Menu
Home
FAQ
Screen Shots
Download
Tutorials
Any Comments?

Comments
Ankit
09 Dec 2011
I don't know how to install from .img fi..
>> show
Masum
22 Nov 2011
How to install Taj os in my pc. and ..
>> show
Me
21 Oct 2011
TAJ seems to lock up in boot using Bochs..
>> show

Booting Process

Intoduction

When a computer starts up (obviously by pressing the power button), the first thing that occurs is it send a signal to motherboard which in turn starts the power supply. After supplying the correct amount of power to each device, it send a signal called "Power OK" to BIOS which resides on motherboard.

Once the BIOS receive the "Power OK" signal, it starts the booting process by first initializing a process called POST (Power On Self Test). POST first check that every device has right amount of power and then it check whether the memory is not corrupted. Then it initialize each devices and finally it gives control to BIOS for further booting.

Now the final process of booting begins. For this the BIOS first find 512 bytes of image called MBR (Master Boot Record) or Bootsector from the floppy disk or hard disk which is used for booting. The priority of boot devices is set by the user in BIOS setting. The normal priority is floppy disk first, then hard disk.

Once BIOS finds the bootsector it loads the image in memory and execute it. If a valid bootsector is not found, BIOS check for next drive in boot sequence until it find valid bootsector. If BIOS fails to get valid bootsector, generally it stops the execution and gives an error message "Disk boot failure".

It is bootsectors responsibility to load the operating system in memory and execute it.

Master Boot Record

A device is "bootable" if it carries a boot sector with the byte sequence 0x55, 0xAA in bytes 511 and 512 respectively. When the BIOS finds such a boot sector, it is loaded into memory at a specific location; this is usually 0x0000:0x7c00 (segment 0, address 0x7c00). However, some BIOS' load to 0x7c0:0x0000 (segment 0x07c0, offset 0), which resolves to the same physical address, but can be surprising.

When the wrong CS:IP pair is assumed, absolute near jumps will not work properly, and any code like mov ax,cs; mov ds,ax will result in unexpected variable locations. A good practice is to enforce CS:IP at the very start of your boot sector.

```
ORG 0x7C00
jmp 0x0000:start
start:
```

or

```
ORG 0
jmp 0x07C0:start
start:
```

Recent Tutorials
OS Glossary
Interrupt Descriptor Table (IDT)
Booting Process
Writing Hello World Bootloader
Real mode
Partition Table
Global Descriptor Table (GDT)
Segmentation
Makefile Tutorial
Programming Floppy Disk Controller
View All »

On a hard drive, the so-called Master Boot Record (MBR) holds executable code at offset 0x0000 - 0x01bd, followed by table entries for the four primary partitions, using sixteen bytes per entry (0x01be - 0x01fd), and the two-byte signature (0x01fe - 0x01ff).

The layout of the table entries is as follows:

Offset	Size (bytes)	Description
0x00	1	Boot Indicator (0x80=bootable, 0x00=not bootable)
0x01	1	Starting Head Number
0x02	2	Starting Cylinder Number (10 bits) and Sector (6 bits)
0x04	1	Descriptor (Type of partition/filesystem)
0x05	1	Ending Head Number
0x06	2	Ending Cylinder and Sector numbers
0x08	4	Starting Sector (relative to begining of disk)
0x0C	4	Number of Sectors in partition

Kernel Image

Now we jump two steps ahead and look at where we want to go: Our kernel image. Your boot record would be easiest if it could just copy the kernel image from disk to memory and jump to some given offset. Unfortunately, unless you take extra precautions, your compiler adds all sort of startup code, relocation tables etc. To get a "flat binary" that can be loaded in this simple copy-and-run way, you have to tell GCC:

```
gcc -c my_kernel.c
ld my_kernel.o -o kernel.bin --oformat=binary -Ttext=0x100000
```

The `-c` switch tells GCC to stop right after compilation, i.e. not to link the object file.

The `--oformat=binary` switch tells the linker you want your output file to be a plain binary image (no startup code, no relocations, ...)

The `-Ttext=0x100000` tells the linker you want your "text" (code segment) address to start at the 1mb memory mark. Since you do not link in any relocation tables, the linker has to resolve all references at link time, and has to know where the executable will be loaded to.

You are of course obliged to load your kernel image to the correct offset, or the references the linker did set up will be invalid.

Back