**《操作系统》课第12次实验报告**

|  |  |
| --- | --- |
| 学院: | 软件学院 |
| 姓名: | 张怡桢 |
| 学号: | 2013747 |
| 邮箱: | 2662765987@qq.com |
| 时间: | 12/08/2022 |

1. **开篇感言**

“你长大后想成为什么人？”

“什么意思？长大后我就不能成为我自己了吗？”

-- 《阿甘正传》

2. **实验题目**

**Linux kernel Module Development**

3. **实验要求**

To implement a Linux kernel module to read and write one specified file

* 选择一个具体文件(xxxfile)，利用该内核机制，实现对该文件的读操作
* cat xxxfile
* 进一步实现对该文件的写操作
* echo "hello 学号姓名日期..." > xxxfile
* 或
* echo "hello 学号姓名日期..." >> xxxfile

4. **实验步骤**

4.1 **Char device based on Linux Kernel Module**

实现字符设备的读写其实就是实现驱动中 file\_operation 结构体里的 read 和 write 成员.

|  |
| --- |
| C #include <linux/init.h> #include <linux/module.h> #include <linux/kernel.h> #include <linux/fs.h> #include <linux/uaccess.h>  MODULE\_LICENSE("GPL"); MODULE\_AUTHOR("Robert W. Oliver II"); MODULE\_DESCRIPTION("A simple example Linux module."); MODULE\_VERSION("0.01");  #define DEVICE\_NAME "lkm\_example" #define EXAMPLE\_MSG "Hello, 2013747 zhangyizhen!\n" #define MSG\_BUFFER\_LEN 128  /\* Prototypes for device functions \*/ static int device\_open(struct inode \*, struct file \*); static int device\_release(struct inode \*, struct file \*); static ssize\_t device\_read(struct file \*, char \*, size\_t, loff\_t \*); static ssize\_t device\_write(struct file \*, const char \*, size\_t, loff\_t \*);  static int major\_num; static char \*name = "lkm\_example"; static struct file \*filp; static int device\_open\_count = 0; static char msg\_buffer[MSG\_BUFFER\_LEN];  module\_param(name, charp, S\_IRUGO);  /\* This structure points to all of the device functions \*/ static struct file\_operations file\_ops = {  .read = device\_read,  .write = device\_write,  .open = device\_open,  .release = device\_release  };  /\* When a process reads from our device, this gets called. \*/ static ssize\_t device\_read(struct file \*s, char \*buffer, size\_t len, loff\_t \*offset) {  int i = 0;  int res = 0;  int j = 0;  filp->f\_pos = 0;  while (1) {  i = kernel\_read(filp, msg\_buffer, 128, &filp->f\_pos);  if (i == 0) {  break;  }  j = 0;  while (i > 0) {  i--;  put\_user(msg\_buffer[res++], buffer + (j++));  }  }  return res; }  /\* Called when a process tries to write to our device \*/ static ssize\_t device\_write(struct file \*s, const char \*buffer, size\_t len,  loff\_t \*offset) {  bool flag;  flag=copy\_from\_user(msg\_buffer, buffer, len);  printk("write to file: %s\n", name);  printk(KERN\_INFO "%s\n", msg\_buffer);  filp->f\_pos = file\_inode(filp)->i\_size;  printk("file size: %lld\n", filp->f\_pos);  kernel\_write(filp, msg\_buffer, len, &filp->f\_pos);  return len; }  /\* Called when a process opens our device \*/ static int device\_open(struct inode \*inode, struct file \*file) {  /\* If device is open, return busy \*/  if (device\_open\_count) {  return -EBUSY;  }  device\_open\_count++;  try\_module\_get(THIS\_MODULE);  return 0; }  /\* Called when a process closes our device \*/ static int device\_release(struct inode \*inode, struct file \*file) {  /\* Decrement the open counter and usage count. Without this, the module wouldnot unload. \*/  device\_open\_count--;  module\_put(THIS\_MODULE);  return 0; }  static int \_\_init lkm\_example\_init(void) {  // 打开文件  filp = filp\_open(name, O\_RDWR, 0);  printk("open file: %s", name);  /\* Try to register character device \*/  major\_num = register\_chrdev(0, "lkm\_example", &file\_ops);  if (major\_num < 0) {  printk(KERN\_ALERT "Could not register device: %d\n", major\_num);  return major\_num;  } else {  printk(KERN\_INFO "lkm\_example module loaded with device major number %d\n",  major\_num);  return 0;  } } static void \_\_exit lkm\_example\_exit(void) {  /\* Remember — we have to clean up after ourselves. Unregister the character  device. \*/  unregister\_chrdev(major\_num, DEVICE\_NAME);  filp\_close(filp, NULL);  printk(KERN\_INFO "Goodbye, 2013747 zhangyizhen!\n"); } /\* Register module functions \*/ module\_init(lkm\_example\_init); module\_exit(lkm\_example\_exit); |

4.2 **Compile the above Linux kernel module**

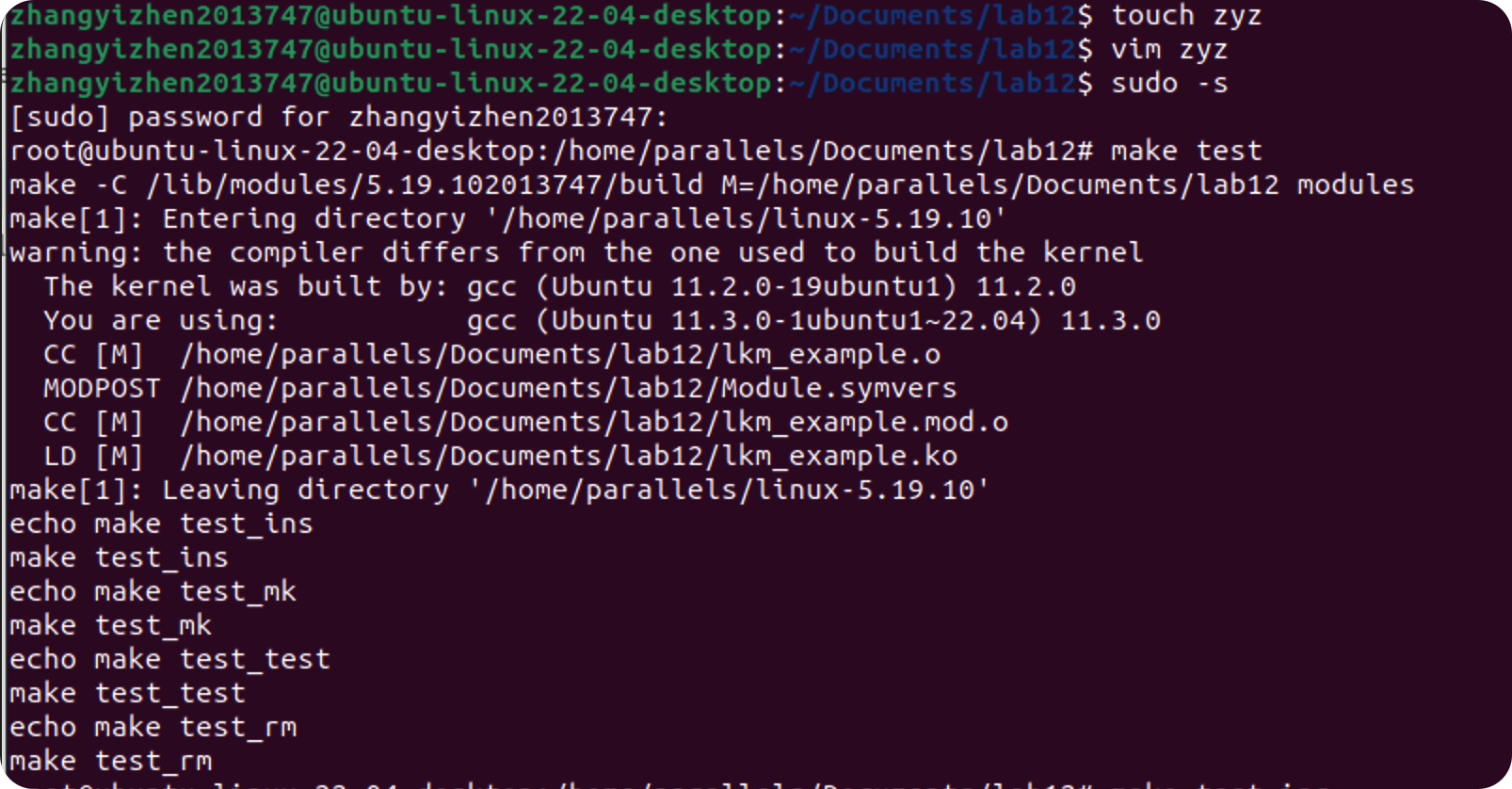
4.2.1 **创建Makefile文件:**

|  |
| --- |
| Makefile ModuleName=lkm\_example obj-m +=${ModuleName}.o all:${ModuleName}.ko ${ModuleName}.ko:${ModuleName}.c  make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules test:${ModuleName}.ko  echo make test\_ins  echo make test\_mk  echo make test\_test  echo make test\_rm test\_ins:${ModuleName}.ko  sudo dmesg -C  sudo insmod ${ModuleName}.ko name=/home/parallels/Documents/lab12/zyz  sudo dmesg test\_mk:${ModuleName}.ko  sudo mknod /dev/${ModuleName} c 509 0 test\_read:${ModuleName}.ko  sudo dmesg -C  cat /dev/${ModuleName}  sudo dmesg test\_write:${ModuleName}.ko  echo "hello 2013747 zhangyizhen 2022/12/8" >> /dev/${ModuleName} test\_rm:${ModuleName}.ko  sudo rmmod ${ModuleName}.ko  sudo dmesg .PHONY:clean clean:  make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean  sudo rm /dev/lkm\_example |

4.2.2 **编译**

|  |
| --- |
| Bash make test |

|  |
| --- |
| Makefile test:${ModuleName}.ko  echo make test\_ins  echo make test\_mk  echo make test\_test  echo make test\_rm |



4.2.3 **执行**

|  |
| --- |
| Bash make test\_ins  make test\_mk  make test\_read make test\_write make test\_rm |

1. make test\_ins ----内核自动分配的设备主号码

|  |
| --- |
| Makefile test\_ins:${ModuleName}.ko  sudo dmesg -C  sudo insmod ${ModuleName}.ko name=/home/parallels/Documents/lab12/zyz  sudo dmesg |

具体操作的文件为/home/parallels/Documents/lab12/zyz

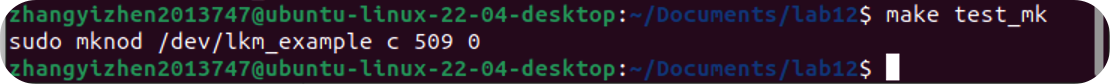


由内核自动分配的设备主号码的输出,可以看到是509号。

1. make test\_mk ----创建一个字符设备文件

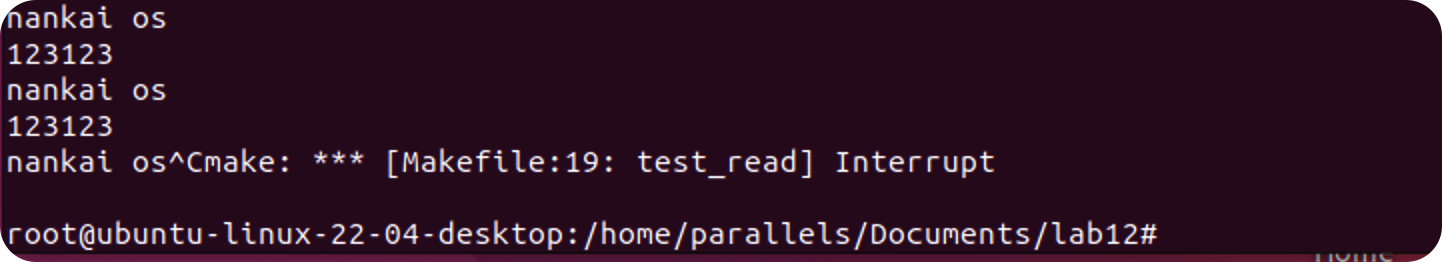
将 test\_mk 中的 MajorNum 替换为运行 make test\_ins 或 dmesg 后得到的值（在本实验中为509）。Mknod 命令中的“c”告诉 mknod 我们需要创建一个字符设备文件。

|  |
| --- |
| Makefile test\_mk:${ModuleName}.ko  sudo mknod /dev/${ModuleName} c 509 0 |



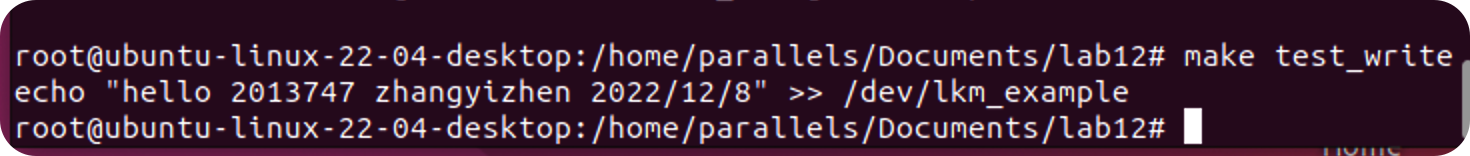
1. make test\_read ----读取zyz文件的具体内容到终端

|  |
| --- |
| Makefile test\_read:${ModuleName}.ko  sudo dmesg -C  cat /dev/${ModuleName}  sudo dmesg |

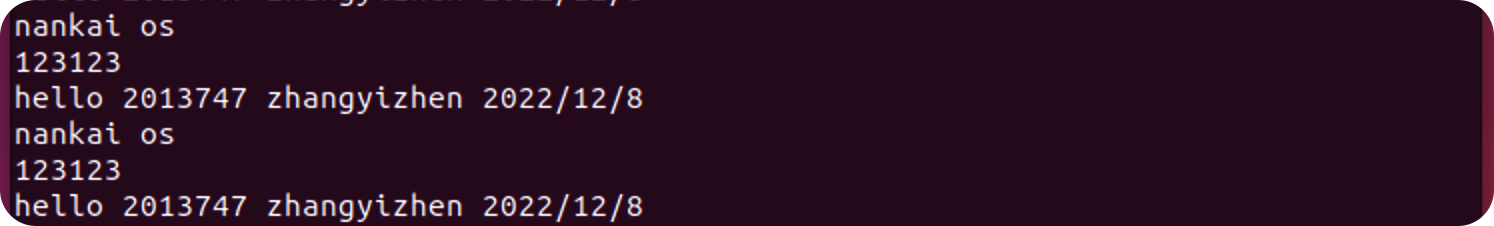


1. make test\_write ----写入文件设置的内容

|  |
| --- |
| Makefile test\_write:${ModuleName}.ko  echo "hello 2013747 zhangyizhen 2022/12/8" >> /dev/${ModuleName} |

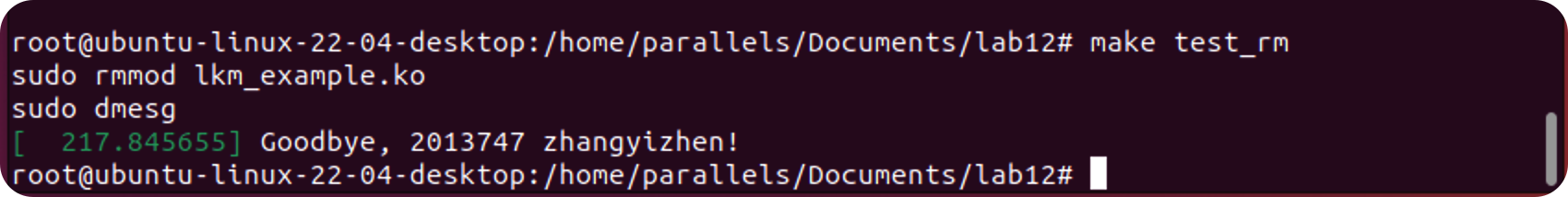


使用make test\_read 读取文件内容，可以看到写入成功:



1. make test\_rm

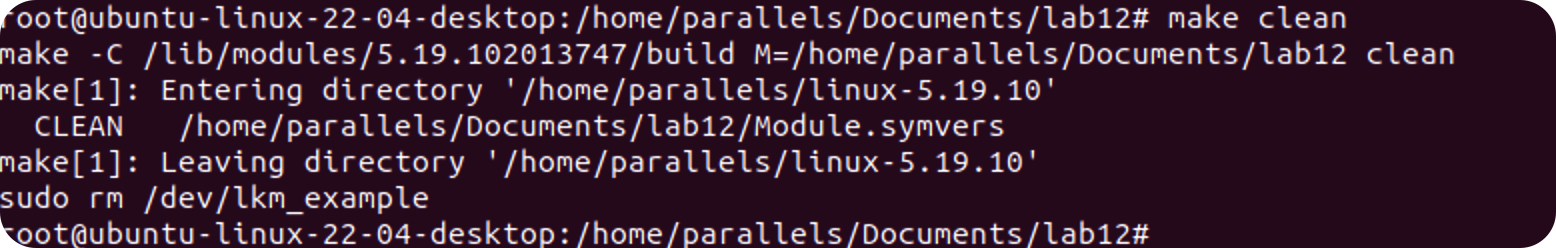
|  |
| --- |
| Makefile test\_rm:${ModuleName}.ko  sudo rmmod ${ModuleName}.ko  sudo dmesg |



4.2.4 **清除**

make clean

|  |
| --- |
| Makefile .PHONY:clean clean:  make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean  sudo rm /dev/lkm\_example |



5. **资料**

[老师的github实验文档](https://github.com/albertleecn/osplab)

6. **附件**

📎代码在压缩包中