Linux Kernel Project1 Report

蒋逸伟 517161910005

1 hello_lab

实验平台：Ubuntu
内核版本：5.4.0-21-generic

1.1 模块一：加载和卸载模块时在系统日志输出信息

这个 lab 旨在了解 Linux 内核模块的接口调用和构成
代码如下

```
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
static int __init hello_init(void)
{
    printk(KERN_INFO "Hello world\n");
    return 0;
}
static void __exit hello_exit(void)
{
    printk(KERN_INFO"Goodbye world\n");
}
module_init(hello_init);
module_exit(hello_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Jiang Yiwei");
MODULE_DESCRIPTION("Hello test");
MODULE_ALIAS("Project1") ;
```

module_init()宏表示程序入口
同理module_exit是程序退出
printk类似于printf() 用以输出第一个参数代表输出级别
内核中共提供了八种不同的日志级别，在 linux/kernel.h 中有相应的宏对应。同样也可以用<level>代替

```

#define KERN_EMERG  "<0>"  /* systemis unusable */
#define KERN_ALERT  "<1>"  /* actionmust be taken immediately */
#define KERN_CRIT   "<2>"  /*critical conditions */
#define KERN_ERR    "<3>"  /* errorconditions */
#define KERN_WARNING "<4>" /* warning conditions */
#define KERN_NOTICE  "<5>" /* normalbut significant */
#define KERN_INFO    "<6>" /*informational */
```

```
28 #define KERN_DEBUG    "<7>"    /*debug-level messages */
```

makefile 的编写

```
1 obj-m = hello.o
2 all:
3         make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) modules
4 clean:
5         make -C /lib/modules/$(shell uname -r)/build M=$(shell pwd) clean
```

Tips:

其中 object 的名字需要和.c 文件相同

`uname-r`可以显示当前内核版本 `pwd` 表示当前目录

同时可以由`file`命令查看`*.ko`的信息

`all`:按下回车键后需要一个`TAB`键在这之间是一个制表符，否则会报错

`file hello.ko`可以显示 module 的信息

实验结果如下:

```
> insmod hello.ko
> rmmod hello
> dmesg|tail -10
[18959.294341] wlp3s0: send auth to fc:d7:33:94:96
:60 (try 1/3)
[18959.303252] wlp3s0: authenticated
[18959.306578] wlp3s0: associate with fc:d7:33:94:
96:60 (try 1/3)
[18959.313810] wlp3s0: RX AssocResp from fc:d7:33:
94:96:60 (capab=0x431 status=0 aid=4)
[18959.320328] wlp3s0: associated
[18959.339992] IPv6: ADDRCONF(NETDEV_CHANGE): wlp3
s0: link becomes ready
[21423.423889] Hello build
[21462.319796] Goodbye world
[21474.539511] Hello build
```

2 parameters

2.1 模块二：支持整型、字符串、数组参数，加载时读入并打印

这个 lab 主要测试参数的传递需要包括头文件`\linux\moduleparam.h`

用到两个宏 `module_param(name,type,perm)`,`module_param_array(name,type,nump,perm)`

其中`nump`是一个指向整数的指针，这个整数是数组中元素的个数

向参数中传递字符串需要制定`type`为`charp`而不是`char`

或者可以使用宏`module_param_string(name, string, len, perm);`

代码如下:

```
29 #include<linux/kernel.h>
```

```c
#include<linux/module.h>
#include<linux/init.h>
#include<linux/moduleparam.h>
static int int_var;
static char* str_var;
static int int_array[8];
static int maxsize = 8;
module_param(str_var,charp,0660);
module_param(int_var,int,0660);
module_param_array(int_array,int,&maxsize,0660);
static int __init hello_init(void)
{
    int i=0;
    printk(KERN_INFO "Hello world\n");
    printk(KERN_INFO "string:%s;\n",str_var);
    printk(KERN_INFO "int:%d;\n",int_var);
    for(i=0;i<(sizeof int_array / sizeof (int));i++)
    {
        printk(KERN_INFO "int_array[%d]:%d",i,int_array[i]);
    }
    return 0;
}
static void __exit hello_exit(void)
{
    printk(KERN_INFO "Goodbye world\n");
}
MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("parameters");
MODULE_AUTHOR("Jiang Yiwei");
module_init(hello_init);
module_exit(hello_exit);
```

实验效果如图:

```
root@austinguish-GL502VSK /h/a/桌/L/project_para#
insmod parameters.ko int_var=9 str_var=helloworld int_ar
ray=1,2,3,5
root@austinguish-GL502VSK /h/a/桌/L/project_para#
dmesg|tail
[28110.493682] Hello world
[28110.493684] string:helloworld;
[28110.493685] int:9;
[28110.493686] int_array[0]:1
[28110.493687] int_array[1]:2
[28110.493688] int_array[2]:3
[28110.493689] int_array[3]:5
[28110.493690] int_array[4]:0
[28110.493691] int_array[5]:0
[28110.493692] int_array[6]:0
```

3 proc_lab

3.1 模块三，在/proc 下创建只读文件

创建只读文件需要先了解 filemode: 共有四位数字第一位是 sticky 位，第二位是文件属主的权限，第三位是组用户的权限，第四位是附加权限数字由下面的表说明。

$$\begin{aligned}
\text{rwx} &= 111 = 7 \\
\text{rw-} &= 110 = 6 \\
\text{r-x} &= 101 = 5 \\
\text{r--} &= 100 = 4 \\
\text{-wx} &= 011 = 3 \\
\text{-w-} &= 010 = 2 \\
\text{--x} &= 001 = 1 \\
\text{---} &= 000 = 0
\end{aligned}$$

这里选择 444 作为只读的权限。

思路是先用`proc_dir_entry *entry`声明一个 proc 文件的入口，紧接着完成`read`函数，通过`proc_create()`函数进行文件创建。

```
63 #include <linux/module.h>
64 #include <linux/init.h>
65 #include <linux/kernel.h>
66 #include <linux/proc_fs.h>
67 static struct proc_dir_entry *entry;
68
69 static struct file_operations ops =
70 {
71 };
72
73 static int proc_init(void)
74 {
75         entry=proc_create("hello_proc",0444,NULL,&ops);
76         return 0;
77 }
78
79 static void proc_exit(void)
80 {
81         proc_remove(entry);
82 }
83
84 module_init(proc_init);
85 module_exit(proc_exit);
```

效果如图：

```
austinguish@austinguish-GL502VSK /proc [SIGKILL]>
cat hello_proc
Hello proc!
austinguish@austinguish-GL502VSK /proc>
echo "hellobit">hello_proc
<W> fish: An error occurred while redirecting file 'hello_p
roc'
open: 权限不够
```

4 可读可写的文件

4.1 在/proc 下创建文件夹，并创建一个可读可写的文件

重点是实现`file_operations`

第一个操作 `.open` 从内核中导出信息到用户空间有很多方法，可以自己去实现 file_operations 的 read 函数，但是这种方法不够简单，而且也会有一些限制，比如一次 read 读取大于 1 页时，模块就不得不去进行复杂的缓冲区管理。为此，就需要学习一下 seq_file 的用法，为了更简单和方便，内核提供了 single_xxx 系列接口，它是对 seq_file 的进一步封装。这里选用了 single_open。同时 my_proc_show 定义了文件打开时的操作，在屏幕上打印文件内容。

`write` 函数使用了`kmalloc`函数用以申请动态内存是 kernel 态的申请内存函数。可以防止内存溢出。

`copy_from_user()`用来从用户态接收信息到内核态

`struct proc_dir_entry *parent=proc_mkdir("",NULL)` 用来创建一个文件夹供接下来的`proc_create()`使用

```
unsigned long copy_from_user (void *to,const void __user *from,unsigned long
n);
Arguments

to

    Destination address, in kernel space.
from

    Source address, in user space.
n

    Number of bytes to copy.
```

代码如下：

```
#include<linux/module.h>
#include<linux/init.h>
#include<linux/proc_fs.h>
#include<linux/uaccess.h>
#include<linux/seq_file.h>
#include<linux/slab.h>
```

```
105
106 static char *str = NULL;
107
108 static int my_proc_show(struct seq_file *m,void *v){
109         seq_printf(m,"%s\n",str);
110         return 0;
111 }
112
113 static ssize_t my_proc_write(struct file* file,const char __user *buffer,size_t count
114         char *tmp = kmalloc((count+1),GFP_KERNEL);
115         if(!tmp)return -ENOMEM;
116         if(copy_from_user(tmp,buffer,count)){
117                 kfree(tmp);
118                 return EFAULT;
119         }
120         str=tmp;
121         return count;
122 }
123
124 static int my_proc_open(struct inode *inode,struct file *file){
125         return single_open(file,my_proc_show,NULL);
126 }
127
128 static struct file_operations fops={
129         .open = my_proc_open,
130         .read = seq_read,
131         .write = my_proc_write
132 };
133 static int __init hello_init(void){
134         struct proc_dir_entry *parent = proc_mkdir("proc_test", NULL);//create direct
135         struct proc_dir_entry *entry;
136         entry = proc_create("Hello_Proc",0666,parent,&fops);
137         if(!entry){
138                 return -1;
139         }
140         return 0;
141 }
142
143 static void __exit hello_exit(void){
144         remove_proc_entry("Hello_Proc",NULL);
145         printk(KERN_INFO "Goodbye world!\n");
```

```
146 }
147
148 module_init(hello_init);
149 module_exit(hello_exit);
150 MODULE_LICENSE("GPL");
```

实验效果如图：