Exercise #2

The aim of this exercise is to get familiar with writing loadable kernel modules.

Task 1

Write a sample loadable kernel module and verify that it works (it should be loaded and removed). Print outputs from the module at a 'syslog' file and also the console.

Step 1. Write a kernel module

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

MODULE_LICENSE("GPL");
MODULE_AUTHOR("tlimkim");

static int __init hello_module(void)
{
    printk(KERN_ALERT "Hello Module! \n");
    return 0;
}

static void __exit bye_module(void)
{
    printk(KERN_ALERT "Bye Module! \n");
}

module_init(hello_module);
module_exit(bye_module);
```

Figure 1. simple_mod.c

1. '__init hello_module'

This function initiation function that called by command 'insmod' that loads this module on the kernel. We use 'printk( )' that works similar to 'printf( )', but it prints on the kernel functions and it can designate log level that for managing message or debugging.

2. '__exit bye_module'

This function called by the command 'rmmod' that remove this module from the kernel.

3. 'module_init( )', 'module_exit( )'

These functions work Macro that each of them are initiate or exit the module functions.

Step 2. Compile the module source file

```
obj-m += simple_mod.o

all:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules
```

Figure 2. Makefile

This is Makefile of module we made and it make object file(.o) and then kernel module file(.ko). 'obj-m' means that we make 'ko' file that is module not the built-in format.

Step 3. Compile the module.

```
tlimkim@tlimkim:~/workspace/kernel_programming/ex2_modules/taeklim$ make
make -C /lib/modules/4.15.1.ownsyscall/build M=/home/tlimkim/workspace/kernel_programming/ex2_
modules/taeklim modules
make[1]: Entering directory '/usr/src/linux-4.15.1'
  CC [M]  /home/tlimkim/workspace/kernel_programming/ex2_modules/taeklim/simple_mod.o
  Building modules, stage 2.
  MODPOST 1 modules
  LD [M]  /home/tlimkim/workspace/kernel_programming/ex2_modules/taeklim/simple_mod.ko
make[1]: Leaving directory '/usr/src/linux-4.15.1'
tlimkim@tlimkim:~/workspace/kernel_programming/ex2_modules/taeklim$ ls
Makefile        modules.order  simple_mod.ko     simple_mod.mod.o
Module.symvers  simple_mod.c   simple_mod.mod.c  simple_mod.o
tlimkim@tlimkim:~/workspace/kernel_programming/ex2_modules/taeklim$
```

Figure 3. Compiling the source file

Step 4. Load and exit the module and check the output at 'syslog', and console.

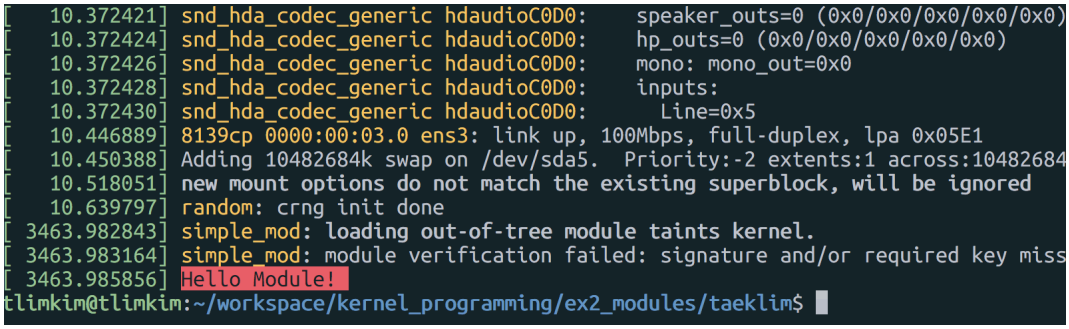

Figure 4. Console output from 'dmesg' after loading the module

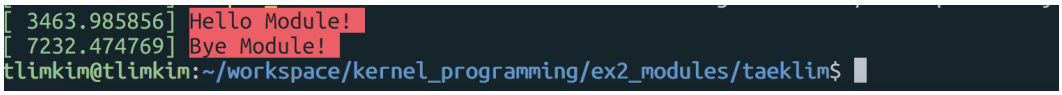

Figure 5. Console output after removing the module

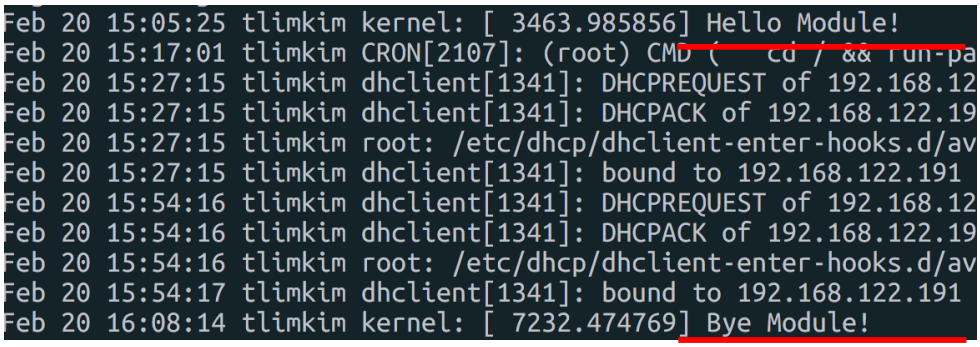

Figure 6. cat /var/log/syslog

Task 2

Extend the basic module to accept command line parameters. Add a simple conditional clause to do perform different tasks (e.g., print different outputs) based on input parameters.

Step 1. Write a kernel module

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

MODULE_LICENSE("GPL");
MODULE_AUTHOR("tlimkim");

static int cond = 1;

module_param(cond, int, 0);

static int __init mod_entry(void)
{
    printk(KERN_ALERT "Module Started \n");

    // Condition
    if (cond == 1) {
        printk("parameter: %d \n", cond);
    } else
        printk("wrong param \n");

    return 0;
}

static void __exit mod_exit(void)
{
    printk(KERN_ALERT "Module Exited");
}

module_init(mod_entry);
module_exit(mod_exit);
~
```

Figure 7. param_mod.c

The new thing in this source file is that 'module_param'. This macro works like a parameter so that we can change the value of parameter outside of the module. In '__init mod_entry', we made a conditional branch so that module can check the parameter's value.

Step 2. Compile and Load the module and check the output

```
[   62.777353] Module Started
[   62.777359] parameter: 1
tlimkim@tlimkim:~/workspace/kernel_programming/ex2_modules/taeklim/mod_parm$
```

Figure 8. Console output from module 'param_mod'