

**课 程 实 验 报 告**

**课程名称： 操作系统原理实验**

**专业班级： 软件2202班**

**学 号： U202217216**

**姓 名： 郑德凯**

**报告日期： 2024/5/7**

**软件学院**

目录

[实验二：第8章设备管理 1](#_Toc166010307)

[一、 实验目的 1](#_Toc166010308)

[二、 实验内容 1](#_Toc166010309)

[三、实验要求 2](#_Toc166010310)

[四、实验指南 2](#_Toc166010311)

[（1）编写一个Linux内核模块，并完成模块的安装/卸载等操作 2](#_Toc166010312)

[（2）在Linux平台编写一个字符设备的驱动程序和测试用的应用程序 2](#_Toc166010313)

[（3）在Linux平台编写一个字符设备的驱动程序和测试用的应用程序 3](#_Toc166010314)

[五、实验过程 3](#_Toc166010315)

[(1) 编写一个Linux内核模块，并完成模块的安装/卸载等操作 3](#_Toc166010316)

[（2）在Linux平台编写一个字符设备的驱动程序和测试用的应用程序 4](#_Toc166010317)

[（3）在Linux平台编写一个字符设备的驱动程序和测试用的应用程序 10](#_Toc166010318)

实验二：第8章设备管理

1. 实验目的

* 理解设备是文件的概念。
* 掌握Linux模块（Module,自学）概念和编程流程
* 掌握Linux驱动程序的编写流程和基本编程技巧

1. 实验内容

* 编写一个Linux内核模块，并完成模块的安装/卸载等操作。
* 在Linux平台编写一个字符设备的驱动程序和测试用的应用程序。驱动程序的功能：可以接收应用程序调用write输入的整数，并用read输出最近输入的两个整数的和或最大值。
* 在Linux平台编写一个字符设备的驱动程序和测试用的应用程序。驱动程序的功能：内部维护一个32字节的缓冲区，应用程序可以多次调用read或write函数读写该缓冲区。 read或write操作相互之间无关联，但要求每次read或write要接着上操作的位置继续。如果读写失败返回错误。

三、实验要求

* 寝室提前做完，老师机房检查和答疑。

四、实验指南

（1）编写一个Linux内核模块，并完成模块的安装/卸载等操作

* 提示1：安装时和退出时在内核缓冲区显示不同的字符串。
* 提示2：相关函数：module\_init( )、 module\_exit( )
* 提示3： MODULE\_LICENSE( )、 MODULE\_AUTHOR ( )等可选提示4：安装命令：insmod XXXX.ko
* 提示5：扩展：编写带参数的模块程序

int mytest = 100;

module\_param(mytest, int, 0644);

（2）在Linux平台编写一个字符设备的驱动程序和测试用的应用程序

* 提示1：参考任务1
* 提示2：至少实现xx\_open,xx\_write,xx\_read等函数
* 提示3：功能：

 xx\_write( )写1个整数

 xx\_read( )读回结果（和或最大值） （注意处理异常，例如，只有一个数时！）

* 提示4: [可选的设备注册方式，其余方式参考baidu]

struct miscdevice mydemodrv\_misc\_device ;

ret = misc\_register( &mydemodrv\_misc\_device );

（3）在Linux平台编写一个字符设备的驱动程序和测试用的应用程序

* 提示1：参考任务1，任务2
* 提示2：至少实现xx\_open,xx\_write,xx\_read等函数
* 提示3：功能：
  + - 内核分配一定长度的缓冲区，比如64字节
    - xx\_write()写进去若干字符，注意维护写入位置。下次继续写的话，接着该位置往后写，直到缓冲区末尾。要返回实际写入字数。
    - xx\_read()读出若干字符串，注意维护读出位置。下次继续读的话，接着该位置往后读，直到缓冲区末尾。要返回实际读回字数。
    - （注意处理异常，例，read或write时剩下的空闲区不够！）

五、实验过程

(1) 编写一个Linux内核模块，并完成模块的安装/卸载等操作

1. 创建一个 C 源文件，命名为 hello\_kernel.c

#include <linux/init.h>

#include <linux/module.h>

#include <linux/kernel.h>

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("zhengdekai");

MODULE\_DESCRIPTION("A simple Linux kernel module");

MODULE\_VERSION("0.1");

static int \_\_init hello\_kernel\_init(void) {

printk(KERN\_INFO "Hello, kernel!\n");

return 0;

}

static void \_\_exit hello\_kernel\_exit(void) {

printk(KERN\_INFO "Goodbye, kernel!\n");

}

module\_init(hello\_kernel\_init);

module\_exit(hello\_kernel\_exit);

1. 编写一个 Makefile 来编译这个模块，创建一个名为 Makefile 的文件

obj-m += hello\_kernel.o

all:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

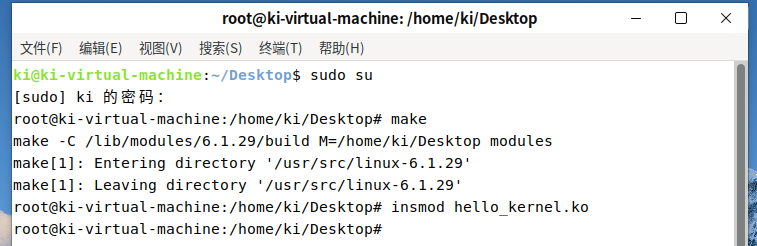
clean:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

1. 编译和安装模块

make

sudo insmod hello\_kernel.ko



1. 查看内核缓冲区中的输出

dmesg | tail



1. 卸载模块

sudo rmmod hello\_kernel



1. 查看内核缓冲区中的输出

dmesg | tail



（2）在Linux平台编写一个字符设备的驱动程序和测试用的应用程序

1. 创建字符设备驱动文件，命名为 char\_dev\_driver.c

#include <linux/module.h>

#include <linux/fs.h>

#include <linux/uaccess.h>

#include <linux/init.h>

#define DEVICE\_NAME "char\_dev"

#define BUF\_SIZE 1024

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("zhengdekai");

MODULE\_DESCRIPTION("A simple character device driver");

static int major\_num;

static int data[2] = {0}; // Array to store the two most recent integers

static int num\_integers = 0; // Number of integers stored

static int char\_dev\_open(struct inode \*inode, struct file \*file) {

pr\_info("Character device opened\n");

return 0;

}

static ssize\_t char\_dev\_read(struct file \*file, char \_\_user \*user\_buf, size\_t count, loff\_t \*ppos) {

int result=0;

// Check if there are enough integers stored

if (num\_integers < 1) {

return -EFAULT; // Error: Not enough integers stored

}

// Check if there is only one integer stored

if (num\_integers == 1) {

result = data[0]; // Return the single integer

} else {

// Calculate the sum of the two most recent integers

result = data[0] + data[1];

}

// Copy the result to the user buffer

if (copy\_to\_user(user\_buf, &result, sizeof(result))) {

return -EFAULT;

}

// Reset the number of integers stored

return sizeof(result);

}

static ssize\_t char\_dev\_write(struct file \*file, const char \_\_user \*user\_buf, size\_t count, loff\_t \*ppos) {

int new\_data;

// Check if the user buffer size is correct

if (count != sizeof(new\_data)) {

return -EINVAL;

}

// Copy the data from the user buffer

if (copy\_from\_user(&new\_data, user\_buf, sizeof(new\_data))) {

pr\_err("Failed to copy data from user space\n");

return -EFAULT; // Return error code

}

// Determine which index to write the new integer to

if (num\_integers % 2 == 0) {

// Even number of integers stored, write to data[0]

data[0] = new\_data;

pr\_info("Data has been written to data[0]: %d\n", data[0]);

} else {

// Odd number of integers stored, write to data[1]

data[1] = new\_data;

pr\_info("Data has been written to data[1]: %d\n", data[1]);

}

// Increment the number of integers stored

num\_integers++;

return sizeof(new\_data);

}

static int char\_dev\_release(struct inode \*inode, struct file \*file) {

pr\_info("Character device closed\n");

return 0;

}

static struct file\_operations char\_dev\_fops = {

.owner = THIS\_MODULE,

.open = char\_dev\_open,

.read = char\_dev\_read,

.write = char\_dev\_write,

.release = char\_dev\_release,

};

static int \_\_init char\_dev\_init(void) {

major\_num = register\_chrdev(0, DEVICE\_NAME, &char\_dev\_fops);

if (major\_num < 0) {

pr\_alert("Failed to register a major number\n");

return major\_num;

}

pr\_info("Registered character device with major number %d\n", major\_num);

return 0;

}

static void \_\_exit char\_dev\_exit(void) {

unregister\_chrdev(major\_num, DEVICE\_NAME);

pr\_info("Unregistered character device\n");

}

module\_init(char\_dev\_init);

module\_exit(char\_dev\_exit);

1. 编写测试应用程序 char\_dev\_test.c

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include <unistd.h>

#define DEVICE\_FILE "/dev/char\_dev"

int main() {

int fd;

int num\_written, num\_read;

int data\_to\_write = 123;

short result\_read;

// Open the device file

fd = open(DEVICE\_FILE, O\_RDWR);

if (fd == -1) {

perror("Failed to open the device file");

return EXIT\_FAILURE;

}

// Write data to the device

num\_written = write(fd, &data\_to\_write, sizeof(data\_to\_write));

if (num\_written == -1) {

perror("Failed to write to the device");

close(fd);

return EXIT\_FAILURE;

}

printf("Data written to device: %d\n", data\_to\_write);

// Read result from the device

num\_read = read(fd, &result\_read, sizeof(result\_read));

if (num\_read == -1) {

perror("Failed to read from the device");

close(fd);

return EXIT\_FAILURE;

}

printf("Result read from device: %d\n", result\_read);

// Close the device file

close(fd);

return EXIT\_SUCCESS;

}

1. 创建makefile文件

obj-m += char\_dev\_driver.o

all:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

gcc char\_dev\_test.c -o char\_dev\_test

clean:

make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

rm -f char\_dev\_test

1. 运行make命令



图形用户界面, 应用程序, Teams

描述已自动生成

图形用户界面, 文本, 应用程序, 电子邮件

描述已自动生成

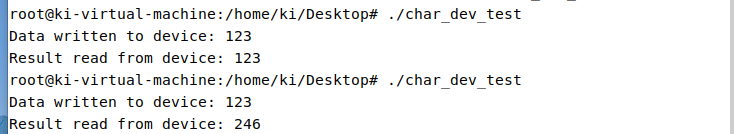
1. 加载sudo insmod char\_dev\_driver.ko
2. 手动创建设备文件

sudo mknod /dev/char\_dev c major\_num minor\_num

其中major\_num是主设备号，可以在系统日志内查看，minor\_num是次设备号，默认为0



1. 运行测试应用程序



成功显示了两个整数的和

（3）在Linux平台编写一个字符设备的驱动程序和测试用的应用程序