# Linux字符设备控制

在用户空间，使用ioctl系统调用来控制设备，原型如下：

int ioctl(int fd,unsigned long cmd,...)

ioctl驱动方法有和用户空间版本不用的原型

int (\*ioctl)(struct inode \*inode,struct file \*filp,unsigned int cmd,unsigned long arg)

struct file:在Linux系统中，每一个打开的文件，在内核中都会关联一个struct file ，它由内核在打开文件时创建，在文件关闭后释放。

struct inode:每一个存在于文件系统里面的文件都会关联一个inode结构，该结构主要用来记录文件物理上的信息。

## ioctl实现

1. 定义命令
2. 实现命令

### 定义命令

命令实质是一个整数，通常分为几个段：类型（8位）、序号，参数传递方向，参数长度。

Type(类型)：表明这是属于哪个设备的命令。

Number(序号)：用来区分同一设备的不同命令。

Direction（参数传递方向）：可能的值\_IOC\_NONE,\_IO\_READ,\_IO\_WRITE。

Size:参数长度

Linux系统提供了下面的宏来帮助定义命令：

\_IO(type,nr) :不带参数的命令

\_IOR(type ，nr ，datatype)：从驱动中读数据

\_IOW(type ，nr ，datatype)：写数据到驱动

\_IOWR(type ，nr ，datatype)：双向传送，tyoe和number成员作为参数被传递

例：#define MEM\_IOC\_MAGIC ‘m’ //定义幻数

#define MEM\_IOCSET

\_IOW(MEM\_IOC\_MAGIC, 0, int)

#define MEM\_IOCGQSET

\_IOR(MEM\_IOC\_MAGIC, 1, int)

### 实现命令

实现命令包括如下3个技术环节

1. 返回值
2. 参数使用
3. 命令操作

### 代码示例

/\*IO操作\*/

int memdev\_ioctl(struct inode \*inode, struct file \*filp,

unsigned int cmd, unsigned long arg)

{

int err = 0;

int ret = 0;

int ioarg = 0;

/\* 检测命令的有效性 \*/

if (\_IOC\_TYPE(cmd) != MEMDEV\_IOC\_MAGIC)

return -EINVAL;

if (\_IOC\_NR(cmd) > MEMDEV\_IOC\_MAXNR)

return -EINVAL;

/\* 根据命令类型，检测参数空间是否可以访问 \*/

if (\_IOC\_DIR(cmd) & \_IOC\_READ)

err = !access\_ok(VERIFY\_WRITE, (void \*)arg, \_IOC\_SIZE(cmd));

else if (\_IOC\_DIR(cmd) & \_IOC\_WRITE)

err = !access\_ok(VERIFY\_READ, (void \*)arg, \_IOC\_SIZE(cmd));

if (err)

return -EFAULT;

/\* 根据命令，执行相应的操作 \*/

switch(cmd) {

/\* 打印当前设备信息 \*/

case MEMDEV\_IOCPRINT:

printk("<--- CMD MEMDEV\_IOCPRINT Done--->\n\n");

break;

/\* 获取参数 \*/

case MEMDEV\_IOCGETDATA:

ioarg = 1101;

ret = \_\_put\_user(ioarg, (int \*)arg);

break;

/\* 设置参数 \*/

case MEMDEV\_IOCSETDATA:

ret = \_\_get\_user(ioarg, (int \*)arg);

printk("<--- In Kernel MEMDEV\_IOCSETDATA ioarg = %d --->\n\n",ioarg);

break;

default:

return -EINVAL;

}

return ret;

}