Linux Bootloader – Report

Vaibhav Devekar

# Introduction

Boot loader is a program traditionally situated at the first sector (512 bytes) of the hard drive; and it is the sector where the boot process starts from. The BIOS on startup loads this sector starting at memory address 7C00:0000 and starts executing the code in 16-bit Real Mode. Since 512 bytes of code may not be enough to load a kernel, the alternative is to use the Stage1 to load Stage2 from the disk. The Stage2 can be of any size and can handle setting up GDT and LDT, entering 32-bit Protected mode and loading and transferring execution to kernel.

Currently the bootloader has been developed such that it needs two hard-disks:

1. The first is a raw hard-disk with the STAGE 1 in the MBR followed by the STAGE 2 and then the kernel image (vmlinuz).
2. The second hard-disk is ext3/ext4 formatted hard-disk containing the /sbin directory with 'init' executable inside the directory.

The entire codebase has been implemented in NASM-style Assembly. The contents of the former hard-disk and the state of memory for this bootloader during the booting process are shown below:

Disk Structure and Memory Mapping during booting

**Boot Disk structure:**   
Sector: 0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Stage 1(MBR) | Stage 2 | Stage 2 | Stage2 | Vmlinuz …. |

**Memory during booting:**

The numbers on the right denote the linear addresses at which the code is loaded in memory.

|  |
| --- |
| Protected Mode Kernel |
|  |
|  |
|  |
| Staging area for Protected Mode Kernel |
| Real Mode Kernel with heap and stack |
| Stage 1 |
| Stage 2 |
|  |

|  |
| --- |
| 100000 |
|  |
|  |
|  |
| 20000 -2FFFF |
| 10000 – 1FFFF |
| 07C00 |
| 00500 |
| 0 |

# Stage1

Initially I had implemented FAT12 code to read FAT and Root directories to locate Stage 2 but later I decided to do away with it since with larger disks, I cannot load their large FATs at once in memory. I would have needed to load the FAT sectors in batches since memory is short. I decided to avoid this altogether and rather load Stage 2 from pre-defined location on disk.

The Stage1 performs following tasks:

1. Set up segment as well as stack registers. Initialize all segment registers to 07C0, stack pointer to FFFF.
2. Actual disk access is in terms of CHS addressing. Hence I used the formulae to convert LBA to CHS. The BIOS Parameter Block/Partition Table part of the MBR contains details about the disk such as number of cylinders, heads , sector per track and so on. These are used to compute the LBA/CHS address when interacting with the disk. [1,2]
3. The Stage 2 is located at sector 1. Accordingly get the CHS address and load the sectors containing Stage 2 at 0050:0000.  
   Disk read is done using BIOS Interrupt - INT 13h. Messages are printed using INT 10h.
4. Transfer execution to Stage2.

Stage2  
The Stage1 loads Stage2 image at 0050:0000. The Stage2 handles setting up GDT, entering 32-bit Protected mode, loading kernel image and transferring execution to kernel. The kernel is copied to the 1MB mark. The bootloader can now start the linux kernel which in turn runs a sample init program.

The Stage2 performs following tasks:

1. Set up segment as well as stack registers. Initialize all segment registers to 0, stack pointer to FFFF.
2. Install GDT: The GDT consisting of null, code and data descriptors is loaded using LGDT instruction [3].
3. Enable A20: To be able to address high memory, we enable the A20 gate. This is done by reading and writing to output port of Keyboard Controller. In this case, we first disable the keyboard and read data from output port. We enable the A20 bit and send the data back to output port and re-enable the keyboard [4].
4. Load the boot sector of kernel located at sector 4 on disk to address 0x10000 in memory.
5. Read the number of sectors of Real code to be loaded from the offset 0x1F1 of the boot sector code. Load the Real mode code starting from 0x10200 [7].
6. The offset 0x1F4 of the boot sector stores the size of Protected mode code in the form of 16-byte paragraphs.
7. Load 1MB chunk of Protected mode code at 0x20000 to 0x2FFFF temporarily. Enter Protected Mode and copy the chunk to 1MB mark.[5]
8. Return to Real Mode and copy the next chunk to 0x20000 and then to high memory in Protected Mode till the entire kernel code is in memory.
9. Return to Real Mode and set the obligatory kernel header fields such as type of bootloader and the kernel command-line.[7]
10. Jump to the Real Mode code at 0x10200, staring the kernel.

The kernel after initialization schedules the init process. The init program displays a "Hello World" message.

# Future Work:

1. Boot a Linux distribution such as Ubuntu.
2. Adapt bootloader to work with existing partitioned hard-disk.
3. Utility to configure and install bootloader.

# References:

1. <http://www.osdever.net/tutorials/view/loading-sectors> - Reading disks
2. <http://www.osdever.net/tutorials/view/lba-to-chs> - LBA to CHS conversion
3. <http://wiki.osdev.org/GDT_Tutorial> - GDT installation
4. <http://wiki.osdev.org/A20> - A20 activation
5. <http://wiki.osdev.org/Protected_mode> - Protected Mode
6. <http://www.brokenthorn.com/Resources/OSDev10.html> - 32-bit Video Programming
7. <https://www.kernel.org/doc/Documentation/x86/boot.txt> - Linux boot and header specification
8. <http://landley.net/writing/rootfs-howto.html> - Initramfs
9. Boot Process:  
   <http://wiki.osdev.org/Boot_Sequence>  
   <http://duartes.org/gustavo/blog/post/how-computers-boot-up>
10. Bootloader examples:  
    Tutorial for 2-Stage bootloader (FAT12):   
    [www.brokenthorn.com/Resources/OSDevIndex.html](http://www.brokenthorn.com/Resources/OSDevIndex.html)   
    Tutorial for single stage bootloader:

<http://megalomaniacbore.blogspot.com/2011/09/writing-your-own-boot-loader.html>   
Single stage bootloader on FAT12 disk:   
<http://anselmjm.blogspot.com/2008/05/my-fat12-bootloader-asm.html>   
Minimal Bootloader for Linux:

<http://sebastian-plotz.blogspot.de/>