The Linux Boot Sequence

How does your kernel get loaded, what is the series of steps that occurs in order to make this happen, what partition to choose? These will be discussed in these slides.

The BIOS

BIOS stands for Basic Input Output System. This is the first system involved in the boot process. BIOS is stored in read-only memory (on a chip)(PROM)

The bios:

- detects hardware.
- Loads contents of NVRAM (CMOS) wherein it can figure out where to look for MBR.
- Finds the boot loader program (MBR) (will search in different devices i.e. floppy, disk, usb, network).
- Video

Master Boot Record

This is step 2 of the load sequence. The MBR is located on the first sector of the boot device.

On most linux devices the MBR would contain (or contain a reference to) the GRUB boot loader. If booting from something other than a hard disk, you might use a different bootloader, but they essentially all do the same thing.

We know that MBR also has partition table information.

The bootloader (GRUB)

- Grand Unified Bootloader
- Allows you to choose what kernel you would like to load.
- Usually has nice GUI with configurable menu and timeouts that you can perform kernel or OS selection.
- Once selected, GRUB loads kernel into RAM
- Also loads the initrd image.
- GRUB video

The kernel

[The kernel] initializes the devices (via their drivers), starts the swapper (it is a “kernel process”, called kswapd in modern Linux kernels), and mounts the root filesystem (/).(from man page)

Kernel then executes the /sbin/init program. (PID 1)

initrd (initial ram disk) is a temporary root fs used by kernel, until the real fs is mounted. Also has drivers in it.

Init

This program first looks at `/etc/inittab` file to decide linux run level. Sometimes it is also referred to as sysvinit. (System 5 Init)

Following are the available run levels

```
0 - halt
1 - Single user mode
2 - Multiuser, without NFS
3 - Full multiuser mode
```

```
4 - unused
5 - X11
6 - reboot
```

taken from here

More about Init

- `/etc/init.d` contains all the start-up scripts for every service at every run level.
- Init has worked well for years... but Ubuntu has moved away from it to `Upstart`. Ubuntu still retains many of the structure of the `Init` just described.
- One of the reasons: init scripts don't automatically have a mechanism to respawn if the service dies. So, for instance, if the cron daemon crashes for some reason, you would have to create some other tool to monitor and restart that process.
- i.e. You can still restart a service by doing `/etc/init.d/apache2 restart`

Upstart

To see how an event-driven system can improve on traditional init scripts, let's take the previous example of a system booted with an unplugged network cable. You could create an Upstart script that is triggered when a network cable is plugged in. That script could then restart the networking service for you. You could then configure any services that require a network connection to be triggered whenever the networking service starts successfully. Now when the system boots, you could just plug in the network cable and Upstart scripts would take care of the rest. [From DevOps Troubleshooting]

More Upstart

- See all upstart jobs `sudo initctl list`
- Start an upstart service `sudo service servicename start`
  - i.e. `sudo service mysql start`
- Can also stop, restart or get status of service.
