Introduction to Kernel Modules()

Paul D. Camarata

July 24, 2018

Abstract

"I do not fear computers. I fear lack of them." -Isaac Asimov

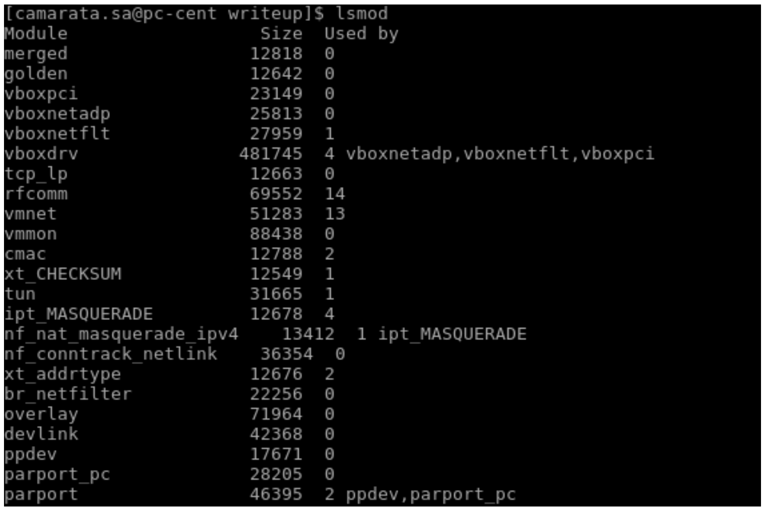
1 Introduction

The purpose of this programming project was to gain a better understanding of kernel modules. It is broken apart into four sections, each of which was a different lesson on different aspects of how an operating system sees and uses kernel modules.

2 Part I: Kernel Modules Overview

Part I is dedicated to listing and compiling of kernel modules. The images included are here to reference the work that was done locally.

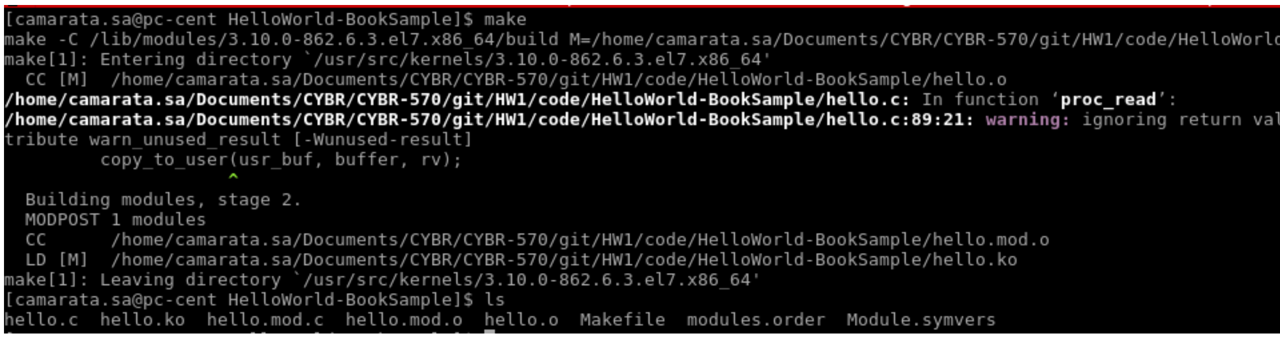
Figure 12 is a reference to running the 'lsmod' command. I have shortened the output, but this command showed that there are many modules currently running on my system.



```
[camarata.sa@pc-cent writeup]$ lsmod
Module                  Size  Used by
merged                 12818  0
golden                 12642  0
vboxpci                23149  0
vboxnetadp             25813  0
vboxnetflt             27959  1
vboxdrv               481745  4 vboxnetadp,vboxnetflt,vboxpci
tcp_lp                 12663  0
rfcomm                 69552  14
vmnet                  51283  13
vmmon                  88438  0
cmac                   12788  2
xt_CHECKSUM            12549  1
tun                    31665  1
ipt_MASQUERADE         12678  4
nf_nat_masquerade_ipv4    13412  1 ipt_MASQUERADE
nf_conntrack_netlink    36354  0
xt_addrtype            12676  2
br_netfilter           22256  0
overlay                71964  0
devlink                42368  0
ppdev                  17671  0
parport_pc             28205  0
parport                46395  2 ppdev,parport_pc
```

Figure 1: lsmod

Figure 2 is a reference to the 'make' command. Upon further examination of the make file, it looks like it is grabbing my currently loaded kernel via 'uname -r' and leveraging that as conditional input to create the binary that will be loaded later.

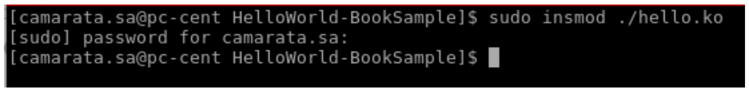


```
[camarata.sa@pc-cent HelloWorld-BookSample]$ make
make -C /lib/modules/3.10.0-862.6.3.el7.x86_64/build M=/home/camarata.sa/Documents/CYBR/CYBR-570/git/HW1/code/HelloWorl
make[1]: Entering directory `/usr/src/kernels/3.10.0-862.6.3.el7.x86_64'
  CC [M]  /home/camarata.sa/Documents/CYBR/CYBR-570/git/HW1/code/HelloWorld-BookSample/hello.o
/home/camarata.sa/Documents/CYBR/CYBR-570/git/HW1/code/HelloWorld-BookSample/hello.c: In function 'proc_read':
/home/camarata.sa/Documents/CYBR/CYBR-570/git/HW1/code/HelloWorld-BookSample/hello.c:89:21: warning: ignoring return val
tribute warn_unused_result [-Wunused-result]
         copy_to_user(usr_buf, buffer, rv);
                     ^
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/camarata.sa/Documents/CYBR/CYBR-570/git/HW1/code/HelloWorld-BookSample/hello.mod.o
  LD [M]  /home/camarata.sa/Documents/CYBR/CYBR-570/git/HW1/code/HelloWorld-BookSample/hello.ko
make[1]: Leaving directory `/usr/src/kernels/3.10.0-862.6.3.el7.x86_64'
[camarata.sa@pc-cent HelloWorld-BookSample]$ ls
hello.c  hello.ko  hello.mod.c  hello.mod.o  hello.o  Makefile  modules.order  Module.symvers
```

Figure 2: make

3 Part II: Loading and Removing Kernel Modules

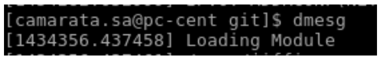
If the title of this section doesn't give it away, then I'm not sure how else I can explain this section. Here we are taking the resulting file(s) from the previous 'make' execution, specifically the '.ko' file and loading it as a kernel module via the 'insmod {module_file_name}' command.



```
[camarata.sa@pc-cent HelloWorld-BookSample]$ sudo insmod ./hello.ko
[sudo] password for camarata.sa:
[camarata.sa@pc-cent HelloWorld-BookSample]$ 
```

Figure 3: insmod

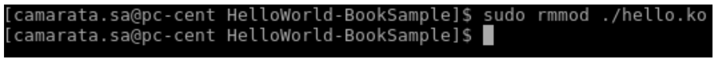
Executing this command adds a log in the 'dmesg' buffer which we examine in Figure 4.



```
[camarata.sa@pc-cent git]$ dmesg
[1434356.437458] Loading Module
```

Figure 4: dmesg - add

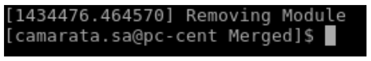
We then remove the module using 'rmmod {module_name}' command...



```
[camarata.sa@pc-cent HelloWorld-BookSample]$ sudo rmmod ./hello.ko
[camarata.sa@pc-cent HelloWorld-BookSample]$ 
```

Figure 5: rmmod

And then we examine 'dmesg' a final time to view the exit message.

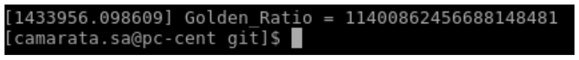


```
[1434476.464570] Removing Module
[camarata.sa@pc-cent Merged]$ █
```

Figure 6: dmesg - remove

From here we start to write some of our own code. All code from this assignment is included here as an appendix as well as delivered as an additional document for the instructor's convenience. As a side note, all deliverables for this assignment are also available via https://github.com/paulcamarata/CYBR-570. Rather than step through every component that needed to be done to complete this portion, I will include the parts that I believe are key and let the included code speak for how this was accomplished.

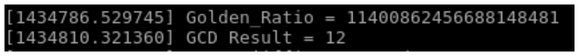
The initial goal was to print the GOLDEN_RATIO_PRIME as part of the module initialization. Figure 7 displays the results when executing this. The one thing to note is that this value appears to be different than what was expected. I worked on this partnered with a couple of other students and this 'delta' had me dig a little bit deeper. I figured the discrepancy existed because I was using a different operating system then my fellow students, but I wanted to understand why my value was so much different, considering this value seemed like it should have been a constant to me. Sure enough, when I dug into the hash.h header file, there were two constants declared, GOLDEN_Ratio_32 and GOLDEN_RATIO_64. When I translated the value of them from hex to binary, the numbers matched up. Also, since I was leveraging a 64 bit kernel, I was defaulting to a value that was different from my classmates. I also tried this on a 64 bit Ubuntu machine, and the value was different yet again. I did not do further research to understand why the 64 bit value is different, even among other seemingly similar operating systems.



```
[1433956.098609] Golden_Ratio = 11400862456688148481
[camarata.sa@pc-cent git]$ █
```

Figure 7: GOLDEN_RATIO

The last portion of part 2 was to have the 'gcd' function return the greatest common denominator of 3,300 and 24. The results of this are below in Figure 8.



```
[1434786.529745] Golden_Ratio = 11400862456688148481
[1434810.321360] GCD Result = 12
```

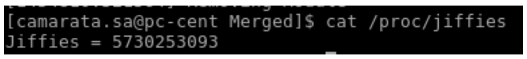
Figure 8: GCD

4 Part III

Part 3 was all about learning how to get something that is dynamically read when called in the proc virtual file system (VFS). This VFS is used to convey all sorts of information that is generally made available by the kernel. The setup of this portion was not part of the deliverable, but it was instrumental for setting up Part 4.

5 Part IV

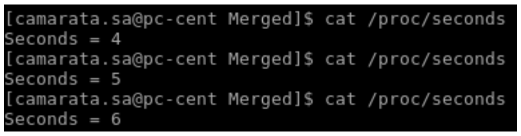
Part 4 is the real meat and potatoes of the assignment. The goal here was to implement two different proc entries that would yield differing results when queried. The first was to create a VFS entry for /proc/jiffies. The results of this are included in Figure 9.



```
[camarata.sa@pc-cent Merged]$ cat /proc/jiffies
Jiffies = 5730253093
```

Figure 9: Jiffies

The second deliverable for Part 4 was to create a VFS entry for /proc/seconds that would report the amount of time since the module had been loaded in seconds. The results of this command can be seen in Figure 10.



```
[camarata.sa@pc-cent Merged]$ cat /proc/seconds
Seconds = 4
[camarata.sa@pc-cent Merged]$ cat /proc/seconds
Seconds = 5
[camarata.sa@pc-cent Merged]$ cat /proc/seconds
Seconds = 6
```

Figure 10: Seconds

It is important to note that in order to accomplish this, I had to create a global variable that the kernel can leverage to pass the value of jiffies on initialization to the application. Reading implied that doing it this way is bad.

6 Conclusion

In conclusion, I think this was an interesting coding project. I would definitely be curious about more practical examples of kernel modules, or a project that dives a bit deeper. I feel like with the code that was provided as a template, this project was overly basic for what I think an engineer in a Master's program should be capable of doing. To attempt to compensate this, I worked with a fellow classmate to try and get all of the components into one C file that is included in the appendix. I also wrote this paper using LaTeX to try and learn something new.

7 Appendix A: Code

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

#define BUFFER_SIZE 128

#define PROC1_NAME "jiffies"
#define PROC2_NAME "seconds"
#define MESSAGE "Hello World\n"

long unsigned jiffies_var = 0;

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


static struct file_operations proc_ops_jiffies = {
        .owner = THIS_MODULE,
        .read = read_jiffies,
};

static struct file_operations proc_ops_seconds = {
        .owner = THIS_MODULE,
        .read = read_seconds,
};

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

        // creates the /proc/golden entry
        // the following function call is a wrapper for
        // proc_create_data() passing NULL as the last argument
        proc_create(PROC1_NAME, 0, NULL, &proc_ops_jiffies);
        proc_create(PROC2_NAME, 0, NULL, &proc_ops_seconds);

        jiffies_var = jiffies;
        printk(KERN_INFO "Loading Module\n");
        printk(KERN_INFO "/proc/%s created\n", PROC1_NAME);
        printk(KERN_INFO "/proc/%s created\n", PROC2_NAME);
        printk(KERN_INFO "Jiffies = %lu\n", jiffies);
        printk(KERN_INFO "Golden_Ratio = %lu\n", GOLDEN_RATIO_PRIME);
return 0;
}

/* This function is called when the module is removed. */
void proc_exit(void) {
        // removes the /proc/golden entry
        remove_proc_entry(PROC1_NAME, NULL);
        remove_proc_entry(PROC2_NAME, NULL);

        printk( KERN_INFO "GCD Result = %lu\n", gcd(3300,24));

        printk( KERN_INFO "/proc/%s removed\n", PROC1_NAME);
        printk( KERN_INFO "/proc/%s removed\n", PROC2_NAME);
        printk(KERN_INFO "Removing Module\n");

}

/**
 * This function is called each time the /proc/golden 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:
 */
ssize_t read_jiffies(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, "Jiffies = %lu\n", jiffies);

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

        return rv;
}

ssize_t read_seconds(struct file *file, char __user *usr_buf, size_t count, loff_t *pos)
{
        int rv = 0;
        long unsigned time = 0;
        char buffer[BUFFER_SIZE];
        static int completed = 0;

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

        completed = 1;
        time = jiffies/HZ - jiffies_var/HZ;
        rv = sprintf(buffer, "Seconds = %lu\n", time);

        // copies the contents of buffer to userspace usr_buf
        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("CYBER-570 Assignment 1");
MODULE_AUTHOR("Paul Camarata");
```