Zagazig University

Faculty of Engineering

Computer and Systems Dept.

Linux Kernel

Project #1 for the course

Operating Systems

To

Dr. Ahmed Amer Shahi

By

Ahmed Moustafa ABD-Elfattah Mohammed Ramadan

20812017100166@eng.zu.edu.eg

2020-2021

Introduction

Linux Kernel is designed by the methodology of Loadable Kernel Modules. This design approach involves that the kernel has a set of core components and can link in additional services via modules. Linux uses loadable kernel modules to support device drivers and file systems. This approach allows an administrator to add functionality only when required. Keeping only what is necessary for kernel memory reduces the kernel's memory footprint and increases its overall performance.

This programming project will introduce dealing with loadable modules. So, in this project, we will be able to develop a kernel module and inject it into the kernel. We will start first by introducing the different steps showed in the project description then showing the actual assignment attaches with some comments.

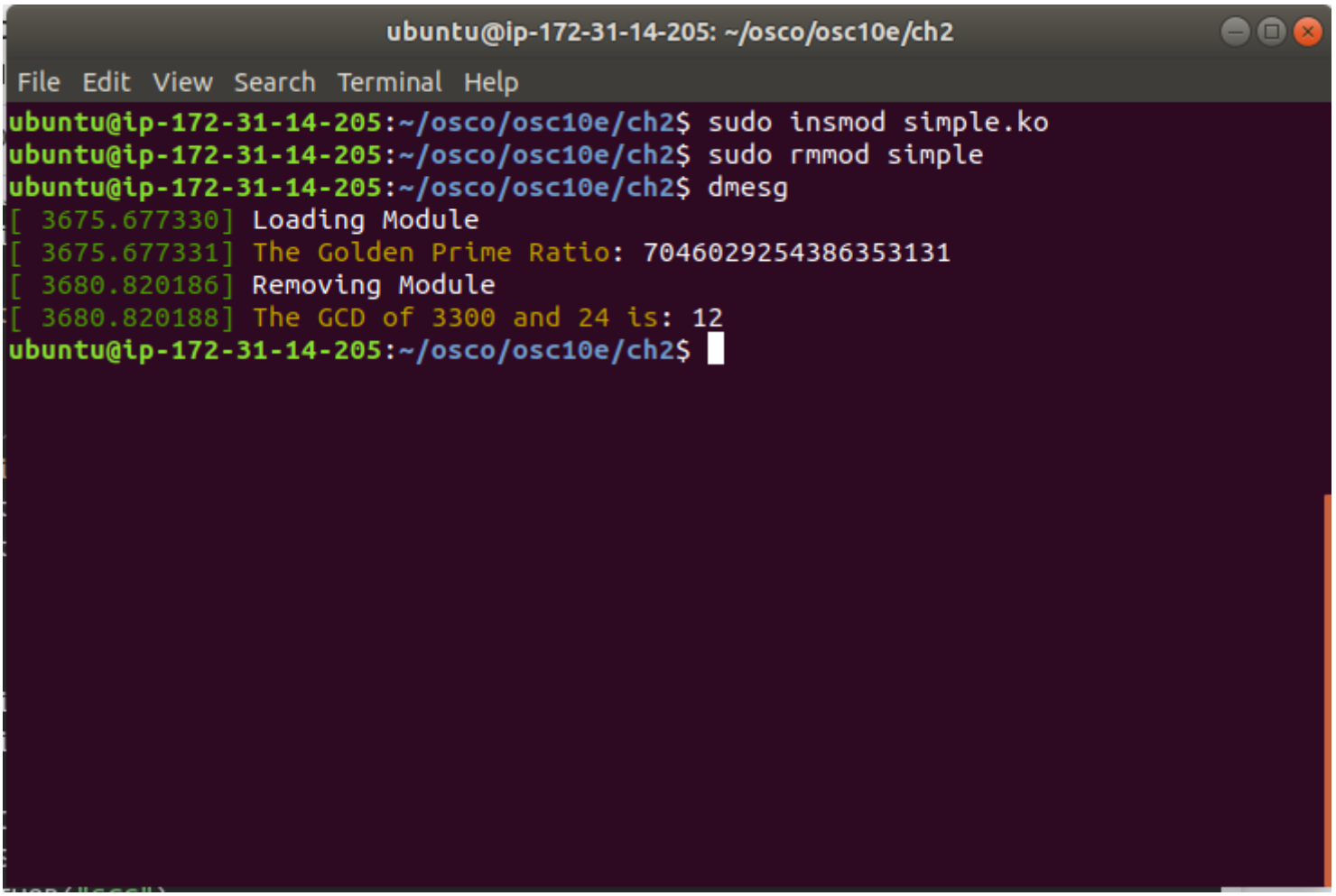
Part 1: Loading and Removing Kernel Modules

A- Printing the golden ratio prime and the gcd.

Code:

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/hash.h>
#include <linux/gcd.h>
/* This function is called when the module is loaded. */
static int simple_init(void)
{
       printk(KERN_INFO "Loading Module\n");
       printk(KERN_INFO "The Golden Prime Ratio: %lu\n", GOLDEN_RATIO_PRIME);

       return 0;
}

/* This function is called when the module is removed. */
static void simple_exit(void) {
     printk(KERN_INFO "Removing Module\n");
     printk(KERN_INFO "The GCD of 3300 and 24 is: %lu\n", gcd(3300, 24));
}

/* 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("Ahmed Moustafa");
```

Preview:



Comment:

In this section, we have started playing with the kernel module by adding one then removing it. This kernel module has two roles only. The first one is to print the GOLDEN_PRIME_RATIO which is found in the “linux/hash.h”, the second role is to print the greatest common divisor of 3300 and 24 using the function gcd found in “linux/gcd.h”. The former is done in the initialization of the module and the second one at the exit. The last picture also provides information about the commands to add and remove modules from the kernel using insmod and rmmod. dmesg is used to show the kernel log.

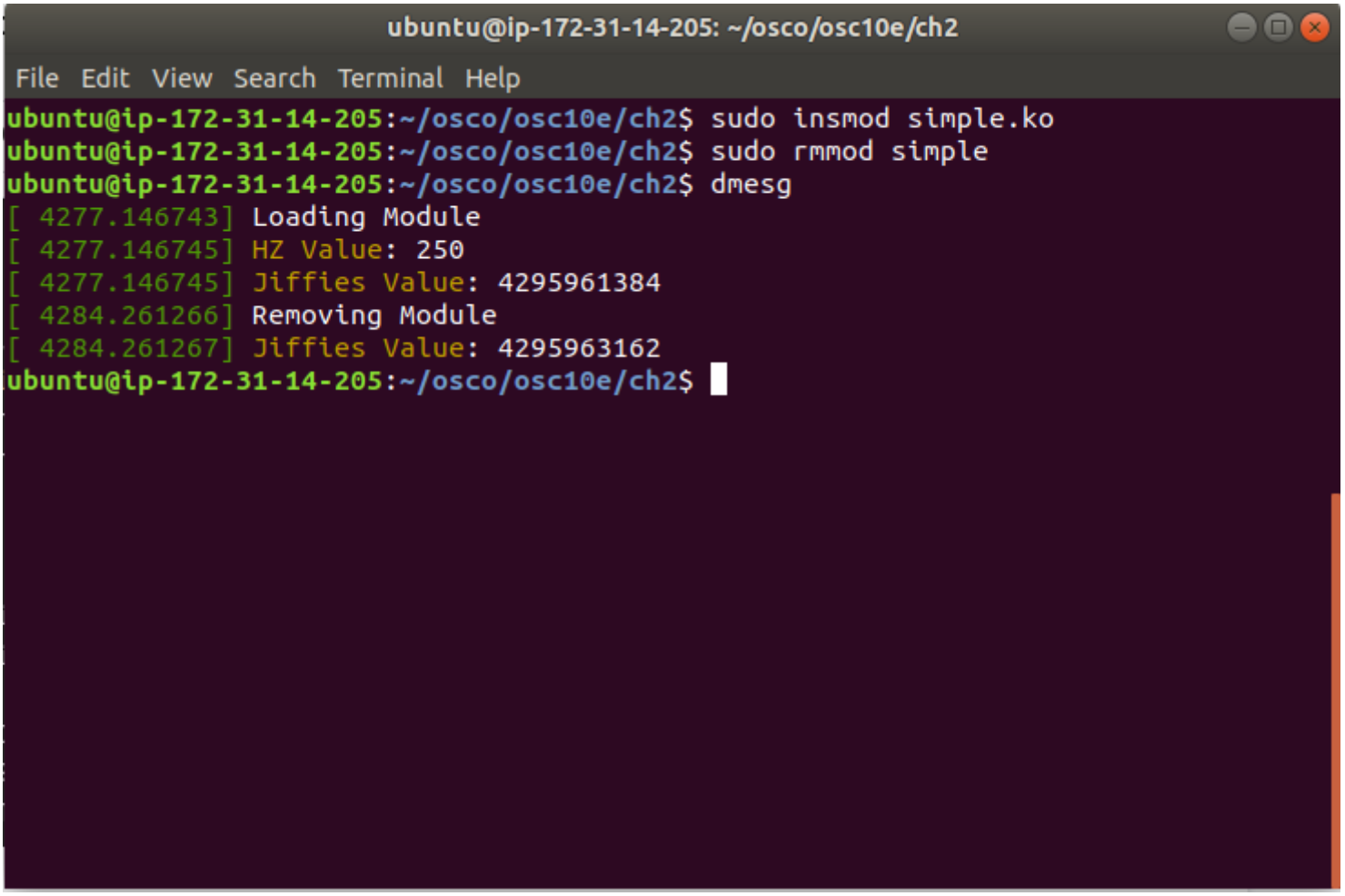
B- Printing the values of jiffies and HZ.

Code:

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/jiffies.h>
#include <asm/param.h>
/* This function is called when the module is loaded. */
static int simple_init(void)
{
        printk(KERN_INFO "Loading Module\n");
     printk(KERN_INFO "HZ Value: %lu\n", HZ);
     printk(KERN_INFO "Jiffies Value: %lu\n",jiffies);
```

```c
return 0;
}

/* This function is called when the module is removed. */
static void simple_exit(void) {
      printk(KERN_INFO "Removing Module\n");
      printk(KERN_INFO "Jiffies Value: %lu\n", jiffies);
}

/* 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("Ahmed Moustafa");
```

Preview:



Comment:

Here we have used two libraries for the linux kernel in order to get the values of the timer interrupt frequency (HZ) and the number of interrupts (jiffies). In order to calculate the number of seconds elapsed since the kernel module was loaded, we should subtract both values of jiffies at init and exit then divide the result by HZ.

Part 2: Assignment

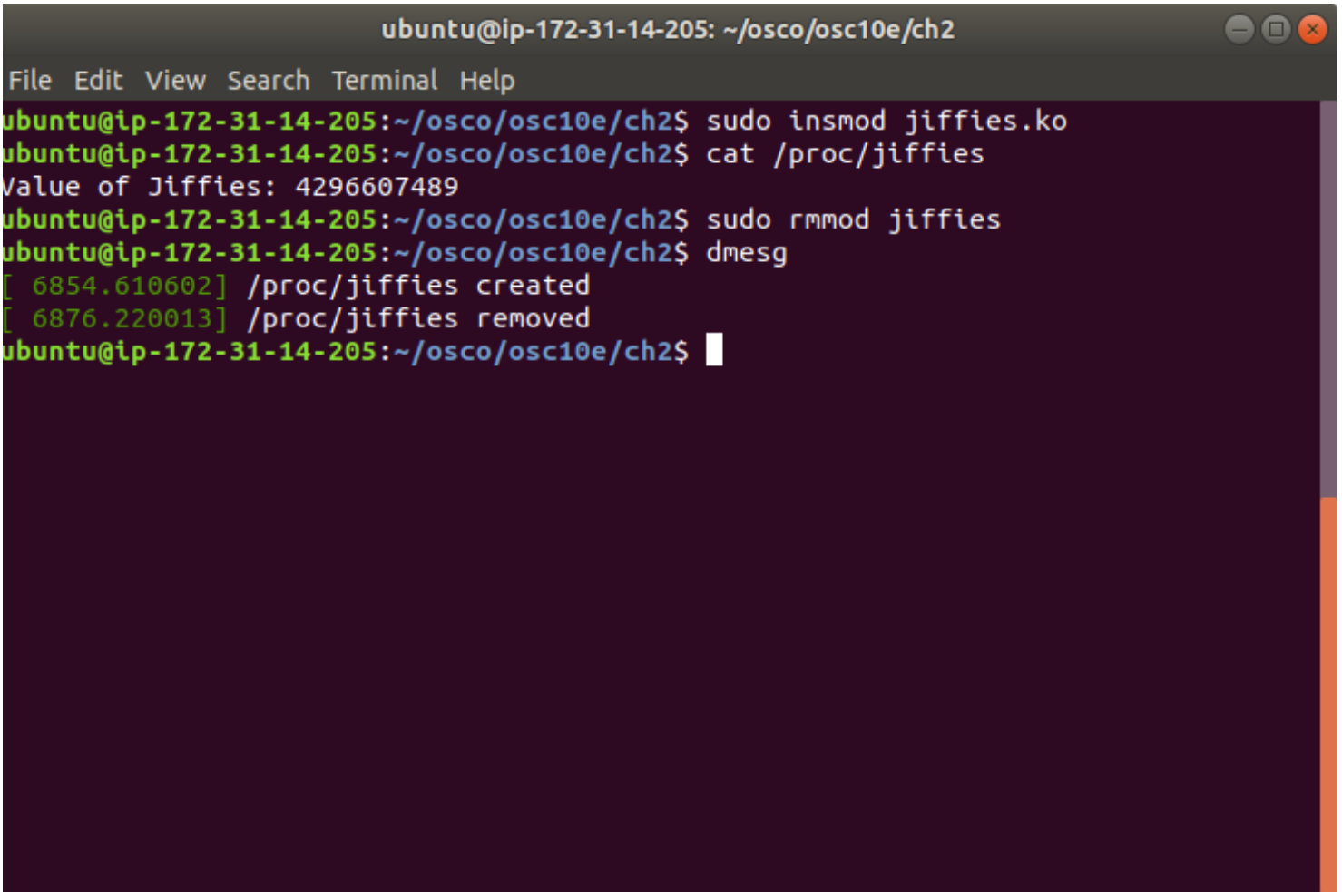
A- Design a kernel module that creates a /proc file named /proc/jiffies that reports the current value of jiffies.

Code:

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <asm/uaccess.h>
#include <linux/jiffies.h>

#define BUFFER_SIZE 128

#define PROC_NAME "jiffies"

/**
 * Function prototypes
 */
static ssize_t proc_read(struct file *file, char *buf, size_t count, loff_t *pos);

static struct file_operations proc_ops = {
        .owner = THIS_MODULE,
        .read = proc_read,
};


/* This function is called when the module is loaded. */
static int proc_init(void)
{

        // creates the /proc/jiffies entry
        // the following function call is a wrapper for
        // proc_create_data() passing NULL as the last argument
        proc_create(PROC_NAME, 0, NULL, &proc_ops);

        printk(KERN_INFO "/proc/%s created\n", PROC_NAME);

     return 0;
}

/* This function is called when the module is removed. */
static void proc_exit(void) {

        // removes the /proc/jiffies entry
        remove_proc_entry(PROC_NAME, NULL);

        printk( KERN_INFO "/proc/%s removed\n", PROC_NAME);
}
```

```c
/**
 * This function is called each time the /proc/jiffies is read.
 *
 * This function is called repeatedly until it returns 0, so
 * there must be logic that ensures it ultimately returns 0
 * once it has collected the data that is to go into the
 * corresponding /proc file.
 *
 * params:
 *
 * file:
 * buf: buffer in user space
 * count:
 * pos:
 */
static ssize_t proc_read(struct file *file, char __user *usr_buf, size_t count,
loff_t *pos)
{
        int rv = 0;
        char buffer[BUFFER_SIZE];
        static int completed = 0;

        if (completed) {
                completed = 0;
                return 0;
        }

        completed = 1;

        rv = sprintf(buffer, "Value of Jiffies: %lu", jiffies);

        // copies the contents of buffer to userspace usr_buf
        raw_copy_to_user(usr_buf, buffer, rv);

        return rv;
}


/* Macros for registering module entry and exit points. */
module_init( proc_init );
module_exit( proc_exit );

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Jiffies Module");
MODULE_AUTHOR("Ahmed Moustafa");
```

Preview:



Comment:

Using the already designed code in the textbook, we have added to it the jiffies printing functionality which has been introduced before.

B- Design a kernel module that creates a proc file named /proc/seconds that reports the number of elapsed seconds since the kernel module was loaded.

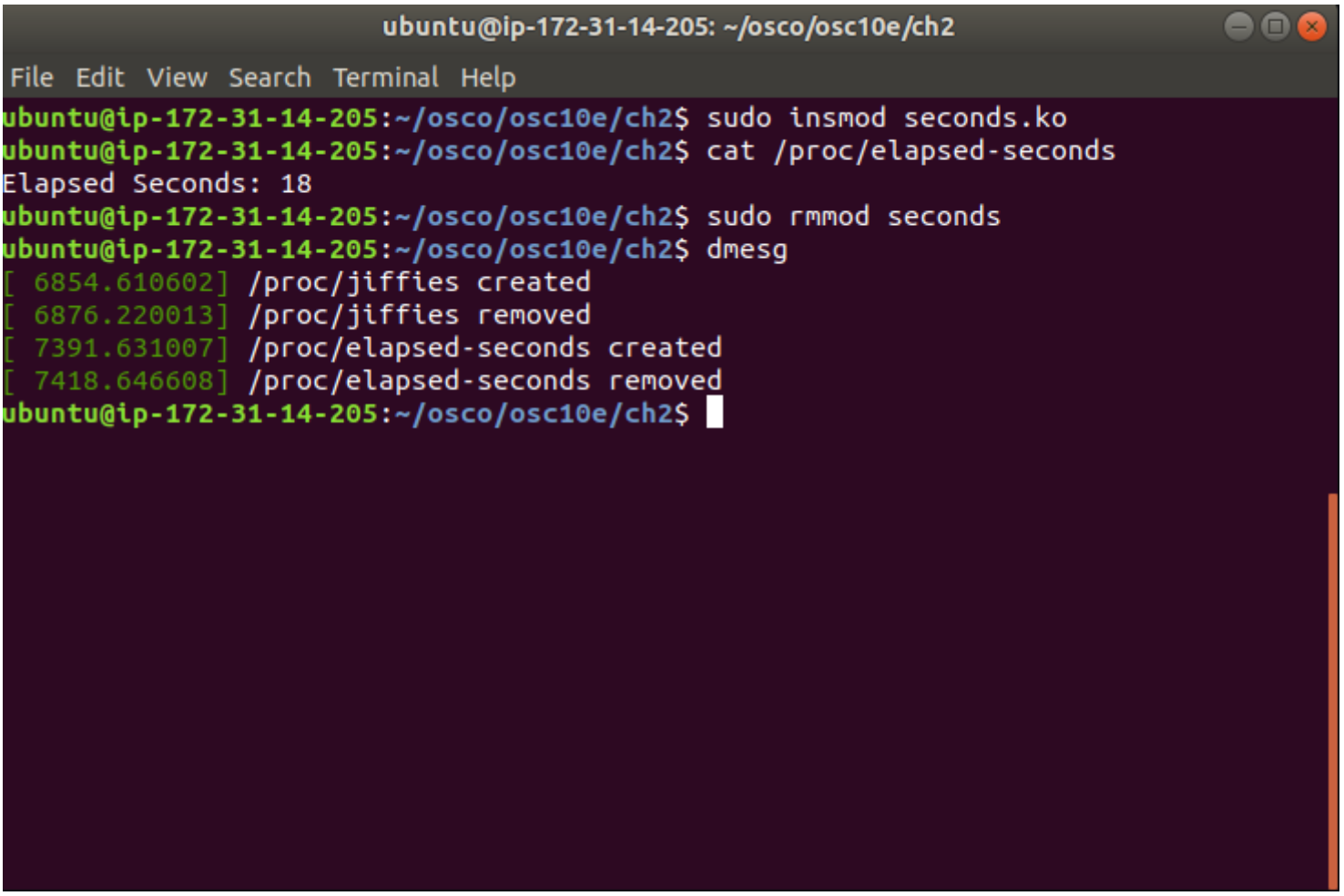
Code:

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <asm/uaccess.h>
#include <linux/jiffies.h>
#include <asm/param.h>

#define BUFFER_SIZE 128

#define PROC_NAME "elapsed-seconds"

unsigned long current_jiffies = 0;
/**
 * Function prototypes
 */
```

```
static ssize_t proc_read(struct file *file, char *buf, size_t count, loff_t *pos);

static struct file_operations proc_ops = {
        .owner = THIS_MODULE,
        .read = proc_read,
};


/* This function is called when the module is loaded. */
static int proc_init(void)
{

        // creates the /proc/elapsed-seconds entry
        // the following function call is a wrapper for
        // proc_create_data() passing NULL as the last argument
        proc_create(PROC_NAME, 0, NULL, &proc_ops);
           current_jiffies = jiffies;
        printk(KERN_INFO "/proc/%s created\n", PROC_NAME);

     return 0;
}

/* This function is called when the module is removed. */
static void proc_exit(void) {

        // removes the /proc/elapsed-seconds entry
        remove_proc_entry(PROC_NAME, NULL);
        printk( KERN_INFO "/proc/%s removed\n", PROC_NAME);
}

/**
 * This function is called each time the /proc/elapsed-seconds is read.
 *
 * This function is called repeatedly until it returns 0, so
 * there must be logic that ensures it ultimately returns 0
 * once it has collected the data that is to go into the
 * corresponding /proc file.
 *
 * params:
 *
 * file:
 * buf: buffer in user space
 * count:
 * pos:
 */
static ssize_t proc_read(struct file *file, char __user *usr_buf, size_t count,
loff_t *pos)
{
        int rv = 0;
```

```
        char buffer[BUFFER_SIZE];
        static int completed = 0;

        if (completed) {
                completed = 0;
                return 0;
        }

        completed = 1;

        rv = sprintf(buffer, "Elapsed Seconds: %lu\n", (jiffies - current_jiffies) /
HZ);

        // copies the contents of buffer to userspace usr_buf
        raw_copy_to_user(usr_buf, buffer, rv);

        return rv;
}


/* Macros for registering module entry and exit points. */
module_init( proc_init );
module_exit( proc_exit );

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("elapsed-seconds Module");
MODULE_AUTHOR("Ahmed Moustafa");
```

Preview:



Comment:

Using the already designed code in the textbook, we have added to it the elapsed time calculation functionality which has been introduced before.