新字符设备驱动框架

一、分配和释放设备号

1、没有指定设备号的话就使用如下函数来申请设备号：

int alloc\_chrdev\_region(dev\_t \*dev, unsigned baseminor, unsigned count, const char \*name)

给定了设备的主设备号和次设备号就使用如下所示函数来注册设备号即可：

int register\_chrdev\_region(dev\_t from, unsigned count, const char \*name)

2、不管是通过alloc\_chrdev\_region 函数还是 register\_chrdev\_region 函数申请的设备号，统一使用如下释放函数：

void unregister\_chrdev\_region(dev\_t from, unsigned count)

参数 from 是要申请的起始设备号，也就是给定的设备号；参数 count 是要申请的数量， 一般都是一个；参数 name 是设备名字。

1. 新字符设备注册

1、Linux 中使用 cdev 结构体表示一个字符设备，cdev 结构体在 include/linux/cdev.h 文件中的定义如下：

struct cdev {

struct kobject kobj;

struct module \*owner;

const struct file\_operations \*ops;

struct list\_head list;

dev\_t dev;

unsigned int count;

};

编写字符设备驱动之前需要定义一个 cdev 结构体变量，这个变量就表示一个字符设备：

struct cdev test\_cdev;

2、定义好 cdev 变量以后就要使用 cdev\_init 函数对其进行初始化，cdev\_init 函数原型如下：

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

定义好 cdev 变量以后就要使用 cdev\_init 函数对其进行初始化，cdev\_init 函数原型如下：

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

参数 cdev 就是要初始化的 cdev 结构体变量，参数 fops 就是字符设备文件操作函数集合。

3、cdev\_add 函数用于向 Linux 系统添加字符设备(cdev 结构体变量)，首先使用 cdev\_init 函数完成对 cdev 结构体变量的初始化，然后使用 cdev\_add 函数向 Linux 系统添加这个字符设备：

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

参数 p 指向要添加的字符设备(cdev 结构体变量)，参数 dev 就是设备所使用的设备号， 参数 count 是要添加的设备数量。

4、卸载驱动的时候一定要使用cdev\_del函数从Linux内核中删除相应的字符设备，cdev\_del 函数原型如下：

void cdev\_del(struct cdev \*p)

参数 p 就是要删除的字符设备。

三、自动创建设备节点

1、自动创建设备节点的工作是在驱动程序的入口函数中完成的，一般在 cdev\_add 函数后面 添加自动创建设备节点相关代码。首先要创建一个 class 类，class 是个结构体，定义在文件 include/linux/device.h 里面。class\_create 是类创建函数， class\_create 是个宏定义，内容如下：

#define class\_create(owner, name) \

({ \

static struct lock\_class\_key \_\_key; \

\_\_class\_create(owner, name, &\_\_key); \

})

struct class \*\_\_class\_create(struct module \*owner, const char \*name, struct lock\_class\_key \*key)

class\_create 一共有两个参数，参数 owner 一般为 THIS\_MODULE，参数 name 是类名字。 返回值是个指向结构体 class 的指针，也就是创建的类。

2、卸载驱动程序的时候需要删除掉类，类删除函数为 class\_destroy，函数原型如下：

void class\_destroy(struct class \*cls);

参数 cls 就是要删除的类。

3、创建好类以后还不能实现自动创建设备节点，我们还需要在这个类下创建一个设 备。使用 device\_create 函数在类下面创建设备，device\_create 函数原型如下：

struct device \*device\_create(struct class \*class, struct device \*parent, dev\_t devt, void \*drvdata, const char \*fmt, ...)

device\_create 是个可变参数函数，参数 class 就是设备要创建哪个类下面；参数 parent 是父设备，一般为 NULL，也就是没有父设备；参数 devt 是设备号；参数 drvdata 是设备可能会使用的一些数据，一般为 NULL；参数 fmt 是设备名字，如果设置 fmt=xxx 的话，就会生成 /dev/xxx 这个设备文件。返回值就是创建好的设备。

4、卸载驱动的时候需要删除掉创建的设备，设备删除函数为 device\_destroy，函数 原型如下：

void device\_destroy(struct class \*class, dev\_t devt)

参数 classs 是要删除的设备所处的类，参数 devt 是要删除的设备号。

四、设置文件私有数据

1、每个硬件设备都有一些属性，比如主设备号(dev\_t)，类(class)、设备(device)、开关状态(state)等等，对于一个设备的所有属性信息我们最好将其做成一个结构体，编写驱动 open 函数的时候将设备结构体作为私有数据添加到设备文件中，在 open 函数里面设置好私有数据以后，在 write、read、close 等函数中直接读取 private\_data 即可得到设备结构体。

/\* 设备结构体 \*/

struct test\_dev{

dev\_t devid; /\* 设备号 \*/

struct cdev cdev; /\* cdev \*/

struct class \*class; /\* 类 \*/

struct device \*device; /\* 设备 \*/

int major; /\* 主设备号 \*/

int minor; /\* 次设备号 \*/

};

struct test\_dev testdev;

/\* open 函数 \*/

static int test\_open(struct inode \*inode, struct file \*filp)

{

filp->private\_data = &testdev; /\* 设置私有数据 \*/

return 0;

}