COSC 519

Linux Kernel Module

12

Problem

When a USB is plugged into a port on the computer, the kernel must know how to handle this device. Since it is impossible for the kernel to have every known USB device, a kernel module must be created for the device. Most of the time, when a USB is unknown to the kernel, the module comes packaged with the device, which then ask the user if they wish to install the device. However, there are cases when the USB is not compatible with an operating system and one must be created to interact with the USB controller.

Description

Loadable kernel modules (LMKs) are pieces of kernel code which are not complied into in the kernel, but they are instead linked to the kernel once they are loaded. A LKM is an executable and linkable object file that can dynamically add/remove functionality into the kernel. LKMs were created to give custom kernel modules the ability to use the system freely. LMKs are compiled separately and can be loaded or unloaded from the running kernel at any time. LMK minimizes the memory footprint of the kernel by loading only elements as needed on the fly. They are used to add modules to support new hardware, file systems or for adding system calls. LKMs also can be inserted at boot when they are needed to support a particular device or at any time by the user.

Advantages	**Disadvantages**
Load on command	Free Run of System
Does not need to recompile kernel	Security
Reduces Kernel Memory Size	Fragmentation Penalty (minor)

There are two ways of adding code to Linux kernel. The most common and basic way to do that is to add some source files to the kernel source tree and recompile the kernel. The kernel configuration process consists mainly of which files to include in the kernel to be compile. Due to the long amount of time that the compiler takes to compile the Linux kernel, typically an hour, it makes this solution undesirable.

Another way of adding code to the Linux kernel would be load the module to the Linux kernel while it is running. A module that is added using this technique is called a Loadable Kernel Module (LKM). These modules can do lots of things, but they are typically one of three things:

1. Device drivers.
2. File system drivers.
3. System calls.
4. Network drivers
5. TTY line disciplines
6. Executable interpreters

Moreover, the Linux kernel isolates LKMs very well so they do not get involved with the rest of the kernel. There are significant benefits of loading modules to the kernel while it is running.

The main advantage of loading the LKMs while the kernel is running is minimized kernel rebuild time. In fact, the user does not have to rebuild the kernel. This will actually save the user time and spares him/her the possibility of introducing a compilation error. Once you have a working base kernel, it is good to leave it untouched as long as possible. In addition, LKMs helps the user to diagnose system problems without added difficulties. Since a bug in the device driver which is bound into the kernel can stop your system from booting. Finally, LKMs are much faster to debug and easier to maintain. For example, what would take a full reboot with file system driver built into the kernel; you can do with a few quick commands with LKMs.

When the module is loaded it prints to log file letting you know that the model was either successfully or unsuccessfully loaded into the kernel. Once successfully loaded the module then registers a device driver and a listener to tell if a USB device is plugged into a USB port. When a device is plugged in to the port, the module registers the device, also giving the device its ID, and prints to the log file that a device has been plugged in. After the device is removed from the port, the module attempts to unregister the device, then writes to log file if the device was unregistered successfully or not. The log file is located in var/log/message folder, this is the default folder for Linux log files.

Implementation

The USB module was created using NetBeans and the linux.usb library. Since the library is not linked until calling the make command, there were numerous compiler errors that NetBeans believed was an issue. Once calling the make command, the module was compiled successfully and created an object (*.o) and a kernel object (*.ko) file. After these two files were created, the kernel object file is loaded into the kernel using "insmod ./module_name.ko" command in the Linux terminal. The file is then added to the module list file which then can be called at any time. In order to remove the module, simply use the remove command, "rmmod

./module_name.ko”. After the module is loaded simple plug in a USB device and the module will pick this up and print to the log file.

Testing/Validation

Testing the module was very straight forward. We would attempt to build/compile the C code in NetBeans, if there was an error the file would not compile and we would have to find the error to take care of it. Once the code was compiled, we loaded the module into the kernel and if the module was loaded successfully it would write to the output log file that it was successfully loaded. After it was loaded, all that was needed was to plug in and unplug different USBs and go to the log file to see what the module wrote. If it was successfully it would write what the USB device was, on the other hand if failed the module would be killed and the error message was printed to the log file.

Conclusion

Thought out this project we learned that loading LKMs while the kernel is running is more efficient for modules that are not used on a normal basis. However if the module is going to be needed every time the machine is powered up, it should be compiled with the kernel. This way the module does not have to be manually loaded every power up. Depending on the functionality and the number of times it may be needed should determine whether to compile the module with the kernel or to load it.

On the other hand, we have learned through our experience that loading LKMs while the kernel is running is the better and less expensive option, due to the fact the kernel does not need to be recompiled to add new functionality. Another plus is that this technic can save memory space in the kernel. If all the functionality was already included in the kernel there would be a lot of wasted memory to kept functions that are not normally used. Although the security and unrestricted access to the system may propose issues, we still feel this to be a more desirable approach.

The significant advantages of LKMs have proven itself to be desired and many operating systems support them, although may be called something different depending on the system. Although they may not be the best solution in every case, it is a very popular technic. Most device driver use LKMs to tell the controller how to communicate and interact with the device.

User Guide

These step-by-step instructions will explain how to get a kernel module to compile and run in a linux environment. This module was compiled on Ubuntu 10.04 LTS using linux kernel version 2.6.32-45.

Step 1: Download Ubuntu iso from http://www.ubuntu.com/download

Step 2 method 1: Install VirutalBox, mount the iso image, then install Ubuntu.

Step 2 method 2: Burn iso image to a cd, insert CD into your PC. You can either install Ubuntu to a separate partition (dual boot) or wipe out all data by reformatting and then install it.

Step 3: Update Ubuntu to get the latest c-compiler updates. Select System menu from top left -> Administration -> Update Manager -> Check -> Install Updates.

Step 4: Donwload and install the Java Development Kit (Newest Version Available)

Step 5: Download and install NetBeans 7.2.1 (or newer) for Linux

Step 6: Download the Kernel Headers for current kernel version by running in the terminal:

```
sudo apt-get update
sudo apt-get install linux-headers-(uname -r)
```

Step 7: Open NetBeans and create a new C/C++ project application project. Give project a name (without spaces), uncheck create main file, and select "C" from the drop down list to the right.

Step 8: Go to Source File folder and right click. Mouse over New -> C Source File and select. Give file a name (without spaces).

Step 9: Copy this kernel module code into the c source file.

Step 10: Open "Important Files" folder below the Source Files folder on the left and open the Makefile. Delete the auto-generated contents and replace with:

```
obj-m += (module name here).o
all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Step 11: Select "Clean and Build" option from the top. If compilation is successful you have just compiled a linux kernel module. Go to the directory it created the .ko file in and use that file to load into the linux kernel.

Appendix

```
/*
* File: testusb.c
* Author: Scott Hosier
*/

#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/usb.h>

MODULE_AUTHOR("Scott Hosier");
MODULE_LICENSE("GPL");


static int usb_notify(struct notifier_block *self, unsigned long action, void *dev)
{
printk(KERN_INFO "## usb_notify called \n");
switch (action)
{
case USB_DEVICE_ADD:
printk(KERN_INFO "# USB device added \n");
printk(KERN_INFO "# This is printed when a usb device is attached \n");
break;
case USB_DEVICE_REMOVE:
printk(KERN_INFO "# USB device removed \n");
printk(KERN_INFO "# This is printed when a usb device is removed \n");
break;
case USB_BUS_ADD:
printk(KERN_INFO "# USB Bus added \n");
break;
case USB_BUS_REMOVE:
printk(KERN_INFO "# USB Bus removed \n");
}
return NOTIFY_OK;
}

static struct notifier_block usb_nb = {
.notifier_call = usb_notify,
};


static int testusb_probe(struct usb_interface *interface, const struct usb_device_id *id)
{
printk("testusb: Probe module\n");
return 0;
}
```

```
static void testusb_disconnect(struct usb_interface *interface)
{
printk("testusb: Disconnect module\n");
}

static struct usb_driver testusb_driver = {
name : "testusb",
probe : testusb_probe,
disconnect : testusb_disconnect,
};

static int __init testusb_init(void)
{

int result;

result = usb_register(&testusb_driver);
if (result) {
printk("testusb: Registering driver failed");
} else {
printk("testusb: Driver registered successfully");
}

printk(KERN_INFO "Init USB hook.\n");

//Hook to the USB core to get notification on any addition or removal of USB devices
usb_register_notify(&usb_nb);
return 0;
}

static void __exit testusb_exit(void)
{

usb_deregister(&testusb_driver);
printk("testusb: Module deregistered");


//Remove the hook
usb_unregister_notify(&usb_nb);
printk(KERN_INFO "Remove USB hook\n");
}

module_init(testusb_init);
module_exit(testusb_exit);
```

Output

```
Dec 8 14:35:00 scott-laptop kernel: [10138.008360] usbcore: registered new interface driver testusb
Dec 8 14:35:00 scott-laptop kernel: [10138.011042] testusb: Driver registered successfully
Dec 8 14:35:00 scott-laptop kernel: [10138.011056] Init USB hook.
Dec 8 14:35:10 scott-laptop kernel: [10147.392061] usb 2-2: new low speed USB device using ohci_hcd and address 7
Dec 8 14:35:10 scott-laptop kernel: [10147.561456] usb 2-2: configuration #1 chosen from 1 choice
Dec 8 14:35:10 scott-laptop kernel: [10147.577192] input: Logitech USB Receiver as /devices/pci0000:00/0000:00:13.0/usb2/2-2/2-2:1.0/input/input13
Dec 8 14:35:10 scott-laptop kernel: [10147.578618] generic-usb 0003:046D:C51B.000B: input,hidraw0: USB HID v1.11 Mouse [Logitech USB Receiver] on usb-0000:00:13.0-2/input0
Dec 8 14:35:10 scott-laptop kernel: [10147.597890] generic-usb 0003:046D:C51B.000C: hiddev96,hidraw1: USB HID v1.11 Device [Logitech USB Receiver] on usb-0000:00:13.0-2/input1
Dec 8 14:35:10 scott-laptop kernel: [10147.598003] ## usb_notify called
Dec 8 14:35:10 scott-laptop kernel: [10147.598007] # USB device added
Dec 8 14:35:10 scott-laptop kernel: [10147.598011] # This is printed when a usb device is attached
Dec 8 14:35:24 scott-laptop kernel: [10161.140072] usb 2-2: USB disconnect, address 7
Dec 8 14:35:24 scott-laptop kernel: [10161.154766] ## usb_notify called
Dec 8 14:35:24 scott-laptop kernel: [10161.154773] # USB device removed
Dec 8 14:35:24 scott-laptop kernel: [10161.154777] # This is printed when a usb device is removed
Dec 8 14:35:30 scott-laptop kernel: [10167.332327] usbcore: deregistering interface driver testusb
Dec 8 14:35:30 scott-laptop kernel: [10167.334609] testusb: module deregistered
Dec 8 14:35:30 scott-laptop kernel: [10167.334623] Remove USB hook
```

Terminal Output

```
Module            Size      Used by
testusb           1541      0
binfmt_misc       6587      1
ppdev             5259      0
fbcon             35102     71
tileblit          1999      1 fbcon
font              7557      1 fbcon
bitblit           4707      1 fbcon
softcursor        1189      1 bitblit
vga16fb           11385     0
```