Installing and Using FreeBSD With Other Operating Systems

Jay Richmond

jayrich@sysc.com

6 August 1996

This document discusses how to make FreeBSD coexist nicely with other popular operating systems such as Linux, MS-DOS®, OS/2®, and Windows® 95. Special thanks to: Annelise Anderson <andrsn@stanford.edu>, Randall Hopper <rhh@ct.picker.com>, and Jordan K. Hubbard <jkh@FreeBSD.org>.

Table of Contents

1 Overview

Most people can not fit these operating systems together comfortably without having a larger hard disk, so special information on large EIDE drives is included. Because there are so many combinations of possible operating systems and hard disk configurations, the Section 5 section may be of the most use to you. It contains descriptions of specific working computer setups that use multiple operating systems.

This document assumes that you have already made room on your hard disk for an additional operating system. Any time you repartition your hard drive, you run the risk of destroying the data on the original partitions. However, if your hard drive is completely occupied by DOS, you might find the FIPS utility (included on the FreeBSD CDROM in the `\TOOLS` directory or via ftp (ftp://ftp.FreeBSD.org/pub/FreeBSD/tools/)) useful. It lets you repartition your hard disk without destroying the data already on it. There is also a commercial program available called **PartitionMagic®**, which lets you size and delete partitions without consequence.

2 Overview of Boot Managers

These are just brief descriptions of some of the different boot managers you may encounter. Depending on your computer setup, you may find it useful to use more than one of them on the same system.

Boot Easy

> This is the default boot manager used with FreeBSD. It has the ability to boot most anything, including BSD, OS/2 (HPFS), Windows 95 (FAT and FAT32), and Linux. Partitions are selected with the function keys.

OS/2 Boot Manager

> This will boot FAT, FAT32, HPFS, FFS (FreeBSD), and EXT2 (Linux). Partitions are selected using arrow keys. The OS/2 Boot Manager is the only one to use its own separate partition, unlike the others which use the master boot record (MBR). Therefore, it must be installed below the 1024th cylinder to avoid booting problems. It can boot Linux using LILO when it is part of the boot sector, not the MBR. Go to Linux HOWTOs (http://www.linuxresources.com/LDP/HOWTO/HOWTO-INDEX.html) on the World Wide Web for more information on booting Linux with the OS/2 boot manager.

OS-BS

> This is an alternative to Boot Easy. It gives you more control over the booting process, with the ability to set the default partition to boot and the booting timeout. The beta version of this programs allows you to boot by selecting the OS with your arrow keys. It is included on the FreeBSD CD in the `\TOOLS` directory, and via ftp (ftp://ftp.FreeBSD.org/pub/FreeBSD/tools/).

LILO, or LInux LOader

> This is a limited boot manager. It will boot FreeBSD, though some customization work is required in the LILO configuration file.

> **About FAT32:** FAT32 is the replacement to the FAT filesystem included in Microsoft's OEM SR2 Beta release, which started replacing FAT on computers pre-loaded with Windows 95 towards the end of 1996. It converts the normal FAT filesystem and allows you to use smaller cluster sizes for larger hard drives. FAT32 also modifies the traditional FAT boot sector and allocation table, making it incompatible with some boot managers.

3 A Typical Installation

Let's say I have two large EIDE hard drives, and I want to install FreeBSD, Linux, and Windows 95 on them.

Here is how I might do it using these hard disks:

- `/dev/wd0` (first physical hard disk)
- `/dev/wd1` (second hard disk)

Both disks have 1416 cylinders.

1. I boot from a MS-DOS or Windows 95 boot disk that contains the `FDISK.EXE` utility and make a small 50 MB primary partition (35-40 for Windows 95, plus a little breathing room) on the first disk. Also create a larger partition on the second hard disk for my Windows applications and data.
2. I reboot and install Windows 95 (easier said than done) on the `C:` partition.
3. The next thing I do is install Linux. I am not sure about all the distributions of Linux, but Slackware (http://www.slackware.com) includes LILO (see Section 2). When I am partitioning out my hard disk with Linux `fdisk`, I would put all of Linux on the first drive (maybe 300 MB for a nice root partition and some swap space).
4. After I install Linux, and are prompted about installing LILO, make *sure* that I install it on the boot sector of my root Linux partition, not in the MBR (master boot record).
5. The remaining hard disk space can go to FreeBSD. I also make sure that my FreeBSD root slice does not go beyond the 1024th cylinder. (The 1024th cylinder is 528 MB into the disk with our hypothetical 720 MB disks). I will use the rest of the hard drive (about 270 MB) for the `/usr` and `/` slices if I wish. The rest of the second hard disk (size depends on the amount of my Windows application/data partition that I created in step 1) can go to the `/usr/src` slice and swap space.
6. When viewed with the Windows 95 `fdisk` utility, my hard drives should now look something like this:

```
-----------------------------------------------------------------------

                  Display Partition Information

Current fixed disk drive: 1

Partition  Status  Type  Volume_Label  Mbytes  System  Usage
C: 1          A   PRI DOS                50    FAT**     7%
   2          A   Non-DOS (Linux)       300             43%

Total disk space is  696 Mbytes (1 Mbyte = 1048576 bytes)

Press Esc to continue

-----------------------------------------------------------------------

                  Display Partition Information

Current fixed disk drive: 2

Partition  Status  Type  Volume_Label  Mbytes  System  Usage
D: 1          A   PRI DOS               420   FAT**    60%
```

```
Total disk space is  696 Mbytes (1 Mbyte = 1048576 bytes)

Press Esc to continue

-----------------------------------------------------------------
```

** May say FAT16 or FAT32 if you are using the OEM SR2 update. See Section 2.

7. Install FreeBSD. I make sure to boot with my first hard disk set at "NORMAL" in the BIOS. If it is not, I will have the enter my true disk geometry at boot time (to get this, boot Windows 95 and consult Microsoft Diagnostics (`MSD.EXE`), or check your BIOS) with the parameter `hd0=1416,16,63` where *`1416`* is the number of cylinders on my hard disk, *`16`* is the number of heads per track, and *`63`* is the number of sectors per track on the drive.
8. When partitioning out the hard disk, I make sure to install Boot Easy on the first disk. I do not worry about the second disk, nothing is booting off of it.
9. When I reboot, Boot Easy should recognize my three bootable partitions as DOS (Windows 95), Linux, and BSD (FreeBSD).

4 Special Considerations

Most operating systems are very picky about where and how they are placed on the hard disk. Windows 95 and DOS need to be on the first primary partition on the first hard disk. OS/2 is the exception. It can be installed on the first or second disk in a primary or extended partition. If you are not sure, keep the beginning of the bootable partitions below the 1024th cylinder.

If you install Windows 95 on an existing BSD system, it will "destroy" the MBR, and you will have to reinstall your previous boot manager. Boot Easy can be reinstalled by using the `BOOTINST.EXE` utility included in the `\TOOLS` directory on the CDROM, and via ftp (ftp://ftp.FreeBSD.org/pub/FreeBSD/tools/). You can also re-start the installation process and go to the partition editor. From there, mark the FreeBSD partition as bootable, select Boot Manager, and then type W to (W)rite out the information to the MBR. You can now reboot, and Boot Easy should then recognize Windows 95 as DOS.

Please keep in mind that OS/2 can read FAT and HPFS partitions, but not FFS (FreeBSD) or EXT2 (Linux) partitions. Likewise, Windows 95 can only read and write to FAT and FAT32 (see Section 2) partitions. FreeBSD can read most filesystems, but currently cannot read HPFS partitions. Linux can read HPFS partitions, but can not write to them. Recent versions of the Linux kernel (2.x) can read and write to Windows 95 VFAT partitions (VFAT is what gives Windows 95 long file names - it is pretty much the same as FAT). Linux can read and write to most filesystems. Got that? I hope so.

5 Examples

(section needs work, please send your example to <`jayrich@sysc.com`>).

FreeBSD + Windows 95: If you installed FreeBSD after Windows 95, you should see `DOS` on the Boot Easy menu. This is Windows 95. If you installed Windows 95 after FreeBSD, read Section 4 above. As long as your hard disk does not have 1024 cylinders you should not have a problem booting. If one of your partitions goes beyond the 1024th cylinder however, and you get messages like "`invalid system disk`" under DOS (Windows 95) and FreeBSD will not boot, try looking for a setting in your BIOS called "> 1024 cylinder support" or

"NORMAL/LBA" mode. DOS may need LBA (Logical Block Addressing) in order to boot correctly. If the idea of switching BIOS settings every time you boot up does not appeal to you, you can boot FreeBSD through DOS via the `FBSDBOOT.EXE` utility on the CD (It should find your FreeBSD partition and boot it.)

FreeBSD + OS/2 + Windows 95: Nothing new here. The OS/2 boot manager can boot all of these operating systems, so that should not be a problem.

FreeBSD + Linux: You can also use Boot Easy to boot both operating systems.

FreeBSD + Linux + Windows 95: (see Section 3)

6 Other Sources of Help

There are many Linux HOW-TOs (http://www.linuxresources.com/LDP/HOWTO/HOWTO-INDEX.html) that deal with multiple operating systems on the same hard disk.

The Linux+DOS+Win95+OS2 mini-HOWTO (http://www.linuxresources.com/LDP/HOWTO/mini/Linux+DOS+Win95+OS2.html) offers help on configuring the OS/2 boot manager, and the Linux+FreeBSD mini-HOWTO (http://www.linuxresources.com/LDP/HOWTO/mini/Linux+FreeBSD.html) might be interesting as well. The Linux-HOWTO (http://www.in.net/~jkatz/win95/Linux-HOWTO.html) is also helpful.

The Windows NT® Loader Hacking Guide (http://www.tburke.net/info/ntldr/ntldr_hacking_guide.htm) provides good information on multibooting Windows NT, Windows 95, and DOS with other operating systems.

And Hale Landis's "How It Works" document pack contains some good info on all sorts of disk geometry and booting related topics. You can find it at ftp://fission.dt.wdc.com/pub/otherdocs/pc_systems/how_it_works/allhiw.zip.

Finally, do not overlook FreeBSD's kernel documentation on the booting procedure, available in the kernel source distribution (it unpacks to /usr/src/sys/i386/boot/biosboot/README.386BSD.

7 Technical Details

(Contributed by Randall Hopper, <`rhh@ct.picker.com`>)

This section attempts to give you enough basic information about your hard disks and the disk booting process so that you can troubleshoot most problems you might encounter when getting set up to boot several operating systems. It starts in pretty basic terms, so you may want to skim down in this section until it begins to look unfamiliar and then start reading.

7.1 Disk Primer

Three fundamental terms are used to describe the location of data on your hard disk: Cylinders, Heads, and Sectors. It is not particularly important to know what these terms relate to except to know that, together, they identify where data is physically on your disk.

Your disk has a particular number of cylinders, number of heads, and number of sectors per cylinder-head (a cylinder-head also known now as a track). Collectively this information defines the "physical disk geometry" for your hard disk. There are typically 512 bytes per sector, and 63 sectors per track, with the number of cylinders and heads varying widely from disk to disk. Thus you can figure the number of bytes of data that will fit on your own disk by calculating:

(# of cylinders) × (# heads) × (63 sectors/track) × (512 bytes/sect)

For example, on my 1.6 Gig Western Digital AC31600 EIDE hard disk, that is:

(3148 cyl) × (16 heads) × (63 sectors/track) × (512 bytes/sect)

which is 1,624,670,208 bytes, or around 1.6 Gig.

You can find out the physical disk geometry (number of cylinders, heads, and sectors/track counts) for your hard disks using ATAID or other programs off the net. Your hard disk probably came with this information as well. Be careful though: if you are using BIOS LBA (see Section 7.3), you can not use just any program to get the physical geometry. This is because many programs (e.g. `MSD.EXE` or FreeBSD fdisk) do not identify the physical disk geometry; they instead report the *translated geometry* (virtual numbers from using LBA). Stay tuned for what that means.

One other useful thing about these terms. Given 3 numbers—a cylinder number, a head number, and a sector-within-track number—you identify a specific absolute sector (a 512 byte block of data) on your disk. Cylinders and Heads are numbered up from 0, and Sectors are numbered up from 1.

For those that are interested in more technical details, information on disk geometry, boot sectors, BIOSes, etc. can be found all over the net. Query Lycos, Yahoo, etc. for `boot sector` or `master boot record`. Among the useful info you will find are Hale Landis's *How It Works* document pack. See the Section 6 section for a few pointers to this pack.

Ok, enough terminology. We are talking about booting here.

7.2 The Booting Process

On the first sector of your disk (Cyl 0, Head 0, Sector 1) lives the Master Boot Record (MBR). It contains a map of your disk. It identifies up to 4 *partitions*, each of which is a contiguous chunk of that disk. FreeBSD calls partitions *slices* to avoid confusion with its own partitions, but we will not do that here. Each partition can contain its own operating system.

Each partition entry in the MBR has a *Partition ID*, a *Start Cylinder/Head/Sector*, and an *End Cylinder/Head/Sector*. The Partition ID tells what type of partition it is (what OS) and the Start/End tells where it is. Table 1 lists a smattering of some common Partition IDs.

Table 1. Partition IDs

ID (hex)	Description
01	Primary DOS12 (12-bit FAT)
04	Primary DOS16 (16-bit FAT)
05	Extended DOS
06	Primary big DOS (> 32MB)
0A	OS/2
83	Linux (EXT2FS)

ID (hex)	Description
A5	FreeBSD, NetBSD, 386BSD (UFS)

Note that not all partitions are bootable (e.g. Extended DOS). Some are—some are not. What makes a partition bootable is the configuration of the *Partition Boot Sector* that exists at the beginning of each partition.

When you configure your favorite boot manager, it looks up the entries in the MBR partition tables of all your hard disks and lets you name the entries in that list. Then when you boot, the boot manager is invoked by special code in the Master Boot Sector of the first probed hard disk on your system. It looks at the MBR partition table entry corresponding to the partition choice you made, uses the Start Cylinder/Head/Sector information for that partition, loads up the Partition Boot Sector for that partition, and gives it control. That Boot Sector for the partition itself contains enough information to start loading the operating system on that partition.

One thing we just brushed past that is important to know. All of your hard disks have MBRs. However, the one that is important is the one on the disk that is first probed by the BIOS. If you have only IDE hard disks, it is the first IDE disk (e.g. primary disk on first controller). Similarly for SCSI only systems. If you have both IDE and SCSI hard disks though, the IDE disk is typically probed first by the BIOS, so the first IDE disk is the first probed disk. The boot manager you will install will be hooked into the MBR on this first probed hard disk that we have just described.

7.3 Booting Limitations and Warnings

Now the interesting stuff that you need to watch out for.

7.3.1 The dreaded 1024 cylinder limit and how BIOS LBA helps

The first part of the booting process is all done through the BIOS, (if that is a new term to you, the BIOS is a software chip on your system motherboard which provides startup code for your computer). As such, this first part of the process is subject to the limitations of the BIOS interface.

The BIOS interface used to read the hard disk during this period (INT 13H, Subfunction 2) allocates 10 bits to the Cylinder Number, 8 bits to the Head Number, and 6 bits to the Sector Number. This restricts users of this interface (i.e. boot managers hooked into your disk's MBR as well as OS loaders hooked into the Boot Sectors) to the following limits:

- 1024 cylinders, max
- 256 heads, max
- 64 sectors/track, max (actually 63, `0` is not available)

Now big hard disks have lots of cylinders but not a lot of heads, so invariably with big hard disks the number of cylinders is greater than 1024. Given this and the BIOS interface as is, you can not boot off just anywhere on your hard disk. The boot code (the boot manager and the OS loader hooked into all bootable partitions' Boot Sectors) has to reside below cylinder 1024. In fact, if your hard disk is typical and has 16 heads, this equates to:

1024 cyl/disk × 16 heads/disk × 63 sect/(cyl-head) × 512 bytes/sector

which is around the often-mentioned 528MB limit.

This is where BIOS LBA (Logical Block Addressing) comes in. BIOS LBA gives the user of the BIOS API calls access to physical cylinders above 1024 though the BIOS interfaces by redefining a cylinder. That is, it remaps your cylinders and heads, making it appear through the BIOS as though the disk has fewer cylinders and more heads than it actually does. In other words, it takes advantage of the fact that hard disks have relatively few heads and lots of cylinders by shifting the balance between number of cylinders and number of heads so that both numbers lie below the above-mentioned limits (1024 cylinders, 256 heads).

With BIOS LBA, the hard disk size limitation is virtually removed (well, pushed up to 8 Gigabytes anyway). If you have an LBA BIOS, you can put FreeBSD or any OS anywhere you want and not hit the 1024 cylinder limit.

To use my 1.6 Gig Western Digital as an example again, its physical geometry is:

(3148 cyl, 16 heads, 63 sectors/track, 512 bytes/sector)

However, my BIOS LBA remaps this to:

(787 cyl, 64 heads, 63 sectors/track, 512 bytes/sector)

giving the same effective size disk, but with cylinder and head counts within the BIOS API's range (Incidentally, I have both Linux and FreeBSD existing on one of my hard disks above the 1024th physical cylinder, and both operating systems boot fine, thanks to BIOS LBA).

7.3.2 Boot Managers and Disk Allocation

Another gotcha to watch out when installing boot managers is allocating space for your boot manager. It is best to be aware of this issue up front to save yourself from having to reinstall one or more of your OSs.

If you followed the discussion in Section 7.2 about the Master Boot Sector (where the MBR is), Partition Boot Sectors, and the booting process, you may have been wondering just exactly where on your hard disk that nifty boot manager is going to live. Well, some boot managers are small enough to fit entirely within the Master Boot Sector (Cylinder 0, Head 0, Sector 0) along with the partition table. Others need a bit more room and actually extend a few sectors past the Master Boot Sector in the Cylinder 0 Head 0 track, since that is typically free. . . typically.

That is the catch. Some operating systems (FreeBSD included) let you start their partitions right after the Master Boot Sector at Cylinder 0, Head 0, Sector 2 if you want. In fact, if you give FreeBSD's sysinstall a disk with an empty chunk up front or the whole disk empty, that is where it will start the FreeBSD partition by default (at least it did when I fell into this trap). Then when you go to install your boot manager, if it is one that occupies a few extra sectors after the MBR, it will overwrite the front of the first partition's data. In the case of FreeBSD, this overwrites the disk label, and renders your FreeBSD partition unbootable.

The easy way to avoid this problem (and leave yourself the flexibility to try different boot managers later) is just to always leave the first full track on your disk unallocated when you partition your disk. That is, leave the space from Cylinder 0, Head 0, Sector 2 through Cylinder 0, Head 0, Sector 63 unallocated, and start your first partition at Cylinder 0, Head 1, Sector 1. For what it is worth, when you create a DOS partition at the front of your disk, DOS leaves this space open by default (this is why some boot managers assume it is free). So creating a DOS partition up at the front of your disk avoids this problem altogether. I like to do this myself, creating 1 Meg DOS partition up front, because it also avoids my primary DOS drive letters shifting later when I repartition.

For reference, the following boot managers use the Master Boot Sector to store their code and data:

- OS-BS 1.35
- Boot Easy
- LILO

These boot managers use a few additional sectors after the Master Boot Sector:

- OS-BS 2.0 Beta 8 (sectors 2-5)
- The OS/2 boot manager

7.3.3 What if your machine will not boot?

At some point when installing boot managers, you might leave the MBR in a state such that your machine will not boot. This is unlikely, but possible when re-FDISKing underneath an already-installed boot manager.

If you have a bootable DOS partition on your disk, you can boot off a DOS floppy, and run:

```
A:\> FDISK /MBR
```

to put the original, simple DOS boot code back into the system. You can then boot DOS (and DOS only) off the hard drive. Alternatively, just re-run your boot manager installation program off a bootable floppy.