CS510 Languages and Low Level Programming: Portfolio submission #3, Topic 5

Due on April 30, 2016 at 11:59pm

Mark P. Jones Spring 2016

Konstantin Macarenco

Attempted Topic:
Describe the key steps in a typical boot process, including the role of a bootloader.

Boot process described in class CPU INIT → BIOS/UEFI (or some other ROM BOOTLOADER) → SCAN DEVICES → FIND MBR → LOAD MBR → MBR CODE (BOOTLOADER) → GRUB → KERNEL + MODULES; is very general and used by almost all computer architectures.

The sequence begins with CPU initialization, followed by self test and then %eip is set to some predefined by the vendor address. Reset vectors used by various vendors:

1. The reset vector for the 8086 processor is at physical address FFFF0h (16 bytes below 1 MB).
2. The reset vector for the 80386 and later x86 processors is physical address FFFFFFF0h (16 bytes below 4 GB).
3. The reset vector for PowerPC/Power Architecture processors is at an effective address of 0x00000100 for 32-bit processors and 0x0000000000000100 for 64-bit processors.
4. The reset vector for SPARC version 8 processors is at an address of 0x00; the reset vector for SPARC version 9 processors is at an address of 0x20 for power-on reset, 0x40 for watchdog reset, 0x60 for externally initiated reset, and 0x80 for software-initiated reset.

Then CPU Executes ROM bootloader (BIOS/EFI), which scans available devices for a Master Boot Record (MBR). Once record is found BIOS copies MBR to RAM and let MBR code to take over further execution. MBR Code - loads a bootloader from some known location on the disk. Bootloader is used to prepare the computer for use. There are various types of bootloader used by different vendors.

1. Microsoft NTLDR
2. Apple BootX
3. Linux GRUB

All implementations have similar task, with, sometimes significant differences in functionality. An OS kernel expects certain information available for further boot like registers set to predefined values, available memory information, boot flags ans system permissions i.e. all system information required for successful work. For example Linux expects boot info in multiboot standard. Multiboot allows to treat a kernel as any other executable file in ELF standard. GRUB loads OS kernel according to the addresses defined in kernel's ELF linker script.
Instead of listing field that multboot provides I extended bootinfo example to print all information that available to kernel.
After bootloader is it sets EIP to the address of the kernel and OS takes over.
At this stage bootloader code is no longer needed and can be deleted/overwritten.
Bootloader abstracts a Particular OS boot details from Vendor specific hardware, making kernel more portable.