Boot Process

When you boot a Linux system, the following steps will be completed before you're greeted with your usual OS

1. BIOS

Basic Input Output System (BIOS) performs a Power On Self Test (POST) to ensure all required hardware is available and functional. If a problem is detected you will see an error message and you will need to attempt to fix the issue and reboot before the system can proceed to the next step in the boot process.

Once the BIOS POST check passes, the BIOS searches for the boolloader program using the MBR. There is a short delay before executing the bootloader where you have a chance to press a key (usually `F12`) to select the location for the BIOS to search for the MBR.

2. MBR

The Master Boot Record is located on the first sector of the bootable disk. On Linux you can see this sector by running `lsblk` , and for me this partition is labeled `nvme0n1p1` because I'm using an M.2 NVME SSD. This SSD is encrypted so the structure might slightly differ from a non-encrypted system. For a normal SSD, the sector would probably appear as `sda` , and for a HDD it would appear as `hda` .

```
lsblk

nvme0n1            259:0    0 931.5G  0 disk
├─nvme0n1p1        259:1    0  c512M  0 part  /boot/efi
├─nvme0n1p2        259:2    0   732M  0 part  /boot
└─nvme0n1p3        259:3    0 930.3G  0 part
  └─nvme0n1p3_crypt   253:0    0 930.3G  0 crypt
    ├─vgkubuntu-root   253:1    0 929.3G  0 lvm   /
    └─vgkubuntu-swap_1 253:2    0   976M  0 lvm   [SWAP]
```

Note that regardless of which type of storage device you're bootloader is on, you will be able to find the device under the `/dev/` directory. That means my bootloader is at `/dev/nvme0n1p1`

```
ls /dev/nvme0*

/dev/nvme0  /dev/nvme0n1  /dev/nvme0n1p1  /dev/nvme0n1p2  /dev/nvme0n1p3
```

The MBR will launch the bootloader which in my case is GRUB2. Your system might use GRUB, or possibly even LILO.

3. GRUB

GRand Unified Bootloader is responsible for loading the kernel for your system. If you want to try out different Linux kernels, see my notes on Linux kernel management On some systems GRUB will not appear by default, so you may need to modify the contents of `/etc/default/grub` to ensure your system will show the GRUB splash screen. To do this, you can run the `sudoedit /etc/default/grub` command and **read the header comment**, then proceed to make your changes in the configuration file. The link to my notes on kernel management will cover this process in more detail since it is required to switch Linux kernels, check it out for more detailed information.

After you make changes to GRUB or install new kernels, you will always need to run `sudo update-grub` in order to apply the changes to your system.

Each valid entry for a kernel in grub will contain full system paths to two files - `vmlinuz` and `initrd`. The `z` in `vminuz` stands for `zip`, since this is the compressed version of your kernel. The system will decompress the kernel and boot into it, and then used `initrd` to initialize required software.

For more information on GRUB, check out the official GRUB Manual - Simple Configuration documentation. This should provide you with all or most of the information you need, but you can feel free to check out the more advanced sections of the guide if needed.

4. Kernel

Once the kernel is decompressed and the file system is mounted, the kernel will execute the `/sbin/init` program, which performs software initialization up to the `runlevel` specified for your system within your local configurations. This can be modified but each distribution may handle this differently, so you should check the relevant documentation on how to do this if you are interested.

5. Init

This section has changed a good bit over the years and I noticed some differences in guides I found online, so this information is just what I collected after a few minutes of checking manpages and searching around my system.

Some useful manpages to checkout -

```
man inittab
man init
man runlevel
man utmp
```

The process is still the same for booting. This part of the boot process will initialize the software required to run the environment specified up to the current `runlevel` setting. Each runlevel will start different groups of software to support different environment features.

6. Runlevel

If you aren't sure what your `runlevel` setting is, just run the command to find out -

```
runlevel

N 5
```

Our current runlevel is set to 5. When we run `man runlevel`, we can see a table describing what the different runlevels mean.

```
OVERVIEW

    "Runlevels" are an obsolete way to start and stop groups of services used in SysV
    init. systemd provides a compatibility layer that maps runlevels to targets, and
    associated binaries like runlevel. Nevertheless, only one runlevel can be "active"
    at a given time, while systemd can activate multiple targets concurrently, so the
    mapping to runlevels is confusing and only approximate. Runlevels should not be used
    in new code, and are mostly useful as a shorthand way to refer the matching systemd
    targets in kernel boot parameters.


    Table 1. Mapping between runlevels and systemd targets
```

Runlevel	Target
0	poweroff.target
1	rescue.target
2, 3, 4	multi-user.target
5	graphical.target
6	reboot.target

But what software is initialized during boot? Check your `/etc/` directory for subdirectories named `/etc/rc0.d`, `/etc/rc1.d`, etc. There is one directory named `/etc/rcS.d` which is always initialized during

Startup.

```
ls /etc/rc*

rc0.d/ rc1.d/ rc2.d/ rc3.d/ rc4.d/ rc5.d/ rc6.d/ rcS.d/
```

Some binaries in these subdirectories start with `K` and others start with `S`. This just means that the `K` binaries are ran when the system is shutdown, and the `S` binaries are ran when the system is started.

```
ls /etc/rc5.d/

K01gdomap          S01cups        S01lvm2-lvmpolld          S01sddm
S01acpid          S01cups-browsed  S01nginx                S01spice-vdagent
S01anacron         S01dbus        S01osspd                S01sysstat
S01apport          S01gdm3        S01plymouth              S01tlp
S01avahi-daemon     S01grub-common  S01postfix              S01trousers
S01binfmt-support   S01haveged      S01pulseaudio-enable-autospawn S01ubuntu-fan
S01bluetooth        S01hddtemp      S01rsync                S01unattended-upgrades
S01console-setup.sh S01irqbalance   S01rsyslog              S01uuidd
S01cron            S01kerneloops   S01saned                S01whoopsie
```

You might noticed that *all* of these binaries are just named symlinks that point to binaries in different directories. This is just one example of how symlinks can be used to organize processes and files on your system.

```
ls /etc/rc5.d/ -l

total 0
lrwxrwxrwx 1 root root 16 Dec  6 09:27 K01gdomap -> ../init.d/gdomap
lrwxrwxrwx 1 root root 15 Dec  6 09:27 S01acpid -> ../init.d/acpid
lrwxrwxrwx 1 root root 17 Dec  6 09:27 S01anacron -> ../init.d/anacron
lrwxrwxrwx 1 root root 16 Dec  6 09:27 S01apport -> ../init.d/apport
lrwxrwxrwx 1 root root 22 Dec  6 09:27 S01avahi-daemon -> ../init.d/avah
...
```

Resources and links

tecmint - Linux Boot Process

thegeekstuff - Linux Boot Process

freecodecamp - Linux Boot Process

Revision #1
Created 25 March 2022 13:26:27 by Shaun Reed
Updated 25 March 2022 22:10:11 by Shaun Reed