Lab 2 Instruction

* You can also find **similar** description on Page 96.

In this project, you will learn how to create a kernel module and load it into the Linux kernel. The project can be completed using the Linux virtual machine that is available with this text. Although you may use an editor to write these C programs, you will have to use the terminal application to compile the programs, and you will have to enter commands on the command line to manage the modules in the kernel.

As you'll discover, the advantage of developing kernel modules is that it is a relatively easy method of interacting with the kernel, thus allowing you to write programs that directly invoke kernel functions. It is important for you to keep in mind that you are indeed writing kernel code that directly interacts with the kernel. That normally means that any errors in the code could crash the system!

Creating Kernel Modules

Linux modules are pieces of code that can be loaded and unloaded into the kernel upon demand. They extend the functionality of the kernel without the need to reboot the system. For example, one type of module is the device driver, which allows the kernel to access hardware connected to the system. Without modules, we would have to build monolithic kernels and add new functionality directly into the kernel image. You can list all kernel modules that are currently loaded by entering the command

lsmod

This command will list the current kernel modules in three columns: name, size, and where the module is being used.

The following program (named simple.c) illustrates a very basic kernel module that prints appropriate messages when the kernel module is loaded and unloaded.

```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>

/* This function is called when the module is loaded. */

int simple init(void)
{
printk(KERN_INFO "Loading Module\n");
return 0;
}
```

```
/* This function is called when the module is removed. */

void simple exit(void)
{
printk(KERN_INFO "Removing Module\n");
}

/* Macros for registering module entry and exit points. */

module_init(simple init);

module_exit(simple exit);

MODULE_LICENSE("GPL");

MODULE_DESCRIPTION("Simple Module");

MODULE_AUTHOR("SGG");
```

A program usually begins with a main() function, executes a bunch of instructions and terminates upon completion of those instructions. Kernel modules work a bit differently. A module always begin with either the init_module or the function you specify with module_init call. This is the entry function for modules; it tells the kernel what functionality the module provides and sets up the kernel to run the module's functions when they're needed. Once it does this, entry function returns and the module does nothing until the kernel wants to do something with the code that the module provides.

The function simple_init() in simple.c is the module entry point, which represents the function that is invoked when the module is loaded into the kernel. Similarly, the simple_exit()function is the module exit point—the function that is called when the module is removed from the kernel.

The module entry point function must return an integer value, with 0 representing success and any other value representing failure. The module exit point function returns void. Neither the module entry point nor the module exit point is passed any parameters. The two following macros are used for registering the module entry and exit points with the kernel:

module_init()

module_exit()

Notice how both the module entry and exit point functions make calls to the printk() function. printk() is the kernel equivalent of printf(), yet its output is sent to a kernel log buffer whose contents can be read by the dmesg command. One difference between

printf() and printk() is that printk() allows us to specify a priority flag whose values are given in the <linux/printk.h> include file. In this instance, the priority is KERN_INFO, which is defined as an informational message.

The final lines—MODULE_LICENSE(), MODULE_DESCRIPTION(), and MODULE_AUTHOR()—represent details regarding the software license, description of the module, and author. For our purposes, we do not depend on this information, but we include it because it is standard practice in developing kernel modules.

Kernel modules need to be compiled a bit differently from regular user-space program. This kernel module simple.c is to be compiled by using the Makefile accompanying the source code with this project. Kernel module can only be compiled with Makefile. Former kernel versions required to care much about settings, which are usually stored in Makefiles. Although hierarchically organized, many redundant settings accumulated in sublevel Makefiles and made them large and rather difficult to maintain. Kbuild can do these things and build process for external loadable modules. It is now fully integrated into the standard kernel build mechanism.

To compile the module, enter the following on the command line, make sure your source file and Makefile are under the same directory:

make

The compilation produces several files. The file simple.ko represents the compiled kernel module. The following step illustrates inserting this module into the Linux kernel.
Loading and Removing Kernel Modules

Kernel modules are loaded using the insmod command, which is run as follows:

sudo insmod simple.ko

sudo allows a permitted user to execute a command as another user as specified by the security policy, by default the superuser. Since you are not a superuser on K200 you probably can not install the new module. K200 will denial access and show an acknowledgement. Alternative, you can try this in a Linux where you have superuser privilege.

To check whether the module has loaded, enter the lsmod command and search for the module simple. Recall that the module entry point is invoked when the module is inserted into the kernel. To check the contents of this message in the kernel log buffer, enter the command
dmesg

You should see the message "Loading Module." Removing the kernel module involves invoking the rmmod command (notice that the .ko suffix is unnecessary):

sudo rmmod simple

Be sure to check with the dmesg command to ensure the module has been removed. Because the kernel log buffer can fill up quickly, it often makes sense to clear the buffer periodically. This can be accomplished as follows:
sudo dmesg –c

Proceed through the steps described above to create the kernel module and to load and unload the module. Be sure to check the contents of the kernel log buffer using dmesg to ensure you have properly followed the steps.

Submission Instruction:

1. Substitute the file name with yourCSUNID.c . Change the file name in your makefile. Install the module after compilation. Take a screenshot of all files after compilation. (5 pts)
2. If it is installed take the screenshot after lsmod command and upload it with the previous screenshot in a PDF file and your source code in .c extension. If not due to access denial upload the screenshot of K200's acknowledgement with the previous screenshot in a PDF file along with your source code in .c extension. (25 pts)

The PDF file comprising your screenshots needs to be named following this format
YourCSUNID_YourLastName.pdf
and submitted here in Moodle.

Submission failed to meet the submission requirement will not be graded. Grade may be forfeited.