9.1.1 Device Drivers

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

Device Drivers 0:00-0:37

In this lesson we're going to be talking about device drivers. If you're going to be responsible for managing Linux systems, then it's very important that you understand how device drivers work within the operating system.

If you've used a Windows system before, then you probably are already familiar with the process required in order to load a driver to support a piece of hardware. Be aware that the way drivers work under Linux is quite a bit different. In order to effectively manage drivers under Linux, you first have to understand what the heck a driver is in the first place.

Device Driver Function 0:38-2:24

The key thing you need to understand when discussing drivers is the fact that the CPU on your motherboard doesn't actually natively know how to communicate with the other hardware devices that are connected to your system.

For example, it doesn't know how to send video data to the video board. It doesn't know how to save data to your hard disk drive. It doesn't know how to send data to the sound board. In order to be able to do these things, the CPU needs instructions in the form of software to tell it how to communicate with these various devices.

For very basic devices in your system, such as your system clock, keyboard, and so on, basic drivers can be stored on a chip on the motherboard. When the system boots up, these drivers can be loaded from the motherboard into memory. The CPU can then use those to communicate with these basic devices, such as the keyboard.

If we're dealing with more complex devices, maybe a network board, sound board, or a storage device, the software needed for the CPU to communicate with these types of devices can't really be stored on a chip on a motherboard.

The problem here is that there are so many different makes and models of these different types of devices that it's completely infeasible to try and store all the software needed for every last device within a chip on the motherboard. Instead, what we do is store the software on the system's hard drive.

Then as the system boots, the operating system loads the software driver from the hard disk into the system memory. Once that's done, the CPU has the information that it needs in order to communicate with the hardware--in this example, a network board.

Kernel Modules 2:25-4:00

It's very important that you understand that there are actually two different ways in which Linux can implement a device driver for a particular piece of hardware in your computer system. One option is to load the driver as a kernel module. Once the Linux kernel itself has been loaded into memory during the boot process, it can then be configured to load kernel modules.

Kernel modules are really just drivers. They allow the CPU in the operating system to communicate with the other hardware that's installed in the system. Kernel modules have an extension of .ko or .o. They're stored within the path that you see here, /lib/modules, followed by the version number of your kernel.

Within that directory there's a subdirectory called kernel. Within that directory there's another subdirectory called drivers. Within the drivers subdirectory, we see several other subdirectories that each contain kernel modules for hardware devices. This is where the device drivers are stored.

For example, the kernel modules that are needed to support SATA storage devices are stored in this folder right here, the /ata directory. The device drivers needed to support a bluetooth network adapter are stored in this folder. The drivers necessary for communicating with firewire devices are stored in this folder.

This is not the only way that drivers can be loaded in a Linux system.

Device Drivers Compiled Into the Kernel 4:01-7:04

A second way that hardware support can be implemented is to actually compile the drivers themselves directly into the Linux kernel. Instead of loading a kernel module externally after the system is booted, instead we take the device drivers themselves and integrate them directly into the Linux kernel. They're in the kernel.

There is no need to load a device driver at system boot, because the driver software is already there. It's in the kernel itself. Understand that doing this requires that you manually recompile the kernel itself from its source code.

When you do, you need to specify which drivers you want directly integrated within the kernel during the recompiling process. After you do this, the kernel modules for those devices will no longer be needed.

The operating system has all the software it needs inside of the kernel itself. At first glance, this probably sounds like a really great way to do things, right?

Why don't we compile all the drivers that we need for the hardware in the system directly into the kernel? This is a good strategy for some drivers. However, as a general rule of thumb, you should limit the drivers compiled into the kernel to only those drivers that the system needs to get the system initially booted up.

Maybe you have a special keyboard driver. Maybe you need a driver for your storage devices. Things like that. The rest of your drivers really should be loaded as kernel modules and not compiled into the kernel itself.

You might be asking, "Why not?" There are actually a couple of really good reasons for doing things this way.

First of all, each driver you compile into the kernel, obviously, will increase the overall size of the kernel. To keep things running fast, you should try to keep your kernel as lean and clean as possible.

Secondly, configuring a kernel--recompiling a kernel--is quite complex. It requires a lot of in-depth knowledge about the hardware that you're going to be integrating support for. You need to know what each configuration element provides. If you don't know what you're doing, it's not going to work properly.

Really, the main issue is that of modularity.

If you're the type of person who never modifies their computer system, never upgrades, never adds new hardware, never reconfigures your computer system, then compiling more drivers directly into the kernel could make sense. However, it doesn't make sense in most situations.

Especially if you're someone like me who likes to constantly tear things apart--add, remove, and reconfigure the hardware inside of the computer system. By loading device drivers as kernel modules at system boot, I can add or remove support from the hardware devices very quickly. It's very easy to do. I can even do it from the command line.

If, on the other hand, I compile all the drivers I use directly into the kernel, what's going to happen after a couple of years is a very bloated kernel that contains support for a lot of hardware that is no longer in the system. Essentially, using kernel modules makes your system much more dynamic.

Summary 7:05-7:14

That's it for this lesson. In this lesson we discussed the role of device drivers in a Linux system. We first looked at kernel modules. Then we looked at device drivers that are directly compiled into the kernel itself.
