编写字符设备驱动实现内核态与用户态通信

2015年11月01日 22:20:04 阅读数：778 标签: linux centos 内核 更多

本次主要实现的是通过编写字符设备，实现从用户态获得只有内核太才有权限访问的进程段地址（比如代码段，其他同理）

初稿主要代码如下：

firstdriver.c

```
/*字符注册模块实现如下*/
#include<linux/slab.h>
#include<linux/module.h>
#include<linux/init.h>
#include<linux/types.h>
#include<linux/errno.h>
#include<linux/fs.h>
#include<linux/mm.h>
#include<linux/cdev.h>
#include<asm/io.h>
#include<linux/sched.h>
#include<asm/uaccess.h>
#include<linux/kernel.h>
#include<linux/list.h>
#include<linux/sem.h>
#include<linux/pid.h>


#define MAJOR_NUM 260
#define MINOR_NUM 0
#define MINIOR_NUM 0
#define DEVICE_NUM 1


//以下为驱动需要实现的四个函数的生命，注册之后，用户态程序即调用此对应函数
static int firstdriver_open(struct inode *, struct file *);
static int firstdriver_release(struct inode *,struct file *);
static ssize_t firstdriver_read(struct file *, char __user *,size_t, loff_t *);
static ssize_t firstdriver_write(struct file *,char __user *,size_t ,loff_t *);
struct ret_type   //定义返回用户态的函数类型
{
        unsigned long start_code;
        unsigned long end_code;
};

static int firstdriver_major = MAJOR_NUM;  //定义主设备号
static int firstdriver_minor = MINOR_NUM;  //定义次设备号
struct file_operations firstdriver_fops =
        {
                .owner = THIS_MODULE,
                .open = firstdriver_open,
                .write = firstdriver_write,
                .release = firstdriver_release,
                .read = firstdriver_read,
        };
struct cdev *cdev;
static pid_t pidno = 1;
static int __init firstdriver_init(void)
{
                int ret = 0;
                dev_t devno = MKDEV(MAJOR_NUM,MINOR_NUM);
                cdev = cdev_alloc();//分配设备空间，这里为了简单，静态指定了设备号，最好动态申请
        if(register_chrdev_region(devno,DEVICE_NUM,"firstdriver"))
        {
                printk(KERN_ALERT"register_chrdev_region fail\n");
                return -1;
        }
        else
        {
                cdev_init(cdev,&firstdriver_fops);
                cdev->owner = THIS_MODULE;
```

```
62                 cdev->ops = &firstdriver_fops;
                                                63                 if((ret = cdev_add(cdev,devno,1)))
64                         printk(KERN_ALERT"Error in adding firstdriver\n");
65                 else
66                         printk(KERN_ALERT"register success\n");
67         }
68         return ret;
69 }
70 static void __exit firstdriver_exit(void)
71 {
72         dev_t devno = MKDEV(MAJOR_NUM,0);
73         cdev_del(cdev);
74         unregister_chrdev_region(devno,1);
75         if(cdev)
76                 kfree(cdev);
77         cdev = NULL;
78 }
79 static int firstdriver_open(struct inode * inode ,struct file * filp)
80 {
81         printk(KERN_ALERT"open succedd\n");
82         return 0;
83 }
84 static int firstdriver_release(struct inode* inode, struct file *filp)
85 {
86         printk(KERN_ALERT"release success\n");
87         return 0;
88 }
89 static ssize_t firstdriver_read(struct file *filp,char *buf,size_t len, loff_t * off)
90 {
91         printk("READING:\n");
92         struct task_struct *p = NULL;
93         struct ret_type *tmp;
94         struct pid *kpid = find_get_pid(pidno);
95         tmp = kmalloc(sizeof(*tmp),GFP_KERNEL);
96         if(tmp == NULL){
97                 return -ENOMEM;
98         }
99         p = pid_task(kpid,PIDTYPE_PID);
100        if(p == NULL){
101   kfree(tmp);
102                printk(KERN_ALERT"find task failed\n");
103                return -1;
104        }
105        if(p->mm == NULL){
106                kfree(tmp);
107                printk(KERN_ALERT"mm = NULL error\n");
108                return -1;
109        } else {
110                tmp->start_code = p->mm->start_code;
111                tmp->end_code = p->mm->end_code;
112                printk(KERN_ALERT"start_code: %lu\n",tmp->start_code);
113                printk(KERN_ALERT"end_code:  %li\n",tmp->end_code);
114        }
115        if(copy_to_user(buf,tmp,sizeof(*tmp)))
116                return -EFAULT;
117        return sizeof(*tmp);
118 }
119 static ssize_t firstdriver_write(struct file *filp,char * buf,size_t len,loff_t * off)
120 {
121        printk("writing:\n");
122        if(copy_from_user(&pidno,buf,sizeof(int)))
123                return -EFAULT;
124        return sizeof(int);
125 }
126 module_init(firstdriver_init);
127 module_exit(firstdriver_exit);
128
129 MODULE_LICENSE("GPL");
130 MODULE_AUTHOR("firstdriver");
131
```

test.c

```c
#include<stdlib.h>
#include<unistd.h>
#include<stdio.h>
#include<fcntl.h>
#include<malloc.h>
struct ret_type
{
        unsigned long start_code;
        unsigned long end_code;
};
int main(int argc,char *argv[])
{
        struct ret_type *tmp;
        tmp = (struct ret_type *)malloc(sizeof(struct ret_type));
        if(tmp == NULL)
        {
                printf("malloc error\n");
                return -1;
        }
        int fd,num;
        num = atoi(argv[1]);
        printf("num is %d\n",num);

        fd = open("/dev/firstdriver",O_RDWR,S_IRUSR|S_IWUSR);
        if(fd != -1){
                int n;
                n = write(fd,&num,sizeof(int));
                if(n != sizeof(int)){
                        printf("write error\n");
                        goto  out;
                }
                n = read(fd,tmp,sizeof(*tmp));
                if(n != sizeof(*tmp)){
                        printf("read error, n =  %d\n",n);
                        goto out;
  }
                n = read(fd,tmp,sizeof(*tmp));
                if(n != sizeof(*tmp)){
                        printf("read error, n =  %d\n",n);
                        goto out;
                }
                printf("start_code is  %lu \n",tmp->start_code);
                printf("end_code is  %lu \n",tmp->end_code);

                close(fd);
        }
        else
                printf("device open failed \n");

        out:free(tmp);
        return 0;
}

```

getchar.c

```c
#include<stdio.h>
int main()
{
        getchar();
        return 0;
}
```

以超级用户身份执行以下事项：
1.make 生成模块文件,firstdriver.ko;

2.insmod firstdriver.ko;

3.cd /dev，mknod firstdriver c 260 0

4.gcc -o getchar getchar.c

5../getchar &

6.gcc -o test test.c

7../test (5生成的后台运行的进程号)

8.可看到结果如下:

start_code: 4194304
end_code： 4196284

参考:

1.http://www.360doc.com/content/14/0123/19/14451193_347404829.shtml

2.http://blog.chinaunix.net/uid-20662363-id-1904086.html

3.http://blog.csdn.net/rainbolide/article/details/7335888

4.http://www.360doc.com/content/10/1130/20/1378815_73825191.shtml

5.http://www.360doc.com/content/12/0901/16/10588621_233591860.shtml