字符设备驱动样例与测试

1.驱动代码

```
/*
 *  使用一个4kb的虚拟内存设备作为字符驱动的测试
*/
#include <linux/module.h>
#include <linux/fs.h>
#include <linux/init.h>
#include <linux/cdev.h>
#include <linux/slab.h>
#include <linux/uaccess.h>

#define GLOBALMEM_SIZE 0x1000       /* 虚拟内存设备的缓存空间大小4kb */
#define MEM_CLEAR 0x1               /* 清0以上4kb内存 */
#define GLOBALMEM_MAJOR 230         /* 虚拟内存设备的主设备号 */

/* 虚拟内存设备 */
struct globalmem_dev {
    struct cdev cdev;
    unsigned char mem[GLOBALMEM_SIZE];
};


static int globalmem_major = GLOBALMEM_MAJOR;
struct globalmem_dev *globalmem_devp;


/* 应用程序用open函数打开设备文件时调用 */
static int globalmem_open(struct inode *inode, struct file *filp)
{
    filp->private_data = globalmem_devp;
    return 0;
}

/* 应用程序用read函数读取打开的设备文件时调用 */
static ssize_t globalmem_read(struct file *filp, char __user *buf,
        size_t size, loff_t *ppos)
{
    unsigned long p = *ppos;
    unsigned int count = size;
    int ret = 0;

    struct globalmem_dev *dev = filp->private_data;

    if(p >= GLOBALMEM_SIZE)
        return 0;
```

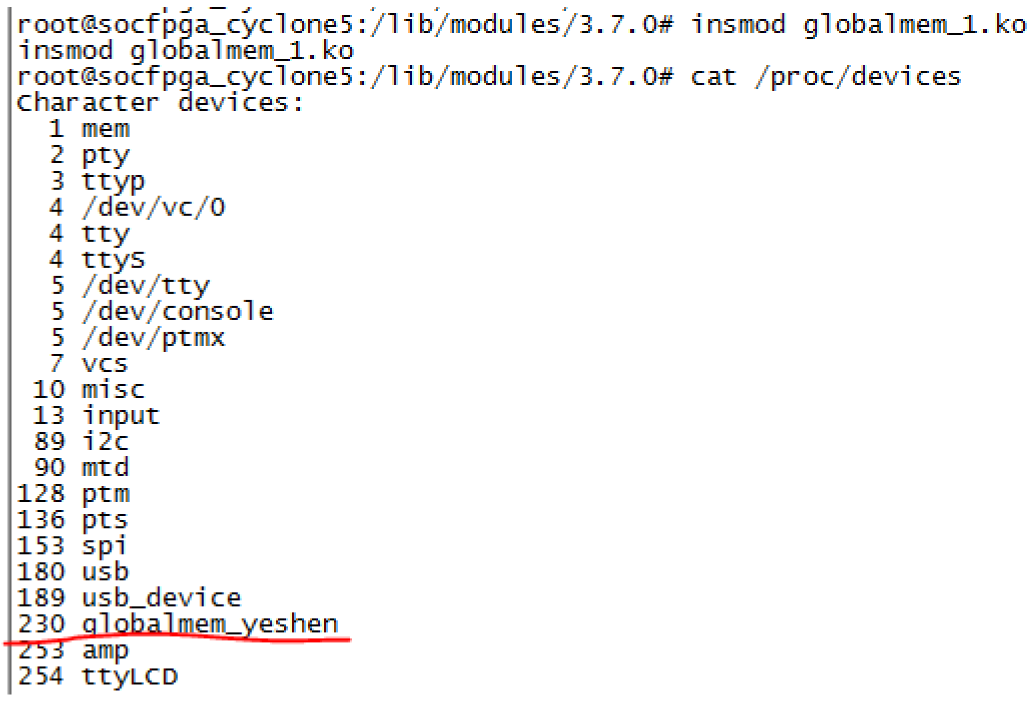
```c
    if(count > GLOBALMEM_SIZE - p)
        count = GLOBALMEM_SIZE - p;

    if(copy_to_user(buf,dev->mem + p,count)){
        ret = - EFAULT;
    }else{
        *ppos += count;
        ret = count;
        printk(KERN_INFO "read %u byte(s) from %lu\n", count, p);
    }

    return ret;
}

/* 应用程序用write函数写入打开的设备文件时调用 */
static ssize_t globalmem_write(struct file *filp, const char __user *buf,
                                size_t size, loff_t *ppos)
{
    unsigned long p = *ppos;
    unsigned int count = size;
    int ret = 0;

    struct globalmem_dev *dev = filp->private_data;

    if(p >= GLOBALMEM_SIZE)
        return 0;

    if(count > GLOBALMEM_SIZE - p)
        count = GLOBALMEM_SIZE - p;

    if(copy_from_user(dev->mem + p, buf, count)){
        ret = - EFAULT;
    }else{
        *ppos += count;
        ret = count;
        printk(KERN_INFO "written %u byte(s) to %lu\n", count,p);
    }

    return ret;
}

/* 应用程序用lseek函数更改打开的设备文件的读写位置时调用 */
static loff_t globalmem_llseek(struct file *filp, loff_t offset, int orig)
{
    loff_t ret;
    switch(orig){
    case 0:     /* 从文件开始处偏移 */
        if(offset < 0) {
            ret = -EINVAL;
            break;
        }
        if((unsigned int)offset >= GLOBALMEM_SIZE){
            ret = -EINVAL;
            break;
        }
        filp->f_pos = offset;
        ret = filp->f_pos;
        break;
    case 1:     /* 从当前位置偏移 */
        if((filp->f_pos + offset) < 0) {
            ret = -EINVAL;
            break;
        }
        if((filp->f_pos + offset) >= GLOBALMEM_SIZE){
            ret = -EINVAL;
            break;
        }
        filp->f_pos += offset;
        ret = filp->f_pos;
        break;
    default:
        ret = -EINVAL;
    }

    return ret;
}

/* 应用程序用ioctl函数更改打开的设备文件的属性时调用 */
static long globalmem_ioctl(struct file *filp, unsigned int cmd, unsigned long arg)
{
    struct globalmem_dev *dev = filp->private_data;

    switch(cmd){
    case MEM_CLEAR:
        memset(dev->mem,0,GLOBALMEM_SIZE);
        printk(KERN_INFO "globalmem is set to zero\n");
        break;
    default:
        return -EINVAL;
    }

    return 0;
}

/* 应用程序用close函数关闭打开的设备文件时调用 */
static int globalmem_release(struct inode *inode, struct file *filp)
{
    return 0;
}



/* fill in the file_operation structure */
static const struct file_operations globalmem_fops = {
    .owner = THIS_MODULE,
    .llseek = globalmem_llseek,
    .read = globalmem_read,
    .write = globalmem_write,
    .unlocked_ioctl = globalmem_ioctl,
    .open = globalmem_open,
    .release = globalmem_release,
};

/* 设置和注册设备 */
static void globalmem_setup_cdev(struct globalmem_dev *dev,int index)
{
    int err;
    dev_t devno = MKDEV(globalmem_major,index);

    cdev_init(&dev->cdev,&globalmem_fops);      /* 绑定globalmem_fops到字符设备，并作些初始化
*/
    dev->cdev.owner = THIS_MODULE;

    err = cdev_add(&dev->cdev, devno, 1);       /* 注册字符设备 */

    if(err)
        printk(KERN_NOTICE "Error %d adding globalmem %d\n", err, index);
}

/* 加载模块时调用 */
static int __init globalmem_init(void)
{
    printk("insmod globalmem_1.ko\n");
    int ret;
    dev_t devno = MKDEV(globalmem_major, 0);

    /* 分配设备号 */
    if(globalmem_major)
        ret = register_chrdev_region(devno, 1, "globalmem_yeshen");     //显示在
/proc/devices
    else{
        ret = alloc_chrdev_region(&devno, 0, 1, "globalmem_yeshen");
        globalmem_major = MAJOR(devno);
    }

    if(ret < 0)
        return ret;

    globalmem_devp = kzalloc(sizeof(struct globalmem_dev), GFP_KERNEL);
    if(!globalmem_devp) {
        ret = -ENOMEM;
        goto fail_malloc;
    }

    /* 设置注册设备 */
    globalmem_setup_cdev(globalmem_devp,0);     /* 0是子设备号 */
    return 0;

fail_malloc:
    unregister_chrdev_region(devno, 1);
    return ret;

}

/* 卸载模块时调用 */
static void __exit globalmem_exit(void)
{
    printk("rmmod globalmem_1.ko\n");
    cdev_del(&globalmem_devp->cdev);    /* 注销虚拟内存设备中的字符设备 */
    kfree(globalmem_devp);              /* 释放虚拟内存设备的空间 */
    unregister_chrdev_region(MKDEV(globalmem_major,0), 1);      /* 释放字符设备号 */
}

module_param(globalmem_major, int, S_IRUGO);

module_init(globalmem_init);
module_exit(globalmem_exit);

MODULE_AUTHOR("Yeshen 569242715@qq.com");
MODULE_LICENSE("GPL v2");
```

2.测试代码

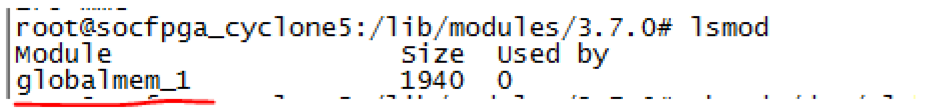
```
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>


#define DEVICE "/dev/globalmem"

int main(void)
{
    int fd,ret = 0;
    unsigned char wbuf[] = "yeshen is fantastic";
    unsigned char rbuf[100] = {0};
    fd = open(DEVICE,O_RDWR);   /* 调用驱动的globalmem_open */
    if(fd < 0){
        printf("open device /dev/globalmem failed!\n");
        return -1;
    }

    ret = write(fd,wbuf,sizeof(wbuf));      /* 调用驱动的globalmem_write */
    if(ret < 0){
        printf("write device /dev/globalmem failed!\n");
        return ret;
```

```c
    }

/*  close(fd);
    fd = open(DEVICE,O_RDWR);
    if(fd < 0){
        printf("open device /dev/globalmem failed!\n");
        return -1;
    }
*/
    lseek(fd,0,SEEK_SET);       /* 调用驱动的globalmem_llseek */
    ret = read(fd,rbuf,10);     /* 调用驱动的globalmem_read */
    if(ret < 0){
        printf("read device /dev/globalmem failed!\n");
        return ret;
    }
    printf("get the string from /dev/globalmem:%s\n",rbuf);


    lseek(fd,2,SEEK_CUR);   /* 调用驱动的globalmem_llseek */
    ret = read(fd,rbuf,5);  /* 调用驱动的globalmem_read */
    if(ret < 0){
        printf("read device /dev/globalmem failed!\n");
        return ret;
    }
    printf("get the string from /dev/globalmem:%s\n",rbuf);

    close(fd);  /* 调用驱动的globalmem_release */

    return ret;
}
```

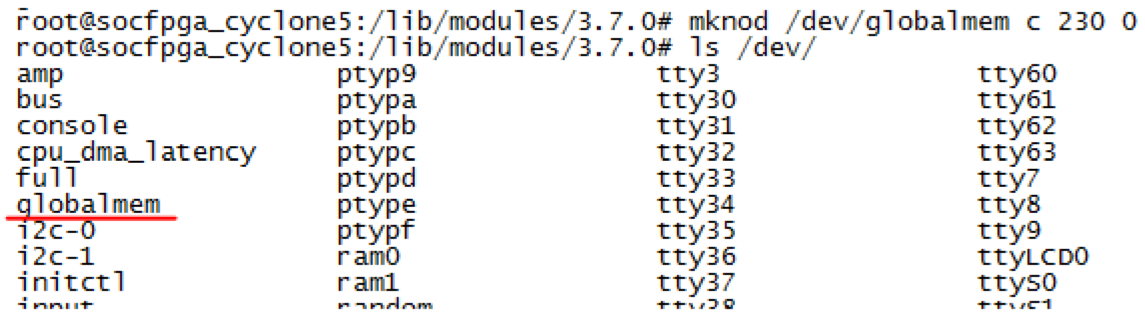
3.简要说明



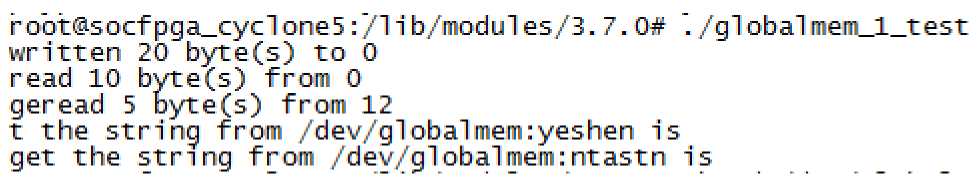
```
root@socfpga_cyclone5:/lib/modules/3.7.0# insmod globalmem_1.ko
insmod globalmem_1.ko
root@socfpga_cyclone5:/lib/modules/3.7.0# cat /proc/devices
Character devices:
  1 mem
  2 pty
  3 ttyp
  4 /dev/vc/0
  4 tty
  4 ttyS
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  7 vcs
 10 misc
 13 input
 89 i2c
 90 mtd
128 ptm
136 pts
153 spi
180 usb
189 usb_device
230 globalmem_yeshen
253 amp
254 ttyLCD
```



```
root@socfpga_cyclone5:/lib/modules/3.7.0# lsmod
Module                  Size  Used by
globalmem_1             1940  0
```



```
root@socfpga_cyclone5:/lib/modules/3.7.0# mknod /dev/globalmem c 230 0
root@socfpga_cyclone5:/lib/modules/3.7.0# ls /dev/
amp                 ptyp9               tty3                tty60
bus                 ptypa               tty30               tty61
console             ptypb               tty31               tty62
cpu_dma_latency     ptypc               tty32               tty63
full                ptypd               tty33               tty7
globalmem           ptype               tty34               tty8
i2c-0               ptypf               tty35               tty9
i2c-1               ram0                tty36               ttyLCD0
initctl             ram1                tty37               ttyS0
```



```
root@socfpga_cyclone5:/lib/modules/3.7.0# ./globalmem_1_test
written 20 byte(s) to 0
read 10 byte(s) from 0
geread 5 byte(s) from 12
t the string from /dev/globalmem:yeshen is
get the string from /dev/globalmem:ntastn is
```

1.通过insmod globalmem_1.ko加载字符驱动模块，然后看到/proc/devices下多出了设备号为230的globalmem_yeshen设备，证明设备已经注册进内核。

2.lsmod显示多出了globalmem_1的模块，证明模块已经加载。

3.mknod /dev/globalmem c 230 0创建字符设备节点

4.最后运行测试程序globalmem_1_test，对打印信息分别说明：

- written 20 byte(s) to 0 ：通过测试程序中的write(fd,wbuf,sizeof(wbuf))最终调用到驱动中的globalmem_write，实现将"yeshen is fantastic"写到虚拟内存设备中去。
- read 10 byte(s) from 0：通过测试程序中的lseek(fd,0,SEEK_SET)和read(fd,rbuf,10)最终调用到驱动的globalmem_read，将虚拟内存中开始的10字节读到用户空间。
- get the string from /dev/globalmem:**yeshen is** ：这是从虚拟内存设备globalmem_dev中读出的10个字节，证明之前成功写入。

- read 5 byte(s) from 12：通过测试程序中的lseek(fd,2,SEEK_SET)和read(fd,rbuf,5)最终调用到驱动的globalmem_read，将虚拟内存中的位置12处的5字节读到用户空间。证明之前的偏移10字节和当前的偏移2字节正确工作。
- get the string from /dev/globalmem:**fanta**：这是从虚拟内存设备globalmem_dev中读出的5个字节，证明之前成功写入。