4.1.1 Linux Boot Process

Click one of the buttons to take you to that part of the video.

Linux Boot Process 0:00-0:57

As a Linux administrator, you need to understand how the operating system itself boots. This is because the Linux boot process is a little more complicated than the boot process that might be used by other operating systems that you may already be familiar with.

So, in this lesson, we're going to spend some time outlining how the boot process works. And this is important because understanding the overall boot process can be really beneficial when you need to configure such things as bootloaders, boot targets, and your service files that govern the way your daemons are loaded. Now, please be aware, before we begin, that the boot process we're going to look at here is generic in nature, and most Linux distributions will follow this process. But the specific implementation of this process could vary between various Linux distributions. And I can guarantee you it will vary between older versions of the Linux operating system and the newer versions of the Linux operating system.

Linux Boot Phases 0:58-1:12

So, to make the Linux boot process a little more digestible, we're going to break it down into the three phases that you see here. We're first going to look at the BIOS phase, when the system initially starts up. We'll look at the bootloader phase, and then we'll end by looking at what we call the kernel phase.

Linux Boot Process 1:13-4:33

This diagram provides a brief summary view of what the Linux boot process looks like. We're going to begin by talking about the BIOS phase of the Linux boot process, which is indicated by this set of boxes, right here. Regardless of what operating system you're using-- whether it's Linux, Windows, or whatever-- when you first power on most X86 or X86-64 personal computers, the system BIOS is the first component that's going to take charge of the boot process. The system BIOS is a ROM chip. It's integrated in your motherboard, and it contains a very small programs and drivers that'll allow your CPU to communicate with other system devices, such as the keyboard, the memory, the system RAM, your I/O ports, your system speaker, and your hard disk drive.

So, the BIOS plays two key roles during the boot process. First, it tests the various system components to make sure they're working properly. This is called the power-on self-test. We just call it POST. Now, if the BIOS encounters any problem with a system device during POST, POST will display an error message on the screen.

If everything works out okay, however, we move on to the second part of the BIOS phase, where we select which storage device we're going to boot the operating system from. Typically, you can configure the order in which the BIOS should look for bootable media in your CMOS setup program. The BIOS will boot from the first bootable device it finds in the list, so you need to make sure that you order your devices properly.

You might be asking, "How does the BIOS know if a device is bootable or not?" Well, it looks in the first sector of the device, which is called the boot sector. On a hard disk drive, the boot sector contains the master boot record, or what we call the MBR. Basically, after running POST, the BIOS really doesn't have anything else to do. It's already done its job. What it needs to do now is turn control of the system over to some other entity. In order to do this, it needs to load programming into memory and run it on the CPU to boot the operating system. In order to do this, it looks for the MBR on your system's hard drive.

The MBR tells the system where a bootloader for the operating system resides, and the bootloader has a very important job. The bootloader is software that the BIOS can load from the MBR or the hard drive that will allow the CPU to access the storage device, probably a hard disk, and load an operating system into RAM. So, in order to do this, the bootloader is configured with the location of the operating system files on the hard disk.

Now, please be aware that the bootloader itself may or may not actually be in the MBR. You can install some bootloaders completely within the MBR, or you can install them within a partition somewhere else on the hard drive and just place a pointer in the MBR. Some bootloaders actually reside in both places; part of it will reside in the MBR, and part of it will reside in a partition on the hard disk drive. So, after loading the bootloaders software into memory, the BIOS turns control of the system over to the bootloader. The bootloader is probably configured to automatically load an operating system from the hard drive. It may also be configured to provide its users with a menu to select which operating system that they want to load.

Unified Extensible Firmware Interface (UEFI) 4:34-6:38

Now, be aware that the process we've talked about so far in this lesson has been used for decades with computer systems. But you need to be aware that the very latest computer systems actually don't use this process anymore. Instead of a traditional BIOS, these systems now use the Unified Extensible Firmware Interface, which we affectionately call UEFI. UEFI is, basically, a BIOS on steroids. Some of the features provided by UEFI are shown here.

First of all, it preserves several of the components that we saw in the traditional BIOS, such as power management and a real-time clock. But UEFI provides support for much larger disks using the Globally Unique Identifier Partition Table, which is just GPT. Probably the key thing you need to be aware of, as far as the boot process is concerned, is the fact that UEFI provides its own boot manager, and this is a significant change.

The old BIOS only had enough intelligence just to load a single block, the first block, from a storage device. So, with advanced bootloaders and advanced operating systems, this really required a multi-staged boot process where we have part of the bootloader in the MBR, and then the rest of it installed in a partition somewhere else on the disk.

UEFI doesn't do this. UEFI has its own command interpreter and its own boot manager. So, with UEFI you actually don't need a dedicated bootloader any longer. All of what you do, instead, is place your operating system's bootable files into the UEFI boot partition. This boot partition contains bootloader files for all of the operating systems that are installed on other partitions on that storage device.

For example, to boot Linux you would probably need to implement a UEFI-aware version of GRUB, the Grand Unified Bootloader. Typically, this is named grub.efi. The reason this is problematic is that you need to make sure that your Linux operating system that you're going to install is compatible with UEFI, if the motherboard in your system is a UEFI motherboard.

Additional Linux Boot Processes 6:39-10:08

So, with this in mind, let's look at the rest of the Linux boot process. With later Linux kernels, the bootloader creates a temporary virtual file system in your system RAM called a RAM disk. And depending upon your distribution, this will either be called the initrd image or the initramfs image.

Now, initrd stands for initial RAM disk, whereas initramfs stands for initial RAM file system. Both of these images perform the same function, which is to provide the linuxrc that is used to set up the system. This image contains a basic file system that's loaded into RAM that can be used to complete a variety of startup tasks. The reason we use a RAM image-- basically a virtual hard drive that's stored in RAM-instead of on an actual storage disk--is because Linux systems can use a wide variety of devices for the root file system. Some of these devices might be created from external USB storage devices or from a software RAID array, or it could even be a device that resides on a different computer and is accessed through MFS, or Samba, or even iSCSI.

Here's the issue; these types of file systems cannot be mounted by the kernel until special software is loaded, which, of course, resides on those file systems. You can't get to the drivers to load the file systems because the drivers reside on those file systems. So, to make the system boot correctly in these situations, the bootloader creates the small virtual hard drive in RAM, the RAM disk we just talked about here, and transfers a temporary root file system from the initrd or initramfs image to that virtual hard disk drive in RAM. The Linux kernel can then use the temporary file system to load all the drivers necessary to complete the tasks required for it to mount the real file system on whatever storage device it is stored upon. So, these processes, here, comprise the bootloader phase of the boot process.

At this point, we're ready to move on to the kernel phase, which is indicated in these boxes here. So, after selecting the operating systems to run the bootloader, we'll load the operating system kernel into RAM from the hard disk drive. After the kernel loads, several key things happen. First of all, the kernel initializes the basic hardware in your system, and then it searches for and uses the initrd or initramfs file system--right here, the virtual root file system-- to run the linuxrc program in order to initially set up the system.

When linuxrc is done, then the initrd or initramfs virtual file system is dismounted, and the RAM disk is destroyed. The kernel then mounts the real root file system and starts probing for new hardware and loading the appropriate drivers. And then, depending upon the distribution, it will load either the init or the systemd process. Which process is used will depend upon your distribution. Older distributions--exclusively, for the most part-- use the init daemon. However, most current distributions are switched to systemd. Regardless of which daemon your distribution uses, the init or systemd processes are the key critical processes that are required to run the rest of the system.

So, at this point in the boot process either init or system, depending upon your distribution, will load the other system processes required to get the system up and running. And once that's done, you can log in and use the system.

Summary 10:09-10:13

That's it for this lesson. In this lesson, we reviewed the Linux boot process. We looked at the BIOS phase, the bootloader phase, and the kernel phase.