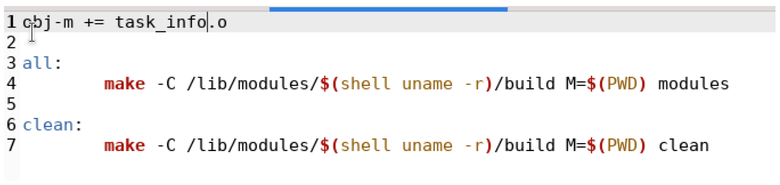
CSC462 Hw#1

Overal Summary experience:

After completing the the loadable kernel modules assignment I feel I gain a better sense on how to work with the linux terminal up closed as before this I few experience performing linux command on the shell. Also I gain better understanding why people would use loadable kernel design for their operating system. Such as I could easily create an implemented different kernels modules like jiffies module, seconds module, and pip module, which allow me to extend the kernels functionality with out having to reboot the system like a monolithic kernel design. I learn how to effiencetly load and unload kernel module with linux command “sudo insmod /path/to/my_module.ko” to load kernel module and to use “sudo rmmod my_module” to unload or remove the module freeing up the system resources. I saw how kernels modules can be developed independently than rest of the kernel such as when my kernel module program crash due to bugs I didn’t had to reboot the whole operating system.

Initial Setup:
For the assignment I used UTM virtual machine that I had downloaded from my previous course csc362 operating system, which was necessary because we are working with program involving with kernel, which a mistake can permanently mess up your main operating system. I created three directory folder. Within each folder their is a kernel module that is program in C along with a Makefile to compile the kernel module.



```
obj-m += task_info.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
```

MakeFile code

Jiffies Kernel Module:
For the functionality for this kernel module was to created and show jiffies. Which a jiffies is a kernel variable that holds the number clock tick since the system has been booted. Where we can access the jiffies when we type the linux command “ cat /proc/jiffies”. The operation in my program I use to achieved this was to defined a ‘proc_ops’ structure containing the ‘.proc_read’ function.

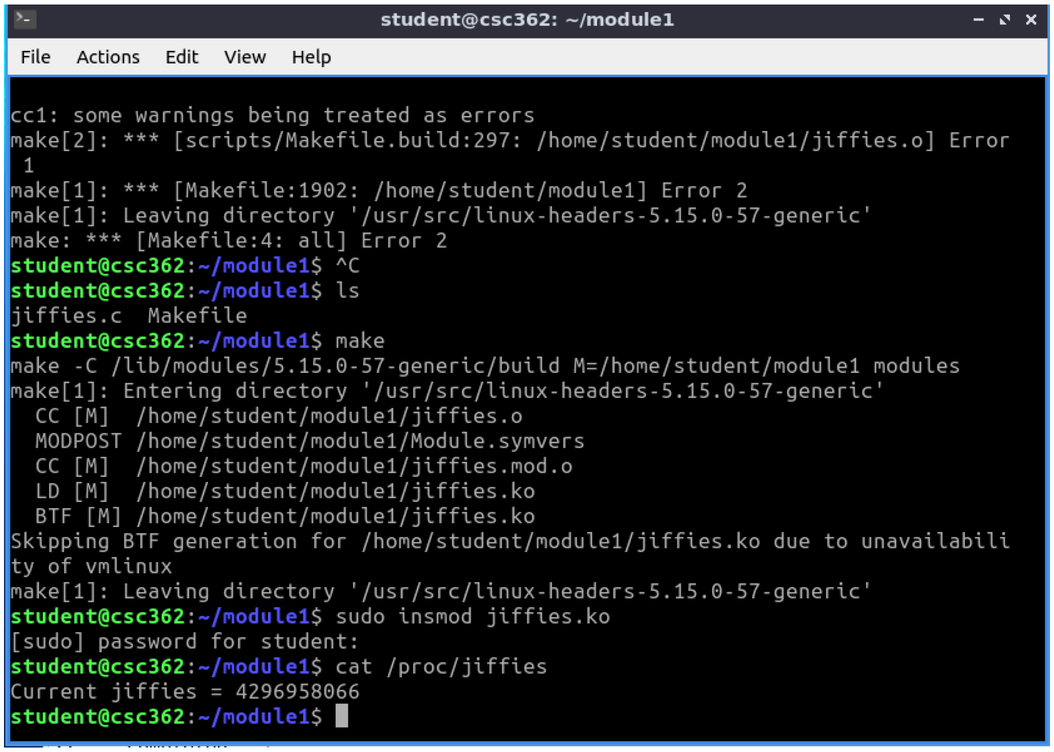
```
static struct proc_ops my_fops = {
    .proc_read = proc_read,
};
```

Used ‘module_init’ and ‘module_exit’ to initialization and cleanup. Implemented ‘proc_read’ to fill a user space buffer with the current value of jiffies.

```
rv = snprintf(buffer, sizeof(buffer), "Current jiffies = %lu\n", jiffies);
```

Used ‘proc_create’ and ‘proc_remove’ functions to manage the ‘/proc/jiffies’ file.

Below is jiffies kernel Module loaded and run on Linux terminal:



```
cc1: some warnings being treated as errors
make[2]: *** [scripts/Makefile.build:297: /home/student/module1/jiffies.o] Error
 1
make[1]: *** [Makefile:1902: /home/student/module1] Error 2
make[1]: Leaving directory '/usr/src/linux-headers-5.15.0-57-generic'
make: *** [Makefile:4: all] Error 2
student@csc362:~/module1$ ^C
student@csc362:~/module1$ ls
jiffies.c  Makefile
student@csc362:~/module1$ make
make -C /lib/modules/5.15.0-57-generic/build M=/home/student/module1 modules
make[1]: Entering directory '/usr/src/linux-headers-5.15.0-57-generic'
  CC [M]  /home/student/module1/jiffies.o
  MODPOST /home/student/module1/Module.symvers
  CC [M]  /home/student/module1/jiffies.mod.o
  LD [M]  /home/student/module1/jiffies.ko
  BTF [M] /home/student/module1/jiffies.ko
Skipping BTF generation for /home/student/module1/jiffies.ko due to unavailabili
ty of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-5.15.0-57-generic'
student@csc362:~/module1$ sudo insmod jiffies.ko
[sudo] password for student:
student@csc362:~/module1$ cat /proc/jiffies
Current jiffies = 4296958066
student@csc362:~/module1$ 
```

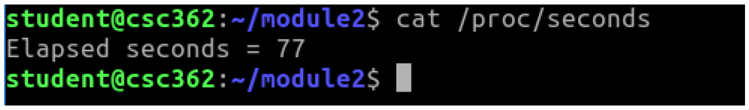
Seconds kernel Module:
For the functionality for this kernel module was to build a kernel module that creates a file named ‘procs/seconds’, which reports the number of seconds elapsed since the modules was first loaded. I achieved this by having a global variable ‘loaded_jiffies’ that stored the jiffies value at the time the module was loaded.

```
static unsigned long loaded_jiffies;
```

I implemented a ‘proc_read’ function that calculates the elapsed seconds.

```
 elapsed_seconds = (jiffies - loaded_jiffies) / HZ;
```

Below is second kernel Module loaded and run on Linux terminal:



```
student@csc362:~/module2$ cat /proc/seconds
Elapsed seconds = 77
student@csc362:~/module2$ 
```

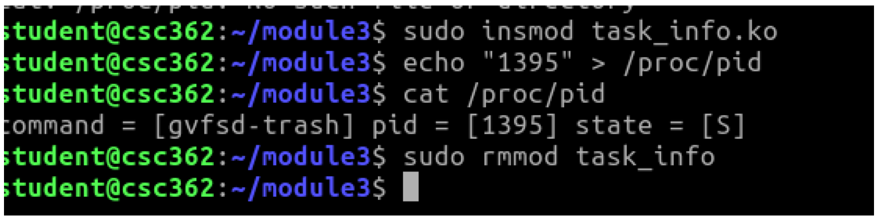
PID kernel Module:
For the functionality for this kernel module was to build a kernel module that use the /proc file system for displaying a task’s information based on its process identifier value pid. I achieved this by writing to ‘proc/pid’ were I utilized ‘kmalloc’ and 'kfree’ functions for kernel space memory management. ‘copy_from_user’ was used to transfer data from user space to kernel space.

```
ssize_t proc_write(struct file *file, const char __user *usr_buf, size_t count, loff_t *pos)
    char *k_mem = kmalloc(count, GFP_KERNEL);

    if (!k_mem) {
        return -ENOMEM;
    }

    if (copy_from_user(k_mem, usr_buf, count)) {
        kfree(k_mem);
        return -EFAULT;
    }

    kstrtol(k_mem, 10, &stored_pid);
    kfree(k_mem);
    return count;
}
```

In the read function I used ‘rcu_read_lock’ and rcu_read_unlock’ to manage concurrency. I used the ‘pid_task’ and ‘find_vpid’ functions to find the ‘task_struct’ which correspond to the given PID.

```
    completed = 1;

    rcu_read_lock();
    task = pid_task(find_vpid(stored_pid), PIDTYPE_PID);
    if(task) {
        rv = snprintf(buffer, sizeof(buffer), "command = [%s] pid = [%ld] state = [%c]\n",
                      task->comm, stored_pid, task_state_to_char(task)); // Use
ask_state_to_char for state
    } else {
        rv = snprintf(buffer, sizeof(buffer), "PID not found\n");
    }
    rcu_read_unlock();
```

Below is PID kernel Module loaded and run on Linux terminal:



```
tudent@csc362:~/module3$ sudo insmod task_info.ko
tudent@csc362:~/module3$ echo "1395" > /proc/pid
tudent@csc362:~/module3$ cat /proc/pid
ommand = [gvfsd-trash] pid = [1395] state = [S]
tudent@csc362:~/module3$ sudo rmmod task_info
tudent@csc362:~/module3$
```

Overall after completing PID kernel module I gain better knowledge of the linux process management system such as preventing memory leak with kmalloc() allocate memory and kfree() to released memory.