

**《操作系统》**

**实验课题十五**

|  |  |
| --- | --- |
|  |  |
| 学 号： |  |
| 姓 名： |  |
| 学 院： |  |
| 日 期： | 2022年 11 月 23 日 |

# 一、开发环境、运行环境、测试环境

使用ubuntu中的vim进行源代码的编写，使用ubuntu 20.0虚拟机运行程序进行测试。

# 二、实验内容：

（1）Linux 设备驱动程序的设计与实现（包括内核模块初始化/退出函数以及设备各类操作功能函数）；

（2）相应的设备驱动测试例程的设计实现，或设备驱动测试所用的现有应用程序的相关功能关联分析说明；

（3）设备驱动的测试验证，包括设备驱动加载和卸载操作、设备基本信息显示、设备驱动运行全过程内核信息输出以及测试例程自身运行的结果截屏。

# 三、实验步骤

### 1.设备驱动程序的设计与实现

**（1）设备驱动程序的功能**

本实验中设计的设备驱动程序是属于字符设备驱动程序，将设备当作文件看待，所以驱动程序就是使用文件操作接口对设备进行控制，本程序中主要使用了read, write, open，release等接口。该驱动程序的功能是实现使用read和write完成整型数据的读取的功能。

完整的程序源代码如下（intDevice.c）

1. #include <linux/module.h>
2. #include <linux/kernel.h>
3. #include <linux/init.h>
4. #include <linux/cdev.h>
5. #include <linux/fs.h>
6. #include <linux/string.h>
7. #include <linux/uaccess.h>
9. MODULE\_LICENSE("GPL");
10. MODULE\_AUTHOR("flygoast");
11. MODULE\_DESCRIPTION("A Simple Character Device driver module");
13. **static** **struct** cdev cdev;
14. **static** dev\_t  devno;
16. **static** **int** tmp[128] = {0};
18. // 打开设备
19. **static** **int**
20. intDevice\_open(**struct** inode \*inodep, **struct** file \*filep)
21. {
22. printk(KERN\_INFO "open\n");
23. **return** 0;
24. }
26. // 释放设备
27. **static** **int**
28. intDevice\_release(**struct** inode \*inodep, **struct** file \*filep)
29. {
30. printk(KERN\_INFO "release\n");
31. **return** 0;
32. }
34. // 设备读操作
35. **static** ssize\_t
36. intDevice\_read(**struct** file \*filep, **char** \_\_user \*buf, **size\_t** count, loff\_t \*offset)
37. {
38. printk(KERN\_INFO "read offset:%lld\\n", \*offset);
39. **if**(count <= \*offset){
40. // 如果count小于偏移量，那么直接读取count个字节
41. **if** (copy\_to\_user(buf, tmp + \*offset - count, count) != 0) {
42. // 读取失败
43. **return** -EFAULT;
44. }
45. // 读取成功，偏移量减去count，并且返回读取字节数
46. \*offset -= count;
47. **return** count;
48. }
49. **else**{
50. // count大于偏移量，无法读取count个字节，读取失败
51. **return** -EFAULT;
52. }
53. }
55. // 设备写操作
56. **static** ssize\_t
57. intDevice\_write(**struct** file \*filep, **const** **char** \_\_user \*buf, **size\_t** count,
58. loff\_t \*offset)
59. {
60. **size\_t**  avail;
62. printk(KERN\_INFO "write offset:%lld\n", \*offset);
63. // 缓冲区剩余字节数
64. avail = **sizeof**(tmp) - \*offset;
66. memset(tmp + \*offset, 0, avail);
68. **if** (count > avail) {
69. // 写入字节数count大于剩余字节数，那么就写入剩余字节数（有多少写多少）
70. **if** (copy\_from\_user(tmp + \*offset, buf, avail) != 0) {
71. // 写入失败
72. **return** -EFAULT;
73. }
74. // 写入成功，偏移量加上写入字节数，并返回
75. \*offset += avail;
76. **return** avail;
78. } **else** {
79. // 写入字节数count小于剩余字节数，直接写入count个字节
80. **if** (copy\_from\_user(tmp + \*offset, buf, count) != 0) {
81. // 写入失败
82. **return** -EFAULT;
83. }
84. \*offset += count;
85. **return** count;
86. }
87. }
89. // 重定位offset
90. **static** loff\_t
91. intDevice\_llseek(**struct** file \*filep, loff\_t off, **int** whence)
92. {
93. loff\_t  newpos;
95. **switch** (whence) {
96. **case** 0: /\* SEEK\_SET \*/
97. newpos = off;
98. **break**;
99. **case** 1: /\* SEEK\_CUR \*/
100. newpos = filep->f\_pos + off;
101. **break**;
102. **case** 2: /\* SEEK\_END \*/
103. newpos = **sizeof**(tmp) + off;
104. **break**;
105. **default**:
106. **return** -EINVAL;
107. }
109. **if** (newpos < 0) {
110. **return** -EINVAL;
111. }
113. filep->f\_pos = newpos;
114. **return** newpos;
115. }
117. // 文件操作结构映射
118. **static** **const** **struct** file\_operations  fops = {
119. .owner = THIS\_MODULE,
120. .open = intDevice\_open,
121. .release = intDevice\_release,
122. .read = intDevice\_read,
123. .llseek = intDevice\_llseek,
124. .write = intDevice\_write,
125. };
127. // 设备注册
128. **static** **int** \_\_init intDevice\_init(**void**) {
129. **int**    ret;
131. printk(KERN\_INFO "Load intDevice\n");
133. devno = MKDEV(112, 0);  // 主设备号
134. ret = register\_chrdev\_region(devno, 1, "intDevice");  // 主设备号，次设备号，设备名称
136. **if** (ret < 0) {
137. **return** ret;
138. }
140. cdev\_init(&cdev, &fops);
141. cdev.owner = THIS\_MODULE;
143. cdev\_add(&cdev, devno, 1);
145. **return** 0;
146. }
147. // 设备注销
148. **static** **void** \_\_exit intDevice\_cleanup(**void**) {
149. printk(KERN\_INFO "cleanup intDevice\n");
150. unregister\_chrdev\_region(devno, 1);
151. cdev\_del(&cdev);
152. }
154. module\_init(intDevice\_init);
155. module\_exit(intDevice\_cleanup);
157. MODULE\_LICENSE("GPL");

intDevice\_open（）的功能是打开设备；

intDevice\_release（）的功能是释放设备；

intDevice\_read（）的功能是实现设备对整形数据进行读取的功能；

intDevice\_write（）的功能是实现设备对整形数据进行写入的功能；

intDevice\_llseek（）的功能是实现缓冲区重定位的功能；

\_\_init intDevice\_init（）的功能是注册设备；

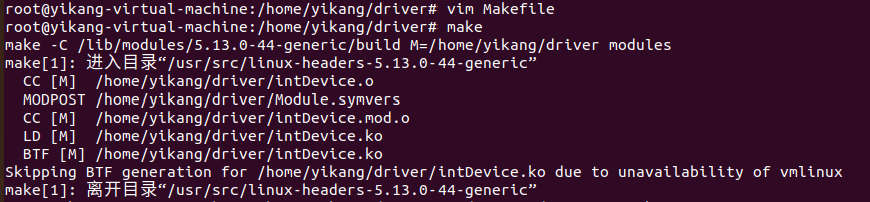
\_\_exit intDevice\_cleanup（）的功能是注销设备；

**（2）设备驱动程序的使用**

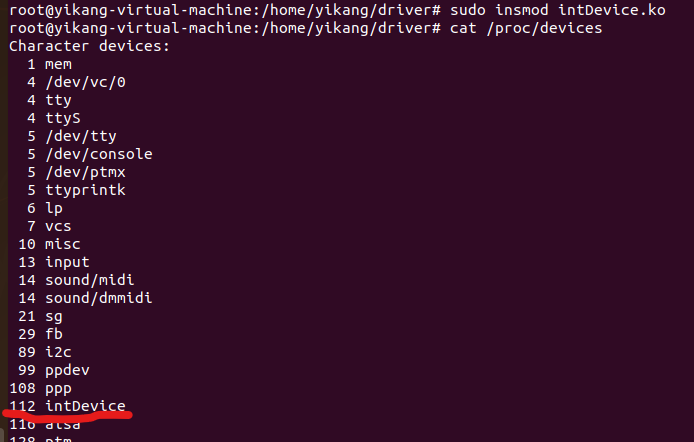
编写如下内容的Makefile文件，对程序进行编译：

1. obj-m += intDevice.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

执行make指令后编译结果如下图所示：



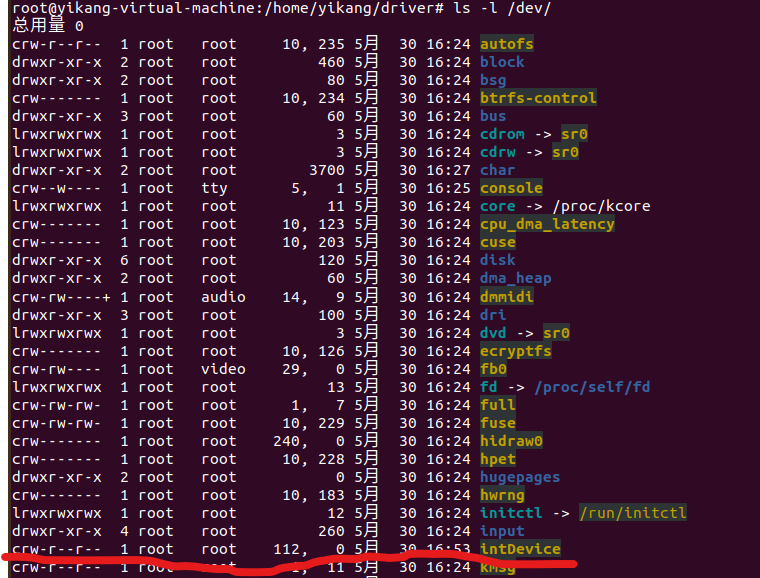
然后通过sudo insmod intDevice.ko命令进行驱动模块的安装，使用cat /proc/devices命令查看安装的驱动程序可以找到刚刚安装的驱动程序intDevice，设备号为112.



然后运行mknod /dev/intDevice c 112 0创建设备文件



使用ls -l/dev/可以查看刚刚创建的设备文件



### 2.对驱动程序进行测试

**（1）编写测试代码文件intDevice\_test.c**

源代码如下：

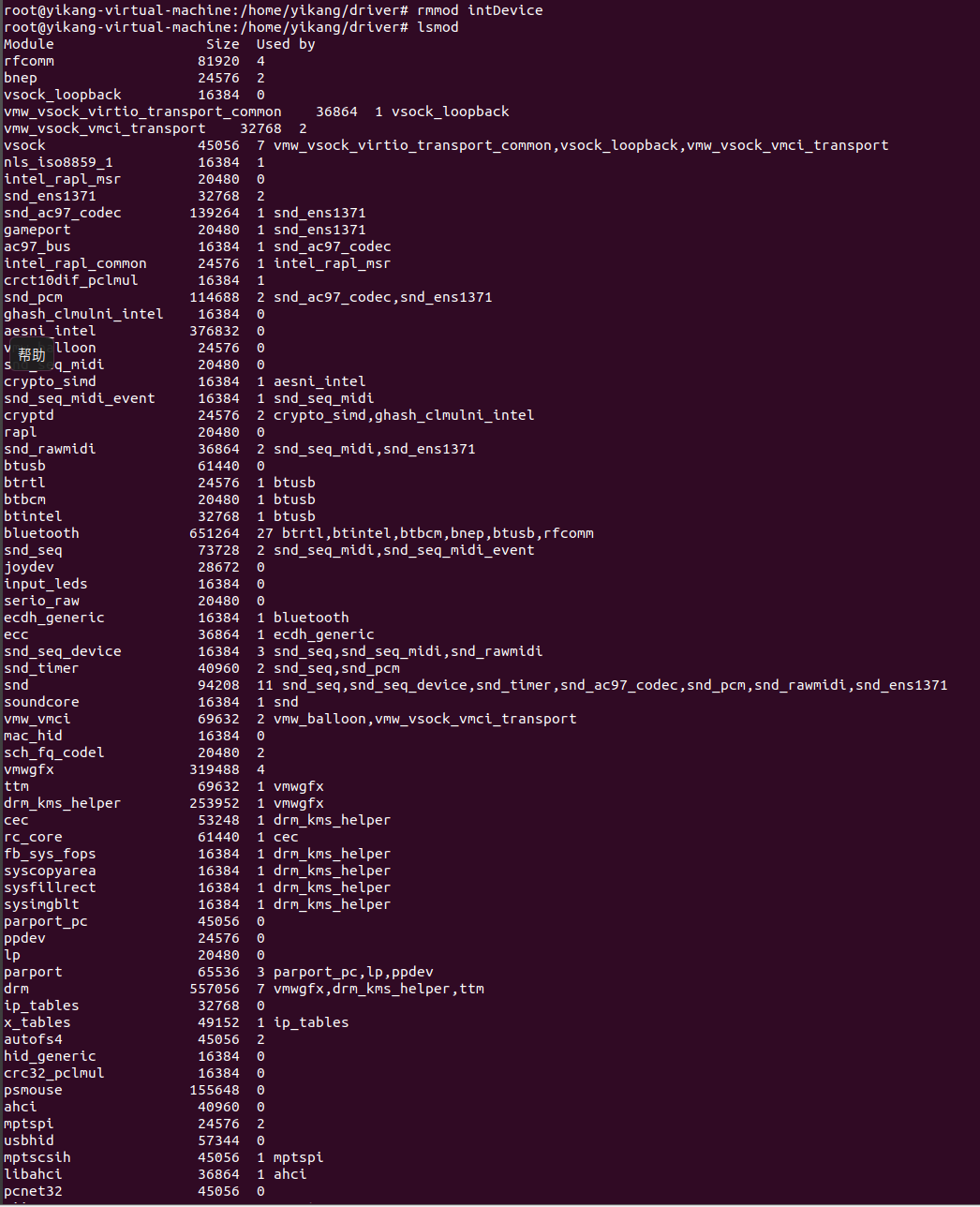
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <sys/types.h>
4. #include <fcntl.h>
6. **int** inputBuf[128], outputBuf[128];
8. **int** main()
9. {
10. **int** fd, m, n;
11. fd = open("/dev/intDevice", O\_RDWR);
12. **if** (fd < 0) {
13. fprintf(stderr, "open file \"/dev/intDevice\" failed\n");
14. exit(-1);
15. }
17. llseek(fd, 0, 0);
18. inputBuf[0] = 666;
19. inputBuf[1] = 333;
20. n = write(fd, inputBuf, 2 \* **sizeof**(**int**));
22. printf("read: ");
23. m = read(fd, outputBuf, 2 \* **sizeof**(**int**));
24. printf("sum = %d\n", outputBuf[0] + outputBuf[1]);
26. close(fd);
27. **return** 0;
28. }

测试程序的主要功能是通过驱动设备对缓冲区进行读写操作，向缓冲区中写入数据后读取出来，结果如下，可见驱动设备成功执行了需要的功能。

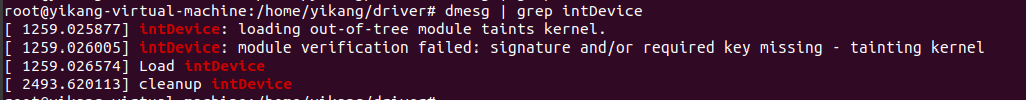


**（2）驱动设备的注销**

可通过命令rmmod intDevice对设备进行注销，结果如下，设备已经不在正在运行的模块中



可以使用命令dmesg | grep intDevice查看日志内容，可以看到设备的注册和注销记录。



# 四、实验总结

通过本次实验，我搞清楚了Linux设备驱动程序的设计原理，编写的代码运行时有一些BUG，通过自己DUBUG和参考网络上经典的代码最终成功完成了实验，在这个过程中学会了很多新的知识和技能，收获颇多。