4.1 UEFI Boot

Phase	Process
UEFI	The following steps take place during the UEFI boot process: **1.** Power is supplied to the processor. The processor is hard-coded to look at a special memory address for code to execute. **2.** This memory address contains a pointer or jump program that instructs the processor where to find the UEFI program. (The mount point for the EFI system partition is usually **/boot/efi**, where its content is accessible after Linux is booted.) **3.** The processor loads the UEFI program. **4.** UEFI runs the power-on self-test (POST). If the POST is successful, UEFI identifies other system devices. It uses the CMOS system clock and information supplied by the devices themselves to identify and configure hardware devices. Plug and Play devices are allocated system resources. The system typically displays information about the keyboard, mouse, and IDE drives in the system. Following this summary, information about devices and system resources is displayed. **5.** UEFI reads the GUID partition table, which is located in the blocks immediately after block 0. The GUID partition table defines the layout of the partition table on the storage device. **6.** Using this information, the UEFI boot loader locates the ESP, which contains the boot loader files or kernel images for all operating systems that are installed on other partitions on the device. ESP also contains device driver files for hardware devices on the computer that are used by the firmware at boot, system utility programs to be run before the operating system is booted, and data files, including error logs. To boot Linux, you would use a UEFI-aware version of the GRUB bootloader and install its boot file (grub.efi) in the EFI system partition.
Boot loader	During the boot loader stage, UEFI gives control to the boot loader program. The following steps take place: **1.** UEFI loads the boot loader code. **2.** When the boot loader is in RAM and executing, a splash screen is commonly displayed, and an optional initial RAM disk (e.g., initrd or initramfs image) is loaded into memory. The initramfs image is used with new distributions. Initramfs: • Is a custom version of the init program, containing all the drivers and tools needed at boot. • Is created by mkinitrd. Mkinitrd uses dracut to reduce boot times by using special tools and enabling udev to create device nodes for system hardware. • Has root permissions that can be used to access the actual /root file system regardless of whether it exists on the local computer or an external device. Without the permissions, the computer could not access the file systems and read information that exists only on those file systems. • Is used to mount the file system and load the kernel into RAM. **3.** With the images ready, the boot loader invokes the kernel image.
OS Kernel	During this stage, the Linux kernel takes over. The kernel: **1.** Resides in the /EFI directory. **2.** Initializes the hardware on the system. **3.** Locates and loads the initrd script to access the linuxrc program, which configures the operating system. **4.** Dismounts and erases the RAM disk image. On older distributions, this is the initrd image. On newer distributions, this is the initramfs image. **5.** Looks for new hardware and loads the drivers. **6.** Mounts the root partition. **7.** Loads and executes either the init (Initial) process (for older distributions) or the systemd process (for newer distributions). These processes then launch all other processes (either directly or indirectly) to finish booting the system. The init (Initial) or systemd processes are always assigned a process ID of 1 because they are always the first processes to run on the system.

4.1 BIOS Boot

Phase	Process
BIOS	In the BIOS stage, BIOS is loaded, and the system hardware is identified. The following steps take place: **1.** Power is supplied to the processor. The processor is hard-coded to look at a special memory address for code to execute. **2.** This memory address contains a pointer or jump program that instructs the processor where to find the BIOS program. **3.** The processor loads the BIOS program. The first BIOS process to run is the power-on self-test (POST). **4.** If the POST is successful, the BIOS identifies other system devices. It uses CMOS settings and information supplied by the devices themselves to identify and configure hardware devices. Plug and Play devices are allocated system resources. The system typically displays information about the keyboard, mouse, and IDE drives in the system. Following this summary, information about devices and system resources is displayed. **5.** The BIOS then searches for a boot sector, using the boot order specified in the CMOS.
Boot loader	During the boot loader stage, BIOS gives control to the boot loader program. The following steps take place: **1.** BIOS searches the boot sector, which contains a Master Boot Record (MBR). **2.** BIOS loads the primary bootloader code from the MBR. **3.** The primary bootloader does one of the following: • It examines the partition table marked as bootable, and then loads the boot sector from that partition. This boot sector contains a secondary boot loader, which locates an OS kernel. • It locates an OS kernel directly without using a secondary boot loader. 4. When the secondary boot loader is in RAM and executing, a splash screen is commonly displayed, and an optional initial RAM disk (e.g., initrd image) is loaded into memory. The initrd image: • Has root permissions that can be used to access the actual /root file system regardless of whether it exists on the local computer or an external device. Without the permissions, the computer could not access the file systems and read information that only exists on those file systems. • Is used to mount the actual file system and load the kernel into RAM. **5.** With the images ready, the secondary boot loader invokes the kernel image.
OS Kernel	During this stage, the Linux kernel takes over. The kernel: **1.** Resides in the /boot directory. **2.** Initializes the hardware on the system. **3.** Locates and loads the initrd script to access the linuxrc program, which configures the operating system. **4.** Dismounts and erases the RAM disk image. On older distributions, this is the initrd image. On newer distributions, this is the initramfs image. **5.** Looks for new hardware and loads the drivers. **6.** Mounts the root partition. **7.** Loads and executes either the init (Initial) process (for older distributions) or the systemd process (for newer distributions). These processes then launch all other processes (either directly or indirectly) to finish booting the system. The init (Initial) or systemd processes are always assigned a process ID of 1 because they are always the first processes to run on the system.

4.2 GRUB Menu

(Common script files in ***/etc/grub.d*** *control the GRUB2 menu)*

Script File	Description
00_header	Sets initial appearance items, such as the graphics mode, default selection, timeout, etc. These settings are typically imported from the /etc/default/grub file.
10_linux	Identifies all Linux kernels installed on the root device and creates corresponding GRUB2 menu entries for each one. This allows you to select which Linux kernel you want to load when you initially boot the system.
30_os-prober	Executes os-prober to search for other operating systems (such as Microsoft Windows) and automatically creates GRUB2 menu items for them.
40_custom	Allows for custom menu entries, which are imported directly into /boot/grub/grub.cfg without any changes.

4.2 GRUB Configuration File

*The **/etc/default/grub** file is the primary configuration file for changing menu display settings*

Option	Description
GRUB_DEFAULT	Sets the default menu entry. Typical entries include: • Numeric (such as 0, 1, 2) • Complete menu entry quotation (such as "Ubuntu, Linux 20.4.2")
GRUB_SAVEDEFAULT	Sets the last selected OS from the menu as the default OS on the next boot. **GRUB_DEFAULT=saved** is also required for this option to work correctly.
GRUB_HIDDEN_TIMEOUT	Determines how long a blank screen will be displayed. While the screen is blank, the user can press the Shift key to display the GRUB2 menu. Options include, 0, X, and (null): **0** disables this functionality. **X** (an integer value) pauses and shows a blank screen for X seconds. **(null)** uses the value specified in the GRUB_TIMEOUT entry.
GRUB_HIDDEN_TIMEOUT_QUIET	Works in conjunction with the GRUB_HIDDEN_TIMEOUT parameter. It displays a counter (countdown) while the screen is blank. Options include true and false: **true** does not display a counter. **false** displays the counter for the duration specified in the GRUB_HIDDEN_TIMEOUT entry.
GRUB_TIMEOUT	Determines how long to wait for user interaction before booting the default operating system. Options include X and -1: **X** (an integer value of 1 or higher) sets the display duration in seconds. **-1** causes the menu to display until the user makes a selection. The GRUB2 menu is hidden by default unless another OS is detected by the system.
GRUB_CMDLINE_LINUX	Passes options to the kernel. With the GRUB Legacy bootloader, this was done by adding options to the end of the kernel line. In GRUB2, this is done using the GRUB_CMDLINE_LINUX parameter.
GRUB_GFXMODE	Sets the resolution of the graphical GRUB2 menu. Multiple resolutions may be specified if they are separated by commas.
GRUB_BACKGROUND	Sets the background image during the GRUB2 menu display. The full path should be used. It must be in the PNG, TGA, or JPG/JPEG file formats.
GRUB_DISABLE_OS_PROBER	Enables and disables the os-prober check of other partitions for operating systems, including Windows and Linux, during the execution of the update-grub command. If the os-prober is enabled, operating systems are placed in the GRUB2 menu.

4.3 Boot Target

Target File	Description
poweroff.target	Halts the system.
rescue.target	Configures the system to run in single-user mode with a text-based user interface. This target sets up a base system and opens a rescue shell for troubleshooting system problems.
multi-user.target	Configures the system to run in multi-user mode with a text-based user interface. This target is commonly used as the default mode for server systems.
graphical.target	Configures the system to run in multi-user mode with a graphical user interface. This target provides all the services of the multi-user target with the addition of a graphical user interface. This target is commonly used as the default mode for desktop systems.
reboot.target	Reboots the system.
emergency.target	Opens a minimal emergency shell for troubleshooting serious system problems.

Equivalent Runlevel Names

Target File	Description
runlevel0.target	Equivalent of **poweroff.target**
runlevel1.target	Equivalent of **rescue.target**
runlevel2.target **runlevel3.target** **runlevel4.target**	Equivalent of **multi-user.target**
runlevel5.target	Equivalent of **graphical.target**
runlevel6.target	Equivalent of **reboot.target**

4.3 Unit Files

<table>
<tr><th>Section</th><th colspan="2">Description</th></tr>
<tr><td rowspan="10">Unit section</td><td colspan="2">The Unit directives provide an overview of the unit.
Unit section directives include:</td></tr>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>Documentation=</td><td>Lists referencing documentation for this unit or its configuration.</td></tr>
<tr><td>DefaultDependencies=</td><td>Lists 'yes' or 'no.' These are similar to implicit dependencies but can be turned on and off by setting this option to yes or no.</td></tr>
<tr><td>Conflicts=</td><td>Lists negative requirement dependencies. If a unit has a Conflicts= setting on another unit, starting the former will stop the latter and vice versa. Note that this setting is independent of the After= and Before= ordering dependencies.</td></tr>
<tr><td>Requires=</td><td>Lists the units that must be activated for a unit to function. By default, the other units listed by a directive are activated at the same time as the unit.</td></tr>
<tr><td>After=</td><td>Lists the units to start before this unit is started.</td></tr>
<tr><td>Before=</td><td>Lists the units to start after this unit is started.</td></tr>
<tr><td>Wants=</td><td>Lists the units recommended to be in effect or started for the unit to function.</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td rowspan="7">Unit-specific section</td><td colspan="2">Systemd categorizes units according to the type of resource they describe. The easiest way to determine the type of a unit is by the suffix type appended to the end of the resource name. For example, a unit file named my.service would be a service unit file that would describe how to manage a service.
Therefore, unit files will typically contain a unit-specific section. The following table shows a few examples of unit files that have directives specific to their type:</td></tr>
<tr><th>Type</th><th>Description</th></tr>
<tr><td>Service</td><td>Describes how to manage a service or application on the server.</td></tr>
<tr><td>Socket</td><td>Describes a network or IPC socket or a FIFO buffer that systemd uses for socket-based activation.</td></tr>
<tr><td>Mount</td><td>Defines a mount point on the system to be managed by systemd.</td></tr>
<tr><td>Automount</td><td>Configures a mount point that will be automatically mounted.</td></tr>
<tr><td colspan="2">When a unit does not include a directive type, it does have unit-specific sections.</td></tr>
<tr><td rowspan="5">Install section</td><td colspan="2">The install section is typically the last section in a unit file. Typically, a unit is installed or enabled by another unit that is started at boot. Directives in the install section specify what happens when a unit is enabled. Directives found in the install section include:</td></tr>
<tr><th>Option</th><th>Descriptions</th></tr>
<tr><td>WantedBy=</td><td>Specifies the unit requesting this unit to be enabled.</td></tr>
<tr><td>RequiredBy=</td><td>Specifies that another unit requires this unit to be enabled.</td></tr>
<tr><td>Also=</td><td>Specifies units to be enabled or disabled as a set.</td></tr>
</table>

4.4 systemctl Command (boot target)

Command	Description
systemctl isolate *boot_target*	Changes the system state to the specified boot target. Changing boot targets with the systemctl command changes only the current system state. If the system is restarted, it will revert back to the default boot target.
systemctl get-default	Displays the current boot target.
systemctl set-default *boot_target*	Sets the default boot target, which is identified by the **/etc/systemd/system/default.target** file. This file is a symbolic link that points to a target file in **/usr/lib/systemd/system** that should be used by default when the system starts. This command modifies the target file that is the **default.target** symbolic link points to.
systemd-analyze blame	Prints a list of running units, listed in the order of time to initialize. Consider that initialization time includes the time a unit must wait for another unit to complete. Does not report results for services that start immediately, as indicated by type=simple.

4.4 systemctl Command

Command	Function
systemctl start service.service	Starts a daemon.
systemctl stop service.service	Stops a daemon.
systemctl restart service.service	Restarts a daemon.
systemctl reload service.service	Reloads a daemon's configuration without actually stopping the daemon itself.
systemctl status service.service	Displays a daemon's status.
systemctl enable service.service	Automatically starts a daemon when the system starts.
systemctl disable service.service	Prevents a daemon from automatically starting when the system starts.
systemctl is-enabled service.service	Looks to see if a daemon is configured to automatically start on system boot.
systemctl mask service.service	Prevents a daemon from starting at all, either automatically or manually, from the shell prompt.
systemctl unmask service.service	Unmasks a previously masked daemon. This allows the daemon to be started either manually or automatically.

4.5 Shutdown Commands

Command	Function
shutdown -h now **halt** **init 0** **systemctl isolate poweroff.target** **systemctl isolate runlevel0.target**	Shuts the system down immediately. Consider the following for the shutdown command: **-h** specifies that the system halt or power off after shutdown. • now forces the system to shut down without a delay. Any of these commands shut the system down immediately.
shutdown -r now **reboot** **init 6** **systemctl isolate reboot.target** **systemctl isolate runlevel6.target**	Shuts the system down immediately and then reboots. Any of these commands reboot the system immediately.
shutdown -h time message **shutdown -r time message**	Shuts the system down in the designated amount of time and sends a message.
shutdown -c **Ctrl+c**	Cancels a pending shutdown.
shutdown -rf time	**-r** parameter issues the reboot command. **-f** parameter stands for reboot fast and skips the fsck utility on reboot.
shutdown -k message	Sends a warning message but does not shut down the system.