有些设备无法使用某个类型框架去实现驱动，内核为这些misc设备提供了miscdevice驱动框架来实现驱动，我们在遇到无法利用某些类型框架去实现驱动的情况时，可以利用此框架来实现驱动。

# misc驱动框架的使用

linux内核使用miscdevice结构来表示一个misc设备：

struct miscdevice {

int minor;//次设备号

const char \*name;//设备名称

const struct file\_operations \*fops;//操作函数集

struct list\_head list;

struct device \*parent;

struct device \*this\_device;

const struct attribute\_group \*\*groups;//属性组

const char \*nodename;

umode\_t mode;

};

minor：misc设备的次设备号，misc设备的次设备号定义于miscdevice.h中，比如看门狗的为WATCHDOG\_MINOR，RTC的为RTC\_MINOR，我们在用miscdevice框架去实现对应驱动时可以使用这些次设备号，当然也可以设置为MISC\_DYNAMIC\_MINOR，那么miscdevice驱动中就会自动找到空闲的此设备号。

misc设备的注册：

int misc\_register(struct miscdevice \* misc)

misc设备的注销：

void misc\_deregister(struct miscdevice \*misc)

# misc驱动框架的实现

misc驱动其实本质还是字符设备：

int misc\_register(struct miscdevice \* misc)

{

//判断是否为动态申请次设备号

if (is\_dynamic) {

...

}

//misc设备主设备号固定为10

dev = MKDEV(MISC\_MAJOR, misc->minor);

//设备节点的创建

misc->this\_device =

device\_create\_with\_groups(misc\_class, misc->parent, dev, misc, misc->groups, "%s", misc->name);

}

可以看到misc\_register只是管理了次设备号和创建了设备节点，设备号的申请以及设备的注册则是通过misc\_init完成：

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

{

//创建/proc/misc节点

ret = proc\_create("misc", 0, NULL, &misc\_proc\_fops);

//创建类

misc\_class = class\_create(THIS\_MODULE, "misc");

//设备注册

register\_chrdev(MISC\_MAJOR,"misc",&misc\_fops)

}

以上就完成了我们在字符设备驱动中讲的创建字符设备的全部工作。但是注意我们在

miscdevice结构中定义的file\_operations还未用到，应用层打开设备时，并不能直接调用到我们注册的file\_operations->open，而是先调用misc\_fops->open:

static int misc\_open(struct inode \* inode, struct file \* file)

{

struct miscdevice \*c;

//获取我们自己定义的file\_operations

list\_for\_each\_entry(c, &misc\_list, list) {

if (c->minor == minor) {

new\_fops = fops\_get(c->fops);

break;

}

}

//文件私有数据

file->private\_data = c;

//将操作函数集替换成我们自己的fops

replace\_fops(file, new\_fops);

//调用我们自己的open

err = file->f\_op->open(inode,file);

}

可以看到misc驱动将字符设备驱动进行了封装，所以相比直接使用字符驱动设备框架，使用misc框架。

# 私有数据传递

我们注意到在misc\_open中文件私有数据被赋值为miscdevice，那么如果我们想获取设备更多的属性，如果miscdevice是某结构的成员，那么我们可以通过container\_of的方式获取这个结构。同时也可以注意到miscdevice中还有this\_device成员，我们同样可以利用dev\_get\_drvdata/dev\_set\_drvdata的方式传递私有数据。

# 示例说明

示例直接在异步IO的示例上进行了修改，只是将设备的注册与创建更改为使用misc驱动方式，私有数据通过dev\_get\_drvdata/dev\_set\_drvdata方式设置与获取，注意在open中我们将默认的file->private\_data进行了替换。应用程序直接使用异步IO的应用程序即可。