# Chapter 26 Kernel Modules - Notes

## 26.2 Introduction

Linux kernel makes extensive use of **modules**, which contains important software that can be loaded/unloaded as needed after system starts. Many modules incorporate **device drivers** to control hardware either inside system, or attached peripherally. Other modules can control network protocols, support different filesystem types, and many other purposes. Parameters can be specified when loading modules to control their behavior. End result: great flexibility and agility in responding to changing conditions and needs.

## 26.3 Learning Objectives:

- List the advantages of utilizing kernel modules.
- Use **insmod**, **rmmod**, and **modprobe** to load and unload kernel modules.
- Use **modinfo** to find out information about kernel modules.

## 26.4 Advantages of Kernel Modules

Many facilities in Linux kernel designed to be built-in to kernel when kernel initially loaded, or to be added (or removed) later as **modules** as necessary. All but the most central kernel components integrated in such fashion.

Such modules may or may not be device drivers. Eg. may implements certain network protocol or filesystem, rather than drive hardware/software device. Even in cases where functionality will virtually always be needed, incorporation of ability to load/unload as module facilitates development, as kernel reboots not required to test changes.

Even with widespread usage of kernel modules, Linux retains **monolithic** kernel architecture, rather than **microkernel** one. This is because once a module is loaded, becomes fully functional part of kernel, with few restrictions. Communicates with all kernel sub-systems primarily through shared resources, such as memory and locks, rather than through message passing as might a microkernel.

Linux hardly the only operating system to use modules. Solaris does it as well, as does AIX, which terms them **kernel extensions**. However, Linux uses them in particularly robust fashion.

## 26.5 Module Utilities

Number of utility programs used with kernel modules:

- **lsmod**: List loaded modules
- **insmod**: Directly load modules
- **rmmod**: Directly remove modules
- **modprobe**: Load or unload modules, using a pre-build module database with dependency and location information
- **depmod**: Rebuild the module dependency database; needed by **modprobe** and **modinfo**
- **modinfo**: Display information about a module

## 26.6 Module Loading and Unloading

While module is loaded, can always see its status with **lsmod**, as shown below.

Module removal can always be done directly with:

```
$ sudo /sbin/rmmod module_name
```

Note: not necessary to supply either full path name or `.ko` extension when removing module.

lsmod

Module loading/unloading must be done as root user. If full path name known, can always load module directly with:

```
$ sudo /sbin/insmod <pathto>/module_name.ko
```

Normal filesystem location for kernel modulesL under directory tree at `/lib/modules/<kernel-version>`. Kernel module always has file extension of `.ko`, as in `.e1000e.ko`, `ext4.ko`, or `usbserial.ko`.

Kernel modules -> kernel version specific, must match running kernel or they cannot be loaded. Must be compiled either when kernel itself compiled, or later, on system which retails enough of kernel source and compilation configuration to do this properly.

## 26.7 modprobe

In most cirucmstances, modules not loaded/unloaded with **insmod** and **rmmod**. Rather, one uses **modprobe**:

```
$ sudo /sbin/modprobe module_name
$ sudo /sbin/modprobe -r module_name
```

with second form being used for removal. For **modprobe** to work, modules must be installed in proper location, generally under `/lib/modules/$(uname -r)`, where `$(uname -r)` gives current kernel version, such as `4.14.2`.

Can use **depmode** to generate/update the file `/lib/modules/$(uname -r)/modules.dep`.

## 26.8 Some Considerations with Modules

Some important things to keep in mind when loading/unloading modules:

- Impossible to unload module being used by one or more other **modules**, can ascertain this from **lsmod** listing
- Impossible to unload module being used by one or more **processes**, can also be seen from **lsmod** listing. However, some modules do not keep track of this reference count, eg. network device driver modules, as it would make it too difficult to temporarily replace module without shutting down and restarting much of the whole network stack
- When module loaded with **modprobe**, system will automatically load any other modules that need to be loaded first (dependencies)
- When module unloaded with **modprobe -r**, system will automatically unload any other modules being used by the module, if they are not being simultaneously used by any other loaded modules

## 26.9 modinfo

**modinfo** can be used to find out information about kernel modules (whether or not they are currently loaded):

```
$ /sbin/modinfo my_module
$ /sbin/modinfo <pathto>/my_module.ko
```

Can see example below, which displays information about version, file name, which hardware devices the device driver module can handle, and what parameters can be supplied on loading.

Much information about modules also seen in `/sys` pseudo-filesystem directory tree. In example, would look under `/sys/module/e1000` and some, if not all parameters, can be read and/or written under `/sys/module/e1000/parameters`. Will show hot to set them next.

modinfo

## 26.10 Module Parameters

Many modules can be loaded while specifying parameter values:

```
$ sudo /sbin/insmod <pathto>/e1000.ko debug=2 copybreak=256
```

or, for module already in proper system location, easier with:

```
$ sudo /sbin/modprobe e1000e debug=2 copybreak=256
```

## 26.11 Kernel Module Configuration

Files in `/etc/modprobe.d` directory control some parameters that come into play when loading with **modprobe**. These parameters include module name **aliases** and automatically supplied options. Can also blacklist specific modules o avoid them being used.

Settings apply to modules as they are loaded/unloaded, and configurations can be changed as needs change.

Format of files in `/etc/modprobe.d` simple: one command per line, with blanks lines and lines starting with `#` ignored (useful for adding comments). Backslash at end of line causes it to continue on next line, which makes file a bit neater.

##

Back to top