4.1.2 Linux Boot Process Facts

Understanding the overall boot process can be beneficial when you need to configure boot loaders, boot targets, and service files that govern the way daemons are loaded. Basic Input/Output System (BIOS) is the firmware specification that was originally used by personal computers to initiate the boot process. The Unified Extensible Firmware Interface (UEFI) was developed to address the shortcomings of BIOS. UEFI has replaced BIOS and is now the standard used by computer manufacturers to govern the boot process.

This lesson covers the following topics:

- UEFI boot
- BIOS boot
- vmlinux/vmlinuz

UEFI Boot

UEFI (previously known as the EFI) is a firmware specification that defines the process for booting computer systems. It is the interface between a computer system's hardware/firmware and the operating system. Some of the features provided by UEFI include the following:

- It provides support for larger disks using the *Globally Unique Identifier Partition Table* (GPT) partition scheme. The MBR partitioning scheme used by BIOS supported only four partitions per disk, with a maximum size of 2 TB per partition. UEFI supports a maximum partition size of 9.4 ZB (9.4 x 10^{21} bytes).
- It provides its own boot manager. This is a significant change. The old BIOS had only enough intelligence to load a single block from the storage device. This required a multi-stage boot process. UEFI has its own command interpreter and boot manager. You no longer need a dedicated boot loader as long as you place the operating system's bootable files into the EFI system partition (ESP), which is formatted with a FAT file system.
- It preserves several components from the traditional BIOS, including power management and a real time clock.

The following table identifies the phases during a 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 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 devices 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.

BIOS Boot

The BIOS process has been used for decades. You may find older hardware that still uses this system. Be aware that recent computer systems no longer use this process but use UEFI instead.

The following table identifies the phases during a 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 boot loader code from the MBR. 3. The primary boot loader 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.

vmlinux/vmlinuz

The vmlinux and vmlinuz Linux kernel executables contain information that can be used for debugging. The vm part of the name stands for virtual memory, indicating that these Linux operating systems can use hard disk space as memory. Be aware of the following:

- Most Linux distributions use a vmlinux or vmlinuz file.
- The vmlinuz file is a compressed file of vmlinux.
  - The vmlinuz file is typically created using bzImage.
  - The bzImage has gzip decompressor code built into it. You cannot use gunzip or gzip -dc to unpack vmlinuz.
  - In the past, zImage was used to create the vmlinuz file. Now, bzImage is used because it uncompresses the kernel into high memory, over 1 MB.
