Eighth Edition Installation Instructions

You received either one or two tapes. If one, it is at 6250 bpi and contains everything. If two, it is 1600 bpi and is divided between Blit (68000 prototype DMD terminal) and Jerq (WE 32000-based DMD terminal) support on the one tape, and everything else on the other. All tapes are *tar*(1) format with block factor 20, and should be extracted like

```
tar xbf 20 /dev/raw-tape-filename
```

into a convenient place. The total is about 52 megabytes; a Blit-Jerq tape will be about 18 MB and an ‘‘everything else’’ tape will be about 52–18 MB.

Almost all files on the tape are owned by root (uid 0) or bin (uid 3) or uucp (uid 48); others are probably oversights. Almost everything is group bin (4). Group sys (2) is used for things that access /dev/mem or the disks, for setgid purposes. You will have to decide whether it is simpler to extract the tape as root (and preserve the numbers) or as someone else, and lose ownership information.

How to bootstrap

The distribution is not set up for bootstrapping to an empty machine. You will have to have a running system, extract the tape, build your own kernel, and then convert to it. Two file system formats are supported by the Eighth Edition; one uses 1KB blocks, the other 4KB blocks. The 1K file system is almost identical to that used by the Berkeley 4.1 (not 4.2!) system; it differs only in the superblock, and the differences can be repaired by use of *fsck*(8). Likewise, it is believed to be similar in structure to the System V file system with 1K blocks, but this has not been tested.

The simplest way to boot the system requires a reasonably empty disk.

1. On a running 4.1BSD system, extract the tape into an empty disk partition. Call it /v8. In /v8/usr/bin, there is a command *v8* that changes root to /v8 and makes you user *bin.* Now you should be able to compile things in the Eighth Edition software environment, and in particular to configure and compile a kernel (/unix). See the configuration and disk-partition sections below. Aside from /unix, this /v8 partition is already a complete and runnable file system.

2. Copy /v8/bin, /v8/lib, /v8/etc, and /v8/boot to an empty 0-origin (root) partition on some disk. Also, copy an appropriate boot block to block 0 on this disk. Boot blocks are found in /v8/usr/sys/boot/bb; see below, and READMEs in neighboring directories. Finally, make a /dev directory with at least console and null in it. /v8/proto-dev is a list of things we have on one of our machines, as a guide.

3A. You should now have a disk that can be booted. When you made your kernel, you might have specified that rootdev and so forth were on some drive other than 0, but it is probably simpler to switch drive plugs or packs to make the v8 disk drive 0. Try to boot it single-user. When that works (you should write-protect the disk for the first few efforts), you will get the familiar # prompt, and basic commands like *ls* will operate correctly. Do not create any files; see step 4.

3B. An alternate strategy is to copy your v8 unix to your own /nunix, and see if it works on your own file system. Make sure it is backed up first.

4. Before creating any files, you should do */etc/fsck /dev/x*, where *x* is your root device. This will convert your new root into an Eighth Edition file system (mostly, create a new free list). Once you have done this, if you want to back out, you must run your own fsck in order to convert it back.

5. Now try the usual things, like making proper entries for stuff in /dev, and compiling "hello world." The latter makes you realize that you have to create /tmp. When you have enough courage, run */etc/fsck* on the file system that was /v8 before. (You should have made a /dev entry for it). One

potential problem here is incompatibility with our disk partitions and yours; see below. Once this has been done, you can come up multi-user.

Configuration

The system configuration mechanisms are mostly taken from 4.1BSD, with a few changes to */etc/config*; the most important is the ability to write a number after a pseudo-device that specifies how many instances are wanted. Other differences of immediate use are that config assumes the standard argument *conf*, runs in the system directory, and does 'make depend' for you. Manual pages *config*(5) and *config*(8) exist in /usr/man, even though they are not in the printed documentation.

Several prototype conf files are supplied; one (alice) is for a large 11/780 with two UBAs, RM03, UDA50/RA81, and dual-ported TU78. Another (forbes) is for a small 11/750 with Fujitsu 160MB disks on an Emulex controller. A third (research) is also an 11/750 and besides Datakit, has two Interlan Ethernet interfaces running TCP/UDP/IP.

To build a new system, generate the files /usr/sys/*sysname*/conf (containing the basic configuration information, and corresponding to BSD's /usr/sys/conf/*SYSNAME*.) Also write, by interpolation or extrapolation, the file /usr/sys/*sysname*/sparam.h, which specifies parameters for the stream IO system. It would indeed be better to have the numbers be generated as a function of *maxusers*. Here is the meaning of the parameters, and a guide to estimating them.

NQUEUE: Two used per active line discipline module, four used per active stream device or pipe. This means six for each terminal connection, 12 for each Blit window.

NSTREAM: One per stream ending at a process, that is one for each active device, two for pipes.

NBLK*: Number of stream data storage blocks of various sizes; BIG is 1KB. They correspond to clist blocks in some systems, mbufs in another. When the other resources run out, the console chatters and the system goes on; when you run out of these, it panics. Things are in fact written so that running out of stream blocks need not be fatal; the panic is there as a debugging tool, because one of the common stream bugs is to lose blocks or treasure them up somewhere. If you get this, look at the output of */etc/showq*. If most of the blocks are missing or on some particular queue, you have a bug, and increasing the numbers will not help. If the blocks are all nicely distributed and well-accounted for, increase the numbers.

Besides configuring devices, you should configure line disciplines; see the samples. In particular, allow one *mesg* for every DMD window, one *tty* for every terminal line and enough spares for DMD windows that have had programs downloaded to them, some *ntty*s if you want to use the C shell. Include the *sp* pseudo-device to make the /dev/pt files work; there are always 256 of them.

Once *conf* and *sparam.h* exist, change directories to /usr/sys/*sysname*, run */etc/config*, and *make*.

Hardware and booting

On the 750, the system boots through the bootstrap ROM and the boot block. The boot block reads in /boot from the filesystem at the beginning of the boot device. /boot reads in /unix.

Boot blocks live in /usr/sys/boot/bb. upboot is for UNIBUS Fujis; 4kboot and 1kboot are for 4K and 1K UDA50/RA disks. The last two aren't really specific to RAs, but use the driver routine provided by the ROM; unfortunately, /boot doesn't.

On the 780 and 785, the boot block isn't used; the console BOOT command executes a command file which loads boot from the floppy. UNIBUS Fujis, Massbus RP-like disks (such as the RM03 and RP07), and UNIBUS UDA50 disks are thought to work.

To set up a 750, copy the appropriate boot block to the first sector of the root filesystem. Make sure you have a useful boot ROM. To set up a 780, cd to /usr/sys/boot/floppy and use *arff*(8) to add the files therein to your console floppy. You might want to look at them first, especially if you have interleaved memory or a non-RP boot device.

The boot program calls UDA disks *ra* when they contain 4K filesystems, *sa* when 1K.

If you don't have one of the devices named above, you'll have to fix things before you can boot. In particular, Fuji Eagles will probably require some fiddling.

Disk partitions

There is almost certainly going to be trouble here: our disk partitions are probably not the same as yours. Check the _sizes arrays in the disk drivers for compatibility. If there is a difference between the distributed numbers and the ones you are accustomed to, you have two choices: change the numbers, or accept them. If you accept them, then the place you extracted the full Eighth Edition tape is in the wrong spot on the disk; a new partition must be initialized for it (using mkfs or mkbitfs(8)), and the tape must be reread.

4K file systems

Each file system may use either 1K blocks or 4K blocks (but not both); see filsys(5). One crucial missing fact is that the kernel recognizes a 4K file system by virtue of the 0100 bit in its minor device number. 4K file systems are indeed about four times faster than 1K file systems, but have larger breakage costs for small files. Also, observe this fine print in filsys(5): 4K file systems must be dismounted before the machine is rebooted, or they must be checked when it comes up, otherwise they cannot be mounted.

Networking

The user-level networking code for Internet (really ethernet) and Datakit is not installed in the distribution. The source is in /usr/src/inet and /usr/src/dk respectively. One peculiarity might be noted with inet: on a whim, because it was done that way in Australia, the name *rlogin* was changed to *rogin.* For a more serious reason, the name *rcp* was changed to *ropy* (rhymes with *copy*). We already had a command *rcp,* which does recursive copies. We will probably resolve this eventually by accepting the 4.2 *rcp* and incorporating our *rcp* into a −R option of *cp.*

Miscellany

The version of the manual on the tape is newer than the printed one. The file dates may be of interest in deciding what is new. Section 2 in particular has changed a lot.

If you have a newer VAX-11/750 with the patchable control store, you should arrange for the microcode to be loaded by calling */etc/ldpcs /etc/pcs750.bin* from /etc/rc. See *ldpcs*(8).

Rarepl(8) will replace bad blocks on RA-style disks. The block to be replaced is the lbn named in the error log packet, not the alleged sector number named by Unix. If you don't know what an MSCP event code is, you shouldn't mess around with this.

Credits

The collection and arrangement of the things in this distribution is the work of Dennis Ritchie, who swore he would never do this again but did, just not as well; Rob Pike, who did all the Jerq/Blit work; Doug McIlroy, who produced and directed the manual with less help from the rest of us than he should have had; and Norman Wilson, who helped enormously, especially with the messy bits.