Project 2A: Kernel Module Programming

Yesheng Ma

March 23, 2017

Abstract

In this project, we are going to write some Linux kernel modules. Kernel modules are code that can be dynamically loaded into the kernel at runtime. In particular, we will write three kernel modules that can output some info, accepts parameters, and create a proc file.

1 Introduction to Kernel Modules

A Linux kernel module is an object file that contains code to extend the running kernel. It is frequently used by kernel hackers to write programs that interacts with Linux kernel. Kernel module code runs in kernel mode. Some frequently used Linux commands related to kernel module include `insmod`, `rmmod`, and `modinfo`.

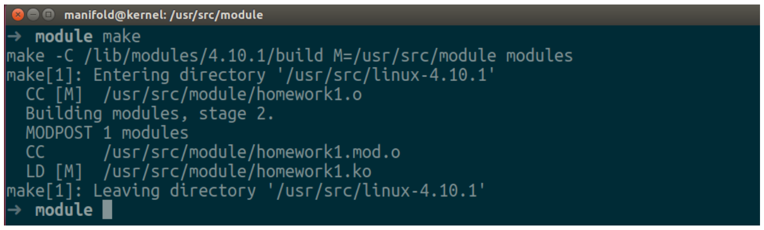
Also, we need to get some idea about the Linux `/proc` filesystem. `/proc` is a *virtual* file system, which is intended to expose some Linux kernel information to user. Also, we can configure the kernel using the `/proc` filesystem.

2 Build a Linux Kernel Module

As we can checkout in the Linux kernel documentation project, there is a typical kind of Makefile script to build a Linux kernel as listed below.

```
obj-m := homework1.o homework2.o homework3.o
KDIR := /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)

all:
	make -C $(KDIR) M=$(PWD) modules
clean:
	rm -f *.o *.ko *.mod.c Module.symvers modules.order
```

As we can see in the script, there is a special kind of Makefile object called `obj-m` and we need to specify the location of Linux header so that the kernel module can be successfully compiled.



```
→ module make
make -C /lib/modules/4.10.1/build M=/usr/src/module modules
make[1]: Entering directory '/usr/src/linux-4.10.1'
  CC [M]  /usr/src/module/homework1.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /usr/src/module/homework1.mod.o
  LD [M]  /usr/src/module/homework1.ko
make[1]: Leaving directory '/usr/src/linux-4.10.1'
→ module 
```

Figure 1: Building a Linux kernel module

3 Core Implementation

3.1 Problem 1

In this problem, we need to write a Linux kernel module that can print message to kernel buffer when it is loaded and cleaned up. The code snippet is shown as follows:

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

static int __init homework1_init(void) {
    printk(KERN_INFO "Hello world!\n");
    return 0;
}

static void __exit homework1_exit(void) {
    printk(KERN_INFO "Goodbye world!\n");
}

module_init(homework1_init);
module_exit(homework1_exit);
```

Listing 1: Module that output info on installation and exit

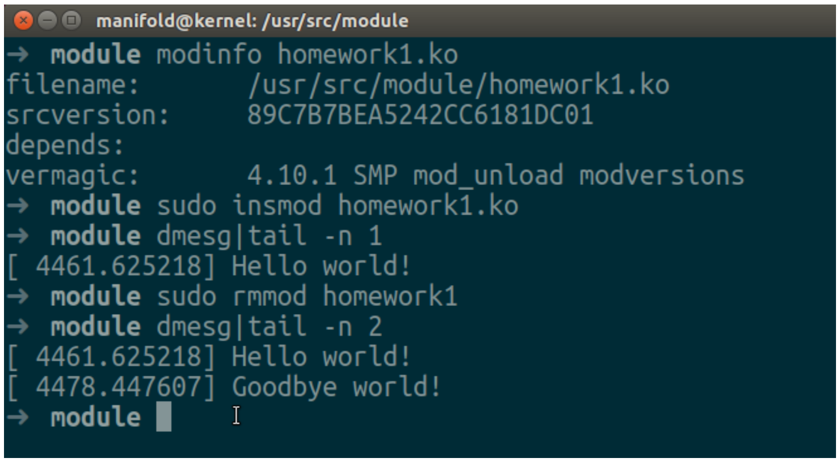
This program mainly includes three parts:

1. header file: these are Linux kernel headers which include necessary information to get kernel module compiled.

2. implementation code: the most important part is the definition of the `init` and `exit` functions, which uses the kernel function `printk` to print messages to kernel log buffer.

3. registration macro: the last two lines of this file registers the init and exit functions to the kernel module.

As we can see in the following figure illustrating the result of running the kernel module. We can use the command `modinfo` to get information about this kernel module. Then we install and remove the module and we can check the last two lines of the kernel log message using `dmesg | tail -n 2`.



```
manifold@kernel: /usr/src/module
→ module modinfo homework1.ko
filename:       /usr/src/module/homework1.ko
srcversion:     89C7B7BEA5242CC6181DC01
depends:
vermagic:       4.10.1 SMP mod_unload modversions
→ module sudo insmod homework1.ko
→ module dmesg|tail -n 1
[ 4461.625218] Hello world!
→ module sudo rmmod homework1
→ module dmesg|tail -n 2
[ 4461.625218] Hello world!
[ 4478.447607] Goodbye world!
→ module
```

Figure 2: Result of homework 1

3.2 Problem 2

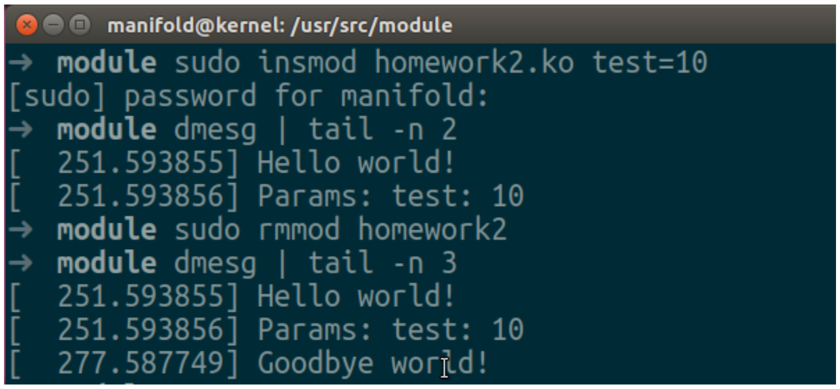
In this problem, we need to write a Linux kernel module that accepts a parameter of type int. We can easily use the kernel macro `module_param` to declare a parameter of the module.

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

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Homework2");
MODULE_AUTHOR("Yesheng Ma");

static int test;
module_param(test, int, 0644);

static int __init homework2_init(void) {
    printk(KERN_INFO "Hello world!\n");
    printk(KERN_INFO "Params: test: %d\n", test);
    return 0;
}
static void __exit homework2_exit(void) {
    printk(KERN_INFO "Goodbye world!\n");
}
```

```

module_init(homework2_init);
module_exit(homework2_exit);
```

Listing 2: Module that takes an integer parameter

We can see that the code is nearly the same as the former one except that it can use the parameter `test` as if it is predefined. The result is shown as follows:



```
manifold@kernel: /usr/src/module
→ module sudo insmod homework2.ko test=10
[sudo] password for manifold:
→ module dmesg | tail -n 2
[  251.593855] Hello world!
[  251.593856] Params: test: 10
→ module sudo rmmod homework2
→ module dmesg | tail -n 3
[  251.593855] Hello world!
[  251.593856] Params: test: 10
[  277.587749] Goodbye world!
```

Figure 3: Result of homework 2

3.3 Problem 3

In this problem, we need to create a readable file in the `/proc` file system. I think this is the most challenging problem since the API for creating kernel module has largely changed and most documents online are quite deprecated. After referring to many documentations, I finally managed to finish this project.

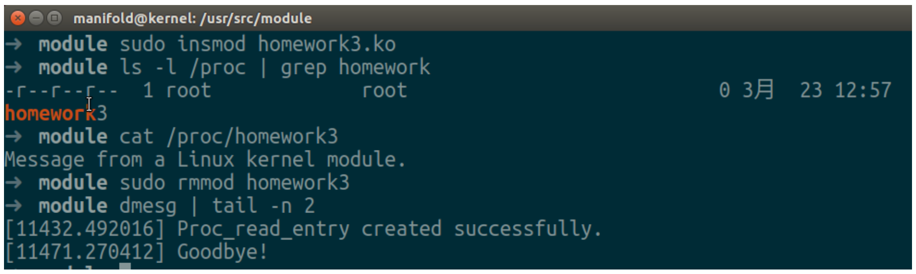
As we can see in the following code, the most significant difference between Linux kernel version 2.6 and 4.10 is that we no longer have the function `create_proc_entry`, on the other hand, we have `proc_create`. The last parameter of this function is a file handle which defines how we can operate on a file. Also, the 4.10 kernel allows us to operate on a proc file as if this is a normal file. Thus we can use the API defined for one special kind of file `seq_file` with related file oprations `seq_printf`, `single_open`. After we are done with the proc file, we can remove it by `remove_proc_entry`.

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

#include <linux/proc_fs.h>
#include <linux/seq_file.h>


MODULE_LICENSE("GPL");
```

```c
MODULE_DESCRIPTION("homework3");
MODULE_AUTHOR("Yesheng Ma");

static int proc_show(struct seq_file *m, void *v) {
    seq_printf(m, "Message from a Linux kernel module.\n");
    return 0;
}

static int proc_open(struct inode *inode, struct file *file) {
    return single_open(file, proc_show, NULL);
}

static const struct file_operations fops = {
    .owner = THIS_MODULE,
    .open = proc_open,
    .read = seq_read,
    .llseek = seq_lseek,
    .release = single_release,
};

static int __init homework3_init(void) {
    struct proc_dir_entry *entry;
    entry = proc_create("homework3", 0, NULL, &fops);
    if (!entry) {
        return -ENOMEM;
    } else {
        printk(KERN_INFO "Proc_read_entry created successfully.\n");
        return 0;
    }
}

static void __exit homework3_exit(void) {
    remove_proc_entry("homework3", NULL);
    printk(KERN_INFO "Goodbye!\n");
}

module_init(homework3_init);
module_exit(homework3_exit);
```

Listing 3: Module to create a readable proc file

The result of homework 3 is shown as follows:



```
→ module sudo insmod homework3.ko
→ module ls -l /proc | grep homework
-r--r--r--  1 root             root                          0 3月  23 12:57
homework3
→ module cat /proc/homework3
Message from a Linux kernel module.
→ module sudo rmmod homework3
→ module dmesg | tail -n 2
[11432.492016] Proc_read_entry created successfully.
[11471.270412] Goodbye!
```

Figure 4: Result of homework 3

4 Conclusion

This is the first project that we do real programming on Linux kernel. Linux kernel module is a very important part of kernel programming and I think during this project I really learned a lot about kernel module development and `/proc` file system. Also, finding information from various documentations is a very important skill for computer science students since many system projects are so large that we have to refer to docs to know what is actually happening and Linux kernel is no exception.

Acknowledgement

Thanks Prof. Chen for guidance on Linux kernel and TAs for their hard work.