Jordan University of Science and Technology
CS375 Operating systems - Assignment 2
Spring 2019-2020

Objectives:

Student should be able to:

- ✓ create a kernel module and load it into the Linux kernel used in Assignment 1.
- ✓ modifying the kernel module so that it uses the kernel linked-list data structure.

What to submit: Due: March 16, 2020

- ✓ You have to work your assignment as a team with at MOST TWO members.
- ✓ At least one of the group members has to submit the report on time.
- ✓ The file name of your report has to be formatted as CS375_Ass#2_X_Y.doc (or .docx or .Pdf) where **X** is the first student ID number and **Y** is the second student ID number.
- ✓ You should submit a short report in **English** that includes the following:
  - o Take a screen shot for each step (point) to support each of your answers.
  - o For each print screen, write a description (**in your words**) for the output.
  - o Submit the code for the two parts along with the report.
  - o Describe the problems that you faced during these steps and how did you solve these problems.
  - o Write a full step by step procedure that you follow to add the module to Linux (Summarization).

After submitting your work, you should schedule an appointment. Check the discussion date(s) on the e-learning to discuss your work **on your personal computer.** Your grade will be given based on your both submission and discussion. You are expected to demonstrate any related question(s) **without refereeing to any supporting material during the discussion.**

Edited: March 7, 2020

Jordan University of Science and Technology
CS375 Operating systems - Assignment 2
Spring 2019-2020

Description

Refer to the textbook for Chapter 2 Programming Project: Linux Kernel Module on page 96 of the 9th edition.

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 *or the one you've created in assignment no. 1*. 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! However, since you will be using a virtual machine, any failures will at worst *only require rebooting the system.*

Part I – Creating Kernel Modules

Follow the instructions on the book and answer the following questions. **Take a screen shot for each step (point) to support each of your answers.**

1. First, enter the **lsmod** command on your Linux machine. What's the output on the screen? Answer: _____.

2. Use **vi** (or you can use **gedit**) to create the following **Makefile** file.

```
obj-m += simple.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
```

3. Similarly, create the following program, named **simple.c** using **vi** (or **gedit**).

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

This kernel module **simple.c** is compiled using the **Makefile** that we just created. To compile the module, enter the following on the command line:

make

The compilation produces several files. List all the files, and which of them represents the compiled kernel module?

Answer: _____.

4. The following step illustrates inserting this module into the Linux kernel.

sudo insmod simple.ko
dmesg

What message do you see? Answer: _____.

5. Removing the kernel module by

sudo rmmod simple

Use the **dmesg** command to ensure the module has been removed.

Answer: _____.

6. Last, clear the kernel log buffer as follows:

sudo dmesg -c

What is the purpose to clear the kernel log buffer?

Answer: _____.

Part II –Kernel Data Structures

The second part of this project involves modifying the kernel module so that it uses the kernel linked-list data structure. Read and follow the instructions below.

7. Use **vi** (or **gedit**) to create the following the **Makefile** file.

```
obj-m += simple-solution.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
```

8. Similarly, create the following program, named **simple-solution.c** using **vi** (or **gedit**)**.**

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

```
struct birthday
{
        int month;
        int day;
        int year;

      struct list_head list;
};

/**
 * The following defines and initializes a list_head object named birthday_list
 */
static LIST_HEAD(birthday_list);

int simple_init(void)
{
        /* the pointer for memory allocation */
        struct birthday *person_one;

        /* the pointer for list traversal */
        struct birthday *ptr;

      printk(KERN_INFO "Loading Module\n");

      person_one = kmalloc(sizeof(*person_one), GFP_KERNEL);
      person_one->month = 8;
      person_one->day = 13;
      person_one->year = 1995;
        INIT_LIST_HEAD(&person_one->list);

      /* add the new node */
        list_add_tail(&person_one->list, &birthday_list);

      person_one = kmalloc(sizeof(*person_one), GFP_KERNEL);
      person_one->month = 9;
      person_one->day = 2;
      person_one->year = 1998;
        INIT_LIST_HEAD(&person_one->list);

      /* add the new node */
        list_add_tail(&person_one->list, &birthday_list);

      person_one = kmalloc(sizeof(*person_one), GFP_KERNEL);
      person_one->month = 8;
      person_one->day = 12;
      person_one->year = 1963;
        INIT_LIST_HEAD(&person_one->list);
```

```
        /* add the new node */
        list_add_tail(&person_one->list, &birthday_list);

      person_one = kmalloc(sizeof(*person_one), GFP_KERNEL);
      person_one->month = 10;
      person_one->day = 22;
      person_one->year = 1963;
        INIT_LIST_HEAD(&person_one->list);

        /* add the new node */
        list_add(&person_one->list, &birthday_list);

      /* now traverse the list */

      list_for_each_entry(ptr, &birthday_list, list) {
            printk(KERN_INFO "Birthday: Month %d Day %d Year %d\n",ptr->month,ptr->day,ptr-
>year);
      }

      return 0;
}

void simple_exit(void) {
        struct  birthday *ptr, *next;

        printk(KERN_INFO "Removing Module\n");

        /* remove allocated memory */
        list_for_each_entry_safe(ptr, next, &birthday_list, list) {
           printk(KERN_INFO "Removing %d %d %d\n",ptr->month, ptr->day, ptr->year);
           list_del(&ptr->list);
           kfree(ptr);
        }

}

module_init( simple_init );
module_exit( simple_exit );

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Kernel Data Structures");
MODULE_AUTHOR("SGG");
```

9. Following the same instructions in part I to load and remove this new kernel module as follows.

```
make
sudo insmod simple-solution.ko
dmesg
```

In particular, what messages were output from the run?

Answer: _____.

10. Removing the kernel module by

```
sudo rmmod simple-solution
```

Use the **dmesg** command to ensure the module has been removed.

Answer: _____.

11. Last, clear the kernel log buffer as

```
sudo dmesg -c
```