

《操作系统原理》实验报告

姓名	鄢湧棚	学号	U201911741	专业班级	网安 1902	时间	2022.1.6
----	-----	----	------------	------	---------	----	----------

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，并完成安装/卸载等操作。
- (2) 编写 Linux 驱动程序并编程应用程序测试。功能：write 几个整数进去，read 出其和或差或最大值。
- (3) 编写 Linux 驱动程序并编程应用程序测试。功能：有序读写内核缓冲区，返回实际读写字节数。

三、 实验过程

- (1) Linux 内核模块编写

先编写模块函数，由如下部分组成：许可证声明，作者，模块描述，安装模块调用函数，删除模块调用函数等。

hello.c 文件内容如下

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

MODULE_LICENSE("GPL"); //模块许可证声明
MODULE_AUTHOR("YYP"); //模块作者（可选）
MODULE_DESCRIPTION("A Simple Linux Module in OS Lab4-1"); //模块描述
```

```
int test;
module_param(test, int, 0644); //模块参数绑定

//安装模块调用函数
static int __init myModuleInit(void) {
    printk("Hello World! This's YYP's module.\n");
    printk("test = %d\n", test);
    return 0;
}

//删除模块调用函数
static void __exit myModuleExit(void) {
    printk("Exit YYP's module.\n");
}

//模块函数注册
module_init(myModuleInit);
module_exit(myModuleExit);
```

然后编写 Makefile 文件，如下：

```
CONFIG_MODULE_SIG=n
obj-m:=hello.o
# generate the path
CURRENT_PATH:=$(shell pwd)
# current linux kernel version
VERSION_NUM :=$(shell uname -r)
# linux path
LINUX_PATH :=/lib/modules/$(VERSION_NUM)/build
```

```
#compile object
all :
    make -C $(LINUX_PATH) M=$(CURRENT_PATH) modules
#clean
clean :
    make -C $(LINUX_PATH) M=$(CURRENT_PATH) clean
```

编写好后，在文件目录下使用如下命令安装模块：

```
make
sudo insmod hello.ko
```

使用“sudo rmmod hello”，将该模块删除。

(2) Linux 驱动程序编写 1

实现了一个简单驱动 driver，功能是向设备写入两个整数然后读取两个整数中的最大值。file_operations 结构定义的是驱动可用函数，此外还要对设备的一些属性进行设置。

```
const int mydev_major = 61;    //主设备号
const char *mydev_name = "yyp's driver";    //设备名
const int mydev_size = 2 * sizeof(int);    //驱动程序缓冲区大小
const int int_size = sizeof(int);    //int 类型大小
char *mydev_buf;    //驱动程序缓冲区
```

然后是驱动安装和驱动删除函数，这里需要对设备进行额外操作，注册，删除，以及缓冲区的设置等。

然后是我们的功能实现函数 mydev_read 和 mydev_write。注意，对驱动缓冲区的操作需要将数据从内核态复制到用户态缓冲区。代码如下：

```
static ssize_t mydev_read(struct file *fp, char __user *buf,
                          size_t len, loff_t *pos) {
    //比较设备中 2 个数据的大小，并设置偏移量
```

```

    int a = (int)mydev_buf[0], b = (int)mydev_buf[int_size];
    int offset = (a >= b) ? 0 : int_size;
    //将数据从内核态复制到用户态缓冲区
    if(copy_to_user(buf, mydev_buf + offset, int_size))
        return -EFAULT;
    printk("<1>mydev: Read %d\n", (int)mydev_buf[offset]);
    //返回实际读取数据字节数
    return int_size;
}

//写入数据：向设备中写入最多 2 个整数
static ssize_t mydev_write(struct file *fp, const char __user *buf,
                           size_t len, loff_t *pos) {
    //最多向设备中写入 2 个整数大小
    if(len > mydev_size) len = mydev_size;
    //将数据从用户缓冲区复制到设备缓冲区
    if(copy_from_user(mydev_buf, buf, len)) return -EFAULT;
    printk("<1>mydev:  write   %d,   %d\n",   (int)mydev_buf[0],
(int)mydev_buf[int_size]);
    //返回实际写入数据字节数
    return len;
}

```

(3) Linux 驱动程序编写 2

该驱动实现的功能是：可以向设备中写入若干字符并在下次写入时数据紧接着上次写入的结尾；可以读取若干字符，并在下次读取时紧跟上次读取的结尾。

代码如下：

```

#include <linux/init.h>
#include <linux/module.h>

```

```

#include <linux/kernel.h> /* printk() */
#include <linux/slab.h> /* kcalloc() */
#include <linux/fs.h> /* everything... */
#include <linux/errno.h> /* error codes */
#include <linux/types.h> /* size_t */
#include <linux/proc_fs.h>
#include <linux/fcntl.h> /* O_ACCMODE */
#include <linux/uaccess.h> /* copy_from/to_user */


MODULE_LICENSE("GPL");
MODULE_AUTHOR("YYP");
MODULE_DESCRIPTION("A Simple Linux Driver in OS Lab4-2");


//函数声明
static int __init mydev2_init(void);
static void __exit mydev2_exit(void);
static ssize_t mydev2_read(struct file *fp, char __user *buf,
                           size_t len, loff_t *pos);
static ssize_t mydev2_write(struct file *fp, const char __user *buf,
                            size_t len, loff_t *pos);
loff_t mydev2_llseek(struct file *filp, loff_t offset, int whence);
    //loff_t 类型用于维护当前读写位置


//驱动可用函数
struct file_operations mydev2_fops = {
    read: mydev2_read,        //读数据
    write: mydev2_write,     //写数据
    llseek: mydev2_llseek    //文件指针移动
};

```

```
const int mydev2_major = 62;    //设备号
const char *mydev2_name = "yyp's driver-2";
const loff_t mydev2_max_size = 64; //驱动程序缓冲区最大值
char *mydev2_buf;    //驱动程序缓冲区
loff_t mydev2_size = 0;    //当前缓冲区数据大小

//安装驱动程序
static int __init mydev2_init(void) {
    int ret;
    //注册设备
    ret = register_chrdev(mydev2_major, mydev2_name, &mydev2_fops);
    if(ret < 0) {
        printk(KERN_EMERG"driver2: cannot obtain major number %d\n",
            mydev2_major);
        return ret;
    }
    //动态分配驱动程序缓冲区
    mydev2_buf = kmalloc(mydev2_max_size, GFP_KERNEL);
    if(!mydev2_buf) {
        ret = -ENOMEM;
        return ret;
    }
    //初始化缓冲区全 0
    memset(mydev2_buf, 0, mydev2_max_size);
    printk("<1>mydev2: Inserting mydev2\n");
    return 0;
}
```

```
//删除驱动程序
```

```
static void __exit mydev2_exit(void) {  
    //取消注册设备  
    unregister_chrdev(mydev2_major, mydev2_name);  
    //释放缓冲区内存  
    if(mydev2_buf) kfree(mydev2_buf);  
    printk("<1>mydev2: Removing mydev2\n");  
}
```

```
//读数据：读取字符
```

```
static ssize_t mydev2_read(struct file *fp, char __user *buf,  
                           size_t len, loff_t *pos) {  
    //计算设备中数据还能读取的大小  
    int left = mydev2_size - *pos;  
    //若文件指针已经指到数据末尾则不能读取，返回  
    if(left == 0) return 0;  
    //若读取数据超过可读取上限，则更新读取字节数  
    else if(len > left) len = left;  
    //从设备缓冲区复制数据到用户缓冲区  
    if(copy_to_user(buf, mydev2_buf+(*pos), len))  
        return -EFAULT;  
    printk("<1>mydev2: Read %ldB, left %ldB\n", len, left-len);  
    //文件指针后移读取字节数个单位  
    *pos += len;  
    //返回实际读取字节数  
    return len;  
}
```

```
//写数据： 写入字符
```

```

static ssize_t mydev2_write(struct file *fp, const char __user *buf,
                           size_t len, loff_t *pos) {
    //计算设备缓冲区还能写入的数据大小
    int left = mydev2_max_size - *pos;
    //若文件指针已经指到缓冲区末尾则不能写入，返回
    if(left == 0) return 0;
    //若写入数据超过可写入上限，则更新写入字节数
    else if(len > left) len = left;
    //从用户缓冲区复制数据到设备缓冲区
    if(copy_from_user(mydev2_buf+(*pos), buf, len))
        return -EFAULT;
    printk("<1>mydev2: Write %ldB, left %ldB\n", len, left-len);
    //文件指针后移写入字节数个单位
    *pos += len;
    //更新当前设备缓冲区中数据字节数
    if(*pos > mydev2_size) mydev2_size = *pos;
    //返回实际写入字节数
    return len;
}

//文件指针移动
loff_t mydev2_llseek(struct file *fp, loff_t offset, int whence) {
    //新文件指针位置
    loff_t newpos = 0;
    //判断起点
    switch (whence) {
        case SEEK_SET: //文件头
            newpos = offset;
            break;

```



```

    case SEEK_CUR: //文件指针当前位置
        newpos = fp->f_pos + offset;
        break;

    case SEEK_END: //文件尾
        newpos = mydev2_size + offset;
        break;

    default:
        return -EINVAL;
}

//若文件指针位置小于 0 或大于数据长度则报错
if(newpos < 0 || newpos >= mydev2_size) return -EINVAL;
//重置文件指针位置并返回
fp->f_pos = newpos;

printk("<1>mydev2: f_pos set to %lld\n", newpos);

return newpos;
}

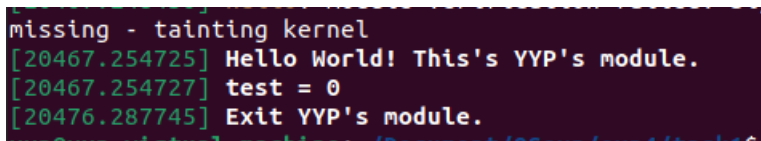
//注册函数
module_init(mydev2_init);
module_exit(mydev2_exit);

```

四、 实验结果

(1) Linux 内核模块编写

使用” sudo dmesg” 查看内核缓冲区



```

missing - tainting kernel
[20467.254725] Hello World! This's YYP's module.
[20467.254727] test = 0
[20476.287745] Exit YYP's module.

```

可以看到模块函数的输出。

(2) Linux 驱动程序编写 1

在编写测试程序测试前，需要将驱动文件写入 linux 的 dev 目录下，使用：

```
sudo mknod /dev/driver c 61 0
```

编写测试程序：

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>

int main() {
    int a[2], maxx;
    int fd = open("/dev/driver", O_RDWR);
    int len;
    if(fd < 0) {
        perror("Open driver fail\n");
        return 0;
    }
    printf("Input 2 int:");
    scanf("%d%d", &a[0], &a[1]);
    len = write(fd, a, 2*sizeof(int));
    if(len == 2*sizeof(int)) printf("Write: %d, %d\n", a[0], a[1]);
    len = read(fd, &maxx, sizeof(int));
    if(len == sizeof(int)) printf("Read max one: %d\n", maxx);
    close(fd);
    return 0;
}
```

运行结果如下：

```
yyp@yyp-virtual-machine:~/Document/0
Input 2 int:22 33
Write: 22, 33
Read max one: 33
```

(3) Linux 驱动程序编写 2

同样也需要写驱动设备文件，编写测试程序：

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <fcntl.h>

char src[256];
char dest[256];
int pc = 0;

int main() {
    int len, cnt;
    int pos;
    int fd = open("/dev/driver2", O_RDWR);
    if(fd < 0) {
        printf("Open driver2 fail\n");
        return 0;
    }
    printf("Write string:\n");
    while(scanf("%s", src) != EOF) {
        len = strlen(src);
```

```

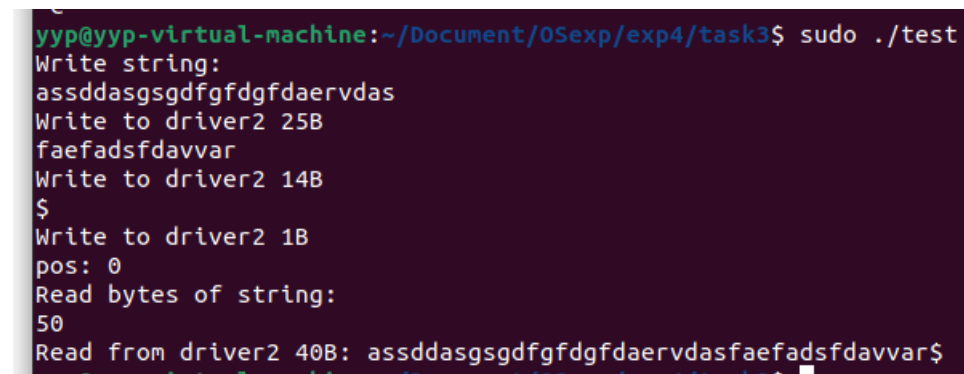
        cnt = write(fd, src, len);
        printf("Write to driver2 %dB\n", cnt);
        if(src[len-1] == '$') break;
    }

    printf("pos: %ld\n", lseek(fd, 0, SEEK_SET));
    printf("Read bytes of string:\n");
    while(scanf("%d", &len) != EOF) {
        cnt = read(fd, dest, len);
        dest[cnt] = 0;
        printf("Read from driver2 %dB: %s\n", cnt, dest);
    }

    close(fd);
    return 0;
}

```

运行如下，未写满 64 字节时：



```

yyp@yyp-virtual-machine:~/Document/0Sexp/exp4/task3$ sudo ./test
Write string:
assddasgsgdfgfdgfdavdas
Write to driver2 25B
faefadsfdavvar
Write to driver2 14B
$
Write to driver2 1B
pos: 0
Read bytes of string:
50
Read from driver2 40B: assddasgsgdfgfdgfdavdasfaefadsfdavvar$

```

写满 64 字节时，不会再写

```

yyp@yyp-virtual-machine:~/Document/05exp/exp4/task3$ sudo ./test
Write string:
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Write to driver2 53B
aaaaaaaaaaaaaaaaaaaaaaaaaaaaa
Write to driver2 11B
aaaaaaaaaaaa
Write to driver2 0B
bbbbbb
Write to driver2 0B
$
Write to driver2 0B
pos: 0
Read bytes of string:
64
Read from driver2 64B: aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaa

```

五、 实验错误排查和解决方法

(1) Linux 内核模块编写

对模块编写的 c 文件组成不是很了解，网上搜样例，按照样例结构写就可以了。

(2) Linux 驱动程序编写 1

驱动写好后，编写的测试程序提示找不到驱动文件，后面发现还得自己使用 `mknod` 命令再 `dev` 文件夹下创建设备文件。

(3) Linux 驱动程序编写 2

没什么问题

六、 实验参考资料和网址

PPT

<https://www.cnblogs.com/lidabo/p/5312827.html>

<https://blog.csdn.net/yeshennet/article/details/82290724>