复习：

注册杂项设备：

misc\_register(&misc\_dev);

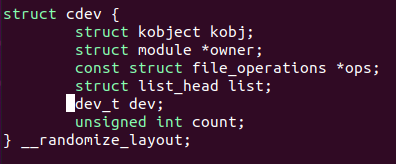
注销杂项设备：

misc\_deregister(&misc\_dev);

在学习注册字符类设备之前，我们需要先学习cdev结构体：

cdev结构体：描述字符设备的一个结构体。

定义在include/linux/cdev.h文件中：



我们主要先关注：

owner（属于哪一个模块）

file\_operations(文件操作集)

dev(设备号)

count（次设备号的数量）

如下的就先可以不用管：

list（链表）

kobj

步骤1：定义一个cdev结构体

步骤2：使用cdev\_init函数，初始化cdev里面的结构体成员变量。（到这儿只完成了初始化，并没有完成注册）

其同样定义在include/linux/cdev.h文件中：

void cdev\_init(struct cdev \*, const struct file\_operations \*);



参数：

cdev：要初始化的cdev结构体。

file\_operations：文件操作集。

由cdev -> ops = fops;可知：

实际上cdev\_init函数 就是 把文件操作集 写给cdev的成员变量：fops。

步骤3：使用cdev\_add函数注册到内核。

其同样定义在include/linux/cdev.h文件中：

int cdev\_add(struct cdev \*, dev\_t, unsigned);

参数：

cdev 结构体指针

dev\_t 设备号

unsigned 次设备号的数量

步骤4：使用cdev\_del函数从内核注销字符设备

其同样定义在include/linux/cdev.h文件中：

void cdev\_del(struct cdev \*);

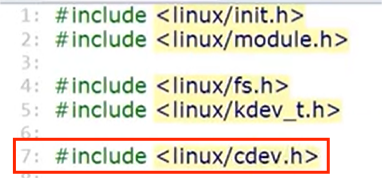
——————————————————————————————————————

实操：

创建第8个实验：拷贝实验7。（上节课分配了设备号，但注册字符设备的流程还没过完）



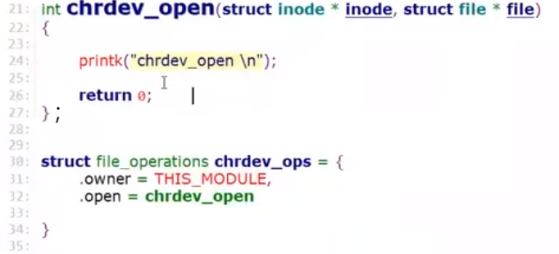
添加头文件：



定义cdev结构体：

IMG_256

定义文件操作集，并将open函数实现：



（在init函数里面的else后面）

填充结构体成员变量：

使用cdev\_init函数初始化cdev的结构体成员变量。

使用cdev\_add函数注册到内核:

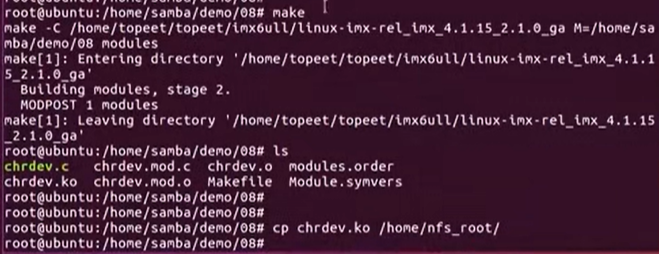


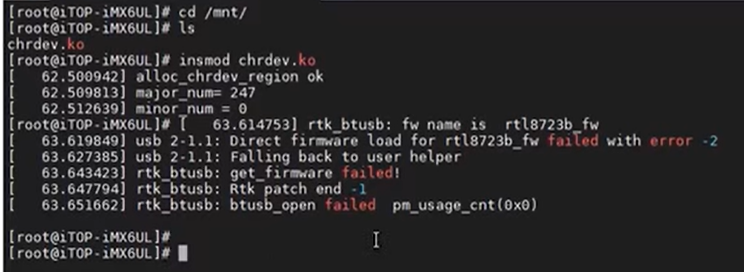
在exit函数里面完成对字符设备的注销：



到此就编写完成。

在虚拟机中进行编译拷贝驱动：





可以看到已经完成注册了。

为了测试字符设备是否正常工作，我们需要编写一个应用app:

将实验5的app拷贝改写：

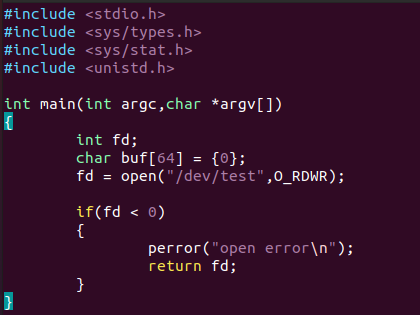
由于我们需要在app里面打开我们的设备节点，

但是字符设备在注册完成后，并不会自动生成一个设备节点。在驱动代码里面我们也可以发现，我们并没有写过设备节点的名字。

虽然我们在驱动里面有两个名称（schrdev和achrdev），但这个名字是设备的名字，也就是我们使用cat 命令（cat /proc/devices）能够查看到的名字。但那并不是设备节点的名字。

在app.c中要打开的设备节点为：/dev/test ：

(在app.c中就执行简单的打开设备节点操作。)



将其拷贝到开发板上。

由于/dev/ 目录下并没有test文件。

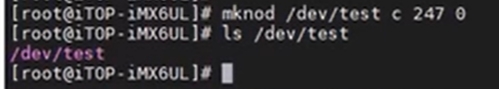
因此，我们需要使用mknod命令创建一个设备节点：

命令格式：

mknod 设备节点名称 设备节点类型（c） 主设备号 次设备号

例如：mknod /dev/test c 247 0

执行后他就会创建我们的设备节点。



测试：

如果整个流程没有问题的话,效果应为：

我们运行app应用 会去打开/dev/test 设备节点，该设备节点对应的：主设备号为247且从设备号为0 的设备 就会调用该设备的 文件操作集对应的open函数。那么就会打印chrdev\_open函数里面的”chrdev\_open\n”

结果也正如我们所想一样：



总结：

虽然我们注册字符设备节点成功了，但美中不足的是不能自动创建其字符设备节点。那么我们使用上就会更加麻烦。

那么下节课将讲述如何让他自动创建字符设备节点。