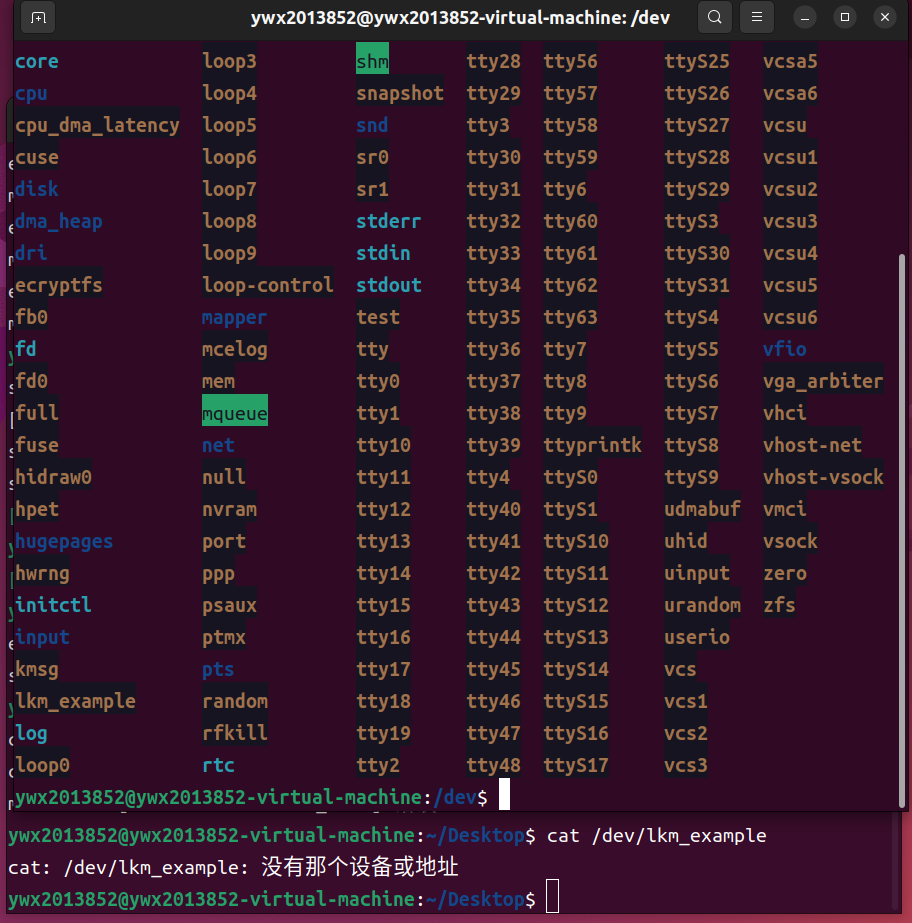
# 《操作系统》课第十二次实验报告

|  |  |
| --- | --- |
| 学院: | 软件学院 |
| 姓名: | 郁万祥 |
| 学号: | 2013852 |
| 邮箱: | yuwanxiang0114@163.com |
| 时间: | 2022.12.2 |

## 开篇感言

此次实验，一直会出现一个问题，就是在/dev目录下虽然已经创建了lkm\_example，但是在进行cat /dev/lkm\_example的时候，会出现问题：cat: /dev/lkm\_example: 没有那个设备或地址，问题如图：



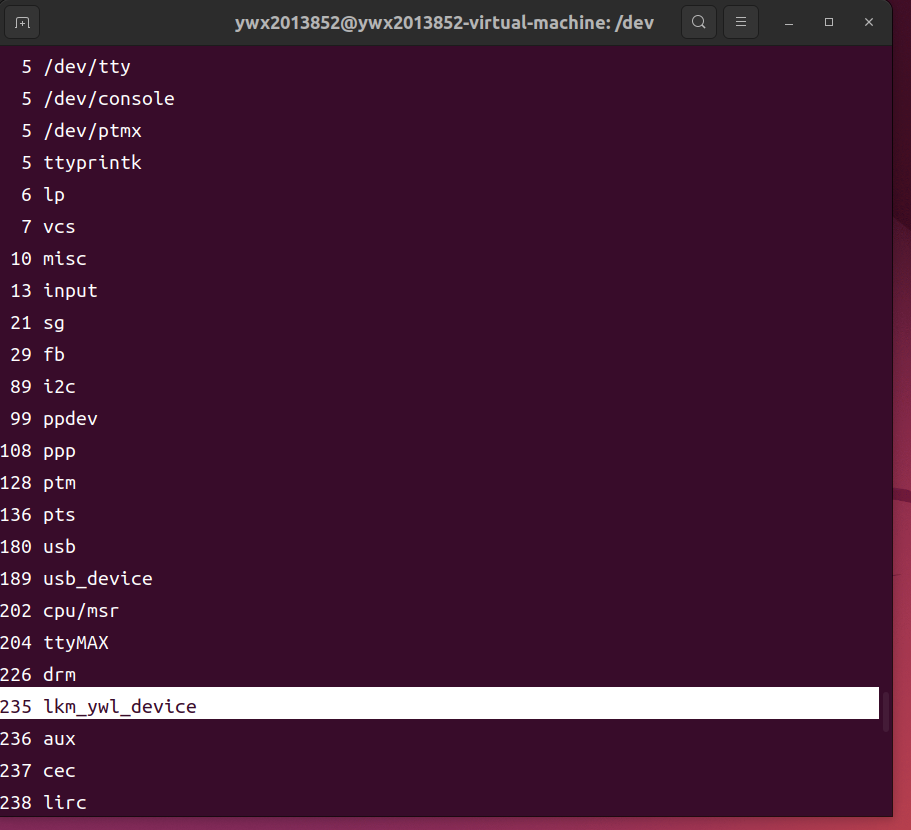
经过查询和通过同学之间的讨论，问题的出现的原因主要可以从两个方面入手：

1. 设备号可能不同，可能是237，也可能是238，(我的甚至是235！！)

这里介绍一个查看主设备号的方法：

先按照正常步骤添加设备，如果能够正常的cat，就没有问题，但是如果不能的话，不要着急sudo rm lab12ywx来删除设备，可以先使用命令cat /proc/devices

查看一下添加设备的主设备号



就像这里，我的是235。

1. 如果你已经在/dev中添加设备了，就需要重新启动，这样添加的设备才可以被删除，才可以更换信息，重新添加设备。

## 实验题目

Linux kernel Module Development（2）

## 实验目标

1. 使用C语言，编写程序，实现一个linux内核模块，可以读取和写一个指定的文件。

2、使用Makefile文件，进行编译命令的编辑。

## 原理方法

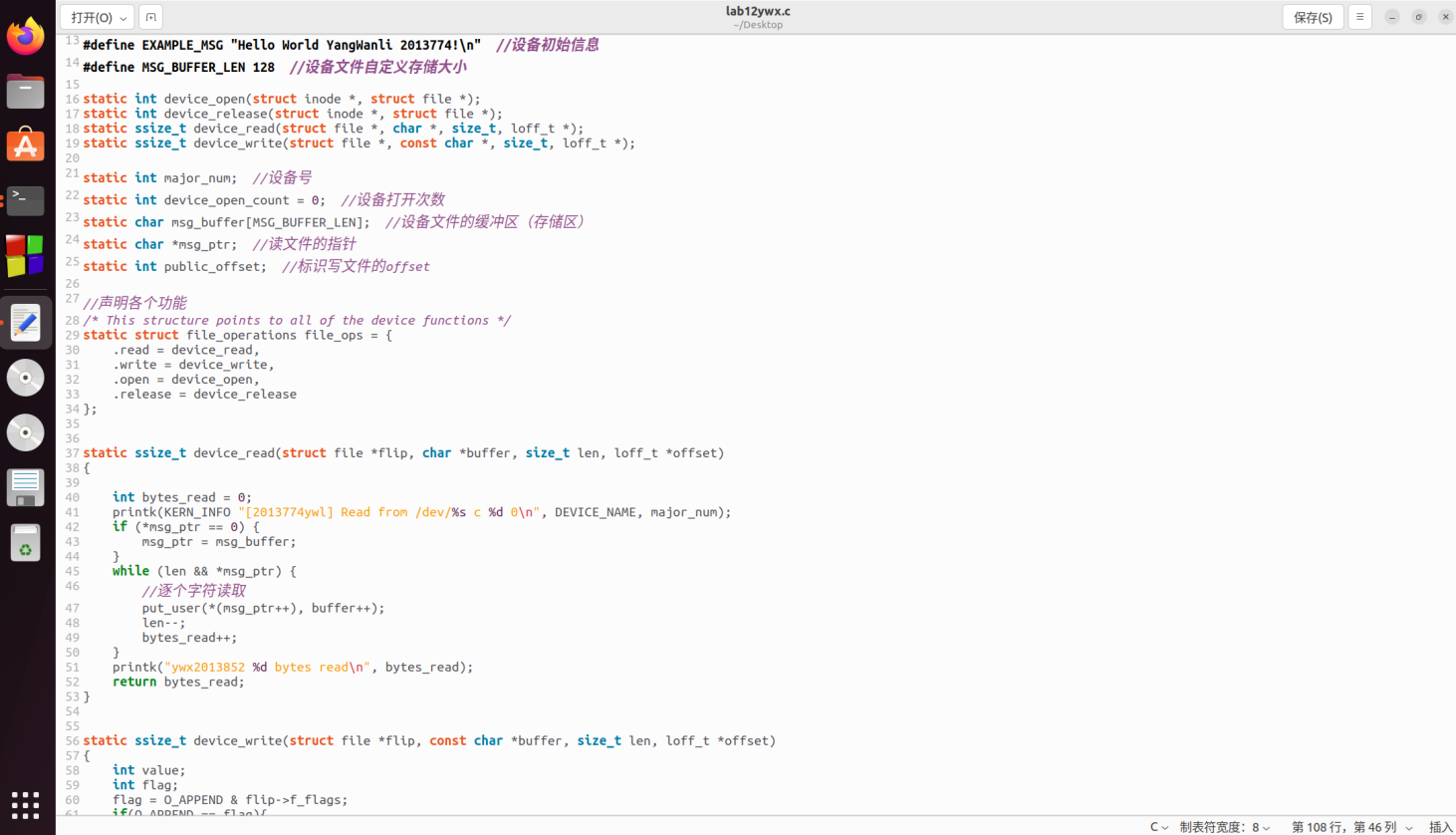
1. linux内核模块：模块是在内核空间运行的程序，实际上是一种目标对象文件，没有链接，不能独立运行，但是其代码可以在运行时链接到系统中作为内核的一部分运行或从内核中取下，从而可以动态扩充内核的功能。这种目标代码通常由一组函数和数据结构组成，用来实现一种文件系统，一个驱动程序，或其它内核

上层的功能。模块机制的完整叫法应该是动态可加载内核模块，一般就简称为模块。

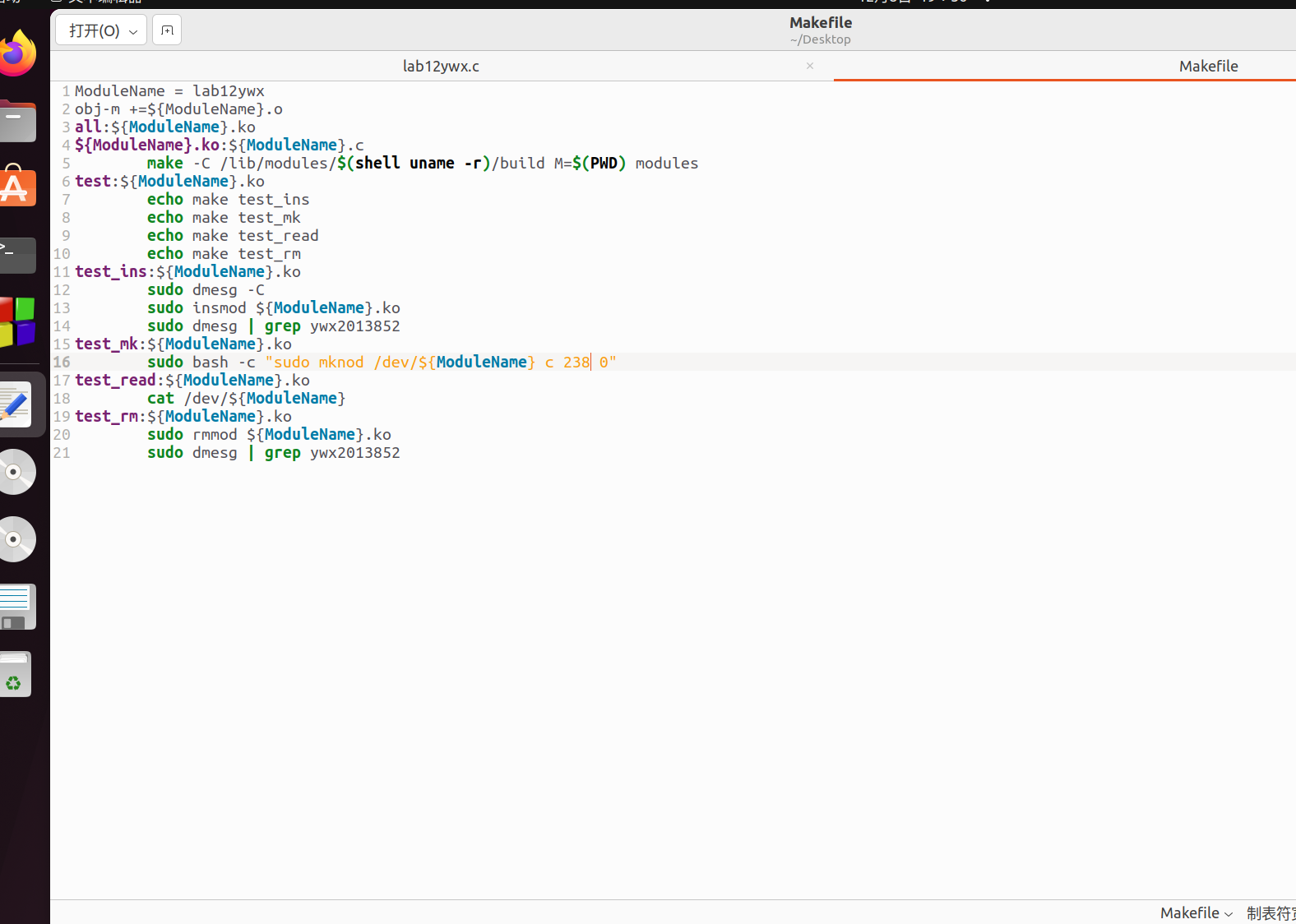
2、sysfs是一个基于内存的文件系统，它的作用是将内核信息以文件的方式提供给用户程序使用。该文件系统的目录层次结构严格按照内核的数据结构组织。除了二进制文件外，sysfs文件内容均以ASCII格式保存，且一个文件只保存一个数据，另外，一个文件不可大于一个内存页（通常为4096字节）。项目要求将char数组保存到属性文件。

## 具体步骤

1. 使用C语言创建一个内核module，实现读取和写一个指定的文件的功能。

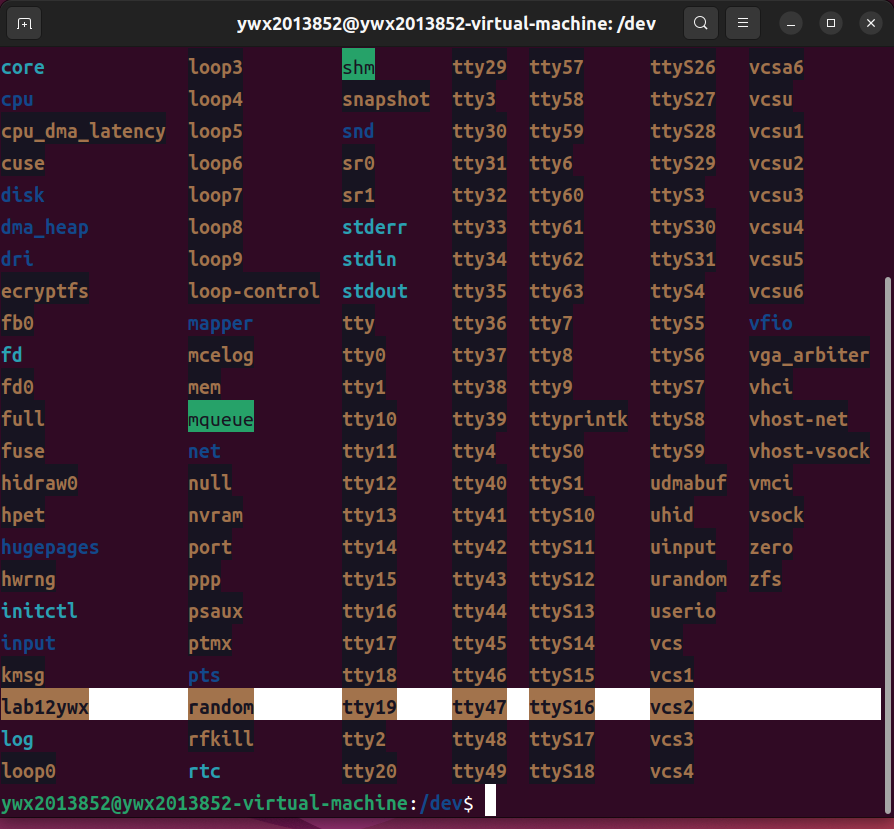


1. 为Module编写一个Makefile。



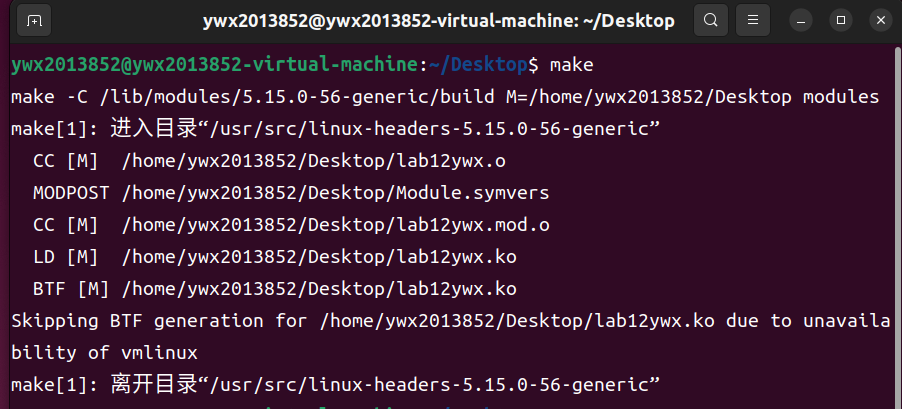
1. Module的插入与卸载。

我们可以进入/dev下使用命令ls查看此时的设备列表里面是否含有刚刚添加的lab12ywx

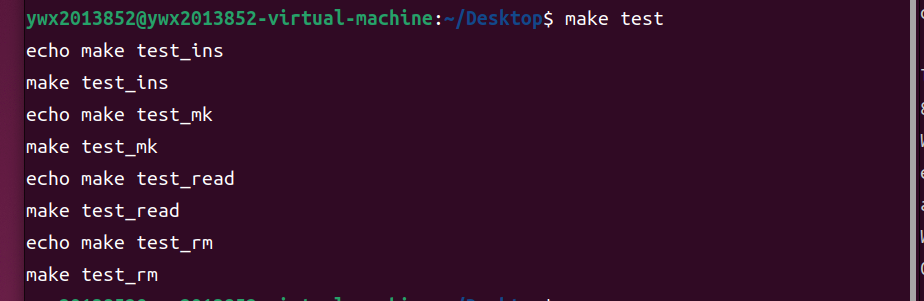


1. 功能的测试

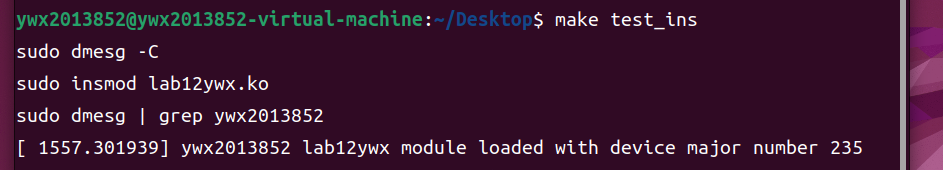
Make：



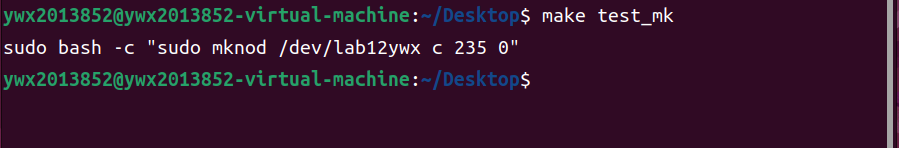
Make test：



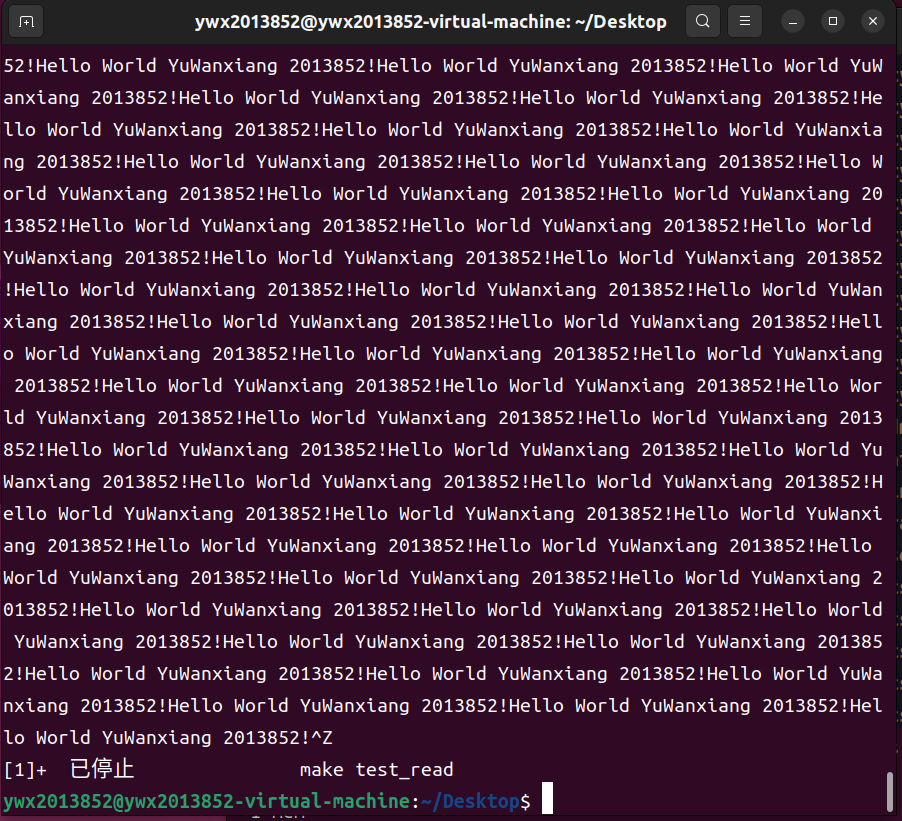
Make test\_ins：



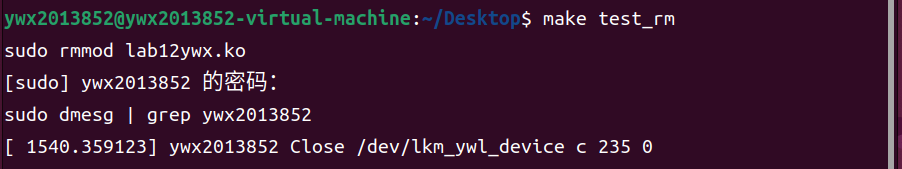
Make test\_mk:



Make test\_test：



Make test\_rm：



## 总结心得

之前的实验过程中，总是抱怨编译内核的时间过于长，每次都在waiting，其实，问题总比困难多，出现了问题，自然就会产生相应的解决方式，无论是操作系统，还是其他的技术，都是在这个过程中不断发展进步的。

## 参考资料

源码：

lab12ywx：

#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("YuWanxiang");

MODULE\_DESCRIPTION("A simple example of linux module.");

MODULE\_VERSION("0.01");

#define DEVICE\_NAME "lkm\_ywl\_device" //设备名称

#define EXAMPLE\_MSG "Hello World YuWanxiang 2013852!\n" //设备初始信息

#define MSG\_BUFFER\_LEN 128 //设备文件自定义存储大小

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; //设备号

static int device\_open\_count = 0; //设备打开次数

static char msg\_buffer[MSG\_BUFFER\_LEN]; //设备文件的缓冲区（存储区）

static char \*msg\_ptr; //读文件的指针

static int public\_offset; //标识写文件的offset

//声明各个功能

/\* 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

};

static ssize\_t device\_read(struct file \*flip, char \*buffer, size\_t len, loff\_t \*offset)

{

int bytes\_read = 0;

printk(KERN\_INFO "[2013774ywl] Read from /dev/%s c %d 0\n", DEVICE\_NAME, major\_num);

if (\*msg\_ptr == 0) {

msg\_ptr = msg\_buffer;

}

while (len && \*msg\_ptr) {

//逐个字符读取

put\_user(\*(msg\_ptr++), buffer++);

len--;

bytes\_read++;

}

printk("ywx2013852 %d bytes read\n", bytes\_read);

return bytes\_read;

}

static ssize\_t device\_write(struct file \*flip, const char \*buffer, size\_t len, loff\_t \*offset)

{

int value;

int flag;

flag = O\_APPEND & flip->f\_flags;

if(O\_APPEND == flag){

printk("ywx2013852 append\n");

value = copy\_from\_user(msg\_buffer + public\_offset, buffer, len);

public\_offset += len;

}

else if(flag == 0){

printk("ywx2013852 overlay\n");

memset(msg\_buffer,'\0',MSG\_BUFFER\_LEN);

value = copy\_from\_user(msg\_buffer, buffer, len);

public\_offset = len;

}

printk(KERN\_INFO "ywx2013852 length of message:%d\n", (int)len);

return len;

}

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

printk(KERN\_INFO "ywx2013852 Open /dev/%s c %d 0\n", DEVICE\_NAME, major\_num);

if (device\_open\_count) {

return -EBUSY;

}

device\_open\_count++;

try\_module\_get(THIS\_MODULE);

return 0;

}

static int device\_release(struct inode \*inode, struct file \*file)

{

printk(KERN\_INFO "ywx2013852 Close /dev/%s c %d 0\n", DEVICE\_NAME,major\_num);

device\_open\_count--;

module\_put(THIS\_MODULE);

return 0;

}

static int \_\_init lkm\_example\_init(void)

{

strncpy(msg\_buffer, EXAMPLE\_MSG, 31);

public\_offset = 31;

msg\_ptr = msg\_buffer;

major\_num = register\_chrdev(0, DEVICE\_NAME, &file\_ops);

if (major\_num < 0) {

printk(KERN\_ALERT "ywx2013852 Could not register device: %d\n", major\_num);

return major\_num;

} else {

printk(KERN\_INFO "ywx2013852 lab12ywx module loaded with device major number %d\n", major\_num);

return 0;

}

}

static void \_\_exit lkm\_example\_exit(void)

{

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

printk(KERN\_INFO "Goodbye World YuWanxiang lab12 !\n");

}

/\* Register module functions \*/

module\_init(lkm\_example\_init);

module\_exit(lkm\_example\_exit);

Makefile：

ModuleName = lab12ywx

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

sudo dmesg | grep ywx2013852

test\_mk:${ModuleName}.ko

sudo bash -c "sudo mknod /dev/${ModuleName} c 238 0"

test\_test:${ModuleName}.ko

cat /dev/${ModuleName}

test\_rm:${ModuleName}.ko

sudo rmmod ${ModuleName}.ko

sudo dmesg | grep ywx2013852