (*bdevsw[])()
{
/* 0*/  &hpopen,  &hpclose,  &hpstrategy,  &hptab,
/* 1*/  &tmopen,  &tmclose,  &tmstrategy,  &tmtab,
};

int     (*cdevsw[])()
{
/* 0*/  &klopen,  &klclose,  &klread,  &klwrite,  &klsgtty,
/* 1*/  &dhopen,  &dhclose,  &dhread,  &dhwrite,  &dhsgtty,
/* 2*/  &nulldev, &nulldev,  &mmread,  &mmwrite,  &nodev,
/* 3*/  &nodev,   &nodev,    &nodev,   &nodev,    &nodev,
/* 4*/  &nodev,   &nodev,    &nodev,   &nodev,    &nodev,
/* 5*/  &nodev,   &nodev,    &nodev,   &nodev,    &nodev,
/* 6*/  &tmopen,  &tmclose,  &tmread,  &tmwrite,  &nodev,
/* 7*/  &hpopen,  &hpclose,  &hpread,  &hpwrite,  &nodev,
/* 8*/  &nodev,   &nodev,    &nodev,   &nodev,    &nodev,
/* 9*/  &nodev,   &nodev,    &nodev,   &nodev,    &nodev,
/*10*/  &nodev,   &nodev,    &nodev,   &nodev,    &nodev,
/*11*/  &nodev,   &nodev,    &nodev,   &nodev,    &nodev,
/*12*/  &nodev,   &nodev,    &nodev,   &nodev,    &nodev,
/*13*/  &syopen,  &nulldev,  &syread,  &sywrite,  &sysgtty,
};
```

The program *mknod*(1M) creates special files. For example, the following would create *part* of the initially-supplied */dev* directory:

```
cd /dev
mknod console c 0 0
mknod error c 20 0
mknod mem c 2 0; mknod kmem c 2 1; mknod null c 2 2
mknod tty c 13 0
mknod rp0 b 0 0; mknod rrp0 c 7 0
mknod mt0 b 1 0; mknod rmt0 c 6 0
mknod mt4 b 1 4; mknod rmt4 c 6 4
```

After the special files have been made, their access modes should be changed to appropriate values by *chmod*(1). For example:

```
cd /dev
chmod 622 console
chmod 444 error
chmod 440 mem kmem
chmod 666 null
chmod 666 tty
chmod 400 rp0 rrp0
chmod 666 mt0 rmt0
chmod 666 mt4 rmt4
```

Assuming the disk drives are major device number 0 (block-type) and major device number 7 (character-type), the following commands can be used to make the special files for disk drives 1-7, each containing sections 0-7:

```
cd /dev
for a in 1 2 3 4 5 6 7
do
   for b in 0 1 2 3 4 5 6 7
   do
      mknod rp$a$b b 0 0$a$b
      mknod rrp$a$b c 7 0$a$b
   done
done
chmod 400 *rp*
```

Note that for disks, an octal number scheme is maintained because each drive is split eight ways. Thus, */dev/rp24* refers to section 4 of physical drive 2.

Minor device numbers for tape are a bit peculiar. The minor device number consists of the following four bits:

0= 800-bpi 1= 1600-bpi	0= rewind 1= no rewind	drive	select

Therefore, the special file for tape drive 1, operating at 1600-bpi, with no rewind would be made as follows:

```
mknod /dev/mt?? b 1 13
mknod /dev/rmt?? c 6 13
```

File names have no meaning to the *operating system* itself; only the major and minor device numbers are important. However, many *programs* expect that a particular file is a certain device. Thus, by convention, special files are named as follows:

block device	*conf.c*	*/dev*
RP03 disk	rp	rp*
RP04/5/6 disk	hp	rp*
RS03/4 fixed head disk	hs	rs*
TU10 tape	tm	mt*
TU16 tape	ht	mt*
character device	*conf.c*	*/dev*
DL11 asynch line	kl	tty*
DH11 asynch line mux	dh	tty*
DZ11 asynch line mux	dz	tty*
DN11 auto call unit	dn	dn*
DU11 synch line	du	du*
DQS11B synch line	dqs	rjei
KMC11 micro	kmc	kmc*
DZ11/KMC11 assist	dzk	tty*
LP11 line printer	lp	lp*
RP03 disk	rp	rrp*
RP04/5/6 disk	hp	rrp*
RS03/4 fixed head disk	hs	rrs*
TU10 tape	tm	rmt*
TU16 tape	ht	rmt*
error	err	error
memory	mm	mem, kmem, null
terminal	sy	tty

For those devices with a */dev* name ending in "*", this character is replaced by a string of digits representing the *minor* device number. For example:

```
mknod /dev/mt1 b 1 1
mknod /dev/rp24 b 0 024
mknod /dev/tty03 c 1 3
```

There is a special file, */dev/swap*, that is used by the program *ps*(1). This file must reflect what device is used for swapping and must be readable. For example:

```
rm /dev/swap
mknod /dev/swap b 0 0
chmod 440 /dev/swap
chown sys /dev/swap; chgrp sys /dev/swap
```

3.4 File Systems

Each physical pack is split into eight logical sections. This split is defined in the operating system by a table with eight entries. Each table entry is two words long. The first specifies how many blocks are in the section, the second specifies the starting cylinder; see *hp*(4) (RP04/5/6) and *rp*(4) (RP03) for default cylinder and block assignments.

These values are described to the system in the header file */usr/include/sys/io.h* and may be changed by using the editor *ed*(1). After such a change, the system must be made again (see Section 3.2).

A file system starts at block 0 of a section of the disk and may be as large as the size of that section; if it is smaller than the size of a section, the remainder of that section is unused. Note that the sections themselves may overlap physical areas of the pack, but the file systems must *never* overlap.

The program *mkfs*(1M) is used to initialize a section of the disk to be a file system. Next, the program *labelit*(1M) is used to label the file system with its name and the name of the pack. Finally, the file system may be checked for consistency by using *fsck*(1M). The file system may then be mounted using *mount*(1M).

4. ADMINISTRATIVE FILES

4.1 /etc/motd

This file contains the *message-of-the-day*. It is printed by *login*(1) after every successful *login*.

4.2 /etc/rc

On the transition between *init* states, */etc/init* invokes */bin/sh* to run */etc/rc* (must have executable modes). In order for */etc/rc* to properly handle the removal of temporary files and the mounting and unmounting of file systems, it is invoked with three arguments: the new state, the number of times this state has been entered, and the previous state. When the system is initially booted, */etc/rc* is invoked with arguments "1 0 0"; when state two (multi-user) is subsequently entered (by the operator typing in **/etc/init 2**), the arguments are "2 0 1".

When state 2 is entered more than one time (via another **/etc/init 2**), no action will be taken by this file. This enables the system administrator to add or delete *login* processes while the system is in multi-user mode (see */etc/inittab* below).

Daemons may be invoked either by */etc/rc* or by including lines for them in */etc/inittab*.

The */etc/rc* file is also used to initialize KMC11 microprocessors (see */etc/dz-kload* below).

This file must be edited to suit local conditions; see *init*(8).

4.3 /etc/inittab

This file indicates to */etc/init* which processes to create in each *init* state. By convention, state 1 is single-user mode, and state 2 is multi-user mode. For example, the following line creates a sample single-user environment:

```
1:co:c:/bin/sh  </dev/console  >/dev/console  2>/dev/console
```

This indicates that for state 1, a process with the arbitrary unique identifier *co* should be created. The program invoked for this process should be the Shell, and when it exits, it should be reinvoked (*c* flag).

To attach a *login* process to the console when in the multi-user state, add the following line:

```
2:co:c:/etc/getty  console 4
```

and for line */dev/tty00* for use by 300/150/110/1200 baud terminals, add the following line:

```
2:00:c:/etc/getty  tty00  0  3600
```

The arguments to *getty*(8) are the device, speed table entry, and the number of seconds to allow before hanging up the line.

This file must be edited to suit local conditions; see *getty*(8), *init*(8), and *inittab*(5).

4.4 /etc/dzkload

This file can be invoked as a command by */etc/rc*. It contains instructions for initializing each KMC11 microprocessor that is to function as a controller for one or more DZ11 communications multiplexors (see *dh*(4)). This file must be edited to suit the configuration.

4.5 /etc/passwd

This file is used to describe each *user* to the system. You must add a new line for each new user. Each line has seven fields separated by colons:

1. Login name:
   Normally 1-6 characters, first character alphabetic, rest alphanumeric, no upper-case characters.
2. Encrypted password:
   Initially null, filled in by *passwd*(1). The encrypted password contains 13 bytes, while the actual password is limited to a maximum of 8 bytes. The encrypted password may be followed by a comma and up to 4 more bytes of "age" information.
3. User-id:
   A number between 0 and 65,535; 0 indicates the super-user. User-ids 0 through 99 are reserved (most of them for future use). Presently, these user-ids are reserved:

daemon::1:	daemons
bin::2:	software administration;
sys::3:	system operation;
adm::4:	system administration;
uucp::5:	UNIX-to-UNIX file copy;
rje::68:	remote job entry administration;
games::19:	miscellaneous; never a real user.

4. Group-id:
   A number between 0 and 65,535; Group-ids 0 through 99 are also reserved. The default group is 1 (*other*).
5. Accounting information:
   This field is used by various accounting programs. It usually contains the user name, department number, and account number.

6. Login directory:
   Full pathname (keep them reasonably short).

7. Program name:
   If null, */bin/sh* is invoked after successful *login*. If present, the named program is invoked in place of */bin/sh*.

For example:

```
ghh::138:11:6824-G.H.Hurtz(4357):/usr/ghh:
grk::266:13:6510-S.P.LeName(4466):/usr/grk:/bin/rsh
```

See also *passwd*(5), *login*(1), *passwd*(1).

4.6 /etc/group

This file is used to describe each *group* to the system. Each line has four fields separated by colons:

group name:
encrypted password:
numerical group-id:
list of all *login* names in the group, separated by commas.

See also *group*(5).

4.7 /etc/profile

When the Shell is executed and is the leader of a process group, as is the case when it is invoked by *login*, it will read and execute the commands in */etc/profile* before executing the commands in the user's *.profile* file. This allows the system administrator to set up a standard environment for all users (e.g., executing *umask*(1), setting Shell variables) and take care of any other housekeeping details (such as *news* −*n*).

4.8 /etc/checklist

This file contains a list of default devices to be checked for consistency by the */sck*(1M) program. The devices normally correspond to those mounted when the system is in *multi-user* mode. For example, a sample checklist would be:

```
/dev/rp0
/dev/rrp1
```

Note that the *root* device is specified as a *block* device, while all others are specified as *character* devices. Character devices can be checked faster than block devices. The *root* device is specified as a block device in order for the */sck* program to detect when the *root* is being checked, so that if this file system is modified, an immediate reboot will be requested.

4.9 /etc/shutdown

This file contains procedures to gracefully shut down the system in preparation for filesave or scheduled downtime.

4.10 /etc/filesave.?

This file contains the detailed procedures for the local filesave.

4.11 /usr/adm/pacct

This file contains the process accounting information; see *acct*(1M).

4.12 /usr/adm/wtmp

This file is the log of all *login* processes.

5. REGENERATING SYSTEM SOFTWARE

System source is found under the */usr/src* directory. The sub-directories are named *cmd* (commands), *lib* (libraries), *uts* (the operating system), *head* (header files), and *util* (utilities); see *mk*(8) for details on how to remake system software.

The accounting routine that deals with holidays and the prime/non-prime time split must be recompiled at the end of the year (it is currently correct for BTL-Murray Hill in 1979). The file is */usr/src/cmd/acct/lib/pnpsplit.c*. A reminder is sent to */usr/adm/acct/nite/log*, the standard place for such messages, starting a week before year-end and continuing until *pnpsplit.c* has been recompiled.

6. UNIX FILE SYSTEM CONVERSION

Procedures have been developed for converting UNIX file systems from the PWB/UNIX Edition 1.2 format to the PWB/UNIX Release 2.0 format.

6.1 Preliminaries

The new system should be generated and decently testing before file system conversion is performed. Do not convert without spare packs– that is courting disaster. It is best to keep the old packs for several days, and to make backup tapes as well.

We strongly advise that a "root pack" of your Edition 1.2 system (*/unix*, */etc*, */bin*, and the *official* part of */usr*) be saved for a long time. Initially, it will be the only way to access Edition 1.2 backup tapes.

The old file systems should be pruned of marginal files. Old object files will not work under Release 2.0, and should be removed (e.g., files named *a.out*, *core*, **.o*), along with your *local* commands stored in */bin*, */etc*, */lib*, */usr/bin*, */usr/lib*. Users should be encouraged to clean house.

6.2 Copying from the Old System

The following steps should be executed by the super-user on an idle, stand-alone (old) system:

```
cd /file-system-name
find . -print | cpio -oB >/dev/rmt0
# For 1,600-bpi, use appropriate device
```

Unless there are a great many linked files, a 1,600-bpi, 2,400-foot reel should hold the maximum Edition 1.2 file system (65K blocks). An 800-bpi tape will hold approximately 40K blocks. *Find* can also be used to pick up *parts* of file systems that can be combined later as described below.

6.3 Copying to the New System

Re-create each file system as follows:

```
mkdir /file-system-name
mkfs /dev/rrp?? blocks:inodes 7 418
# For RP03 disks, the last argument above should be 200
labelit /dev/rrp?? file-system-name pkname
mount /dev/rp?? /file-system-name
cd /file-system-name
# Mount the tape created on the old system
cpio -idmB6 </dev/rmt0
# For 1,600-bpi, use appropriate device
# If you are combining the smaller Edition 1.2 file systems,
# you may copy in more than one tape per new file system
```

After the tapes have been copied in, the new file system should be unmounted and checked (using *fsck*(1M)). It would then be wise to make a tape or disk backup of the new file system.

January 1980