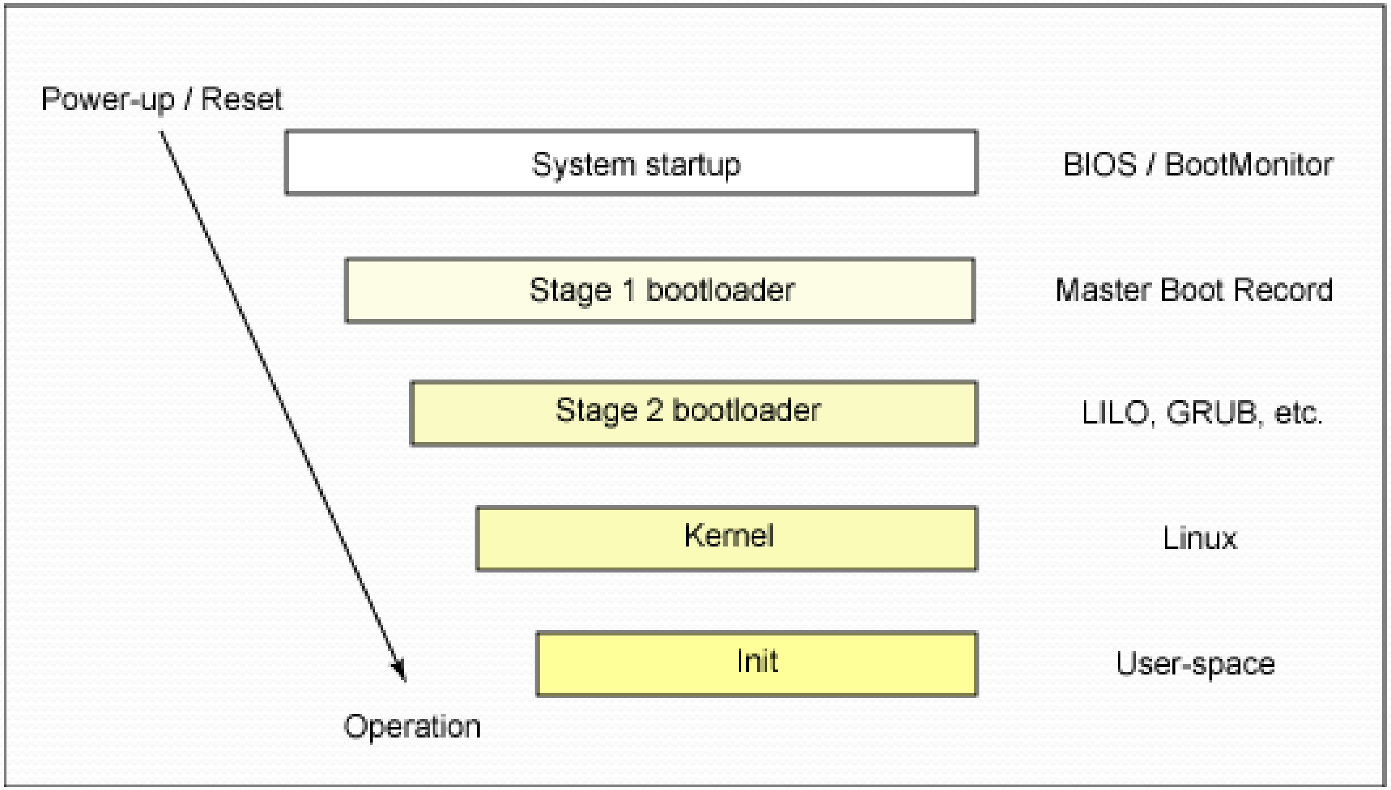
Boot Process

Presented By:

Amir Reza Niakan Lahiji

Boot Process Overview



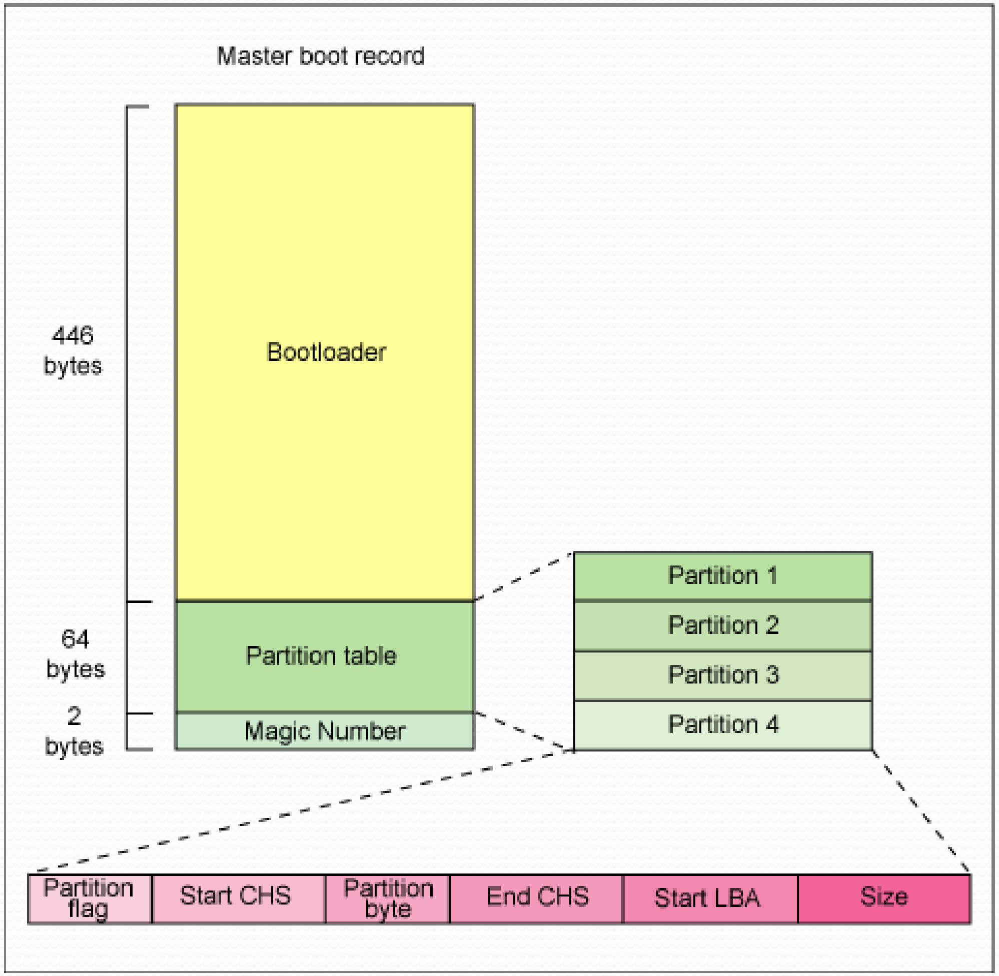
System Startup in PC

- booting Linux begins in the BIOS at address oxFFFFo
  - First step: Power-On Self Test (POST)
  - Second step: local device enumeration and initialization
    - Searches for devices that are both active and bootable based on the preferred order which is determined in CMOS
    - If hard disk must be boot(Suppose Linux resides in hard disk)
      - BIOS loads MBR (Master Boot Record) in RAM and yields control to it
      - MBR contains the primary boot loader
      - MBR is a 512-byte sector, located in the first sector on the disk (sector 1 of cylinder 0, head 0)

Stage 1 boot loader

- The primary boot loader is a 512-byte image which contains
  - Program Code (first 446 bytes)
  - Partition Table (64 bytes)
  - Magic Number (2 bytes)
  - The job of the primary boot loader is to find and load the secondary boot loader
    - First looks through the partition table for an active partition
    - When it finds an active partition, it scans the remaining partitions in the table to ensure that they're all inactive
    - When this is verified, the active partition's boot record is read from the device into RAM and executed

Stage 1 boot loader



Stage 2 boot loader (Kernel loader)

- The task at this stage is to load the Linux kernel and optional initial RAM disk.
- The first- and second-stage boot loaders combined are called Linux Loader (LILO) or GRand Unified Bootloader (GRUB) in the x86 PC environment.
- The great thing about GRUB is that it includes knowledge of Linux file systems.
  - GRUB can load a Linux kernel from an ext2 or ext3 (and also ext4) file system
  - It does this by making the two-stage boot loader into a three-stage boot loader.
  - Stage 1 (MBR) boots a stage 1.5 boot loader that understands the particular file system containing the Linux kernel image

Kernel loader

- Kernel image is compressed and typically this is
  - a zImage (compressed image, less than 512KB) or
  - a bzImage (big compressed image, greater than 512KB)
- At the head of this kernel image is a routine that does some minimal amount of hardware setup and then decompresses the kernel contained within the kernel image and places it into high memory.
- If an initial RAM disk image is present, this routine moves it into memory and notes it for later use.
- The routine then calls the kernel and the kernel boot begins
- During the boot of the kernel, the initial-RAM disk (initrd) that was loaded into memory by the stage 2 boot loader is copied into RAM and mounted.

Kernel loader

- This initrd serves as a temporary root file system in RAM and allows the kernel to fully boot without having to mount any physical disks
- initrd is mainly designed to allow system startup to occur in two phases, where the kernel comes up with a minimum set of compiled-in drivers, and where additional modules are loaded from initrd.
- After the kernel is booted, the root file system is pivoted where the initrd root file system is unmounted and the real root file system is mounted.
- The initrd function provides a means of bootstrapping to gain access to the disk and mount the real root file system.

Kernel loader

- Let see
  - ls –l /boot
    - vmliuz-2.6.28-11-generic → compressed kernel image
    - initrd.img-2.6.28-11-generic → initial RAM disk
  - ls –l /boot/grub
    - stag1 -> 512 byte -> primary boot loader
    - e2fs_stage1_5 -> ex2, ex3 or ext4 file systems
    - fat_stage1_5 -> fat file systems
    - …
    - stage2

Init

- After the kernel is booted and initialized, the kernel starts the init (first user-space application).
- System V init process
  - Run levels concept
  - /etc/inittab
- Upstart init process
  - Event-driven init process
  - Ubuntu 6.10 and later
  - Fedora 9.0 and later

System V init

- Runlevel
  - is a number which indicates what "mode" you want to computer to boot into
  - 0 — Halt
  - 1 — Single-user mode
  - 2 — Not used (user-definable)
  - 3 — Full multi-user mode
  - 4 — Not used (user-definable)
  - 5 — Full multi-user mode (with an X-based login screen)
  - 6 — Reboot

System V init

- The default runlevel for a system to boot to and stop is configured in /etc/inittab. Like:
  - id:3:initdefault:
- The /etc/init.d directory contains the scripts executed by init at boot time and when the init state (or "runlevel") is changed
- These scripts are referenced by symbolic links in the /etc/rc*n*.d directories
- The names of the links all have the form S*mmscript* or K*mmscript* where *mm* is a two-digit number and *script* is the name of the script (this should be the same as the name of the actual script in /etc/init.d).
- When changing runlevels, init looks in the directory /etc/rc*n*.d for the scripts it should execute, where *n* is the runlevel that is being changed to, or S for the boot-up scripts.

System V init

- When init changes runlevel first the targets of the links whose names start with a K are executed, each with the single argument stop, followed by the scripts prefixed with an S, each with the single argument start.
- The two-digit number *mm* is used to determine the order in which to run the scripts: low-numbered links have their scripts run first.

System V init

- the chain of events for a SysV init boot is as follows:
  - The kernel looks in /sbin for init
  - init runs the /etc/rc.d/rc.sysinit script
  - rc.sysinit handles most of the boot loader's processes and then runs rc.serial (if it exists)
  - init runs all the scripts for the default runlevel
  - init runs /etc/rc.d/rc.local

Upstart init

- The Upstart init daemon is event-based and runs specified programs when something on the system changes
  - Instead of starting and stopping services only when the runlevel changes, Upstart can start and stop services upon receiving information that something on the system has changed.
- An *event* is a change in system state that init can be informed of.
  - the boot loader triggers the startup event
  - the system entering runlevel 2 triggers the runlevel 2 event
  - a filesystem being mounted triggers the path-mounted event
  - You can also trigger an event manually by using the initctl emit command.

Upstart init (jobs)

- A *job* is a series of instructions that init reads.
  - A *task* is a job that performs its work and returns to a waiting state when it is done
  - A *service* is a job that does not normally terminate by itself.
  - The /etc/event.d directory holds *job definition files* (files defining the jobs that the Upstart init daemon runs)
  - You can run and stop a job manually using the initctl start and stop commands, respectively.

Upstart init (task example)

- /etc/event.d/testtask

```
start on helloevent
script
        echo “hello world” > /home/test
        date >> /home/test
end script
```

- #initctl emit helloevent

Upstart init (service example)

- /etc/event.d/testtask

```
start on helloevent

respawn
exec  /bin/service1  > output1.txt
```

- #initctl emit helloevent