9.2.1 Kernel Module Management

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

Kernel Module Management 0:00-0:22

If you're going to be responsible for managing Linux systems, then you've got to understand how to manage the hardware drivers that are used by the operating system. As with most other operating systems, on Linux you can manually list, load, or unload kernel modules, which are our device drivers. Let's take a look at how you do that.

View Information About Currently Loaded Modules 0:23-2:28

Let's begin by talking about how you can view a list of currently loaded kernel modules. One way to do this is to use the lsmod command. This command pulls data from the /proc/modules file and reformats it so that it displays nicely on the screen. To use this command, all you have to do is type 'lsmod' at the shell prompt, and as you can see, each kernel module or device driver used by the system is listed on the screen. It tells you the size of the kernel module and it also displays dependency information for that module. In this example, we see that for this kernel module--named bnep--to load, the Bluetooth kernel module must already be loaded first.

As you can see, the lsmod command only displays summary information. If you need more detailed information about one particular module, then you want to use the modinfo command instead. Generally speaking, what I do is use the lsmod command first to find the name of the module in question, and then I can use modinfo to get more information about it. I type 'modinfo', followed by the module name. In this example we're looking for more information about that Bluetooth driver we just looked at with lsmod. This is the device driver that's used by the Linux system to support Bluetooth devices. In the output of the modinfo command, I can see the name of the kernel module file. I can see its licensing information, version information, a description of the module. I can see the name of the programmer who wrote it, I can see dependency information right here. In this case, in order for Bluetooth to load, this kernel module, rfkill has to be loaded first.

There's also a digital signature on the module so we know that it has not been tampered with or modified. Here's the hashing mechanism that was used to create the signin key, it's SHA256, and so on. That's how you view information about kernel modules.

Load Kernel Modules 2:29-10:33

If, on the other hand, you need to load a kernel module, we have to use a different command. Before we actually load a kernel module, we first need to run the command that you see here, 'depmod'. This is very important. Recall that just a minute ago we saw that different kernel modules have different dependencies. One module has to be loaded before another one can load. The depmod command is used to build a file named modules.dep right here and it's stored in the /lib/modules directory. Within modules there is a subdirectory named after your kernel version, and within that directory is modules.dep.

Within this file, depmod lists all of these intricate dependencies between modules. This is very important because it helps other kernel module management utilities ensure that all these dependencies are accounted for. We need to make sure that dependent modules are loaded first whenever you try to load a new module, otherwise we're just going to have problems. Once we've created modules.dep with the depmod command, it is now safe to go ahead and load kernel modules, because we know that any module dependencies are going to be automatically calculated and accounted for. Be aware that there are actually two different commands that you can use to load kernel modules, one of which I never use, the other one of which I use all the time.

The one I never use is insmod. The syntax is 'insmod', followed by the filename of the module you want to load. An example is shown here, where I run insmod and then I point to the parallel port driver located in the /lib/modules and then the kernel version number /kernel/driver/parport directory. The name of the kernel module that I want to load is parport_pc.kl, and this driver supports the parallel port hardware in my system. By loading this driver, I can then connect devices to my parallel port and send or receive data from that parallel port. I never use insmod. Instead, I use the modprobe command to load kernel modules. The reason why I use modprobe and I never use insmod is the fact that the insmod command ignores dependencies.

Remember, we used the depmod command to go through and identify all that dependency information to make sure that dependent kernel modules are loaded before we try to load a new kernel module. Well, insmod doesn't care. If we tell it to load a particular kernel module, it just loads it. Or, at least it tries to load it, and it doesn't take into account any dependent kernel modules that need to be loaded first. The modprobe utility is a much better utility for loading kernel modules because it does take into account dependency information. The syntax is similar to that which we used for the insmod command. We enter 'modprobe', followed by the name of the module that we want to load. For example, within the /lib/module/kernel/version/driver/net/ethernet directory, can't see the whole path here.

There are kernel modules that are used for a wide variety of network boards, and you can see all of those right here. For example, let's suppose in this system that we installed a 3Com 3c59x network board, and we want to load the necessary kernel module to support that new network card. So we use the modprobe command to load the 3c59x kernel module. You're probably wondering at this point if the loading of that kernel module will be persistent across system restarts, and the answer is no, it won't. I have to qualify that. It won't, unless that device is automatically detected by the kernel at boot, in which case the answer is yes, the kernel module will be loaded. Be aware that the modprobe command is actually automatically run every time the Linux kernel loads, and it's going to read the information contained in this file right here: /etc/modprobe.conf. It will read this file in order to determine which kernel modules need to be loaded at startup.

Please be aware that some newer Linux distributions will not actually use this file. Instead, they will use a series of files located in the /etc/modprobe.d directory, but these files do basically the same thing that the modprobe.conf file does. That is, tell modprobe which kernel modules need to be loaded at system boot. If you don't see this one, look for this one, or vice versa. Within the modprobe.conf file, there are several directives that are used to tell modprobe which kernel modules need to be loaded. The first one is the install directive. It's followed by the name of the module. This tells modprobe to load the specified module. Be aware, a neat little trick is that you can actually use the install directive to not just load modules, but to also run shell commands. This can provide you with a high degree of flexibility when you're trying to load kernel modules.

For example, if there is a particular command that has to be run before a kernel module will load, that can be done right there. Another directive found within the modprobe.conf file is the alias directive. This directive gives a kernel module an alias name that can be used to reference the kernel module. The alias directive specifies the alias name, followed by the name of the module that we want to assign the alias to. Lastly, we have the options directive, which allows us to specify a particular module name and the options we want to use with that module when the kernel loads it.

Now that you know what directives are used in the modprobe.conf file, I need to tell you that you should not edit that file at all. In fact, if you open up the modprobe.conf file, you'll probably see a big nasty note at the beginning of the file that tells you, "Please do not edit this file directly," because you're probably going to mess things up. Instead, if you need to manually specify that a particular kernel module be loaded at startup, assuming that the kernel does not detect the hardware itself and automatically load the necessary kernel modules, then you can put the necessary commands in one in one of these three different files. One option is to put the commands in the /etc/modprobe.conf.local file; this is basically just a user-editable version of the modprobe.conf file. It uses the exact same syntax as the modprobe file. For example, to load a module, you would use the install directive within this file to specify that a particular kernel module be loaded.

Alternatively, you could also place the commands in the /rc.local or boot.local files as well. Be aware that these two files do not use the same syntax as modprobe.conf or modprobe.conf.local. Instead, what you do is just run the modprobe command, followed by the name of the module that you want to load. This will ensure that that module is loaded every time the system boots. Understanding this, I want you to be aware that you probably won't ever need to do it. My experience has been that most distributions will run a hardware detection routine at system boot; it will scan for new hardware. And when it finds it, it will automatically load the appropriate kernel module for you. The only time I've ever had to manually load a kernel module at setup is to support really, really, really old non-plug-and-play hardware--like an old ISA network board. I haven't had to do that in about a decade. Be aware that if that situation arises, this is how you do it.

Unload Kernel Modules 10:28-11:44

There may be times when you do need to unload a kernel-loaded kernel module, and you do that using one of two different commands. You can use the rmmod command, in which case you enter 'rmmod', followed by the module name that you want to unload. I actually don't use this command, because if the device serviced by this module is currently in use--which frequently is all the time--then rmmod will give up and say, "Hey, I can't unload this. It's in use." In addition, rmmod does not take module dependencies into account. If I'm trying to unload a module that another kernel module is dependent on being present, rmmod doesn't care. If it can take the kernel module out, it will; and that's going to cause major problems with your system. Instead, I strongly recommend that you use the modprobe command. The syntax is 'modprobe -r', followed by the name of the kernel module that you want to unload. modprobe will take a look at all the dependency information and then cleanly unload the kernel module, instead of just ripping it out like rmmod will.

Summary 11:40-11:55

That's it for this lesson. In this lesson we reviewed how you manage kernel modules. We first talked about how you view information about currently loaded kernel modules. We talked about how to load kernel modules, and then we talked about how to unload kernel modules.
