

# Guide to x86 Bootstrapping (and Partitioning)

*``Behold ye scoffers,  
For I will work wonders in your days,  
Which ye will not believe."  
-- Book of Habakkuk*

## Important Note

The NetBSD information is probably out of date since around NetBSD-1.3 they have started using a block-list to load the secondary bootstrap and a stand-alone library (libsa), and another program (installboot) to set it up.

## Preface

This guide will attempt to describe partitioning, boot sequences, and the programs which manage them on the Intel 80x86 platform. Perhaps the simplest way to explain partitions and the bootup sequence is to start with simple cases, and add complexity as needed. We will start with a single-OS setup, and proceed to more complicated examples. The author takes no responsibility for any damages caused by following these directions. This is for informational purposes only.

I am in the process of writing a unified version of this document which will read sort of like a *choose your own adventure* book, in that you see how the computer gets to its final destination partition. Let me know what you think of this idea. I may keep these around as common case examples for the impatient.

I suppose I should start posting this regularly, but it isn't quite a FAQ. It's also terribly organized. Sorry.

## Processor Startup Sequence

The processors in Intel's x86 clan begin life by performing a hardware reset, initializing all the little bits of cache, registers, and buffers to a known value. It goes into [real mode](#) shortly. The EIP register is initialized to 0000FFF0H, and the CS register is a segment selector (value F000H) which points to a base address of FFFF0000H. Thus, execution begins at address FFFFFFFF0H, sixteen bytes from the top of physical memory, in an EPROM. The EPROM is usually located at a much lower physical address, but is being remapped to a high address by the system chipset (e.g. Intel 430HX). Note that the selector/base correspondence here is not the usual relationship when programming in real mode. Typically, in a PC this EPROM will set up a real-mode IDT and jump to the BIOS.

# Single OS, Single Disk

The simplest setup is a single OS on a single disk. For example, you might have [NetBSD](#) on your first and only SCSI disk (you do use [SCSI](#), don't you?).

## Bootup Sequence

1. The [BIOS](#) bootstrap routine generates an int 0x19 which usually loads the first [sector](#) of the floppy or hard disk (0:0:1 in CHS [disk addressing](#) format) in memory at [segment address](#) 0000:7C00H. The first sector (in this example) is the [primary bootstrap loader](#).

The BIOS checks that the last two bytes of that sector are AA55H. If they are not, you will probably get a BIOS-dependent message (maybe Non-System or Non-Bootable Disk) or a hang.

2. This primary or *first stage* bootstrap loader mainly has the job of loading the [secondary bootstrap loader](#). The code for the primary bootstrap is in `/usr/mdec/??boot`, `/usr/mdec/sdboot` in this case. This is generated from code in a directory, in this case `/sys/arch/i386/boot/`.

The primary loader examines the first sector of the disk and looks at something I call the [hard partition table](#). It examines this table for a partition with the [system id](#) of 165 (Net/Free/Open/386BSD). In our case, this partition should start at the beginning (0:0:1) of the disk.

It would load the 1-14th sectors of the NetBSD hard partition it found above. In this case, it would load the sectors 0:0:1 through 0:0:14. It loads this batch (which includes itself and the secondary bootstrap loader) into the area 0000:1000H (FreeBSD and Mach load it much higher). This number was chosen because the bottom 4k of memory is used as a scratch area for the BIOS.

If the secondary bootstrap loader is absent, you may get the following error messages from the NetBSD primary bootstrap:

- o No bootable partition
- o Read Error

Assuming all goes well, it then jumps to a pre-determined location in the code it just located -- the secondary bootstrap loader.

3. The NetBSD secondary bootstrap loader is recognizable because it provides the user with a prompt. It is represented by the file `/usr/mdec/boot??`, `/usr/mdec/sdboot` in this case.

The typical prompt looks like:

```
>> NetBSD BOOT: xxxx/yyyy k []
use hd(1,a)/netbsd to boot sd0 when wd0 is also installed
Boot: [[[sd(0,a)]/netbsd][-adrs] :-
```

The NetBSD secondary bootstrap is designed to load the kernel. It has no device drivers, so that means you'll have to boot off stuff your BIOS recognizes (i.e. the first two drives). The stuff about `hd` is only relevant if you have both SCSI and IDE controllers in your machine, and is a kluge to compensate for the way the BIOS works.

The bootstrap looks for the kernel of your choice. It would then load it into memory. Since the kernel might be bigger than 512K or 640K, the secondary bootstrap has to switch between real mode (so it can use BIOS calls) and protected mode (so it can access lots of memory). When you select a drive, or let it default, it goes and reads the partition table from the disklabel on that drive. That statement glosses over a lot of fine details, so let me elaborate. Suppose it defaults, and goes to boot the file `sd(0,a)/netbsd`.

Using a procedure similar to the primary bootstrap, it finds a hard partition on `sd0` with a NetBSD system ID. It then searches this hard partition for the disklabel. This contains an embedded [Unix partition table](#).

The Unix partition table is just a list of "partitions" (called slices by the FreeBSD people, but it's the same thing no matter how you, uh, slice it (ouch)) and their disk address ranges. It finds Unix partition `a` (the `a` in the `sd(0,a)` above) and finds the beginning of that Unix partition. It then looks for the file `netbsd` within the root of this filesystem (due to the leading slash).

It can respond with one of these error messages if something goes wrong:

```
Can't find KERNELFILE
```

```
    You specified a filename for the kernel which doesn't exist in the drive/partition
    combination you specified.
```

```
invalid format
```

```
    The partition you selected does not have the proper magic number (a special code used to
    identify formatted partitions). It could be damaged, unformatted, or not even a NetBSD
    partition.
```

```
Wangtek unsupported
```

```
    I'm not sure what planet this came from
```

```
kernel too large
```

```
    Your kernel exceeds 1MB or whatever the limit is.(TODO: what is it)
```

```
unknown device
```

You have to pick `fd*`, `sd*`, `hd*` or `wd*`

`bad unit`

Give something like `sd0`, not `sdq`.

## How to set your disk up this way

1. Run NetBSD's `fdisk`, and create a single hard partition that starts at 0:0:1 for NetBSD. (TODO: This step may or may not be necessary (or possible)) See step three.
2. Use `disklabel -w -r sd0 disktabentry volumelabel` to write a disklabel to the drive. Note that this (using `-r`) will overwrite the secondary and primary bootstrap loader area. In this situation, this means that it will overwrite the area from 0:0:1 to about 0:0:14 or so. The `"-r"` option must be used when writing to a disk that doesn't have a label already, unfortunately. (TODO: I think I'm a bit fuzzy here. What options really blow away what parts of the disk?)
3. Use `disklabel -B sd0` to write primary and secondary bootstrap loaders to the drive. Note that this will overwrite the hard partition table, but it has a "fake" compiled-in hard partition table that has a 386/NetBSD hard partition starting at the beginning of the disk. It will not overwrite the Unix partition table. (Note: you could simply have used the `-B` option in the previous `disklabel` command, but I seperated them here for clarity).
4. You can optionally go back and use NetBSD's `fdisk` to adjust the hard partition table to accurately reflect the size of the partition, but only the starting disk address is really necessary. The starting address is only really used when loading the secondary bootstrap, [step 2](#) above.

## Multiple OS, Single Disk

### Bootup Sequence

Let's say (since it is very common) that you have set up MS-DOS and NetBSD on your only disk. The bootup sequence is as follows:

1. Same as [first step](#) of Single OS, Single Disk above.
2. On all the MS-DOS/NetBSD systems I have experience with (excepting boot managers, see below), the MBR is the MS-DOS primary bootstrap code placed there when it was initially formatted for MS-DOS. Should the MS-DOS primary bootstrap be lost, the (once undocumented) MS-DOS command "FDISK /MBR" can re-write it to the (first sector of the) disk. This hasn't blown away the embedded partition table on the rare occasions I have done it.

The MS-DOS primary boot loader would then proceed to scan its embedded hard partition table for an [active](#) hard partition (TODO: what if there are more than one, or none? is that ok?). The primary loader then relocates itself and loads the first sector from the active hard partition. In a sense, this parallels step one above.

If the first sector of the active hard partition doesn't have the right magic number, it might say:

```
Non-System disk or disk error
Replace and press any key when ready
```

- Let's assume the active hard partition was the MS-DOS partition. Now we load the first sector of this partition, which could be a full track into the disk (that is, address 0:1:1). We then load the second and subsequent sectors until we have bootstrapped ourselves into MS-DOS. Some versions of MS-DOS (6.x and up?) will print a message at this point similar to:

```
Starting MS-DOS.
```

On the other hand, let us assume the active hard partition was the NetBSD partition. Then it would proceed by loading the first sector of the NetBSD partition, which we will assume is at location (100:0:1). This is the primary bootstrap as described [above](#). This time, when it examines the hard partition table, it will not be the one embedded within itself. Remember, the primary NetBSD bootstrap reads the hard partition table from (0:0:1), even though it resides at some other location, in this case (100:0:1). It searches for the NetBSD partition, as before, which starts at location (100:0:1). It then loads the 1st-14th sectors from this location.

- At this point, if the active partition was MS-DOS, your config file and junk is parsed and run. If NetBSD was your active partition, the actions are the same as part 3 (secondary bootstrap) [above](#).

## How to set your disk up like this:

- Use MS-DOS's `FDISK . EXE` program (usually run from floppy) to create a primary MS-DOS (hard) partition that does NOT fill the entire drive. It will usually start at (0:1:1) and end at some cylinder boundary, let's say (100:0:1). Since MS-DOS's `FDISK.EXE` is brain damaged and broken, you may have to tell it in some other bizarre measurements (e.g. MBs rather than cylinders). It leaves a blank track at the beginning for the boot sector and anything else (viruses, etc) that you may want to put there -- with no way to override it.
- Then, use NetBSD's `fdisk` program to create another partition that spans the rest of the drive. Make sure to give it a system id of 165, and mark one of the two (hard) partitions as active.
- `disklabel -B -w -r sd0 disktabentry`  
This command looks up the NetBSD partition and writes the primary and secondary bootblocks for NetBSD into the first sectors of the NetBSD hard partition (for example, they might be put in (100:0:1) through (100:0:14)). The Unix partition table (the `disklabel`, as it is called) is stored starting in the first sector of the NetBSD area of the disk (100:0:1).

## Multiple OS, Single Disk, Boot Manager

## Boot Sequence

1. Same as above
2. At this point, the boot manager's primary bootstrap will take over. Some may fit entirely in the first sector, but most will have a two-part procedure, where the first sector will load its secondary bootstrap in a manner reminiscent of OSes. For example, OS-BS 2.0 Beta 8 loads several of the following sectors into memory, and presents a menu (which can be edited with an external utility). This menu allows the user to pick any hard partition off the disk. After the boot manager decided on a partition (either by user decision or timeout, etc) then it would start an OS by jumping to the first sector of that hard partition and executing it.

### To set your disk up like this:

Same as above; set up each OS, then install the boot manager last (or as per its instructions).

## Multiple OS, Multiple Disks, Boot Manager

This procedure is essentially the same. There appear to be two ways of booting of the second drive. The first way, used by O/S Boot Select, is to directly load the first sector of the partition off the second drive. The other (slightly more indirect) way, taken by pboot, is to "cascade", loading the MBR of the second drive as though it were the first. I think that O/S Boot Select can do this also.

The documentation that comes with OS-BS discusses some issues with booting from drives other than the first. It appears that the main issue is that the OS must be prepared to load from another disk (i.e. not assume that it's on the first disk). You can be reasonably sure that most Microsoft OSes will not boot off the second disk. I'm pretty sure NetBSD, FreeBSD, and Linux can.

## Tools

This section is devoted to programs which will help you deal with boot sequences or partitioning.

### Boot Managers

#### [bteasy](#)

I consider "boot easy" a fair boot manager. Current version is 17.

#### [pboot.zip](#)

This is a good boot manager.

#### [osbsBETA.exe \(self extracting pkzip archive\)](#)

This is a very good boot manager. Get more info on it from its [home page](#). His README file has a detailed section of how your OS should compute the partition to load from, etc. Note that it has

been in beta for years, but it is quite stable.

### [yapboot](#)

This is the "Yet Another Partition Sector Boot Program" editor and boot manager. Get it from [ftp. ee.und.ac.za](ftp://ee.und.ac.za).

### System Commander

This is a commercial product which you can find out about from [V Communications' web site](#). It allows you to boot any partition and doesn't require reformatting when resizing partitions. I haven't used it personally.

## Partition Table Editors

### [PFDISK.EXE](#)

An excellent MS-DOS program which gives you more control over the hard partition table in the MBR than the native FDISK.EXE; could be used in lieu of NetBSD fdisk. Download it from the [FreeBSD archives](#). However, read what the author has to say:

The most recent version is 1.3 (from Oct. 1990). That works OK. Note that an incomplete distribution of pfdisk has been propagated around, somewhat to my dissapointment, because it omitted what I consider some of the more clever parts of package. The full release of pfdisk is available from [ftp.mc.com](ftp://mc.com).

### [yapboot](#)

This is the "Yet Another Partition Sector Boot Program" editor and boot manager. Get it from [ftp. ee.und.ac.za](ftp://ee.und.ac.za).

### Norton Diskedit

You can't get it for free, but it is a swell program. It comes with the [Norton Utilities](#) set. If you're trapped in a uSoft world, you should probably have it. You can probably get more information from [Symantec's home page](#).

### Partition Magic

This is a commercial solution made by [PowerQuest](#). This tool appears to let you resize partitions and do all kinds of stuff. I've never used it myself, but they asked politely. Go visit [their web site](#) for more info.

### System Commander

This is a commercial product which you can find out about from [V Communications' web site](#). It allows you to boot any partition and doesn't require reformatting when resizing partitions. I haven't used it personally.

## Partition Resizers

### [FIPS](#)

This appears to shrink MS-DOS FAT partitions. You can FTP fips11.zip from [the FreeBSD archives](#), or fips15.zip from [the Linux system/install archive](#). Supposedly there's a Linux HOWTO on it as well, but it apparently isn't in the archives yet.

### Partition Magic

Appears to be able to resize partitions. Visit [their web site](#) for more info.

### System Commander

This is a commercial product which you can find out about from [V Communications' web site](#). It allows you to boot any partition and doesn't require reformatting when resizing partitions. I haven't used it personally.

## References and Other Information

See also:

- The NetBSD source tree, especially the files in [NetBSD /usr/src/sys/arch/i386/boot](#) and [NetBSD /usr/src/sys/arch/i386/stand](#).
- OpenBSD's source tree, especially the files in [libsa \(standalone x86 boot code\)](#)
- [FreeBSD's DOS based tools section](#)
- [NetBSD's DOS utils section](#)
- [OpenBSD's DOS based tools section](#)
- Erich Boleyn's [multiboot proposal](#), which is a protocol between a bootstrap program and an OS kernel.
- [GRUB](#) could very well be the end-all be-all of kernel loaders, with a nice interface and all kinds of powerful features (loading kernel modules independently, etc). It can chain-load things like MS-DOS, OS/2 and other OSes, as well as interpreting many file systems and booting multiboot compliant kernels as well as natively booting several OS kernels. It's currently in beta-test.
- [SOLO](#), the Shag/OS bootloader
- Hale Landis' [How It Works](#) documents the exact structure of the hard/extended partition table(s) which MS-DOS understands, and contains disassembly of different bootstrap programs and a good section on geometry translation. This is a must-read (although it has a few errors).
- Ralf Brown's [interrupt list](#) is a very extensive treatise on interrupts which covers many topics including BIOS calls, MS-DOS/Win calls, and calls to software-based drivers. A copy of his web pages may also be found at [CMU](#).
- Robert Collins' [x86.org](#) is a web site devoted to giving you an, uh, alternative but very detailed view of Intel's microprocessor line.
- Seagate has a [FAQ list](#), a very nice archive of various disk-related FAQs.
- Evan's [partitioning how-to](#) has a definite MS-\* slant.
- The Linux [Assembly HOWTO](#); this document describes how to program in assembly using FREE programming tools, focusing on development for or from the Linux Operating System on i386 platforms, mostly 32-bit mode. You may wish to look at [the most recent version of the](#)

## [Linux Assembly HOWTO](#).

- The Linux [Disk HOWTO](#)
- The Linux [Bootdisk HOWTO](#)
- The Linux [Installation HOWTO](#)
- The Linux [Partitioning Mini-HOWTO](#)
- The Linux [LILO Mini-HOWTO](#)
- [comp.lang.asm.x86 FAQ \(HTML\)](#), [comp.lang.asm.x86 FAQ \(zip\)](#) for x86 assembler questions
- The LILO (Linux LOader) documentation pages
  - [Technical Overview \(postscript, gzipped\)](#)
  - [Users Guide \(postscript, gzipped\)](#)
- My stab at a [NetBSD stage 1 bootloader \(shar\)](#) which is probably outdated or unnecessary.
- [Intel's Pentium Pro Family Developer's Manual Volume 3: Operating System Writer's Manual](#), especially Chapter 8 (Processor Management and Initialization). You may also want Intel's [Pentium Manuals](#), particularly the [Intel Architecture Software Developer's Manual, Volume 3: System Programming Guide](#). Note that they charge about US\$25 for each printed volume, whereas [Motorola](#) gives their manuals away for free (unless you work for them). Older manuals (even the 386) are probably fine. Also see [Intel's developer web site](#).
- The [80x86 Assembly Pages](#), tons of links and information on Intel 80x86 assembly.
- The [Unix Boot Process](#), covering what comes after the kernel gets loaded, but only vaguely.
- The [Red Hat Linux Boot Process Tips](#), covering Unix's init program. You may also want to search for the /etc/default thread in [the NetBSD-current mailing list archives](#), dates 19950724-19950819.
- The [sformat manual page](#) covers a SCSI disklabelling/fixing/diagnosing program for SunOS/Solaris. You may also [download sformat itself](#).
- Donald Knuth's "The Art of Computer Programming Volume 1", the best book I've ever seen from which to learn assembly language programming (albeit in a generic sense). It's like Drano for your mind.
- "Plug-and-Play OPROMs and the BIOS Boot Specification", by Tom Roden and Scott Townsend, Dr. Dobbs Journal, #265, May 1997, pp 38-

## Caveats and Warnings

Beware, some products do not stick with the c:h:s numbering. For example, Norton's excellent diskedit utility has h:c:s in its partition table layout.

Make frequent backups.

Print out partition tables (of all types) when you make a change, and above all, KEEP GOOD RECORDS. When stuff starts getting weird, take your time and document every change. Always read first, then write.

NetBSD's disklabel is rampantly destructive. Don't assume it won't destroy your data. Remember, if you ASSUME, it... um, hm,... I know there's a trick in here somewhere...

No, really, I have been burned yet AGAIN by NetBSD's disklabel. I've got to tear apart that source one day and figure out how it really works. If it can happen to me...

This bears repeating; make frequent backups.

The exact boot sequence of your OS may vary; FreeBSD and Mach will probably be very close to what I have described here, while Linux will be less so.

## Glossary

### Active Partition

One of the hard partitions on a disk should be marked as the active partition, which designates the [hard partition](#) to boot.

### ATA

AT attachment, the politically correct term for [IDE](#).

### Boot Blocks

See [Primary](#) and [Secondary](#) Bootstrap Loaders.

### BIOS

Basic Input/Output System, a slow and antiquated ROM which is found in most x86 PCs and consists of 16-bit code. It is important because it is ubiquitous and because it is a very easy way to get your kernel and device drivers loaded in the first place.

### Cylinder

A concentric area around the disk spindle, spanning multiple [platters](#) (fixed *r*).

### Disk Addressing

A sector of the disk is identified by an ordered triple, consisting of cylinder, head, sector. Traditionally, cylinders and head numbering starts with zero, while sectors start with one. Most people think of the disk as being laid out in "row-major order". This means that the numbering goes in the following order; 0:0:1, 0:0:2, 0:0:3, and so on. Although the disk is laid out in three dimensions, it may help to think of it this way.

Another way of phrasing the order is that all the sectors are in order, then you proceed to the next head. After every head's set of sectors is the next cylinder. Physically, heads are aligned vertically, cylinders are concentric cylinders about the axis of rotation, and sectors are radial units that form a complete circle about the axis of rotation. My document uses *CHS* numbering, so cylinders come first, heads next, sectors last.

### Disklabel

A bunch of data about a drive that tells Unix how to use the drive. In NetBSD, it is currently stored at the beginning (but not the first sector) of your "a" or "c" partition -- I'm not sure which (TODO: find out).

## EIDE

The more advanced form of [IDE](#).

## Extended Partition

Although I'd like to forget about these, M\$-DOS had a technique for getting around the four [hard partition](#) limitation. Apparently, (only) one of the hard partitions can be marked as an extended partition, and the "extended partition table", essentially another hard partition table, is stored somewhere as the first sector of the hard partition which was marked extended. For some reason, you can only make one of these four new partitions point to an actual data-containing area. However, you can also have one of these four new partitions point to *another* extended partition, so in effect this forms a linked list that you can extend as much as you like. Don't ask me why the other two entries get wasted.

Linux respects these extra partitions, and can use them as easily as hard partitions (NetBSD can not). In fact, I have heard Linux LILO **must** be installed in one of these partitions (or on floppy). I haven't looked at the LILO docs yet myself so I'll get back to you when I do.

See also the [hard partition table](#).

## Filesystem

A single Unix Partition.

## First Sector

See [master boot record](#).

## Geometry Translation

Originally and still a kluge, this is a term for what is done by a drive controller BIOS to allow you to use large disks (over 512MB) using old mainboard BIOS routines, which speak in terms of CHS (see [disk addressing](#)). What I mean is that originally there were BIOSes that read from disk using supplied parameters which were packed into registers. Unfortunately, they weren't very forward-looking and didn't leave enough space for future drives. When the number of bits required to identify a cylinder (10) exceeded the number of bits provided by the BIOS, something had to be done. Well, there were 6 bits used to identify the drive head, and it was well-nigh impossible to change the BIOS interface at this point (curse backwards binary compatibility!). So the drive controller BIOS designers decided to emulate a disk with less cylinders and more heads. Thus, the controller BIOS translates a disk address using this fictitious geometry to the real drive geometry, usually using some quick bit shifts. As long as you ALWAYS use this fictitious geometry, and never change controllers, no big deal. See the [How It Works](#) document for more info about this.

This is basically all pretty bogus since modern drives use variable density bit packing on the cylinders to ensure maximum storage capacity and low magnetic drift. The Macintosh low-

density floppy drives did this in a rather crude manner (very cool of them). The geometry they report is already disconnected from reality, so don't put too much credibility in the whole mess.

Personally, I think the whole thing would be *so* much better if everyone used [LBA](#) all the time. Unfortunately, MS-DOG doesn't pay attention to the LBAs in the hard partition table (as far as I know) so we're stuck with it. Actually, while I'm ranting, a protected-mode BIOS and/or a hack to start the machine in protected mode would be pretty cool.

## Hard Partition Table

A listing of disk address ranges that is always stored in the first sector of the disk (the [master boot record](#)). Limited in capacity to four partitions. Sometimes called the old-style, standard, or MS-DOS partition table. Each partition table entry takes up 16 bytes. It is this fundamental limitation which inspired the [Unix partition table](#). See also [Extended Partitions](#). Note that floppy disks can't have partition tables. Sorry.

## Head

See [Platter](#).

## IDE

IDE stands for "Integrated Drive Electronics", and it's also called ATA. IDE is brain-damaged and lame. A recent extension to IDE, called EIDE or ATA-2, attempts to come near the functionality that [SCSI](#) has had for years. If you can't take my word for it, go read about it in [Seagate's excellent ATA FAQ](#). I'll wait here; come back when your brain is full. Be sure to thank the people at Seagate for being so candid with us!

By the time IDE controller designers add all the features of SCSI into (E)IDE (which they have already started doing), it's going to be even *more* complicated than SCSI. Besides all this, IDE has trouble coexisting with other drive types, and it is x86 I/O bus specific, making it difficult (but not impossible) to design other computer architectures to use IDE drives. One of its few advantages are that due to its overall simplicity, the overhead of setting up a command (controller latency) is much lower than SCSI, so you are going to have better access times. Oh yeah, they're also cheap, in more ways than one. On the downside, the on-disk buffer to controller speed in the best EIDE setup doesn't even come close to the bandwidth of Ultra-Wide SCSI.

## LBA

LBA is an acronym for "Linear Block Address[ing]". This acronym was invented by IDE drive manufacturers to describe the method of addressing disk blocks that was independent of the disk geometry. SCSI has used LBA from the very beginning. Note that in computing the LBA from a CHS [disk address](#), every operating system I know of makes sure that sectors within a track are numbered contiguously, then that tracks within a cylinder are numbered contiguously. This means that even if two different software systems disagree about the geometry of a disk, their LBAs are the same.

## Magic Number

Forty-two. Also, a special binary signature placed within a file or partition (usually at a fixed

place), used to identify such a beast. For example, the "#!" at the beginning of a Unix script are a form of magic numbers. Linux keeps magic numbers at the beginning of most kernel data structures to detect corruption.

## Master Boot Record

The "first" sector of a disk (0:0:1). See [Disk Addressing](#).

## MBR

See [master boot record](#).

## Physical Partitions

See [Hard Partitions](#).

## Physical Partition Table

See [Hard Partition Table](#).

## Platter

Also called a side or head (fixed  $z$ ).

## Primary Bootstrap Loader

A piece of code generated by NetBSD, and stored in (e.g.) `/usr/mdec/sdboot`.

## Protected Mode

Intel's name for an internal state of the processor which causes it to work in what is commonly referred to as "32-bit mode". The name refers to extra memory protection mechanisms which are not enabled in [real mode](#).

## Real Mode

Whoa boy are you a curious one. Real mode refers to an essentially extinct form of backwards compatibility in the x86 architectures, and is sometimes called "16-bit mode". If you want more information than this, you'll have to look at some [Intel processor reference manuals](#).

## SCSI

SCSI stands for Small Computer Systems Interface - the One True Controller Type. SCSI is a real peer-to-peer kind of protocol, meaning you can do cool things like hook two machines up to the same bus with six other devices (disk drives, scanners, etc) shared between them. Some people and companies (Microbrain) claim SCSI is hard to set up. Bullsh1t!!! It's just that SCSI can sling a ton of data around, and so it is usually set up to take advantage of things like DMA, and the only reason this is hard is because of the ISA bus. If you're ever going to have more than one disk drive, or you want high performance, get SCSI. Know ye that the SCSI spec may take more than one hour to read. :)

related info

- o [FAQ, HTML, old](#)
- o [FAQ part 1, new](#)
- o [SCSI game rules \(clever, useful\)](#)

## Secondary Bootstrap Loader

A complement to the Primary Bootstrap Loader, (e.g.) stored in `/usr/mdec/bootsd`.

## Side

See [Platter](#).

## Sector

A fixed place on a hard drive (fixed  $r$ ,  $theta$ ,  $z$ ).

## Segment Notation

The x86 has a much-maligned property of being a *segmented* architecture. The  $xxxx:yyyy$  notation means you multiply the  $xxxx$  by a factor of 16, and add to the  $yyyy$  to get the physical address, *in certain modes*. Despite being dragged through the mud, segments can let you do pretty neat things like per-segment protection (rather than per-page), and checking array bounds in hardware. Nevertheless, it takes a long time to load segment descriptor registers (9 cycles, with potential hidden costs).

## Soft Partitions

See [Unix Partition Table](#).

## System ID

A per-partition, one-byte code in the hard partition table which identifies the OS/FS type of that partition. NetBSD, for example, has a system id of 165. Linux has two types; one for Linux swap, one for Linux primary, since these must occupy different hard partitions. See my (rather incomplete) [table](#) of known values. A more complete set of web pages can be found on [A.E. Brouwer's site](#).

There is another table (currently #549, under INT 19) within Ralf Brown's [interrupt list](#), which you *might* be able to find at one of the following locations. Note that the table numbers move around, and so do the URLs.

- [DJ Delorie's conversion](#) for djgpp
- [Marc Perkel's conversion](#)

## Track

A ring-shaped surface where a cylinder intersects the side of a platter (fixed  $r$ ,  $z$ ).

## Unix Partition Table

A listing of "partitions" (letters a-h) and their disk address ranges, as well as some other information. Part of the disklabel. NetBSD stores all of its partitions in such a way that they usually exist contiguously within a single hard partition, although they can point outside of it if you so desire (not recommended for novices). They basically take the place of the [hard partition table](#), although a trivial hard partition table must be used to find one. FreeBSD calls these "slices". I have been told Linux can read them, too. One cool thing is that a unix partition table can contain the hard partition table, and be located on the [master boot record](#).

## Postamble

Send any corrections, kudos, additions, or "needs clarification" to my email address below. Any additions about other OSes would be cool. Please submit this to relevant web indexes and search engines where it is not already listed.

---

Go to this level's [index](#)

[VaX#n8\\_vax@linkdead.paranoia.com](mailto:vax#n8_vax@linkdead.paranoia.com)

Original date: 20 July 1995

Updated: Sun Jan 18 06:41:55 CST 1998

Uploaded to Server: Thursday, 08-Feb-2001 20:41:39 CST