

Linux内核 实验报告 - 模块编程

1. 模块一：加载和卸载模块时在系统日志输出信息

过程

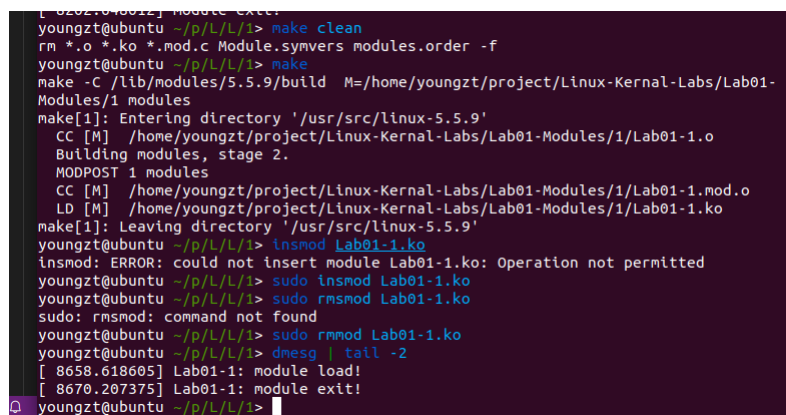
仅需在init和exit的相关函数中输出信息

```
static int __init init_fun(void)
{
    printk(KERN_INFO "Lab01-1: module load!\n");
    return 0;
}

static void __exit exit_fun(void)
{
    printk(KERN_INFO "Lab01-1: module exit!\n");
}

module_init(init_fun);
module_exit(exit_fun);
```

效果



```
[ 8628.040012] Module Exit!
youngzt@ubuntu ~/p/L/L/1> make clean
rm *.o *.ko *.mod.c Module.symvers modules.order -f
youngzt@ubuntu ~/p/L/L/1> make
make -C /lib/modules/5.5.9/build M=/home/youngzt/project/Linux-Kernal-Labs/Lab01-Modules/1 modules
make[1]: Entering directory '/usr/src/linux-5.5.9'
CC [M] /home/youngzt/project/Linux-Kernal-Labs/Lab01-Modules/1/Lab01-1.o
Building modules, stage 2.
MODPOST 1 modules
CC [M] /home/youngzt/project/Linux-Kernal-Labs/Lab01-Modules/1/Lab01-1.mod.o
LD [M] /home/youngzt/project/Linux-Kernal-Labs/Lab01-Modules/1/Lab01-1.ko
make[1]: Leaving directory '/usr/src/linux-5.5.9'
youngzt@ubuntu ~/p/L/L/1> insmod Lab01-1.ko
insmod: ERROR: could not insert module Lab01-1.ko: Operation not permitted
youngzt@ubuntu ~/p/L/L/1> sudo insmod Lab01-1.ko
youngzt@ubuntu ~/p/L/L/1> sudo rmmod Lab01-1.ko
sudo: rmmod: command not found
youngzt@ubuntu ~/p/L/L/1> sudo rmmod Lab01-1.ko
youngzt@ubuntu ~/p/L/L/1> dmesg | tail -2
[ 8658.618605] Lab01-1: module load!
[ 8670.207375] Lab01-1: module exit!
youngzt@ubuntu ~/p/L/L/1>
```

2. 模块二：支持整型、字符串、数组参数，加载时读入并打印

过程

在上一模块的基础上改动代码，增加传递参数的命令：

```
static int para_int = 0;
module_param(para_int, int, 0644);

static int para_int_array[10] = {0};
static int para_int_num = 0;
module_param_array(para_int_array, int, &para_int_num, 0644);

static char* para_chars = "default";
module_param(para_chars, charp, 0644);
```

并在加载模块时打印：

```
static int __init init_fun(void)
{
    int i = 0;
    printk(KERN_INFO "Lab01: module load!\n");

    printk(KERN_INFO "para_int=%d\n", para_int);
    printk(KERN_INFO "para_int_array: \n");
    for (i = 0; i < para_int_num; ++i)
        printk(KERN_INFO "%d ", para_int_array[i]);
    printk(KERN_INFO "\n");
    printk(KERN_INFO "para_chars=%s\n", para_chars);

    return 0;
}
```

编译完成后，在命令行加载模块，传递一些参数：

```
sudo insmod Lab01.ko para_int=2020 para_int_array=4,9,14,42 para_chars="Ziteng-
Yang"
```

然后打印内核日志查看模块输出信息：

```
dmesg | tail -10
```

效果

```
youngzt@ubuntu ~/p/L/L/1> sudo rmmod Lab01
youngzt@ubuntu ~/p/L/L/1> sudo insmod Lab01.ko para_int=2020 para_int_array=4,9,14
,42 para_chars="Ziteng-Yang"
youngzt@ubuntu ~/p/L/L/1> dmesg | tail -10
[12061.491524] Lab01: module exit!
[12064.871909] Lab01: module load!
[12064.871910] para_int=2020
[12064.871910] para_int_array:
[12064.871910] 4
[12064.871911] 9
[12064.871911] 14
[12064.871911] 42
[12064.871912] para_chars=Ziteng-Yang
youngzt@ubuntu ~/p/L/L/1>
```

3. 模块三：在/proc下创建只读文件

过程

首先再多创建一些参数, 使得可以在加载模块的时候选择文件名、读写权限等

```
// create files in "/proc"
static bool mkdir = false;
module_param(mkdir, bool, 0644);

static bool mkfile = false;
module_param(mkfile, bool, 0644);

static bool writable = false;
module_param(writable, bool, 0644);

static char* dir_name = "mydir";
module_param(dir_name, charp, 0644);

static char* file_name = "myfile";
module_param(file_name, charp, 0644);
```

然后设置部分文件操作的函数, 使得文件创建好后可以通过cat读取文件

```
static int proc_show(struct seq_file *m, void *v) {
    seq_printf(m, "YZT's proc!\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 mkfile_fops = {
    .owner= THIS_MODULE,
    .open = proc_open,
    .read = seq_read,
    // .write =
    .llseek = seq_lseek,
    .release = single_release
};
```

在加载模块时创建文件夹以及文件, 参数0x0444表示只读, 参数0x0666表示可读可写:

```
if(mkdir && dir_name)
{
    my_dir = proc_mkdir(dir_name, NULL);
    if(mkfile)
        if(!writable)
            my_file = proc_create(file_name, 0x0444, my_dir, &mkfile_fops);
        else
            my_file = proc_create(file_name, 0x0666, my_dir, &mkfile_fops);
}
```

效果

```
youngzt@ubuntu ~/p/L/Lab01-Modules> sudo rmmod Lab01
youngzt@ubuntu ~/p/L/Lab01-Modules>
sudo insmod Lab01.ko mkdir=1 dir_name="YZT" mkfile=1 file_name="YZT-File"
youngzt@ubuntu ~/p/L/Lab01-Modules> cat /proc/YZT/YZT-File
YZT's proc!
youngzt@ubuntu ~/p/L/Lab01-Modules> echo "test">>/proc/YZT/YZT-File
An error occurred while redirecting file '/proc/YZT/YZT-File'
open: Permission denied
youngzt@ubuntu ~/p/L/Lab01-Modules>
```

(写操作被拒绝)

4. 模块四：在/proc下创建可读可写文件

过程

首先需要修改输出函数的内容，增加一个字符指针来表示文件中的内容

```
static char *content = NULL;
static int proc_show(struct seq_file *m, void *v) {
    seq_printf(m, "YZT's proc:\n");

    if(content!=NULL)
        seq_printf(m, content);
    else
    {
        seq_printf(m, "Content is Empty!\n");
    }
    return 0;
}
```

然后需要增加一个写函数，来设定从命令行的IO接口如何获取内容：

```
// get content from IO
// using echo
static ssize_t proc_write(struct file *file, const char __user *buffer, size_t
count, loff_t *f_pos) {
    char *tmp = kzalloc((count+1), GFP_KERNEL);
    if(!tmp) return -ENOMEM;
    if(copy_from_user(tmp, buffer, count)){
        kfree(tmp);
        return EFAULT;
    }
    content = tmp;
    return count;
}
```

并将其加入文件管理的接口函数中：

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

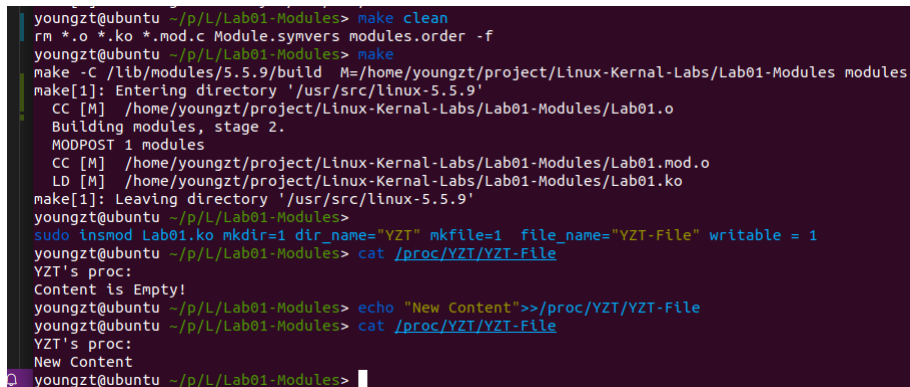
加载模块后，在命令行中先读一次文件，再写入内容，然后再读一次文件：

```
sudo insmod Lab01.ko mkdir=1 dir_name="YZT" mkfile=1 file_name="YZT-File"
writable = 1

cat /proc/YZT/YZT-File
echo "New Content">>/proc/YZT/YZT-File
cat /proc/YZT/YZT-File
```

New Content将被写入字符指针中

效果



```
youngzt@ubuntu ~/p/L/Lab01-Modules> make clean
rm *.o *.ko *.mod.c Module.symvers modules.order -f
youngzt@ubuntu ~/p/L/Lab01-Modules> make
make -C /lib/modules/5.5.9/build M=/home/youngzt/project/Linux-Kernal-Labs/Lab01-Modules modules
make[1]: Entering directory '/usr/src/linux-5.5.9'
  CC [M] /home/youngzt/project/Linux-Kernal-Labs/Lab01-Modules/Lab01.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M] /home/youngzt/project/Linux-Kernal-Labs/Lab01-Modules/Lab01.mod.o
  LD [M] /home/youngzt/project/Linux-Kernal-Labs/Lab01-Modules/Lab01.ko
make[1]: Leaving directory '/usr/src/linux-5.5.9'
youngzt@ubuntu ~/p/L/Lab01-Modules>
sudo insmod Lab01.ko mkdir=1 dir_name="YZT" mkfile=1 file_name="YZT-File" writable = 1
youngzt@ubuntu ~/p/L/Lab01-Modules> cat /proc/YZT/YZT-File
YZT's proc:
Content is Empty!
youngzt@ubuntu ~/p/L/Lab01-Modules> echo "New Content">>/proc/YZT/YZT-File
youngzt@ubuntu ~/p/L/Lab01-Modules> cat /proc/YZT/YZT-File
YZT's proc:
New Content
youngzt@ubuntu ~/p/L/Lab01-Modules>
```

5. 实验总结

- 内核模块编程和普通程序相比有一个困难点在于，一旦出现空指针访问的错误，就需要重启系统；这在调试上是一个挑战；
- 实验中可以切实地体会到/proc文件系统如何形成的一个虚拟文件系统：事实上命令行上对其进行的读写操作都转化为了访问内存中的变量值；