字符设备驱动

linux设备驱动基本知识点

1.嵌入式设备包含三类:字符设备, 块设备, 网络设备

2.在linux系统中,一切设备皆文件

3.文件类型(7种):

l-链接文件

p-管道文件

s-套接字文件

d-目录文件

b-块设备文件

c-字符设备

"-"-普通文件

其中字符设备文件与块设备文件统一称为设备文件,设备文件都存在于/dev目录下

设备文件的属性:

1.文件类型

2.文件访问权限

3.设备号 4(主设备号:哪类设备) 68(次设备号:哪个设备)

4.文件名字

所有的设备文件都在/dev目录下

（1）关于struct file

在linux系统中，**每打开一个文件（要打开才会有）**，在内核中都会关联一个struct file文件，这是由Linux内核在打开文件的时候创建的。在文件关闭后会释放这个struct file结构体。

其中重要的成员有：

loff_t f_pos---文件读写指针

struct file_operations* f_ops----该文件所对应的操作。

（2）关于struct inod

每一个**存在于文件系统的文件**都会有一个inod结构体和它对应。主要是用来记录文件的物理信息。不管文件是否打开，都会关联。

重要成员

inode->i_rdev---设备号。

字符设备编写流程

实现模块加载函数

a.申请主设备号,注册硬件操作方法

```
  dev_major=register_chrdev(0,"dev_module",&dev_fops);
```

b. 动态创建设备文件

```
dev_class=class_create(THIS_MODULE,"dev_class");
dev_device=device_create(dev_class,NULL,MKDEV(dev_major, 0),NULL,"xxx");实现模块卸载函数
```

c.释放主设备号

```
 unregister_chrdev(dev_major,"dev_module");
```

d.构建file_operations结构体变量

```
struct file_operations dev_fops={
    .owner   =THIS_MODULE,  //属于当前模块所拥有
    .open    =dev_open,
    .write   =dev_write,
    .read    =dev_read,
};
```

4). 实现操作硬件的方法

设备号

由于设备号对于内核来说是一种有限的资源，所以在驱动和设备号进行绑定之前，驱动应该向内核去申请设备号。

设备号在系统中用dev_t来表示/描述

typedef __kernel_dev_t dev_t; typedef **u32** kernel_dev_t; 由上面两个宏,可知:dev_t其实就是一个无符号的32位整型 其中高12位为主设备号,一般用major来表示,主设备号表示哪类设备 其中低20位为次设备号,一般用minot来表示,次设备号表示哪个设备

对设备号的一些操作宏

```
/*从设备号中分离出主设备号*/
#define MAJOR(dev)  ((unsigned int) ((dev) >> MINORBITS))
/*从设备号中分离出次设备号*/
#define MINOR(dev)  ((unsigned int) ((dev) & MINORMASK))
/*由主设备号与次设备号组成一个设备号*/
#define MKDEV(ma,mi)    (((ma) << 20) | (mi))
```

主设备号的申请

```
//方法一:静态申请,由驱动中指定。首先通过cat /proc/devices命令查看当前运行中的内核中哪个主设备号是空闲的，如果是空闲状态，驱动就可以向内核申请分配这个主设备号
//旧版本内核申请主设备号
ret=register_chrdev(100,"dev_module",&dev_fops);
    __register_chrdev(major, 0, 256, name, fops);

//新版本内核注册
struct cdev *cdev;      //表示一个字符设备
        /*申请主设备号*/
register_chrdev_region(major, baseminor, count, name);
        /*分配cdev*/
        cdev = cdev_alloc();
        cdev->owner = fops->owner;
        cdev->ops = fops;
        /*注册字符设备*/
        cdev_add(cdev, MKDEV(cd->major, baseminor), count);
```

```
//方法二:动态申请,由系统动态分配
//旧版本内核
dev_major=register_chrdev(0,"dev_module",&dev_fops);

//新版本内核
int alloc_chrdev_region(dev_t *dev, unsigned int firstminor,unsigned int count, char *name)
```

设备文件

设备文件也叫做设备节点

如何创建设备文件

方法一:手动创建,在终端执行如下命令

mknod /dev/xxx c dev_major 0

方法二:由系统动态帮我们创建设备文件,需要满足如下两个条件

a. 要求根文件系统中实现了mdev机制或者udev机制

b. 在驱动中必须要调用以下两个函数

```
/*创建一个设备类
 *param1:设备类的拥有者 THIS_MODULE
 *param2:设备类的名字,在系统中此名字不用重复
 */
struct class *=class_create(owner, name)
/*创建一个设备/设备文件/设备节点
 *param1:设备类
 *param2:一般为NULL
 *param3:设备号
 *param4:一般为NULL
 *param5:设备文件名字
 */
struct device*=device_create(struct class * cls, struct device * parent, MKDEV(major,0), void *
drvdata, xxx)
```

c.释放设备类与设备结构体

```
device_destroy(dev_class,MKDEV(dev_major, 0));
class_destroy(dev_class);
```

介绍新的结构体：cdev

```
struct cdev {            //在系统中用来表示一个字符设备
    struct kobject kobj;
    struct module *owner;
    const struct file_operations *ops;  //操作方法
    struct list_head list;
    dev_t dev;                          //设备号
    unsigned int count;
};
```

```
/*为cdev分配空间*/
struct cdev *cdev_alloc(void)
/*初始化struct cdev*/
cdev_init(struct cdev * cdev, const struct file_operations * fops);
/*注册cdev,告诉内核*/
cdev_add(struct cdev * p, dev_t dev, unsigned count);
/*注销/释放*/
cdev_del(struct cdev *dev);
```

用新的方法来实现设备号的申请

新的静态的方式来申请设备号

```
/*param1:设备号
 *param2:个数
 *param3:模块的名字
 *返回值:成功返回0,失败返回负值
 */
ret=register_chrdev_region(dev_t from, unsigned count, const char * name)
    /*申请设备号*/
    __register_chrdev_region(MAJOR(n), MINOR(n),next - n, name);


/*1.新的动态的方式来申请设备号
 *param1:存储设备号的变量指针
 *param2:起始次设备号
 *param3:此设备号个数
 *param4:模块的名字
 *返回值:成功返回0,失败返回负值
 */
alloc_chrdev_region(dev_t * dev, unsigned baseminor, unsigned count, const char * name);

/*释放设备号*/
unregister_chrdev_region(dev_t from, unsigned count);
```

如果用新的方法来申请设备号的驱动编程流程

1) 实现模块加载函数 a. 申请设备号

```
ret=alloc_chrdev_region(&dev_no,0,1,"dev_module");

或者

ret=register_chrdev_region(dev_no,1,"dev_module");
```

b. 创建字符设备struct cdev

```
cdev_alloc()

cdev_init(dev_cdev,&dev_fops);

cdev_add(dev_cdev,dev_no,1);
```

c. 创建设备文件

```
dev_class=class_create(THIS_MODULE,"dev_class");

dev_device=device_create(dev_class,NULL,dev_no,NULL,"xxx");
```

2) 实现模块卸载函数 release 3) 构建操作方法struct file_operations 4) 实现操作硬件的方法xxx_open xxx_write xxx_read

在系统中为指针分配空间的函数

/*param1:大小

```
 *param2:标号  GFP_KERNEL:表示分配不成功,则休眠
 *              GFP_ATOMIC:表示分配不成功,则不休眠
 */
kmalloc(size, flags)

/*释放空间*/
kfree(fs210_buf_dev);
```

用户空间与内核空间进行数据交互

```
#include<asm/uaccess.h>
    /*获取用户空间的数据
     *param1:目的
     *param2:源
     *param3:大小
     *返回值:失败返回>0,返回剩余的没有拷贝成功的字节数
     *       成功返回0
     */
    ret =copy_from_user(dev->buf,buf,size);

    /*将数据返回给用户空间*/
    ret =copy_to_user(buf,dev->buf,size);
```