Booting and System Management Daemons

Unix – Chapter 2

Bjarte Wang-Kileng

HVL

January 15, 2025

Western Norway University of Applied Sciences

Boot process and configuration

Select boot device:

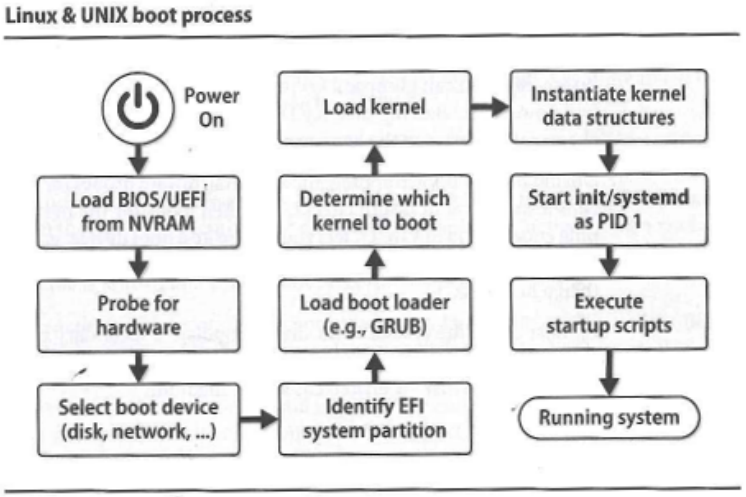
- Enable disk as boot device.
- Choose boot device.
- Specify boot device priorities.

Determine which kernel to boot:

- Select what kernel to load.
- Specify default kernel.

Execute startup scripts:

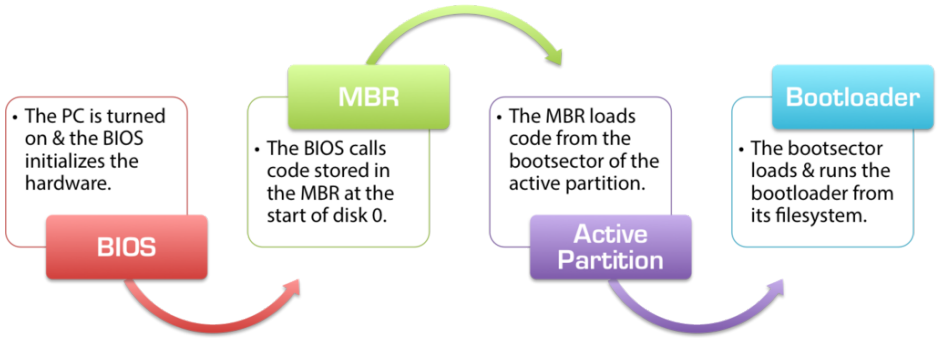
- Specify what init scripts to run.
- Run order or dependencies between init scripts.



BIOS and UEFI

- The acronyms:
  - BIOS = *Basic Input/Output System*
  - UEFI = *Unified Extensible Firmware Interface*
- BIOS and UEFI are firmware that is run when the computer starts.
  - CPU is hardwired to run the firmware.
- Usually stored in EEPROM on the motherboard.
- Typically includes a GUI to e.g. select the boot disk, or specify a disk priority for boot.
- Initializes the computer and runs the next stage of the bootstrapping code (e.g. GRUB).
- UEFI has replaced BIOS on modern PCs.
- UEFI firmwares usually implement some kind of BIOS compatibility.
  - UEFI can boot from a MBR (MBR on next slide).

BIOS



- Boot disk must start with MBR (*Master Boot Record*) that contains:
  - First-stage boot loader, *boot block*.
  - Disk partition table.
- The second-stage boot loader, e.g. GRUB:
  - Includes the necessary file system drivers to load the OS.
  - Loads the OS.
- The second-stage boot loader is read either from:
  - The active partition, or
  - the dead zone before the first partition (64 disk blocks),
- All BIOS configurations are done inside the firmware.
  - BIOS has no concept of OS or file systems.

UEFI acronyms

- UEFI = Unified Extensible Firmware Interface
- GUID = Globally Unique Identifier
- GPT = GUID Partition Table
- ESP = EFI System Partition
- FAT = File Allocation Table (MS_DOS file system)
- NVRAM = Non-Volatile RAM

UEFI

- UEFI firmware can handle FAT partitions.
- ESP is a FAT partition that stores the target application, e.g. GRUB.
  - Usually mounted as "/boot/efi".
- UEFI parameters are stored in NVRAM, accessible by UEFI and OS.
  - Accessible as "/sys/firmware/efi/efivars", or "/sys/firmware/efi/vars".
- GPT identifies the ESP and the target application.
  - Stored as UEFI parameters.
  - Default target application is "/efi/boot/bootx64.efi".
- UEFI configurations can be modified in user space using e.g. the command **efibootmgr** or through the "/sys/firmware/" file system.

GRUB

- Default boot loader for Linux.
- Can boot multiple operating systems.
- GRUB code location:
  - UEFI – ESP
    - E.g. “EFI/almalinux/grubx64.efi”.
  - BIOS – Usually in the dead zone before the first partition.
- GRUB can read its configurations from a file.
  - Usually found as **grub.cfg** in the ESP or “/boot/grub2” partition.

Modify GRUB

- Modifications of **grub.cfg** do not survive updates.
  - Configuration file "/etc/default/grub".
  - Command grub2-editenv – modify GRUB parameters
- Transfer the changes to the **grub.cfg** file with the command:

```
grub2-mkconfig -o <path-to>/grub.cfg
```

  - Before using *grub2-mkconfig*, package "os-prober" must be installed.
- To specify a default kernel, see e.g. GRUB 2 - Fedora Project Wiki.

GRUB command line

- Lets us modify the boot entries at boot time.
  - Changes are not saved.
- Modify a GRUB menu entry:
  - Locate the GRUB menu entry.
  - Press the “e” key.
  - Modify the kernel line.
  - Use “F-10” to boot from the modified GRUB stanza.
- See the book or Internet for details.

systemd

- Started by the kernel as process with pid equal to 1.
- Starts and stops system services and daemons.
- The systemd process is the top process in the process hierarchy.

systemd and init

- Most major distros uses systemd to initialize the system.
  - RedHat did use the init system of System V before RedHat 7.
- Startup sequence:
  - System V init – the scripts are run in a set sequence.
  - systemd – dependencies determine the startup sequence.
- Parallelism:
  - System V init – the scripts are run in a strict sequence, one at a time.
  - systemd – can run the startup scripts in parallel.
- systemd is a more complicated system.

systemd units and unit files

- A unit is an entity that is managed by systemd:
  - Can be a service started by a script.
  - Can be a target (more later).
  - And much more.
- A systemd unit is described by a unit file:
  - Path of executable.
  - Specify how to start and stop.
  - Specify dependencies.
- Unit files are read from:
  - “/etc/systemd/system” – Highest priority.
  - “/usr/local/lib/systemd/system”
  - “/usr/lib/systemd/system”
  - “/lib/systemd/system”
- Unit-file suffix specify unit type:
  - E.g. “.service”, “.target”, “.timer”, “.socket”, “.mount”, ...

Find systemd units

- List all loaded units:

```
systemctl list-units
```

- List loaded services only:

```
systemctl list-units --type=service
```

- List all service unit files:

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

Enable/disable/mask

- An enabled service will be started at “boot” by systemd.

```
systemctl enable mariadb.service
```

- A disabled service will not be started at boot by systemd.
  - Can still be started manually.
  - Can also be started due to a depending enabled service.

```
systemctl disable mariadb.service
```

- A masked service can not be started.
  - Unit file is linked to “/dev/null”.

```
systemctl mask mariadb.service
```

Enabled unit and boot

An enabled unit is started if any of the *WantedBy* units are started.

Usually, this means that the unit is started at boot.

Start/stop/status

- A service can be started.

```
systemctl start mariadb.service
```

- A service can be stopped.

```
systemctl stop mariadb.service
```

- We can check the status of a service.

```
systemctl status -l mariadb.service
```

Enablement state

- States are e.g. *enabled*, *disabled* or *static*.
- Only unit files with an `Install` section can be *enabled* or *disabled*.
- The **list-unit-files** sub command list service and enablement state.
  - See the man page for all the enablement states.
- Services with state *static* have no `Install` section.
  - Can only be started by hand, or if named as a dependency by another service.

Targets

- Targets are used to gather units.
  - Action on target will act on all units of the target.
- List loaded targets:

```
systemctl list-units --type=target
```

- List all targets:

```
systemctl list-unit-files --type=target
```

- Some targets have special meaning.
  - Run automatically by systemd at specific events.

Some special targets

ctrl-alt-del.target: Run when Control+Alt+Del is pressed.

default.target: Default unit to start at boot.

emergency.target: Run if boot fails, e.g. due to a failing local disk mount.

- Play with this mode **before** you need it.

graphical.target: Sets up the graphical login screen.

multi-user.target: Sets up a multi-user system.

reboot.target: Shutdown and reboot the system.

shutdown.target: Shutdown the system.

rescue.target: Pulls in the base system for administrative purposes.

Using targets

- The sub command **isolate** changes the current mode:

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

  - Activate the target with its dependencies, and deactivate all others.

- Show the default target:

```
systemctl get-default
```

- Set the default target:

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

- Can also specify a target on the kernel boot line
  - Emergency mode: `systemd.unit=emergency.target`
    - Short form: `emergency`
  - Rescue mode: `systemd.unit=rescue.target`.
    - Short form: `rescue`

Unit dependencies

- Dependencies are specified in the unit files.
- Keywords are e.g. *Wants*, *Requires*, *Requisite*, *BindsTo*, *PartOf*, *Conflicts*
  - Does not imply any sequence for processing.
  - See the book page 51 and manual pages for their meaning.
- *Wants* are a weaker form of *Requires*.
  - Success of unit does not depend on sucess of *Wants*.
  - Success of unit does does depend on sucess of *Requires*.
- Keywords to serialize processing are e.g. *After*, *Before*.
  - If no serializing keyword, systemd will try to run processes in parallel.
- Systemd assumes a default set of dependencies for units.
  - Turn off assumption with `DefaultDependencies=false`

Unit file

- New unit files can be created in an text editor.
- The systemctl sub command **edit** lets us modify an existing unit-file.

```
systemctl edit mariadb.service
```

  - Creates an override in "/etc/systemd/system/mariadb.service.d/".
- Do not modify existing unit-files.
  - Use an override, e.g. "/etc/systemd/system/mariadb.service.d/override.conf".
- Using unit-file overrides:
  - The original and the override will be merged by systemd at use.
  - If collisions, the override has higher priority.
- Install section can not be modified by an override.

Show unit properites

- List unit, with overrides:

```
systemctl cat mariadb.service
```

- List specfic property of unit:

```
systemctl show -p After mariadb.service
```

Some comments on logging

- Systemd has its own logging system, manged by the *journald* daemon.
- Systemd log messages are stored in the "/run" directory.
  - Typically, the rsyslog daemon will process also the systemd logs.
- Can be accessed with the command **journalctl**.
  - Show log of the Bluetooth unit:

```
journalctl -u bluetooth.service
```

- System logging will be covered later.