Overview

Learners will be able to…

- Describe each step of the **Linux Boot Process**
- Define **Kernel Panic**
- Describe **Device Types in `/dev`**
- Perform **Basic Package Compilation**
- Describe **Basic Storage Concepts**

info

Make Sure You Know

This assignment assumes the learner has no previous experience with Bash scripting.

Limitations

This assignment is an overview of fundamental information about Linux Environment, which is provided with this assignment.

Initializing additional kernels and file systems in this environment will not be possible.

Introduction

When we go to use our Linux machine, its likely that we:
- Press the Power Button
- Wait a few seconds, then
- See the login screen to enter some credentials

The **boot process** describes what happens while we're waiting those few seconds for our login screen to appear.

When a Linux system boots, it goes through 6 different phases:

1. BIOS
2. MBR
3. GRUB
4. Kernel
5. Init
6. Runlevel

In this section, we'll look at these phases and see what happens during each.

The Linux system keeps detailed information about what happens during the boot process in its `/var/log`

Click the button below to display the contents of our `/var/log` directory.

We can see that there isn't just 1 log file, but information on many different processes.

We can use the `dmesg` command to display all of the boot commands contained in our log. This can be helpful when troubleshooting hardware problems

Use the button below to display our boot commands with `dmesg` piped through the `less` command, to make it more readable.

Checkpoint

BIOS & UEFI

BIOS is an abbreviation for **Basic Input/Output System**. It runs from the **Read-Only Memory (ROM)**, located on the motherboard of your machine. This makes it independent from any OS you might have installed in your computer.

The **BIOS**' main objective is to locate, load, run, and give control to the **MBR Bootloader**.

When you first power on your machine, BIOS:

- Does a few system integrity checks on the **Hard Disk Drive (HDD**) or S**olid State Drive (SSD)**.
- Looks for the **Bootloader** software in a number of different storage locations.
- Once found, BIOS loads the file and runs the program.
- After the bootloader software has been loaded into memory, the BIOS then transfers control to it.

info

BIOS Configuration

If you want to change the boot device, most BIOS have a special key that you can press to do so. All BIOS have a special key that you can use to get into the BIOS configuration screen (from which you can define the boot order).

These keys depend on the BIOS and are often written on the screen briefly during the start-up process.

UEFI

Newer computers may use **Unified Extensible Firmware Interface (UEFI)**, instead of **BIOS** at this stage of the boot process.

UEFI and **BIOS** are both low-level programs that start when your computer starts up. **UEFI** is a more modern solution because it can support bigger hard drives, faster boot times, more security features, and graphics and mouse cursors.

UEFI is quicker than BIOS because it has greater addressable address space than BIOS. Graphics and mouse cursor support are also smoother in UEFI than in BIOS.

Many of these PCs still arrive with text-mode UEFI setup screens that resemble classic BIOS screens.

Checkpoint

Master Boot Record

The **Master Boot Record (MBR)** can be found in the first sector of the bootable disk. This is commonly found in the `/dev/hda` or `dev/sda` directories.

The **MBR** contains.
1. ~ 446 bytes of Primary boot loader information
2. ~ 64 bytes of Partition Table Information
3. ~ 2 bytes of MBR Validation Checks
4. Information about **GRUB**

All of this information takes up less than $512% bytes of usable space.

The Master Boot Record's primary goal is to load and execute the GRUB bootloader.

Initial Ram Disk

As the bootloader is starting up, it's common for it to begin a task called **initial ram disk (Init RD)**, contained on the `/boot` directory.

Init RD is used as a temporary root file system by the kernel until kernel is completely booted and the real root file system is mounted.

This allows us to use certain parts of the operating systems (like loading drivers over a network device) before the actual operating system has been started by our machine.

▼ **Lilo**

Linux Loader (Lilo) is another bootloader program, similar to GRUB, that you may find while working on Linux systems.

For this lesson, we will focus on the **GRUB 2** bootloader

Checkpoint

GRUB

GRUB stands for **Grand Unified Bootloader**. It starts the operating system and holds information about the file system. Most modern Linux systems use the **grub2** bootloader.

Grub's main objective is to read its configuration file and boot the Linux kernel. If you have multiple kernel images available, you can chose which one you'd like to use.

When executed, GRUB will display a splash and wait for input. If nothing is received, it runs the **default kernel image.**

The location of the configuration file depends on the system distribution and whether it uses **BIOS** or **UEFI**.

In BIOS system, it usually reads inside `/boot/grub2/grub.cfg`. This location may be different for UEFI systems.

When this bootloader stage begins:
- Firmware executes the bootloader program `grub2`
- The bootloader reads its configuration filed
- the bootloader executes and passes control to the **kernel**

Installing Grub 2

GRUB2 is normally installed automatically on a Linux system, so it is rare that you will have to install GRUB2 manually.

However, there are situations where we may need to repair the existing GRUB2 installation or change its configuration. This requires us to install GRUB2 manually.

Kernel

The Linux **kernel** is the main interface between a computer's hardware and its processes. It's contained within an executible file that you may see in two formats:

- `vmlinux` = Uncompressed format
- `vmlinuz` = Compressed format

In the boot process, the kernel is responsible for mounting the root file system based on the specifications within the GRUB configuration file `/boot/grub2/grub.cfg`.

The kernel also runs the init program, located at `/sbin/init`. This becomes the system's first running process and it is assigned a **process ID** of 1.

We can find this process by using the `ps` command and searching for the `init` process using `grep`

Copy the code below and paste it into the terminal to locate the `init` process.

```
ps -ef | grep init
```

The `ps` command gives us the name of the process along with its process ID number. When we run this command, we find the `/sbin/init` location with a process ID of 1.

The kernel's primary goal is to load the file system and execute the `init` process.

Checkpoint

Init

The **init** process looks at the inittab file to decide the Linux run level. Runlevels decide which initial programs will be loaded and started.

The initial RAM disk (initrd) is a temporary root file system that is mounted until the true root file system becomes accessible. The initrd is loaded at kernel boot.

The kernel mounts this initrd during the two-stage boot procedure to load the modules necessary to access the real file systems or as well as the permanent root file system.

The run levels that are available include:

0. Halt
1. Single User Mode
2. Multiuser, without NFS
3. Full Multiuser
4. Unused
5. X11
6. reboot

`mkinitrd`

`mkinitrd` is a utility that **creates an initial ramdisk image that the kernel uses to preload block device modules** (IDE, SCSI, or RAID) required to access the root filesystem. These images are normally found within files named `initrd.img`.

`mkinitrd` automatically installs filesystem modules, IDE modules, and raid modules if necessary, making it easy to design and utilize modular kernels. This command is also helpful for reinstalling the bootloader.

For more information, check out the `mkinitrd` man page by clicking the button below.

Runlevel

The run level stage executes the appropriate programs based on the selected run level. Each run level has it’s own directory at the following locations:

0. `/etc/rc.d/rc0.d/`
1. `/etc/rc.d/rc1.d/`
2. `/etc/rc.d/rc2.d/`
3. `/etc/rc.d/rc3.d/`
4. `/etc/rc.d/rc4.d/`
5. `/etc/rc.d/rc5.d/`
6. `/etc/rc.d/rc6.d/`

The programs inside the directory will run in association with the selected run level.

PXE

A **boot source** is a file that contains a bootable operating system (OS) as well as all of the OS's configuration options, including drivers.

We can use a **boot source** to make templates for virtual machines that have certain settings. These templates may be used to generate an endless amount of accessible virtual machines.

Preboot eXecution Environment

A **Preboot eXecution Environment (PXE)** boots systems without a hard disk or OS. PXE is one of the ways that computers can start up over a network by making the system's **network interface card (NIC)** work like a boot device. It's used nearly exclusively in systems with a central server that run virtual operating systems.

Several network protocols are used to make PXE network boot work, such as:
- **Network Interface Controller (NIC)**

- PXE-enabled NIC is the accepted standard for data center-grade servers, but many network cards for consumers do not have PXE capabilities.
- **Internet Protocol (IP)**
- **Dynamic Host Configuration Protocol (DHCP)**
    - There are two kinds of actors in DHCP: the DHCP server and the DHCP client.
- **User Datagram Protocol (UDP)**
- **Trivial File Transfer Protocol (TFTP)**
    - A protocol based on UDP for getting or sending files. Its simplicity makes it useful for putting into a limited firmware environment.

PXE network booting is useful in situations where there are no hard drives. Booting is done with the help of routers and other devices that are managed centrally.

PXE Booting

The PXE booting process begins when the PXE-enabled network interface card (NIC) on the client system sends a broadcast request to the DHCP server. A “discover” packet is sent out with this request. When the DHCP server gets this packet, it sends back the IP address of the client and the address of the TFTP server, which is where the boot files are.

When the PXE Process is executed:

- The NIC sends a “discover” packet containing a broadcast request to the DHCP server.
- The request is received by the DHCP server.
- It sends back a “offer” packet with the address of the TFTP server, the boot image (pxelinux.0), and standard information like the IP address, subnet mask, domain name system (DNS).
- This information is brought from the server to the client system.
- It looks at what the server is offering and is then given network parameters such as an IP address and a subnet mask.
- The client asks for the boot image from the PXE boot server, which is a TFTP server.
- The boot image is sent by the PXE boot server using TFTP.
- The client runs it.
- The boot image looks for the boot configuration files in the TFTP server’s pxelinux.cfg directory.
- The client gets the files for the kernel and root system and loads them.
- The client system restarts.

Some of the most important benefits of PXE booting include:

- It’s possible to store and control data in a central location, which makes information security more reliable.
- The client doesn’t need an operating system or a device for storing information.
- Since PXE doesn’t care about the vendor, it’s easy to add new clients to the network as business needs change.
- Most tasks are done remotely, which saves IT managers a lot of time and effort.

Booting from USB

Sample content New Page

Booting from ISO

Sample content New Page

Kernel Panic Error

Many times, commonly after purging, we may receive a **Kernel Panic** error.

It can be alarming to have our computer flash the word “panic” at us when it’s trying to do something, but, we’re not so easily shaken!

The key in this situation is to:

- **Relax** the initial feeling of surprise. Someone has to stay level, here!
- **Assess** the situation by reading any error messages for information
- **Recover** the kernel by taking necessary steps to clear the error.

Kernel Panic Causes

There are several reasons to use the `mkinitrd` command to correct a Kernel Panic error:

Kernel Panic Solutions

Sample content New Page

Block Devices

Sample content New Page

Character devices

Sample content New Page

Special Character Devices

Sample content New Page