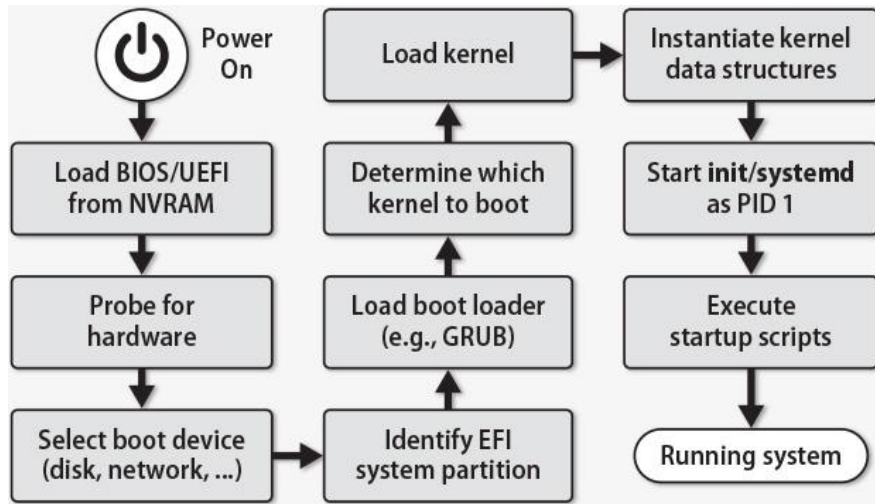


The Boot Process

BIOS (basic input output system)

1. Firmware that initializes hardware on the MB.
2. Makes system ready to load and run OS.

General process overview



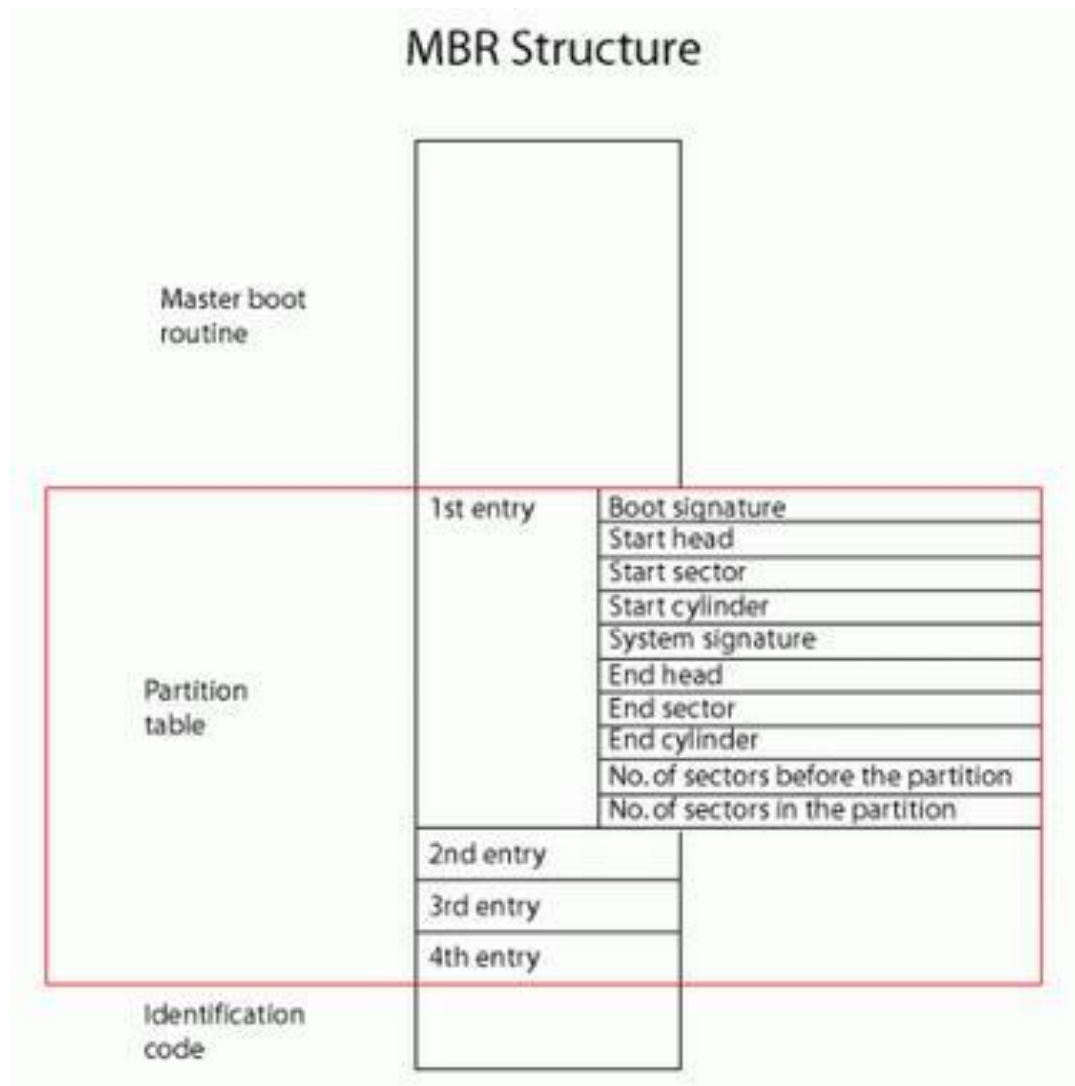
(Image src: our textbook)

What happens in legacy systems (BIOS)

1. Each device starts with MBR. MBR (512B) was introduced in 1983 with PC DOS 2.0, lives at cylinder 0, head 0, sector 1. Max addressable space is about 2TB. How large are HDs today? Being replaced by GPT. End signature is AA55 or 55AA.

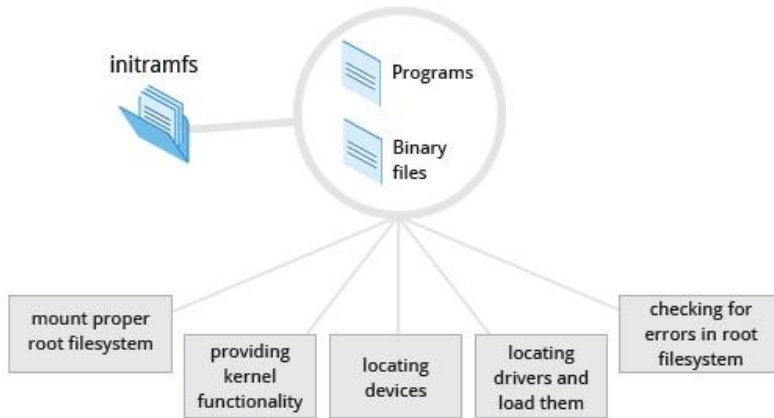
BIOS checks hardware and looks at MBR. *MBR contains 1st stage boot block (boot loader) and partition table (64B), each partition entry is 16 B long*. Partition table contains info about “active” disk partition. Space too small for anything else. MBR is OS-agnostic.

- a. Looks for a bootable volumes in the table (HD, CD-ROM, USB, etc.).
 - i. Boot order can be changed in firmware UI.
 - ii. Device must be set to bootable, or “active”.
2. BIOS loads 2nd stage boot code from the beginning of “active” partition into memory. Active partition starts at 64th disk block. (Volume boot record scheme).
 3. In case of GRUB - 2nd stage boot code lives in the space between MBR and the first disk partition. Filesystem driver lives in that space and is used by GRUB. This stages boot loader knows about OSs and FSs and supports several of them.



4. Boot code determines which kernel to boot, locates kernel on the disk and loads into memory. Boot loader loads both kernel and initial RAM-based FS to memory.

The *initramfs* filesystem image performs all actions needed to mount the proper root filesystem (providing kernel functionality for the needed filesystem and device drivers for mass storage controllers with a facility called *udev* (for user device), which is responsible for figuring out which devices are present, locating and loading device drivers that they need. After the root filesystem has been found, it is checked for errors and mounted.



Img src: google.com

5. When control is passed to the Linux kernel, kernel data structures are initialized, and systemd daemon starts as PID 1.
6. Startup shell scripts are executed, filesystems checked/mounted, system daemons started.
7. System is up!!!!!!

■ Intel was working at the Itanium systems in 1990s and noticed that BIOS can be a limiting factor. Their work became foundation for the UEFI.

UEFI (Unified Extensible Firmware Interface)

1. Replaced BIOS in modern systems, BUT may be an issue for single-OS Ubuntu installations.
2. Can fall back to BIOS implementation, if system does not support UEFI.
3. Virtualized environments often prefer BIOS.
4. Modern disk partitioning scheme is different:
 - a. uses GPT (GUID PT – Globally Unique Identifier Partition Table, not MBR).
 - b. Compatible with FAT (File Allocation Table) filesystems.
 - c. Combined together produce ESP (EFI System Partition) can be written, read, and mounted by any OS (like FAT).
 - d. No boot blocks are required.
 - e. Loader-less bootstrap - boot target can be a kernel configured for direct UEFI loading. (most systems use loader to be backwards compatible with older systems)
 - f. EFI has API to access system's hardware. Resembles a miniature OS in its own. Can modify boot menu entries from user space on a running system:

`efibootmgr -v`

- g. Allows UEFI-level add-on device drivers written in processor independent language and stored in UEFI.

(Most people do not care which one they use!!!!)

How to get to your computer UEFI settings on a win10 machine:

Start, Settings, Updates and Security, Restart now. This will bring up a blue color screen (this is not the BSOD), Troubleshoot, Advanced Options, UEFI Firmware settings. Restart to change UEFI settings. Computer will boot into boot manager UI. Setup under Diagnostics.

System Hardening Tip:

- Protect BIOS of the host machine with a password, so the end-user cannot change and override the security settings of the BIOS.
- Each computer manufacturer has a different set of keys to enter the BIOS mode, and they are often posted online on manufacturer's website!
- Do not forget your BIOS password. It is a headache to crack it! Occasionally none of the manufacturer's keys will work, and you will end up with unusable computer!
- Disable the booting from external media devices (USB/CD/DVD). If you omit to change this setting, anyone can use a USB stick with a bootable OS and can access your OS data.

Boot Loaders

- Small program that identifies and loads OS kernel into memory.
- Pass arguments to the kernel (eg.: boot in single user mode).
- Can allow to select between OSs.
- loadlin was an early loader; LILO and ELILO came later, still in use by some.
- Most distros use GRUB2.

GRUB 2 (Grand Unified Boot Loader)

1. Default for most Linux distros, Ubuntu adopted it since version 9.10.
2. Developed by GNU Project.
Info about copyleft: www.gnu.org/licenses
GNU GPL latest license 3.0: www.gnu.org/licenses/gpl-3.0.html
3. Legacy GRUB and GRUB 2.
4. Allows to select bootable kernel and mode.
5. May present a boot-time UI in multiple boot environments.

6. Lives in a text file !!!!! in `/boot/grub/grub.cfg`
7. `grub.cfg` file will be auto generated by `grub-mkconfig` or a wrapper `update-grub`.
8. `/etc/default/grub` allows to change GRUB variables. Run `update-grub` after making any changes to system, or directly to `/boot/grub/grub.cfg`
9. Backup custom config file, as major updates overwrite it automatically.
10. Supports command-line interface (type `c` at GRUB boot screen)
11. Can boot OSs not listed in config file from command line.

How to boot into GRUB?

BIOS: pressing and holding the Shift key after power-on will bring up the GNU GRUB menu. (If you see the Ubuntu logo, you've missed the point where you can enter the GRUB menu.)

UEFI: press (perhaps several times) the Escape key at power-on to get into GRUB menu.

3 init systems:

1. sysVinit and BSD init – only allows sequential startup of services.
 - devs were adding services, but it was still becoming obsolete,
 - to maintain compatibility with newer systems - enter the Upstart.
2. Upstart (Canonical, 2006) – allows concurrent service start-up (unless dependent on other services), supports multicore processing.
 - essential subsystems have deep hooks in the kernel, a large number of management scripts, software package dependencies, so replacing one component with another is not a trivial task.
 - Users are pretty much stuck with what devs implement.
3. systemd (2011) allows concurrent service start-up (unless dependent on other services), supports multicore processing.
 - Since 2015 Ubuntu uses `systemd`.

Sysadmins can control every aspect of behavior *after* boot by configuring and ordering scripts!!!!

Daemons

1. Kernel background processes that start autonomously after booting.
2. Part of kernel implementation.
3. Not configurable, do not require administrative attention.
4. Have a **d** at the end of the name: `sshd`, `journald`, etc.
5. Have nothing to do with directories that end with “.d”.
6. Have low PIDs.
7. Have brackets around their [names] in `ps -ef`.

8. **systemd** is a daemon with PID 1.
9. What entity has a PID of 0?

Traditional *init*

1. Ensures all necessary services and daemons are running, according to their mode:
 - a. Single user (min set of filesystems, no svcs, root shell on console)
 - b. Multiuser mode (needed filesystem, window system, graphical login, nw svcs)
 - c. Server mode (as above, but no GUIs)

Simplified Boot Process

1. The system powers up. The BIOS does minimal hardware initialization and hands over control to the boot loader.
2. The boot loader calls the kernel.
3. The kernel loads an initial RAM disk that loads the system drives and then looks for the root file system.
4. Once the kernel is set up, it begins the **systemd** initialization system.
5. **systemd** takes over and continues to mount the host's file systems and start services.

Systemd

- Adopted by Fedora in 2011
- Adopted by RHEL 7 and SUSE
- Replaced Upstart in Ubuntu 16.04
- Written by Ken Sievers and Lennart Poettering.

System features

1. Advanced process management.
2. A collection of programs, daemons, libraries and kernel components.
3. Aggressive parallelization capabilities.
 - a. Starts faster than system with init. **/sbin/init** now points to **/lib/systemd/systemd**
4. Dependency-based service control logic.
5. Backwards compatible with sysVinit.
6. Configuration files live in **/etc/systemd/system**.
 - a. Have symlinks to files with init scripts located in **/lib/systemd/system**.
 - b. To start a service at boot time it must be linked to **/etc/systemd/system/**
The **systemctl** command does this for you when you enable a new service.
7. Defines a dependency model.
8. Manages processes in parallel.
9. Manages
 - a. network connections (**networkd**),
 - b. kernel log entries (**journald**),

- c. logins (**logind**).
- d. 12 unit types:
 - a. service
 - b. socket (IPC socket)
 - c. device (kernel device names in sysfs and udev)
 - d. target (group of units)
 - e. mount (filesystem mount point)
 - f. automount (filesystem automount point)
 - g. swap (swap file or partition)
 - h. path (file or directory)
 - i. scope (external processes not started by system)
 - j. slice (a management unit of processes)
 - k. timer (system timer)
 - l. snapshot (system saved state)
- e. Unit behavior is in a unit config file.

10. Too complex? Complexity = weakness???

11. You can check how long each service takes to start (and *blame* the service that takes the longest for the long time it takes to boot 😊)

systemd-analyze blame

- **systemd-analyze** is used to
 - to determine system boot-up performance statistics,
 - to retrieve state and tracing info from the system and service mgr,
 - to verify the correctness of unit files.
 - to access special functions useful for advanced system manager debugging.

systemd-analyze time

- time spent in the kernel before userspace has been reached,
- time spent in the initial RAM disk (initrd) before reaching system userspace. (see end of file for brief explanation of initrd)
- time normal system userspace took to initialize.

Printed times indicate time passed up to the point where all system services were spawned, not until they fully finished initialization, or the disk is in idle state.

```
$ systemd-analyze time
Startup finished in 2.331s (kernel) + 18.236s (initrd) + 42.838s (userspace) = 1min
3.406s
multi-user.target reached after 42.811s in userspace
```

systemd tools

systemctl – various info about status of system;
journalctl – centralized logging tool;

systemctl

Command for investigating the status of **systemd** and changing its configs.

systemctl - without args shows all loaded and active units that have an init script

systemctl list-units --type=service - list only services

systemctl list-unit-files --type=service - list their files

systemctl reboot - yes, there is more than one way to get the same things done in Linux

Subcommand	Function
list-unit-files [<i>pattern</i>]	Shows installed units; optionally matching <i>pattern</i>
enable <i>unit</i>	Enables <i>unit</i> to activate at boot
disable <i>unit</i>	Prevents <i>unit</i> from activating at boot
isolate <i>target</i>	Changes operating mode to <i>target</i>
start <i>unit</i>	Activates <i>unit</i> immediately
stop <i>unit</i>	Deactivates <i>unit</i> immediately
restart <i>unit</i>	Restarts (or starts, if not running) <i>unit</i> immediately
status <i>unit</i>	Shows <i>unit</i> 's status and recent log entries
kill <i>pattern</i>	Sends a signal to units matching <i>pattern</i>
reboot	Reboots the computer
daemon-reload	Reloads unit files and systemd configuration

Service-Related Command Examples:

Command	Description
<code>systemctl start servicename</code>	Start a service
<code>systemctl stop servicename</code>	Stop a service
<code>systemctl restart servicename</code>	Restart a service
<code>systemctl reload servicename</code>	Reload a service (reloads config files, not restarts the service)

systemctl status <i>servicename</i>	Show service status
systemctl condrestart <i>servicename</i>	Restart a service if it is already running
systemctl enable <i>servicename</i>	Enable a service at startup
systemctl disable <i>servicename</i>	Disable at startup (remove from boot list)
systemctl halt	Halt the system
systemctl reboot	Reboot the system

Troubleshooting and Logging systemd

1. implements universal logging framework
2. Includes ALL service and kernel messages from [boot to shutdown](#).
3. Journal managed by **journald** daemon
4. **journalctl** by default only shows all messages from current boot.
 - a. Can make it retain all messages from all boots. Edit **/etc/systemd/journald.conf**:

[Journal]

Storage=persistent

- a. After making changes restart the journald:

sudo systemctl restart systemd-journald

5. Can control the size of the journal.
 - a. View size of file:

journalctl --disk-usage systemd-journald

6. Can also query journalctl with the following options:
 - a. -u show log for some unit:

journalctl -u ssh

- b. -k show kernel messages only

journalctl -k

Booting Problems

3 approaches:

1. Do not waste time debugging, restore and redo.
 - a. You should have made frequent backups!
 - b. Sometimes this approach saves time.

2. Run a shell, if you can, and debug interactively.
 - a. Should have backups!
 - b. Single-user or rescue mode (rescue.target),
 - i. pass **-s** to boot loader
 - ii. no nw,
 - iii. root filesystem mounted as /usr, read only
 - iv. bare min of svcs, daemons and processes
 - v. cannot reset forgotten passwords
 - vi. have to mount partitions manually

 - c. systemd provides emergency mode if cannot enter recovery mode:
systemctl emergency

 - d. Usually no networking

3. Boot another image, mount non-booting system, debug.
 - a. Should have backups!

Reboot and Shutdown

Should reboot after every configuration change.

Be nice! Abruptly shutting down the system is not nice.

- Enter at the command line:
 - a. halt - logs shutdown, stops services, flushes cached data to disk, halts kernel.
 halt -p - powers down the system.
 - b. reboot – reboots system.
 - c. shutdown - usually scheduled, gives time to save files and shut down.
 shutdown - analogous to halt, power-off and reboot.

shutdown [options] [time] [wall]

where: options – ask the man, time – usually “now”, wall – message that users see

- Systemctl allows to halt, shutdown and reboot system as well!!!

systemctl halt

systemctl shutdown

systemctl reboot

Ubuntu Runlevels (??)

Run levels are operational levels that determine the state of the system with respect to what services are available on the system when it is running. They used to be used by the *init* in the past that used config file in `/etc/init/rc-sysinit.conf`. Now Ubuntu uses *systemd* instead of *init* and hence the concept of *runlevels* is replaced by the term *targets*.

runlevels and targets

- 0 – poweroff.target (and **runlevel0.target** is a symbolic link to **poweroff.target**)
- 1 – rescue.target (runlevel1.target)
- 2,3,4 – multi-user.target (runlevel3.target)
- 5 – graphical.target (runlevel5.target)
- 6 – reboot.target (runlevel6.target)
- Emergency – emergency.target

When system boots, by default *systemd* activates the default.target unit. Its main work is to activate services and other units via their dependencies.

To view the default target, type the command below.

```
# systemctl get-default  
  
graphical.target
```

To change the “runlevels”, use command:

```
sudo systemctl isolate multi-user.target
```

In manual:

```
isolate NAME
```

Start the unit specified on the command line and its dependencies and stop all others. If a unit name with no extension is given, an extension of ".target" will be assumed.

This is similar to changing the runlevel in a traditional init system. The isolate command will immediately stop processes that are not enabled in the new unit, possibly including the graphical environment or terminal you are currently using

To make default, use this:

```
sudo systemctl enable multi-user.target  
sudo systemctl set-default multi-user.target
```

Targets

- Targets help systems to determine which unit files are necessary to produce a certain system state.
- Targets are represented by target units that have extension `.target`.
- They group together other systemd units through a chain of dependencies.
 - a. `graphical.target` indicates when the system's graphical session is ready. Units that are required to start to achieve the state have in their config file:
WantedBy=graphical.target or *RequiredBy= graphical.target*
 - b. to make themselves available at the correct time units that depend on `graphical.target` can include in their config file:
Wants=, Requires=, or After=
- A target can have a corresponding directory with syntax *target_name.target.wants* (e.g. `graphical.target.wants`) in **`/etc/systemd/system`**.
- When you enable a service (using `systemctl enable`), symlinks to the service are created inside the *target_name.target.wants* directory for each target listed in that service's *WantedBy= configuration*. This is actually how the *WantedBy=* option is implemented.

Old commands still work:

To check your runlevel use ***runlevel*** command

```
# runlevel
N 5
```

To edit permanently:

Edit option in `/etc/defaults/grub`:

```
GRUB_CMDLINE_LINUX="5"
```

And then run `update-grub`

Discovery Exercises:

1. Figure out how to display GRUB on startup for a single-OS system. In multiboot system it will be shown automatically. Research which grub variables you need to change to do this.
2. Change background image for GRUB menu. There is more than one way to do this. Which way did you use?

3. Read the GNU GPL license, version 3.0. www.gnu.org/licenses/gpl-3.0.html
4. Research the differences between *more* and *less*, and discover that *less* is actually more, and *more* is, well, less.
5. In one of the sections above we mention the *initrd* (initial RAM disk). According to Wikipedia, “*initrd* is a scheme for loading a temporary root file system into memory, which may be used as part of the Linux startup process. *initrd* and *initramfs* refer to two different methods of achieving this. Both are commonly used to make preparations before the real root file system can be mounted.” (Wikipedia). Do a little of research and find out more about *initrd*.
6. Command **ps** (which stands for process status) shows a system snapshot of processes. It takes a number of different options to format the output the way user wants it. **ps** can take 3 different kinds of options:
 - a. UNIX options, which may be grouped and must be preceded by a dash.
 - b. BSD options, which may be grouped and must not be used with a dash.
 - c. GNU long options, which are preceded by two dashes.Check the man to see what options would be useful. How do you list all processes using UNIX options?