字符设备驱动ioctl实现用户层内核层通信

测试代码实现

memdev.h

```
#ifndef _MEMDEV_H_
#define _MEMDEV_H_
#include<linux/ioctl.h>
#ifndef MEMDEV_MAJOR
#define MEMDEV_MAJOR 0
#endif
#ifndef MEMDEV_NR_DEVS
#define MEMDEV_NR_DEVS 2
#endif
#ifndef MEMDEV_SIZE
#define MEMDEV_SIZE 4096
#endif
struct mem_dev
{
    char *data;
    unsigned long size;
};
#define MEMDEV_IOC_MAGIC 'k'
#define MEMDEV_IOCPRINT _IO(MEMDEV_IOC_MAGIC,0)
#define MEMDEV_IOCGETDATA _IOR(MEMDEV_IOC_MAGIC,1,int)
#define MEMDEV_IOC_MAXNR 3
#endif
```

memdev.c

```
#include<linux/module.h>
#include<linux/kernel.h>
#include<linux/types.h>
#include<linux/fs.h>
#include<linux/errno.h>
#include<linux/mm.h>
#include<linux/sched.h>
#include<linux/init.h>
#include<linux/cdev.h>
#include<linux/slab.h>
#include<asm/io.h>
#include<asm/system.h>
#include<asm/uaccess.h>
#include "memdev.h"
static int mem_major = MEMDEV_MAJOR;
module_param(mem_major,int,S_IRUGO);
struct mem_dev *mem_devp;
struct cdev cdev;
int mem_open(struct inode *inode,struct file *filp)
{
    struct mem_dev *dev;
    int num = MINOR(inode->i_rdev);
    if(num >= MEMDEV_NR_DEVS)
        return -ENODEV;
    dev = &mem_devp[num];
    filp->private_data = dev;
    return 0;
}
int mem_release(struct inode *inode,struct file *filp)
{
    return 0;
}
long memdev_ioctl(struct file *filp,unsigned int cmd,unsigned long arg)
{
    int err = 0;
    int ret = 0;
    int ioarg = 0;
    printk("kernel cmd is : %ld\n",cmd);
    switch(cmd)
    {
    case MEMDEV_IOCPRINT:
        printk("CMD MEMDEV_IOCPRINT DONE\n\n");
        break;
    case MEMDEV_IOCGETDATA:
        ioarg = 1101;
        if(copy_to_user((int *)arg,&ioarg,sizeof(int)))
            return -EFAULT;
        break;
    default:
        return -EINVAL;
    }
    return ret;
}
static const struct file_operations mem_fops =
{
    .owner = THIS_MODULE,
    .open = mem_open,
    .release = mem_release,
    .unlocked_ioctl = memdev_ioctl,
};
static int memdev_init(void)
{
    int result;
    int i;
    dev_t devno = MKDEV(mem_major,0);
    if (mem_major)
    {
        result = register_chrdev_region(devno,2,"memdev");
        printk("first mem_major is : %ld\n",mem_major);
    }
    else
    {
        result = alloc_chrdev_region(&devno,0,2,"memdev");
        mem_major = MAJOR(devno);
        printk("second mem_major is : %ld\n",mem_major);
    }
    if(result < 0)
```

```
        return result;
    cdev_init(&cdev,&mem_fops);
    cdev.owner = THIS_MODULE;
    cdev.ops = &mem_fops;
    cdev_add(&cdev,MKDEV(mem_major,0),MEMDEV_NR_DEVS);
    mem_devp = kmalloc(MEMDEV_NR_DEVS * sizeof(struct mem_dev),GFP_KERNEL);
    if(!mem_devp)
    {
        result = -ENOMEM;
        goto fail_malloc;
    }
    memset(mem_devp,0,sizeof(struct mem_dev));
    for(i=0;i<MEMDEV_NR_DEVS;i++)
    {
        mem_devp[i].size = MEMDEV_SIZE;
        mem_devp[i].data = kmalloc(MEMDEV_SIZE,GFP_KERNEL);
        memset(mem_devp[i].data,0,MEMDEV_SIZE);
    }
    return 0;
    fail_malloc:
    unregister_chrdev_region(devno,1);
    return result;
}
static void memdev_exit(void)
{
    cdev_del(&cdev);
    kfree(mem_devp);
    unregister_chrdev_region(MKDEV(mem_major,0),2);
}
MODULE_LICENSE("GPL");
module_init(memdev_init);
module_exit(memdev_exit);
Makefile
obj-m += memdev.o
CURRENT_PATH := $(shell pwd)
LINUX_KERNEL := $(shell uname -r)
LINUX_KERNEL_PATH := /usr/src/linux-headers-$(LINUX_KERNEL)
all:
    make -C $(LINUX_KERNEL_PATH) M=$(CURRENT_PATH) modules
clean:
    make -C $(LINUX_KERNEL_PATH) M=$(CURRENT_PATH) clean
app-ioctl.c
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include "memdev.h"
int main(void)
{
    int fd = 0;
    int cmd;
    int arg = 0;
    char Buf[4096];
```

```
        fd = open("/dev/memdev0",O_RDWR);
        if(fd < 0)
        {
            printf("Open Dev Mem Erro\n");
            return -1;
        }

        printf("call memdev_iocprint\n");
        cmd = MEMDEV_IOCPRINT;
        printf("userspace cmd is : %ld\n",cmd);
        if(ioctl(fd,cmd,&arg) < 0)
        {
            printf("call cmd MEMDEV_IOCPRINT fail\n");
            return -1;
        }
        printf("call MEMDEV_IOCGETDATA\n");
        cmd = MEMDEV_IOCGETDATA;
        if(ioctl(fd,cmd,&arg) < 0)
        {
            printf("call cmd MEMDEV_IOCGETDATA fail\n");
            return -1;
        }
        printf("in user space MEMDEV_IOCGETDATA get data is %d\n\n",arg);
        close(fd);
        return 0;
    }
```

编译memdev，对应的设备驱动

make
insmod memdev.ko
在dmesg中会有输出结果init，在/proc/devices中生成对应的设备驱动号249

创建设备节点
cat /proc/devices中生成的memdev节点编号249
mknod /dev/memdev0 c 249 0创建设备节点，并将对应的设备节点与设备驱动号进行绑定。
当打开该设备节点进行后续操作时，将会由设备驱动文件进行具体实现

编译app ioctl测试文件
gcc -o app-ioctl app-ioctl.c
./app-ioctl
正常执行结果会显示，dmesg也会显示内核结果