Using Kernel Modules

Zilogic Systems

1. Kernel Modules

The Linux kernel consists of a large base kernel image usually located in `/boot/vmlinuz-x.y.z`, and many small kernel modules located in `/lib/modules/x.y.z/`.

```
[~]$ file /boot/vmlinuz-2.6.18
/boot/vmlinuz-2.6.18: Linux kernel x86 boot executable RO-rootFS,
root_dev 0x303, swap_dev 0x1, Normal VGA
[~]$ find -name "*.ko" /lib/modules/2.6.18/ | head -n 5
/lib/modules/2.6.18/kernel/lib/crc-itu-t.ko
/lib/modules/2.6.18/kernel/lib/crc7.ko
/lib/modules/2.6.18/kernel/lib/crc16.ko
/lib/modules/2.6.18/kernel/lib/zlib_inflate/zlib_inflate.ko
/lib/modules/2.6.18/kernel/lib/crc-ccitt.ko
...
...
```

Kernel modules are lot similar to plug-ins commonly found in web browsers. Modules can be dynamically loaded and unloaded. By loading modules features can be added to the kernel. And by unloading modules modules features can be removed from the kernel.

2. Advantages of Kernel Modules

There a lot of advantages in implementing features as modules:

Smaller Base Kernel Image	With kernel modules, only the required features can be loaded into the kernel. For example, there are 100s of different graphics cards. GNU/Linux distributions will have to provide generic kernels that support all the graphics cards. If there were no module support, all the drivers for the 100 different cards would have to be part of the base kernel image. But with module support, all the drivers for the 100 different cards, will be available as modules in the filesystem. The graphics card would be detected at boot time and only the required modules will be loaded into the kernel.
Shorter Development Cycle	Without modules, driver development would have a long Edit - Compile - Reboot cycle. With modules the driver development reduces to an Edit - Compile - Unload - Load cycle, making development and testing of drivers easier and faster.
Cleaner Design	As the name suggests, modules encourage a cleaner design style. There is a clear interface exported to and from modules. Usually, it is possible to understand a module without having to understand other parts of the kernel.

3. What Can be Implemented as Modules?

Theorotically, any feature that is not required to boot the system to the point of being able to load modules, can be implemented as modules. Most commonly found modules fall into the following categories.

- Device Drivers
- Filesystems - `ext3`, `fat`, `ntfs`
- Network Protocols - `ipv6`, `ppp`
- Executable File Formats - `binfmt_aout`, `binfmt_misc`
- Cryptographic Algorithms - `aes`, `des`, `blowfish`, `sha1`
- Scheduling Algorithms - `cfq-iosched`, `deadline-iosched`

4. Working with Modules

The list of currently loaded kernel modules can be obtained using the `lsmod` command.

```
[~]$ lsmod
Module                  Size  Used by
psmouse                35016  0
radeon                 99744  2
drm                    61332  3 radeon
rfcomm                 34584  0
l2cap                  21696  5 rfcomm
bluetooth              45956  4 rfcomm,l2cap
... clip ...
usbcore               112644  2 uhci_hcd
processor              28840  0
```

The output format is,

- first column - name of the kernel module
- second column - the size of the module
- third column - usage count. When the kernel or user space uses the interfaces provided by a module, the usage count is incremented. In the case of a sound driver, when `/dev/audio` is opened the `soundcore` module’s usage count is incremented.

```
[~]$ lsmod | grep "soundcore"
soundcore               9248  2 snd
[~]$ cat /dev/audio &
[~]$ lsmod | grep "soundcore"
soundcore               9248  3 snd
```

- fourth column - modules that use the interfaces provided by this module. In other words, the modules listed in the fourth column *depend* on the module given in the first column.

An unwanted module can be removed using the `rmmod` command.

```
[~]$ lsmod | grep rtc
rtc                    12372  0
[~]$ sudo rmmod rtc
[~]$ lsmod | grep rtc
```

A module can be removed only when it’s usage count is at zero. In other words, a module can be removed only when nobody is using it’s interfaces.

The module can be loaded back into the kernel using the `insmod` command. The `insmod` command takes the complete path to the module as argument.

```
[~]$ sudo insmod /lib/modules/2.6.18/kernel/drivers/char/rtc.ko
[~]$ lsmod | grep rtc
rtc                    12372  0
```

Kernel module behaviour can be customised using parameters. For example the kernel loopback driver, is available as a kernel module `loop`. The number of loopback devices provided by the driver is limited, since memory has to be allocated for each device. The default no. of devices is 8. The following command will fail.

```
[~]$ sudo losetup /dev/loop15 linux.img
```

But if we would like to use more than 8 devices, the kernel module can be reloaded with the required no. of devices as argument.

The command `modinfo` provides details about a module, including the parameters accepted by a module.

```
[~]$ /sbin/modinfo /lib/modules/2.6.18/kernel/drivers/block/loop.ko
filename:       /lib/modules/2.6.18/kernel/drivers/block/loop.ko
license:        GPL
alias:          block-major-7-*
vermagic:       2.6.18 SMP mod_unload 686 REGPARM gcc-4.1
depends:
parm:           max_loop:Maximum number of loop devices (1-256) (int)
```

The module accepts one parameter `max_loop`, the maximum number of loop back devices required. To enable 16 loopback devices the following command sequence can be used.

```
[~]$ sudo /sbin/insmod /lib/modules/2.6.18/kernel/drivers/block/loop.ko max_loop=16
```

The following command will now succeed.

```
[~]$ sudo losetup /dev/loop15 linux.img
```