

导航

博客园

首页

新随笔

联系

订阅 XML

管理

| | | | | | | | |
|----|---------|----|----|----|----|----|---|
| < | 2014年7月 | | | | | | > |
| 日 | 一 | 二 | 三 | 四 | 五 | 六 | |
| 29 | 30 | 1 | 2 | 3 | 4 | 5 | |
| 6 | 7 | 8 | 9 | 10 | 11 | 12 | |
| 13 | 14 | 15 | 16 | 17 | 18 | 19 | |
| 20 | 21 | 22 | 23 | 24 | 25 | 26 | |
| 27 | 28 | 29 | 30 | 31 | 1 | 2 | |
| 3 | 4 | 5 | 6 | 7 | 8 | 9 | |

公告

昵称: LoveFM

园龄: 2年9个月

粉丝: 33

关注: 3

+加关注

Linux设备驱动之Ioctl控制

大部分驱动除了需要具备读写设备的能力之外，还需要具备对硬件控制的能力。

一、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,...);
/*
fd:文件描述符
cmd:控制命令
...:可选参数:插入*argp, 具体内容依赖于
cmd
*/
```

用户程序所作的只是通过命令码告诉驱动程序它想做什么，至于怎么解释这些命令和怎么实现这些命令，这都是驱动程序要做的事情。

二、驱动ioctl方法：

```
int (*ioctl) (struct inode
*inode,struct file *filp,unsigned
int cmd,unsigned long arg);
/*
inode与filp两个指针对应于应用程序传递
的文件描述符fd，这和传递open方法的参数
一样。
cmd 由用户空间直接不经修改的传递给驱动
程序
arg 可选。
*/
```

在驱动程序中实现的ioctl函数体内，实际上是有一个switch {case}结构，每一个case对应一个命令码，做出一些相应的操作。怎么实现这些操作，这是每一个程序员自己的事情，因为设备都是特定的。关键在于怎么样组织命令码，因为在ioctl中命令码是唯一联系用户程序命令和驱动程序支持的

统计

随笔 - 23

文章 - 0

评论 - 13

引用 - 0

搜索

找找看

谷歌搜索

常用链接

我的随笔

我的评论

我的参与

最新评论

我的标签

我的标签

ADS1.2(1)

NFS(1)

probe(1)

volatile(1)

随笔分类

ARM(3)

C/C++语言(2)

Linux内核分析OS(2)

Linux驱动开发(7)

Linux系统管理(2)

Linux应用程序

嵌入式软件(1)

数据结构与算法(6)

随笔档案

2011年12月 (11)

2011年11月 (12)

最新评论

1. Re:Linux设备驱动之Ioctl控制
感谢。。。

--yangzhaoiiii

2. Re:Linux设备驱动之Ioctl控制

```
#!/bin/sh#unstall_mod.shmodule="memdev"device="memdev"# invoke rmmod with all arguments we got/sbin/rmmod $module $*
```

途径。

在Linux核心中是这样定义一个命令码的:

| 设备类型 | 序列号 | 方向 | 数据尺寸 |
|-------|-------|-------|----------|
| ----- | ----- | ----- | ----- |
| 8 bit | 8 bit | 2 bit | 8~14 bit |
| ----- | ----- | ----- | ----- |

这样一来,一个命令就变成了一个整数形式的命令码。但是命令码非常的不直观,所以Linux Kernel中提供了一些宏,这些宏可根据便于理解的字符串生成命令码,或者是从命令码得到一些用户可以理解的字符串以标明这个命令对应的设备类型、设备序列号、数据传送方向和数据传输尺寸。

1、定义命令:

内核提供了一些宏来帮助定义命令:

```
//nr为序号, datatype为数据类型,如int
_IO(type, nr) //没有参数的命令
_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
```

定义命令例子:

```
#define MEM_IOC_MAGIC 'm' //定义类
型
#define MEM_IOCSET
_IOW(MEM_IOC_MAGIC,0,int)
#define MEM_IOCTLQSET
_IOR(MEM_IOC_MAGIC, 1, int)
```

2、实现命令:

```
|| exit 1# remove nodesrm -f
/dev.....
```

--燕云

3. Re:Linux设备驱动之Ioctl控制

很棒!谢谢楼主! 下面这些自动化脚本,锦上添花 :)#!/bin/sh# install_mod.shmodule="memdev"device="memdev"mode="664"# Group: since distributions do it differently, look for w.....

--燕云

4. Re:linux设备驱动程序之简单字符设备驱动

楼主写得好详细,谢谢啦!正在做这个课程设计呢

--简单的信仰

5. Re:如何计算Nand Flash要传入的行地址和列地址

图片数不出来呀

--爱不孤单

阅读排行榜

1. ubuntu 10.04下的配置tftp服务器(8028)
2. Linux设备驱动之Ioctl控制(7698)
3. linux设备驱动程序之简单字符设备驱动(5139)
4. S3C2440的LCD编程(3127)
5. (百度笔试)简要说明树的深度优先、广度优先遍历算法,及非递归实现的特点(2925)

评论排行榜

1. Linux设备驱动之Ioctl控制(5)
2. linux设备驱动程序之简单字符设备驱动(4)
3. 求子数组的最大和____<3>(2)
4. s3c2440存储控制器和地址以及启动的理解(1)
5. 如何计算Nand Flash要传入的行地址和列地址(1)

推荐排行榜

1. linux设备驱动程序之简单字符设备驱动(4)
2. Linux设备驱动之Ioctl控制(3)

定义好了命令，下一步就是要实现ioctl函数了，ioctl的实现包括三个技术环节：

1) 返回值；

ioctl函数的实现是根据命令执行的一个switch语句，但是，当命令不能匹配任何一个设备所支持的命令时，通常返回-EINVAL(非法参数)；

2) 参数使用；

用户使用 `int ioctl(int fd, unsigned long cmd, ...)` 时，...就是要传递的参数；

再通过 `int (*ioctl)(struct inode *inode, struct file *filp, unsigned int cmd, unsigned long arg)` 中的arg传递；

如果arg是一个整数，可以直接使用；

如果是指针，我们必须确保这个用户地址是有效的，因此，使用之前需要进行正确检查。

内部有检查的，不需要检测的：

```
copy_from_user
copy_to_user
get_user
put_user
```

需要检测的：

```
__get_user
__put_user
```

检测函数 **access_ok()**：



```
static inline int access_ok(int
type, const void *addr, unsigned
long size)
/*
type :是VERIFY_READ 或者
VERIFY_WRITE用来表明是读用户内存还是
写用户内存；
addr:是要操作的用户内存地址；
size:是操作的长度。如果ioctl需要从用户
空间读一个整数，那么size参数就等于
sizeof(int)；
```

3. Linux设备驱动之I/O端口与I/O内存(2)

4. Linux设备驱动之mmap设备操作(1)

5. Linux高级字符设备之Poll操作(1)

返回值: `Access_ok`返回一个布尔值: 1, 是成功(存取没问题); 0, 是失败, `ioctl`返回 `-EFAULT`;

*/



3) 命令操作;

```
switch(cmd)
{
    case:
        ... ..
}
```

三、ioctl实例分析:

(1) memdev.h:

```
#ifndef _MEMDEV_H_
#define _MEMDEV_H_

#include <linux/ioctl.h>

#ifndef MEMDEV_MAJOR
#define MEMDEV_MAJOR 0    /*预设的
mem的主设备号*/
#endif

#ifndef MEMDEV_NR_DEVS
#define MEMDEV_NR_DEVS 2    /*设备
数*/
#endif

#ifndef MEMDEV_SIZE
#define MEMDEV_SIZE 4096
#endif
```



```
/*mem设备描述结构体*/
struct mem_dev
{
    char *data;
```

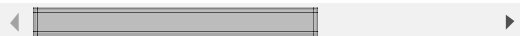
```
    unsigned long size;
};

/* 定义幻数 */
#define MEMDEV_IOC_MAGIC 'k'


/* 定义命令 */
#define MEMDEV_IOCPRINT
    _IO(MEMDEV_IOC_MAGIC, 1)
#define MEMDEV_IOCGETDATA
    _IOR(MEMDEV_IOC_MAGIC, 2, int)
#define MEMDEV_IOCSETDATA
    _IOW(MEMDEV_IOC_MAGIC, 3, int)

#define MEMDEV_IOC_MAXNR 3

#endif /* _MEMDEV_H_ */
```



(2) memdev.c: (驱动程序)

 View Code



```
static int mem_major =
MEMDEV_MAJOR;

module_param(mem_major, int,
S_IRUGO);

struct mem_dev *mem_devp; /*设备结
构体指针*/

struct cdev cdev;

/*文件打开函数*/
int mem_open(struct inode *inode,
struct file *filp)
{
    struct mem_dev *dev;

    /*获取次设备号*/
    int num = MINOR(inode-
>i_rdev);
```

```
    if (num >= MEMDEV_NR_DEVS)
        return -ENODEV;

    dev = &mem_devp[num];

    /*将设备描述结构指针赋值给文件私有
数据指针*/
    filp->private_data = dev;

    return 0;
}

/*文件释放函数*/
int mem_release(struct inode
*inode, struct file *filp)
{
    return 0;
}

/*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;
}

/*文件操作结构体*/
static const struct
file_operations mem_fops =
{
    .owner = THIS_MODULE,
    .open = mem_open,
    .release = mem_release,
    .ioctl = memdev_ioctl,
};
```

```
/*设备驱动模块加载函数*/
static int memdev_init(void)
{
    int result;
    int i;

    dev_t devno = MKDEV(mem_major,
0);

    /* 静态申请设备号*/
    if (mem_major)
        result =
register_chrdev_region(devno, 2,
"memdev");
    else /* 动态分配设备号 */
    {
        result =
alloc_chrdev_region(&devno, 0, 2,
"memdev");
        mem_major = MAJOR(devno);
    }

    if (result < 0)
        return result;

    /*初始化cdev结构*/
    cdev_init(&cdev, &mem_fops);
    cdev.owner = THIS_MODULE;
    cdev.ops = &mem_fops;

    /* 注册字符设备 */
    cdev_add(&cdev, MKDEV(mem_major,
0), MEMDEV_NR_DEVS);

    /* 为设备描述结构分配内存*/
    mem_devp =
kmalloc(MEMDEV_NR_DEVS *
sizeof(struct mem_dev),
GFP_KERNEL);
    if (!mem_devp) /*申请失败*/
    {
        result = - ENOMEM;
        goto fail_malloc;
    }
    memset(mem_devp, 0,
```



```
sizeof(struct mem_dev));

/*为设备分配内存*/
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;
}

/*模块卸载函数*/
static void memdev_exit(void)
{
    cdev_del(&cdev);    /*注销设备*/
    kfree(mem_devp);    /*释放设备结
构体内存*/

unregister_chrdev_region(MKDEV(mem
_major, 0), 2); /*释放设备号*/
}

MODULE_AUTHOR("David Xie");
MODULE_LICENSE("GPL");

module_init(memdev_init);
module_exit(memdev_exit);
```



(3)app-ioclt.c（应用程序）



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

#include "memdev.h" /* 包含命令定义 */

int main()
{
    int fd = 0;
    int cmd;
    int arg = 0;
    char Buf[4096];

    /*打开设备文件*/
    fd =
open("/dev/memdev0",O_RDWR);
    if (fd < 0)
    {
        printf("Open Dev Mem0
Error!\n");
        return -1;
    }

    /* 调用命令MEMDEV_IOCPRINT */
    printf("<--- Call
MEMDEV_IOCPRINT --->\n");
    cmd = MEMDEV_IOCPRINT;
    if (ioctl(fd, cmd, &arg) < 0)
    {
        printf("Call cmd
MEMDEV_IOCPRINT fail\n");
        return -1;
    }

    /* 调用命令MEMDEV_IOCSETDATA */
    printf("<--- Call
MEMDEV_IOCSETDATA --->\n");
    cmd = MEMDEV_IOCSETDATA;
    arg = 2007;
    if (ioctl(fd, cmd, &arg) < 0)
    {
        printf("Call cmd
```

```
MEMDEV_IOCSETDATA fail\n");\n\n    return -1;\n\n}\n\n/* 调用命令MEMDEV_IOCGETDATA */\nprintf("<--- Call\nMEMDEV_IOCGETDATA --->\n");\n    cmd = MEMDEV_IOCGETDATA;\n    if (ioctl(fd, cmd, &arg) < 0)\n    {\n        printf("Call cmd\nMEMDEV_IOCGETDATA fail\n");\n        return -1;\n    }\n    printf("<--- In User Space\nMEMDEV_IOCGETDATA Get Data is %d -\n-->\n\n", arg);\n\n    close(fd);\n    return 0;\n}
```



分类: **Linux驱动开发**

绿色通道:

好文要顶

关注我

收藏该文

与我联系



LoveFM

关注 - 3

粉丝 - 33

+加关注

3

0

(请您对文章做出评价)

« 上一篇: **linux设备驱动程序中的阻塞机制**

» 下一篇: **Linux高级字符设备之Poll操作**

posted on 2011-12-04 13:30 [LoveFM](#)

[阅读\(7697\)](#) [评论\(5\)](#) [编辑](#) [收藏](#)

评论

#1楼 2013-10-27 09:32 会游泳的灰机

我再调用ioctl函数的时候 提示 error:
expected expression before 'int'的错误, 怎么回事呢?

[支持\(0\)](#) [反对\(0\)](#)

#2楼 2013-12-27 12:05 [itfanr](#)

感谢 我收藏了啊

[支持\(0\)](#) [反对\(0\)](#)

#3楼 2014-06-25 19:15 燕云

很棒! 谢谢楼主!

下面这些自动化脚本, 锦上添花 :)

```
1  #!/bin/sh
2  # install_mod.sh
3  module="memdev"
4  device="memdev"
5  mode="664"
6
7  # Group: since
8  distributions do it
9  differently, look for wheel
10 or use staff
11 if grep '^staff:'
12 /etc/group > /dev/null;
13 then
14     group="staff"
15 else
16     group="wheel"
17 fi
18
19 # remove stale nodes
20 rm -f /dev/${device}?
21
22 # invoke insmod with all
23 arguments we got
24 # and use a pathname, as
25 newer modutils don't look
26 in . by default
```

```
27 /sbin/insmod -f
28 ./ $module.ko $* || exit 1
29
major=`cat /proc/devices |
awk "\\$2==\"$module\"
{print \\$1}"`

mknod /dev/${device}0 c
$major 0
mknod /dev/${device}1 c
$major 1
ln -sf ${device}0
/dev/${device}

# give appropriate
group/permissions
chgrp $group /dev/${device}
[0-1]
chmod $mode /dev/${device}
[0-1]
```

支持(0) 反对(0)

#4楼 2014-06-25 19:16 燕云

```
1 #!/bin/sh
2 #uninstall_mod.sh
3 module="memdev"
4 device="memdev"
5
6 # invoke rmmod with all
7 arguments we got
8 /sbin/rmmod $module $* ||
9 exit 1
10
11 # remove nodes
12 rm -f /dev/${device}[0-1]
/dev/${device}

exit 0
```

支持(0) 反对(0)

#5楼 2014-07-10 09:54 yangzhaoiiii

感谢。。。

支持(0) 反对(0)

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请
[登录](#) 或 [注册](#)，[访问](#)网站首页。

[博客园首页](#) [博问](#) [新闻](#) [闪存](#) [程序员招聘](#) [知识库](#)



最新IT新闻：

- [科普：飞机被击中可能发生什么？](#)
 - [“神奇小子”GeoHot加入Google Project Zero项目组](#)
 - [阿里巴巴要卖汽油了？](#)
 - [开发人员差距和技术债务危机](#)
 - [如果Google重返中国，能够击倒百度吗？](#)
- » [更多新闻...](#)

最新知识库文章：

- [程序员的工作不能用“生产效率”这个词来衡量](#)
 - [通过设计让APP变快的6个方法](#)
 - [打造你自己的程序员品牌](#)
 - [领域驱动设计实现之路](#)
 - [前端开发与项目管理](#)
- » [更多知识库文章...](#)

Powered by:

[博客园](#)

Copyright © LoveFM