cdev/file\_operations/dev\_t

**一、背景介绍**

模块成功被加载之后，模块就要开始工作。

驱动模块的工作就是创建设备，并让设备能够正常工作。

本节主要讲解设备相关的知识。

**二、知识点：**

1.dev\_id

设备id，其实是个32bit的数，高12bit存储主设备号，低20位存储次设备号。在<linux/types.h>中声明。

2.申请和释放设备id

以下两个函数在<linux/fs.h>中声明。

alloc\_chrdev\_region(dev\_t \*dev, 0, int minor\_Count,char \*DeviceName);

unregister\_chrdev\_region(dev\_t dev, int minorCount);

设备注册成功之后，使用cat /proc/devices命令查看。

然后可在/dev目录下创建对应的设备节点。

3.主设备号和次设备号

主设备号对应驱动程序，

次设备号对应该驱动具体实现的设备。

MAJOR(dev\_t dev); //可从设备id中析取出主设备号。

MINOR(dev\_t dev); //可从设备id中析取出次设备号。

4.字符设备

下面内容都在<linux/cdev.h>中。

struct cdev {

struct kobject kobj;

struct module \*owner;

const struct file\_operations \*ops;

struct list\_head list;

dev\_t dev;

unsigned int count;

};

struct cdev \*cdev\_alloc(void);//申请一个cdev结构空间

void cdev\_init(struct cdev \*, const struct file\_operations \*);//初始化cdev，首先给cdev清零，然后用参数设置file\_operations字段

初始化完成之后，需要设置owner字段为THIS\_MODULE.

int cdev\_add(struct cdev \*, dev\_t, unsigned);//此步将cdev和dev\_t管理起来并在内核中激活。最后一个参数是次设备号。

设备激活后，就可以通过系统调用操作设备节点了。

void cdev\_del(struct cdev \*);//从内核中去掉设备。

**三、示例程序：**

#include <linux/init.h>

#include <linux/module.h>

#include <linux/types.h>

#include <linux/fs.h>

#include <linux/cdev.h>

MODULE\_LICENSE("Dual BSD/GPL");

dev\_t devNo;

int temp\_open(struct inode \*inode, struct file \*filp)

{

printk(KERN\_NOTICE "temp:temp\_open=====>");

printk(KERN\_NOTICE "temp:temp\_open<=====");

return 0;

}

int temp\_release(struct inode \*inode, struct file \*filp)

{

printk(KERN\_NOTICE "temp:temp\_release<=====");

printk(KERN\_NOTICE "temp:temp\_release=====>");

return 0;

}

ssize\_t temp\_read(struct file \*filp, char \_\_user \*buf, size\_t count, loff\_t \*f\_pos)

{

printk(KERN\_NOTICE "temp:temp\_read<=====");

printk(KERN\_NOTICE "temp:temp\_read=====>");

return 0;

}

ssize\_t temp\_write(struct file \*filp, const char \_\_user \*buf, size\_t count, loff\_t \*f\_pos)

{

printk(KERN\_NOTICE "temp:temp\_write<=====");

printk(KERN\_NOTICE "temp:temp\_write=====>");

return 0;

}

struct file\_operations filp={

.owner = THIS\_MODULE,

.open = temp\_open,

.release= temp\_release,

.read = temp\_read,

.write = temp\_write,

};

struct cdev mycdev;

static int \_\_init temp\_init(void)

{

int err;

printk(KERN\_INFO "TEMP: hello kernel!");

alloc\_chrdev\_region(&devNo,0,1,"tempDevice");

cdev\_init(&mycdev,&filp);

mycdev.owner=THIS\_MODULE;

err=cdev\_add(&mycdev,devNo,1);

return 0;

}

static void \_\_exit temp\_exit(void)

{

unregister\_chrdev\_region(devNo,1);

cdev\_del(&mycdev);

printk(KERN\_INFO "TEMP: Bye kernel!");

}

module\_init(temp\_init);

module\_exit(temp\_exit);