Day 5-Linux Boot Process and Run Levels

Linux Boot Process

The **boot process** is the sequence of steps from the moment a computer is powered on until the operating system is fully loaded and the user interface becomes operational.

Power-On Self-Test (POST)

- Performed by the system’s **BIOS** or **UEFI firmware** immediately after the system is powered on.
- Initializes hardware components such as the screen, keyboard, and storage devices.
- Tests the system memory (RAM) for faults.

CMOS/RTC (Real-Time Clock)

- Stores persistent system settings like **date**, **time**, and **hardware configuration**.
- This data is retained using a small battery, even when the system is powered off.

Bootloader Execution

- After POST, the **bootloader** is loaded from the **boot sector** (in BIOS systems) or **EFI System Partition** (in UEFI systems) of the primary storage device (SSD, HDD).
- Common bootloaders include:
  - `GRUB` (GNU GRUB, which is widely used in Linux)
  - `ISOLINUX` (used in bootable ISO images)
  - `U-Boot` (commonly used in embedded systems)

Loading the Kernel and Initial RAM Disk (initrd/initramfs)

- The kernel uses the `udev` **system** to detect available hardware, identify which drivers are required, and load them accordingly.
- Once the **real root filesystem** is located, it is checked for errors using tools like `fsck`, and then it is **mounted** as the new root.
- After that, control is handed over to the system's init process, which begins initializing user-space processes.

Init Process

Once the kernel is set up and the **root filesystem is mounted**, the kernel executes `/sbin/init`, which becomes the **first user-space process**. This process:

- Starts all other user-space processes (excluding kernel threads)
- Becomes the **ancestor of all other processes**
- Keeps the system running and handles **graceful shutdown**

Traditionally, this `init` process ran **sequentially**, processing startup scripts one after another. While functional, this made it **slower** and **less efficient** on modern multi-core systems.

Systemd

To overcome the limitations of `init`, most modern Linux distributions have replaced the traditional `init` with `systemd`.

- On modern systems, `/sbin/init` is typically a symbolic link to `/lib/systemd/systemd`, meaning **systemd has taken over as the init system**.
- Unlike the traditional init system, `systemd`:
  - Uses **unit configuration files** instead of complex shell scripts
  - Supports **parallel startup**, improving boot times
  - Tracks dependencies between services

    - Defines **clear states and conditions** for service initialization
- To check the init system, run the command:
    - `ls -l /sbin/init`

Run Levels

In traditional **SysVinit-based systems**, **runlevels** define the system's operating state (e.g., shutdown, single-user mode, graphical interface). In `systemd`**-based systems**, runlevels are replaced by **targets**, which offer more flexibility and better dependency management.

Common Runlevels and Their Systemd Equivalents

	Runlevel	Description	systemd Target
1	0	Halt / Shutdown	`poweroff.target`
2	1	Single-user mode	`rescue.target`
3	3	Multi-user, CLI only	`multi-user.target`
4	5	Multi-user + GUI (graphical)	`graphical.target`
5	6	Reboot	`reboot.target`

Commands and Examples

- To check current runlevel (SysV systems or legacy support) run:
    - `runlevel`
- To check current default systemd target run:
    - `systemctl get-default`
- To view the symbolic link for the default target run:
    - `ls -ltr /etc/systemd/system/default.target`
- To change the default target to CLI run:
    - `sudo systemctl set-default multi-user.target`
- To change the default target to GUI run:
    - `sudo systemctl set-default graphical.target`