

# 《操作系统原理》实验报告

姓名	侯皓斐	学号	U202010851	专业班级	软工 2003 班	时间	2022.5.24
----	-----	----	------------	------	-----------	----	-----------

## 一、实验目的

- 1) 理解设备是文件的概念。
- 2) 掌握 Linux 模块、驱动的概念和编程流程
- 3) Windows /Linux 下掌握文件读写基本操作

## 二、实验内容

- 1) 编写一个 Linux 内核模块，并完成模块的安装/卸载等操作。
- 2) 编写 Linux 驱动程序（字符类型或杂项类型）并编程应用程序测试。功能：有序读和写内核缓冲区，可以重复读，可以覆盖写。返回实际读写字节数。
- 3) Linux 中文件软连接和硬链接的验证实验。

## 三、实验过程

### 3.1 Linux 内核模块实验

开发环境：银河麒麟操作系统 v10 桌面版

编写一个 Linux 内核模块，并完成安装/卸载等操作。安装时和退出时在内核缓冲区显示不同的字符串。主要实现 `module_init()`、`module_exit()`。而且需要编写 Makefile 文件。

利用 `module_param(mytest, int, 0644)`宏，实验文件编译生成的 ko 模块可接收 charp（字符串）型参数，默认为"yuechuhaoxi"。当用户加载模块时在内核调试信息输出"Hello, yuechuhaoxi!\n"，退出时输出"Goodbye, yuechuhaoxi!\n"。

我们编写好如下的 Makefile 文件与 code.c 程序后，使用 make 进行编译。

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

```

C code.c
1  #include <linux/init.h>
2  #include <linux/kernel.h>
3  #include <linux/module.h>
4  #include <linux/moduleparam.h>
5  #include <linux/sched.h>
6
7  MODULE_LICENSE("GPL");
8  MODULE_AUTHOR("Haofei Hou HUST");
9  MODULE_DESCRIPTION("A simple example Linux module.");
10
11 static char* name="yuechuhaoxi";
12 module_param(name, charp, 0644);
13
14 static int code_init(void) {
15     printk(KERN_INFO"Hello, %s!\n", name);
16     return 0;
17 }
18
19 static void code_exit(void){
20     printk(KERN_INFO"Goodbye, %s!\n", name);
21 }
22
23 module_init(code_init);
24 module_exit(code_exit);

```

编译得到一系列文件，我们首先使用 `sudo dmesg -C` 指令清空内核信息。然后使用 `sudo insmod code.ko name='Haofei'` 修改参数值并将模块装入内核。使用 `dmesg` 显示内核信息。再使用 `sudo rmmod code.ko` 卸载内核，再 `dmesg` 显示当前内容。与预期相符。

### 3.2 Linux 字符类型驱动程序编写

**开发环境：**银河麒麟操作系统 v10 桌面版

编写 Linux 字符类型设备驱动程序，实现 `Test_open`，`Test_release`，`Test_read`，`Test_write` 等函数。并通过文件操作结构体 `file_operations test_fops` 完成接口映射。也实现 `Test_init_module` 驱动入口函数，`Test_exit_module` 驱动出口函数。

要实现有序读和写内核缓冲区，可以重复读，可以覆盖写的功能。

需要声明和申请内核缓冲区。

```

19 static char *device_buffer;
20 static size_t pos;
21 #define MAX_DEVICE_BUFFER_SIZE 64
22
23 static int Test_open(struct inode *inode, struct file *filp)
24 {
25     device_buffer = kmalloc(MAX_DEVICE_BUFFER_SIZE, GFP_KERNEL);
26     pos = 0;
27     return 0;
28 }

```

`Test_write()`负责写进去若干字符直到缓冲区末尾，维护写入位置，返回实际写入字数。

```

49 static ssize_t
50 Test_write(struct file *file, const char __user *buf, size_t count, loff_t *f_pos)
51 {
52     if(pos + count > 64)
53         count = 64-pos;
54     if(copy_from_user(device_buffer+pos, buf, count)) {
55         printk("write failed\n");
56         return -EFAULT;
57     }
58     pos = pos + count;
59     printk("%s: 写入%d字节, 写完后缓冲区为%s\n", __func__, count, device_buffer);
60     return count;
61 }

```

Test\_read()负责读出若干字符，维护读出位置。应返回实际读回字数。

```

35 static ssize_t |
36 Test_read(struct file *file, char __user *buf, size_t len, loff_t *ppos)
37 {
38     if(pos < len)
39         len = pos;
40     if(copy_to_user(buf, device_buffer+pos-len, len)) {
41         printk("read failed\n");
42         return -EFAULT;
43     }
44     pos = pos-len;
45     printk("%s: 读出%d字节, 读出后指针位置为%d\n", __func__, len, pos);
46     return len;
47 }

```

设备可以使用和文件相同的调用接口来完成打开、关闭、读写和 I/O 控制等操作。file\_operations 是一个完成文件操作和设备操作映射关系的抽象结构体。Linux 内核为设备建立一个设备文件，这样就使得对设备文件的所有操作，就相当于对设备的操作。用户程序可以用访问普通文件的方法访问设备文件，进而访问设备。

```

63 static struct file_operations test_fops = {
64     .owner = THIS_MODULE,
65     .open = Test_open,
66     .release = Test_release,
67     .read = Test_read,
68     .write = Test_write,
69 };

```

设备注销时，应注意不要忘记释放内核缓冲区，显式的使用了 device\_destroy()函数注销设备，删除设备文件。

```

94 static void Test_exit_module(void)//驱动退出函数
95 {
96     kfree(device_buffer);
97     if (0 != mem_class) {
98         device_destroy(mem_class,dev);
99         class_destroy(mem_class);
100         mem_class = 0;
101     }
102     cdev_del(&cdev);
103     printk("dev close\n");
104 }

```

为了创建一个字符设备，我们编写了如下驱动入口函数，声明与注册设备。显式的使用了 device\_create()函数建立设备文件。

```

71 static int Test_init_module(void) { //驱动入口函数
72     //动态分配设备号
73     int result = alloc_chrdev_region(&dev, 0, 2, DevName);
74     if (result < 0) {
75         printk("Err:failed in alloc_chrdev_region!\n");
76         return result;
77     }
78     //创建class实例
79     mem_class = class_create(THIS_MODULE,ClassName); // /dev/ create devfile
80     if (IS_ERR(mem_class)) {
81         printk("Err:failed in creating class!\n");
82     }
83     //动态创建设备描述文件 /dev/test
84     device_create(mem_class,NULL,dev,NULL,DevName);
85
86     cdev_init(&cdev,&test_fops);
87     _cdev.owner = THIS_MODULE;
88     _cdev.ops = &test_fops; //Create Dev and file_operations Connected
89     result = cdev_add(&cdev,dev,1);
90     printk("dev open\n");
91     return result;
92 }

```

我们编写适当的测试程序。使用文件命令打开设备，读写设备。

```
int fd = open(DEV_NAME, O_RDWR);
```

make 得到一系列文件，我们首先使用 `sudo dmesg -C` 指令清空内核信息。然后使用 `sudo insmod demomod.ko` 将模块装入内核，同时注册设备。使用 `dmesg` 显示内核信息。gcc test.c 并运行(注意一定使用 `sudo`)，最后使用 `sudo rmmod demomod.ko` 注销设备，最后使用 `dmesg` 显示内核信息。

### 3.3 Linux 文件软连接与硬链接验证

开发环境：银河麒麟操作系统 v10 桌面版

在 Linux 命令行使用 `ln` 或 `ln -s` 两个命令创建若干的硬链接和软链接文件。

Linux `ln`（英文全拼：link files）命令的功能是为某一个文件在另外一个位置建立一个同步的链接。硬链接的意思是一个档案可以有多个名称(目录文件中文件项指向同一个 inode)，而软链接的方式则是产生一个特殊的档案（其指向对应文件的路径）。

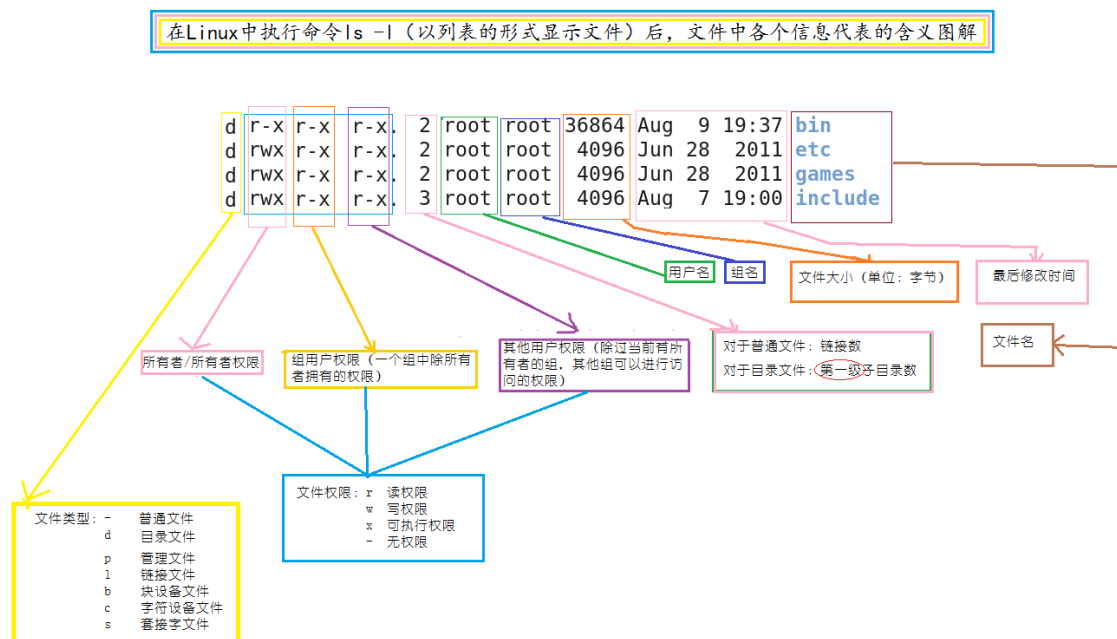
```

moonlight@moonlight-VMware:~/test4/4$ ls -li
总用量 1728
6686121 -rw-rw-r-- 2 moonlight moonlight 1765983 5月 24 10:20 output.txt
moonlight@moonlight-VMware:~/test4/4$ ln output.txt f1
moonlight@moonlight-VMware:~/test4/4$ ln output.txt f2
moonlight@moonlight-VMware:~/test4/4$ ln f2 f3
moonlight@moonlight-VMware:~/test4/4$ ln output.txt f4 -s
moonlight@moonlight-VMware:~/test4/4$ ls -li
总用量 6912
6686121 -rw-rw-r-- 5 moonlight moonlight 1765983 5月 24 10:20 f1
6686121 -rw-rw-r-- 5 moonlight moonlight 1765983 5月 24 10:20 f2
6686121 -rw-rw-r-- 5 moonlight moonlight 1765983 5月 24 10:20 f3
6685799 lrwxrwxrwx 1 moonlight moonlight 10 5月 24 11:17 f4 -> output.txt
6686121 -rw-rw-r-- 5 moonlight moonlight 1765983 5月 24 10:20 output.txt

```

使用 `ls -li` 命令查看目录或文件的信息。

其中总用量（total）表示该目录下所有文件所占的文件块的总数。而第一列表示文件的 inode 编号。而后面则表示文件中的各个信息，具体信息如下：



最后通过 `rm` 指令删除部分文件后，再次使用 `ls -li` 命令观察指令运行效果。

## 四、实验结果

### 4.1 Linux 内核模块实验

编译得到一系列文件，我们首先使用 `sudo dmesg -C` 指令清空内核信息。然后使用 `sudo insmod code.ko name='Haofei'` 修改参数值并将模块装入内核。使用 `dmesg` 显示内核信息。再使用 `sudo rmmod code.ko` 卸载内核，再 `dmesg` 显示当前内容。与预期相符。

```
moonlight@moonlight-VMware:~/test4/1$ make
make -C /lib/modules/5.17.5/build M=/home/moonlight/test4/1 modules
make[1]: 进入目录 "/home/moonlight/linux-5.17.5"
CC [M] /home/moonlight/test4/1/code.o
MODPOST /home/moonlight/test4/1/Module.symvers
CC [M] /home/moonlight/test4/1/code.mod.o
LD [M] /home/moonlight/test4/1/code.ko
make[1]: 离开目录 "/home/moonlight/linux-5.17.5"
moonlight@moonlight-VMware:~/test4/1$ sudo dmesg -C
[sudo] moonlight 的密码:
moonlight@moonlight-VMware:~/test4/1$ sudo insmod code.ko name="Haofei"
moonlight@moonlight-VMware:~/test4/1$ dmesg
[28027.731466] code: loading out-of-tree module taints kernel.
[28027.731983] code: module verification failed: signature and/or required key missing - tainting kernel
[28027.733711] Hello, Haofei!
moonlight@moonlight-VMware:~/test4/1$ sudo rmmod code.ko
moonlight@moonlight-VMware:~/test4/1$ dmesg
[28027.731466] code: loading out-of-tree module taints kernel.
[28027.731983] code: module verification failed: signature and/or required key missing - tainting kernel
[28027.733711] Hello, Haofei!
[28046.078222] Goodbye, Haofei!
```

## 4.2 Linux 字符类型驱动程序编写

make 得到一系列文件，我们首先使用 `sudo dmesg -C` 指令清空内核信息。然后使用 `sudo insmod demomod.ko` 将模块装入内核，同时注册设备。使用 `dmesg` 显示内核信息。`gcc test.c` 并运行（注意一定使用 `sudo`），最后使用 `sudo rmmod demomod.ko` 注销设备，最后使用 `dmesg` 显示内核信息。

```
moonlight@moonlight-VMware:~/test4/3remake$ make
make -C /lib/modules/5.17.5/build M=/home/moonlight/test4/3remake modules
make[1]: 进入目录 "/home/moonlight/linux-5.17.5"
CC [M] /home/moonlight/test4/3remake/demomod.o
MODPOST /home/moonlight/test4/3remake/Module.symvers
LD [M] /home/moonlight/test4/3remake/demomod.ko
make[1]: 离开目录 "/home/moonlight/linux-5.17.5"
moonlight@moonlight-VMware:~/test4/3remake$ sudo dmesg -C
[sudo] moonlight 的密码:
moonlight@moonlight-VMware:~/test4/3remake$ sudo insmod demomod.ko
moonlight@moonlight-VMware:~/test4/3remake$ gcc test.c
moonlight@moonlight-VMware:~/test4/3remake$ sudo ./a.out
1:520 10It is fate?
2:1206 204 337I can not believe?
moonlight@moonlight-VMware:~/test4/3remake$ sudo rmmod demomod.ko
moonlight@moonlight-VMware:~/test4/3remake$ dmesg
[29881.292272] dev open
[29891.654119] Test_write: 写入6字节，写完后缓冲区为 520 10
[29891.654124] Test_write: 写入11字节，写完后缓冲区为 520 10It is fate?
[29891.654126] Test_read: 读出17字节，读出后指针位置为 0
[29891.654208] Test_write: 写入12字节，写完后缓冲区为 1206 204 337fate?
[29891.654212] Test_write: 写入18字节，写完后缓冲区为 1206 204 337I can not beli
ve?
[29891.654213] Test_read: 读出30字节，读出后指针位置为 0
[29901.541271] dev close
```

## 4.3 Linux 文件软连接与硬链接验证

我们通过 `ln`，`ln -s` 等命令创建了下方一系列文件。

```
moonlight@moonlight-VMware:~/test4/4$ ls -li
总用量 1728
6686121 -rw-rw-r-- 2 moonlight moonlight 1765983 5月 24 10:20 output.txt
moonlight@moonlight-VMware:~/test4/4$ ln output.txt f1
moonlight@moonlight-VMware:~/test4/4$ ln output.txt f2
moonlight@moonlight-VMware:~/test4/4$ ln f2 f3
moonlight@moonlight-VMware:~/test4/4$ ln output.txt f4 -s
moonlight@moonlight-VMware:~/test4/4$ ls -li
总用量 6912
6686121 -rw-rw-r-- 5 moonlight moonlight 1765983 5月 24 10:20 f1
6686121 -rw-rw-r-- 5 moonlight moonlight 1765983 5月 24 10:20 f2
6686121 -rw-rw-r-- 5 moonlight moonlight 1765983 5月 24 10:20 f3
6685799 lrwxrwxrwx 1 moonlight moonlight 10 5月 24 11:17 f4 -> output.txt
6686121 -rw-rw-r-- 5 moonlight moonlight 1765983 5月 24 10:20 output.txt
```

我们可以清楚地看出 `f1`，`f2`，`f3` 和 `output.txt` 互为硬链接，其指向了同一个 `inode`，而且文件属性全部一致，由于有 4 个文件均为 1728 个块，故总用量为  $1728 \times 4 = 6912$  个块。而软连接的 `f4` 属性与前四者完全不同，而且指向了 `output.txt` 这个文件的路径。

当我们删除 `output.txt` 文件，我们发现 `f4` 文件失效，其指向的路径不存在。可得出结论，删除软连接连接的文件，会使源文件对应的软连接文件失效。



```
moonlight@moonlight-VMware:~/test4/4$ rm output.txt
moonlight@moonlight-VMware:~/test4/4$ ls -li
总用量 5184
6686121 -rw-rw-r-- 4 moonlight moonlight 1765983 5月 24 10:20 f1
6686121 -rw-rw-r-- 4 moonlight moonlight 1765983 5月 24 10:20 f2
6686121 -rw-rw-r-- 4 moonlight moonlight 1765983 5月 24 10:20 f3
6685799 lrwxrwxrwx 1 moonlight moonlight 10 5月 24 11:17 f4 -> output.txt
```

当我们删除 f4 文件,我们发现没有任何影响,即删除软连接文件不会对源文件产生影响。

```
moonlight@moonlight-VMware:~/test4/4$ ls -li
总用量 6912
6686141 -rw-rw-r-- 4 moonlight moonlight 1765983 5月 24 10:20 f1
6686141 -rw-rw-r-- 4 moonlight moonlight 1765983 5月 24 10:20 f2
6686141 -rw-rw-r-- 4 moonlight moonlight 1765983 5月 24 10:20 f3
6686141 -rw-rw-r-- 4 moonlight moonlight 1765983 5月 24 10:20 output.txt
```

当我们删除 f1 硬链接文件,我们发现总用量减少,而其他并未变化,即删除硬链接文件对其它文件没有影响,减少了一个指向该 inode 的文件。

```
moonlight@moonlight-VMware:~/test4/4$ rm f1
moonlight@moonlight-VMware:~/test4/4$ ls -li
总用量 5184
6686141 -rw-rw-r-- 3 moonlight moonlight 1765983 5月 24 10:20 f2
6686141 -rw-rw-r-- 3 moonlight moonlight 1765983 5月 24 10:20 f3
6686142 lrwxrwxrwx 1 moonlight moonlight 10 5月 24 11:27 f4 -> output.txt
6686141 -rw-rw-r-- 3 moonlight moonlight 1765983 5月 24 10:20 output.txt
```

当我们同时删除 f1, f2, f3, output.txt 文件,整个文件会真正的被删除。

```
moonlight@moonlight-VMware:~/test4/4$ rm output.txt f1 f2 f3
moonlight@moonlight-VMware:~/test4/4$ ls -li
总用量 0
6686142 lrwxrwxrwx 1 moonlight moonlight 10 5月 24 11:27 f4 -> output.txt
```

## 五、实验错误排查和解决方法


### 5.1 Linux 内核模块实验

由于是第一次编写 Makefile 文件,导致出现了不少乱子。

首先是 make 无法找到对应的 Makefile。

```
moonlight@moonlight-VMware:~/test4/1$ make
make -C /lib/modules/5.17.5/build M=/home/moonlight/test4/1 modules
make[1]: 进入目录 "/home/moonlight/linux-5.17.5"
scripts/Makefile.build:44: /home/moonlight/test4/1/Makefile: 没有那个文件或目录
make[2]: *** 没有规则可制作目标 "/home/moonlight/test4/1/Makefile"。 停止。
make[1]: *** [Makefile:1831: /home/moonlight/test4/1] 错误 2
make[1]: 离开目录 "/home/moonlight/linux-5.17.5"
make: *** [makefile:3: all] 错误 2
```

经过仔细检查发现是 Makefile 的大小写是由严格规定的。而下面的拼写是不对的。

 makefile

而且也出现了一些怪异的错误。

```
moonlight@moonlight-VMware:~/test4/1$ make
makefile:3: *** 遗漏分隔符 遗漏分隔符 %s。 停止。
```

其原因在于 Makefile 有着严格的格式要求，将部分空格替换为 Tab 后即可。

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

不过这个 Charp 类型的字符串有些小小怪异，空格后的文字会被自动忽略。

```
moonlight@moonlight-VMware:~/test4/1$ sudo insmod code.ko name='Haofei Hou'
moonlight@moonlight-VMware:~/test4/1$ dmesg
[ 2449.422626] code: loading out-of-tree module taints kernel.
[ 2449.422698] code: module verification failed: signature and/or required key missing - tainting kernel
[ 2449.423201] code: unknown parameter 'Hou' ignored
[ 2449.423253] Hello, Haofei!
moonlight@moonlight-VMware:~/test4/1$ sudo rmmod code.ko
moonlight@moonlight-VMware:~/test4/1$ dmesg
[ 2449.422626] code: loading out-of-tree module taints kernel.
[ 2449.422698] code: module verification failed: signature and/or required key missing - tainting kernel
[ 2449.423201] code: unknown parameter 'Hou' ignored
[ 2449.423253] Hello, Haofei!
[ 2473.733844] Goodbye, Haofei!
```

## 5.2 Linux 字符类型驱动程序编写

编写驱动程序的过程中出了不少乱子。

首先是一个小小的语法错误。

```
moonlight@moonlight-VMware:~/test4/3$ make
make -C /lib/modules/5.17.5/build M=/home/moonlight/test4/3 modules
make[1]: 进入目录"/home/moonlight/linux-5.17.5"
CC [M] /home/moonlight/test4/3/miscdevice.o
/home/moonlight/test4/3/miscdevice.c: In function 'miscdevice_exit':
/home/moonlight/test4/3/miscdevice.c:68:18: error: passing argument 1 of 'misc_deregister' from incompatible pointer type [-Werror=incompatible-pointer-types]
   68 |     misc_deregister(&my_miscdevice);
      |                      ^~~~~~
      |                      |
      |                      struct device **
In file included from /home/moonlight/test4/3/miscdevice.c:5:
./include/linux/miscdevice.h:92:48: note: expected 'struct miscdevice *' but argument is of type 'struct device **'
   92 |     misc_deregister(struct miscdevice *m);
```

应该是注册设备时出现了乱子。我明白过来，我们注册的应该是我们的设备文件，而不是注册我们实例化后的设备。即 `misc_register(&mydevice); misc_deregister(&mydevice);`



```

53 static struct miscdevice mydevice = {
54     .minor = MISC_DYNAMIC_MINOR,
55     .name = "buffer",
56     .fops = &mydevice_fops,
57 };
58
59 static int __init miscdevice_init(void) {
60     misc_register(&mydevice);
61     my_miscdevice = mydevice.this_device;
62     return 0;
63 }
64
65 static void __exit miscdevice_exit(void)
66 {
67     kfree(buffer);
68     misc_deregister(&my_miscdevice);
69 }

```

我发现一直不能正常打开设备文件，正常使用设备。（设备已成功注册）

```

moonlight@moonlight-VMware:~/test4/3$ ./a.out
open device failed
moonlight@moonlight-VMware:~/test4/3$ cd /dev
moonlight@moonlight-VMware:/dev$ ls
ashmem          kmsg            sda5            tty24
autofs          log             sda6            tty25
binder          loop0           sda7            tty26
block           loop1           sda8            tty27
bsg             loop2           sg0             tty28
btrfs-control  loop3           sg1             tty29
buffer          loop4           shm             tty3

```

而且内核输出信息莫名的奇怪，明明是先开启再关闭，为何内核信息反过来？

```

moonlight@moonlight-VMware:~/test4/3$ sudo insmod miscdevice.ko
moonlight@moonlight-VMware:~/test4/3$ dmesg
[ 462.889288] miscdevice close!
moonlight@moonlight-VMware:~/test4/3$ sudo insmod miscdevice.ko
insmod: ERROR: could not insert module miscdevice.ko: File exists
moonlight@moonlight-VMware:~/test4/3$ sudo rmmod miscdevice
moonlight@moonlight-VMware:~/test4/3$ dmesg
[ 462.889288] miscdevice close!
[ 476.747042] miscdevice open!
_

```

而且有时也会出现下面这样的报错。

```

moonlight@moonlight-VMware:~/test4/3$ dmesg
[ 201.610202] misckermmod: loading out-of-tree module taints kernel.
[ 201.610235] misckermmod: module verification failed: signature and/or required
key missing - tainting kernel
[ 201.611297] misckerbuffer open
[ 256.886995] misckerbuffer close
_

```

我良久调试无果后选择不使用杂项类型的设备，使用字符类型的设备进行模拟。没有情况没有发生改变。

后来发现是因为运行程序时，没有为其加上 `sudo` 导致的。而 `printk` 内核输出信息需要加上 “\n”，否则其可能输出出现乱子。而为了避免内核禁止加载模块，应该在 `Makefile` 中加上如此一行。

```
CONFIG_MODULE_SIG=n
```

还有两个非常傻，但是却想不到自己会犯的低级错误。

```
1 #include <stdio.h>
2 #include <fcntl.h>
3 #include <string.h>
4 #include <unistd.h>
5 char buffer[64];
6 #define DEV_NAME "/dev/kernelbuf"
```

`test.c` 的编写中 `buffer` 应该放在全局变量，否则 `buffer` 会被乱码覆盖，而且改换设备类型后 `DEV_NAME` 应当对应改变。

## 六、实验参考资料和网址

- (1) 教学课件：感谢华中科技大学软件学院苏曙光老师
- (2) 苏曙光. 操作系统原理（慕课版）[M]. 北京：人民邮电出版社，2022.
- (3) [https://blog.csdn.net/qq\\_46106285/article/details/121938689](https://blog.csdn.net/qq_46106285/article/details/121938689)
- (4) <https://zhuanlan.zhihu.com/p/420194002>
- (5) <https://xknote.com/blog/130126.html>
- (6) <https://www.cnblogs.com/fellow1988/p/6235080.html>
- (7) <http://c.biancheng.net/view/1039.html>
- (8) <https://blog.csdn.net/renlonggg/article/details/80701949>
- (9) <http://blog.chinaunix.net/uid-20758472-id-3368165.html>
- (10) <https://blog.csdn.net/duanlove/article/details/8225624>
- (11) <https://blog.csdn.net/mike8825/article/details/105420609>
- (12) [https://blog.csdn.net/zhlw\\_199008/article/details/81386875](https://blog.csdn.net/zhlw_199008/article/details/81386875)
- (14) <https://cloud.tencent.com/developer/article/1690636>