Shanghai Jiao Tong University

CS353 Linux kernel

Project 2A:Module Programming

Chao Gao

5142029014

March 29, 2017

1 Introduction

This project is aimed to have a better understanding about linux module programming. Modules are pieces of code that can be loaded and unloaded into the kernel upon demand. It allows the dynamic insertion and removal of code from the kernel at run-time.
We are required to write three simple modules in this project.
Module 1: Load/unload the module can output some info.
Module 2: Accept a parameter, load the module and output the parameter's value.
Module 3: Create a proc file, read the proc file and return some info.

2 File Structure

basic part
2A_1 ——output some info
2A_2 ——accept an integer parameter
2A_3 ——create a proc file, read the file and return info

optional part
option_1 ——accept the string and array parameter
option_2 ——create the directory, create a writable file, deal with write buffer

3 Environment

Virtual OS: Ubuntu 16.04.
kernel version: 4.9.13.

4 Basic part

Here I will present the whole implementation process of basic part.

4.1 Module 1

First, include some necessary header files.

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

Write the entrance.

```c
static int __init hello_init(void)
{
    printk(KERN_INFO "Hello world\n");
    return 0;
}
```

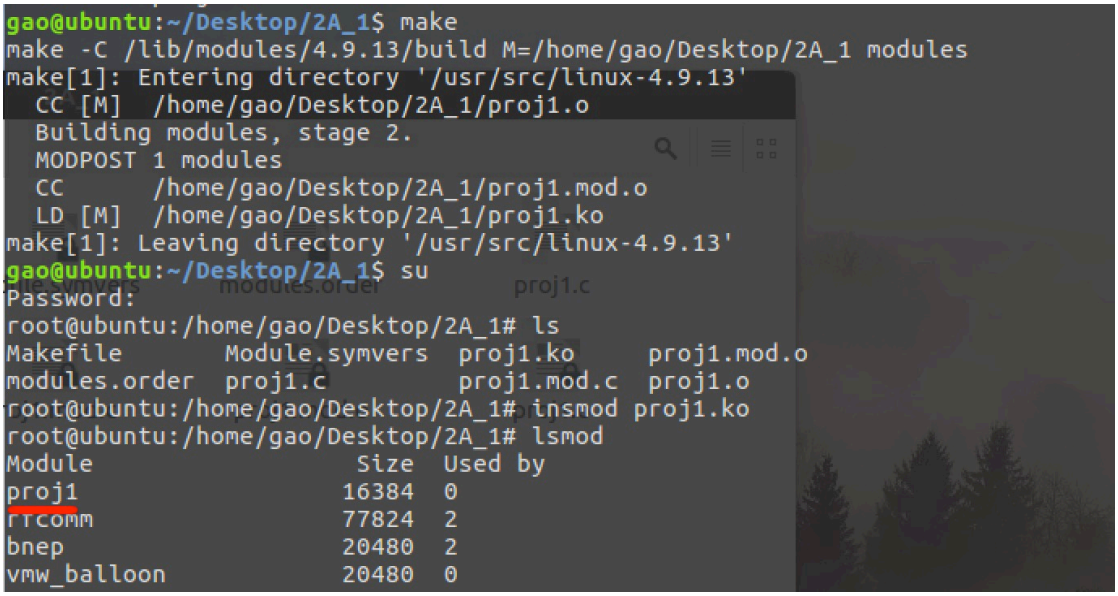
Write the exit.

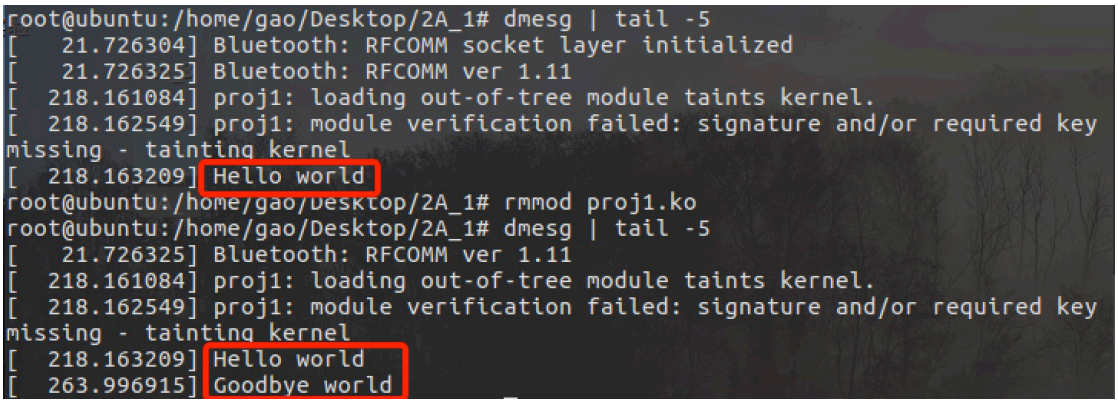
```c
static void __exit hello_exit(void)
{
    printk(KERN_INFO "Goodbye world\n");
}
```

Write the Makefile.

```
obj-m:=proj1.o
all:
	make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) modules
clean:
	make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) clean
```

Run and test the module.





We can see the messages are successfully displayed when the module is loaded and unloaded.

4.2 Module 2

Something different between module 1 and module 2 is that module 2 needs to accept a parameter and output parameter's value.

Use **module_param()** to pass the parameter.

```
1 static int test;
2 module_param(test, int, 0644);
```

Add one line of code to the entrance.

```
1 printk(KERN_INFO "Params: test:%d;\n", test);
```

Run and test the module.

```
root@ubuntu:/home/gao/Desktop/2A_2# make
make -C /lib/modules/4.9.13/build M=/home/gao/Desktop/2A_2 modules
make[1]: Entering directory '/usr/src/linux-4.9.13'
  CC [M]  /home/gao/Desktop/2A_2/proj2.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/gao/Desktop/2A_2/proj2.mod.o
  LD [M]  /home/gao/Desktop/2A_2/proj2.ko
make[1]: Leaving directory '/usr/src/linux-4.9.13'
root@ubuntu:/home/gao/Desktop/2A_2# insmod proj2.ko test=10
root@ubuntu:/home/gao/Desktop/2A_2# dmesg | tail -5
[18503.338676] Hello world
[18503.338678] Params: test: 10;
```

4.3 Module 3

In this part, I'm going to create a proc file and read the file.

```
1 static int my_proc_show(struct seq_file *m, void *v)
2 {
3     seq_printf(m, "I am Chao Gao. Hello proc!\n");
4     return 0;
5 }
6 static int my_proc_open(struct inode *inode, struct file *file)
7 {
8     return single_open(file, my_proc_show, NULL);
9 }
```

my_proc_show() is to output the kernel data to the user sapce. And I use **single_open()** function to bind the **my_proc_show()** function pointer. When we open the proc file, **single_open()** will be invoked.

So how to create a proc file, we can use **proc_create()** function.
static inline struct proc_dir_entry *proc_create(const char *name, mode_t mode,struct proc_dir_entry *parent, const struct file_operations *proc_fops).

name means the file name that we want to create, mode means the access authority, parent means the parent dictory(if we want to create a file in /proc, we can use 'NULL'), proc_fops means the file operation function.
I have to define file operation function first.

```
1 static const struct file_operations hello_proc_fops ={
2       .owner = THIS_MODULE,
3       .open = my_proc_open,
4       .read = seq_read,
5       .llseek = seq_lseek,
6       .release = single_release,
7 };
```

my_proc_open is defined by me, and other functions are existing functions in the kernel.

Create the file.

```
1 static int __init hello_proc_init(void)
2 {
3       proc_create("hello_proc",0644,NULL,&hello_proc_fops);
4       printk(KERN_INFO "hello_proc.\n");
5       return 0;
6 }
```

Remove the file.

```
1 static void __exit hello_proc_exit(void)
2 {
3       remove_proc_entry("hello_proc", NULL);
4       printk(KERN_INFO "Goodbye proc.\n");
5 }
```

Run and test the module.

```
gao@ubuntu:~/Desktop/2A_3$ make
make -C /lib/modules/4.9.13/build M=/home/gao/Desktop/2A_3 modules
make[1]: Entering directory '/usr/src/linux-4.9.13'
  CC [M]  /home/gao/Desktop/2A_3/proj3.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/gao/Desktop/2A_3/proj3.mod.o
  LD [M]  /home/gao/Desktop/2A_3/proj3.ko
make[1]: Leaving directory '/usr/src/linux-4.9.13'
gao@ubuntu:~/Desktop/2A_3$ su
Password:
root@ubuntu:/home/gao/Desktop/2A_3# insmod proj3.ko
root@ubuntu:/home/gao/Desktop/2A_3# cat /proc/hello_proc
I am Chao Gao, Hello proc!
```

Now we can see the file is successfully created and return the data.

5 Analysis

Since the project is not that difficult and the instructions provided are very detailed, the process is smooth in module 1 and module 2.
However, due to the version of kernel updates quickly, I meet some challenges in the implementation of module 3.
First I try to use the function **create_proc_read_entry()** which is provided in the handbook to create the proc file, error occurs.

```
/home/gao/Desktop/fs/fs.c: In function 'hello_init':
/home/gao/Desktop/fs/fs.c:11:9: error: implicit declaration of function 'create_
proc_read_entry' [-Werror=implicit-function-declaration]
  entry = create_proc_read_entry ("hello", 0444, NULL, hello_call, NULL); if(!ent
          ^
```

I want to search for some solutions in the Internet and find that this function has been outdated from kernel version 3.10, there is no definition of this function in proc_fs.h. I have to use **proc_create()** function. There are some differences between them, and the implementation details are provided in previous part.

6 Optional part

Since the basic requirement helps me a lot to understand the module programming, the optional part is not that difficult to figure out. Here I just briefly talk about this part.

6.1 String and array parameter

```
1 module_param(charp_arg, charp, 0644);
2 module_param_array(arr_arg, int, &arr_argc, 0644);
```

```
1 printk(KERN_INFO "Get_string:%s\n", charp_arg);
2 for (i = 0; i < arr_argc; i++)
3 {
4     printk(KERN_INFO "arr_arg[%d]=%d", i, arr_arg[i]);
5 }
6 printk(KERN_INFO "array_args:", arr_argc);
```

Test result

```
root@ubuntu:/home/gao/Desktop/option1# insmod param.ko charp_arg="hello" arr_arg=3,2,4
root@ubuntu:/home/gao/Desktop/option1# dmesg | tail -5
[21393.184317] Get_string: hello
[21393.184318] arr_arg[0]= 3
[21393.184318] arr_arg[1] = 2
[21393.184319] arr_arg[2]=4
[21393.184319] array args: 3
```

6.2 Create a directory, write the file, buffer overflow

Simply use **proc_mkdir()** to create a directory, change the authority to 0666 to make the file writable.

```
1 myTest = proc_mkdir("test", NULL);
2 file = proc_create("abc", 0666, myTest, &my_fops);
```

For write buffer overflow problem(here assume the buffer maximum size is 10), I simply take the first 10 contents.

```
1 {
2     buffer_size = count;
3     if(buffer_size >= BUFFER)
4         buffer_size = BUFFER;   //no more than 10
5     if(copy_from_user(tmp, buffer, buffer_size))
6     {
7         return EFAULT;
8     }
9     tmp[buffer_size] = '\0';
10    return count;
11 }
```

Test result

```
root@ubuntu:/home/gao/Desktop/5142029014_高超/2A_3# echo "today is good" > /proc/test/abc
root@ubuntu:/home/gao/Desktop/5142029014_高超/2A_3# cat /proc/test/abc
I am Chao Gao, hello proc!
current_size is 10
the content is today is g
```

7 Summary

From this project, I have a brief understanding about kernel module programming. Due to the detailed handbook, I start to program quickly, but linux kernel is

updating quickly, some functions have been changed or even disappeared, this time I should read the source code to know the new interface, and along this process I really gain a lot. Finally I accomplish this project.

Thanks Prof.Chen for the guidance about the linux kernel and TAs for answering my questions.