Linux/Android开发记录 学习、记录、分享Linux/Android开发技术

个人资料

liuhaoyutz

访问： 80609次
积分： 1673分
排名： 第7877名

原创： 83篇 转载： 0篇
译文： 0篇 评论： 59条

博客声明

本博客文章均为原创，欢迎转载交流。转载请注明出处，禁止用于商业目的。

博客专栏

Android应用开发学习笔记
文章：30篇
阅读：17067

LDD3源码分析
文章：17篇
阅读：29965

文章分类

- LDD3源码分析 (18)
- ADC驱动 (1)
- 触摸屏驱动 (1)
- LCD驱动 (1)
- Linux设备模型 (8)
- USB驱动 (0)
- Android架构分析 (12)
- Cocos2d-x (1)
- C陷阱与缺陷 (3)
- Android应用开发 (30)
- Linux设备驱动程序架构分析 (8)

LDD3源码分析之ioctl操作

分类： LDD3源码分析 2012-03-23 10:56 1647人阅读 评论(0) 收藏 举报

ioc cmd user exchange access buffer

作者：刘昊昱

博客：http://blog.csdn.net/liuhaoyutz

编译环境：Ubuntu 10.10

内核版本：2.6.32-38-generic-pae

LDD3源码路径：examples/scull/main.c

本文分析LDD3第六章中关于ioctl操作的代码，并编写测试程序对ioctl功能进行测试。

一、ioctl操作

驱动程序中ioctl函数的函数原型如下：

```
int (*ioctl)(struct inode *inode, struct file *filp,

          unsigned int cmd, unsigned long arg);
```

其中cmd和arg参数是ioctl与其它驱动程序函数不同的地方。cmd是预先定义好的一些命令编号，对应要求ioctl执行的命令。arg是与cmd配合使用的参数。

ioctl函数的功能比较繁琐，从函数名可以看出，它一般是实现对设备的各种控制操作。可以这样理解，通过常规的read，write，lseek等等函数实现不合理的功能，就交给ioctl来实现。例如：要求设备锁门，弹出介质，改变波特率，甚至执行自我破坏，等等。

ioctl的实现一般是通过一个大的switch语句，根据cmd参数执行不同的操作。所以，在实现ioctl函数之前，要先定义好cmd对应的命令编号。为了防止发生混淆，命令编号应该在系统范围内是唯一的。为此，Linux内核将命令编号分为4个部分，即4个位段，分别是：

type: 幻数(magic number)，它占8位。个人理解幻数就是一个标志，代表一个(类)对象。后面我们会看到，scull使用字符'k'作为幻数。

number：序数，即顺序编号，它也占8位。

direction：如果相关命令涉及到数据的传输，则这个位段表示数据传输的方向，可用的值包括_IOC_NONE(没有数据传输)，_IOC_READ(读)、_IOC_WRITE(写)、_IOC_READ | _IOC_WRITE(双向传输数据)。注意，数据传输方向是从应用程序的角度看的，也就是说_IOC_READ意味着从设备中读数据，所以驱动程序必须向用户空间写数据。

size：所涉及的用户数据大小。这个位段的宽度与体系结构有关，通常是13或14位。

最新评论

LDD3源码分析之内存映射
wzw88486969: @fjlhlonng:unsigned long offset = vma->vm_pgoff <v...

Linux设备驱动程序架构分析之I2
teamos: 看了你的i2c的几篇文章，真是受益匪浅，虽然让自己写还是ie不出来。非常感谢

LDD3源码分析之块设备驱动程
elecfan2011: 感谢楼主的精彩讲解，受益匪浅啊！

LDD3源码分析之slab高速缓存
donghuwuwei: 省去了不少修改的时间，真是太好了

LDD3源码分析之时间与延迟操作
donghuwuwei: jit.c代码需要加上一个头文件。

LDD3源码分析之slab高速缓存
捧灰: 今天学到这里了，可是为什么我没有修改源码一遍就通过了额。。。内核版本是2.6.18-53.el5-x...

LDD3源码分析之字符设备驱动程
捧灰: 参照楼主的博客在自学~谢谢楼主！

LDD3源码分析之调试技术
fantasyhujian: 分析的很清楚，赞一个！

LDD3源码分析之字符设备驱动程
fantasyhujian: 有时间再好好读读，真的分析的不错！

LDD3源码分析之hello.c与Makef
fantasyhujian: 写的很详细，对初学者很有帮助！！！

阅读排行

LDD3源码分析之字符设	(3143)
LDD3源码分析之hello.c	(2701)
S3C2410驱动分析之LCI	(2527)
Linux设备模型分析之kse	(2435)
LDD3源码分析之内存映!	(2336)
LDD3源码分析之与硬件!	(2333)
Android架构分析之Andrc	(2093)
LDD3源码分析之时间与!	(1987)
LDD3源码分析之poll分析	(1972)
S3C2410驱动分析之AD(	(1948)

评论排行

LDD3源码分析之字符设	(12)
S3C2410驱动分析之触摸	(7)
LDD3源码分析之内存映!	(5)
LDD3源码分析之hello.c	(4)
Linux设备模型分析之kob	(4)
LDD3源码分析之slab高:	(4)
S3C2410驱动分析之LCI	(3)
LDD3源码分析之阻塞型I	(3)
LDD3源码分析之时间与!	(3)
LDD3源码分析之poll分析	(2)

文章存档

2014年06月 (1)
2014年05月 (4)
2014年04月 (1)

<linux/ioctl.h>中包含的<asm/ioctl.h>头文件定义了一些构造命令编号的宏：

_IO(type, nr)，用于构造无数据传输的命令编号。

_IOR(type, nr, datatype)，用于构造从驱动程序中读取数据的命令编号。

_IOW(type, nr, datatype)，用于构造向设备写入数据的命令编号。

_IOWR(type, nr, datatype)，用于双向传输命令编号。

其中，type和number位段从以上宏的参数中传入，size位段通过对datatype参数取sizeof获得。

另外，<asm/ioctl.h>头文件中还定义了一些用于解析命令编号的宏，如_IOC_DIR(cmd)，_IOC_TYPE(cmd)，_IOC_NR(cmd)，_IOC_SIZE(cmd)。

首先我们来看一下scull是如何定义命令编号的，理解scull的ioctl函数的实现，关键是理解这些命令是什么含义，即要求完成什么工作。在scull.h中有如下定义：

[cpp]

```cpp
135/*
136 * Ioctl definitions
137 */
138
139/* Use 'k' as magic number */
140#define SCULL_IOC_MAGIC  'k'
141/* Please use a different 8-bit number in your code */
142
143#define SCULL_IOCRESET    _IO(SCULL_IOC_MAGIC, 0)
144
145/*
146 * S means "Set" through a ptr,
147 * T means "Tell" directly with the argument value
148 * G means "Get": reply by setting through a pointer
149 * Q means "Query": response is on the return value
150 * X means "eXchange": switch G and S atomically
151 * H means "sHift": switch T and Q atomically
152 */
153#define SCULL_IOCSQUANTUM _IOW(SCULL_IOC_MAGIC,  1, int)
154#define SCULL_IOCSQSET    _IOW(SCULL_IOC_MAGIC,  2, int)
155#define SCULL_IOCTQUANTUM _IO(SCULL_IOC_MAGIC,   3)
156#define SCULL_IOCTQSET    _IO(SCULL_IOC_MAGIC,   4)
157#define SCULL_IOCGQUANTUM _IOR(SCULL_IOC_MAGIC,  5, int)
158#define SCULL_IOCGQSET    _IOR(SCULL_IOC_MAGIC,  6, int)
159#define SCULL_IOCQQUANTUM _IO(SCULL_IOC_MAGIC,   7)
160#define SCULL_IOCQQSET    _IO(SCULL_IOC_MAGIC,   8)
161#define SCULL_IOCXQUANTUM _IOWR(SCULL_IOC_MAGIC, 9, int)
162#define SCULL_IOCXQSET    _IOWR(SCULL_IOC_MAGIC,10, int)
163#define SCULL_IOCHQUANTUM _IO(SCULL_IOC_MAGIC,  11)
164#define SCULL_IOCHQSET    _IO(SCULL_IOC_MAGIC,  12)
165
166/*
167 * The other entities only have "Tell" and "Query", because they're
168 * not printed in the book, and there's no need to have all six.
169 * (The previous stuff was only there to show different ways to do it.
170 */
171#define SCULL_P_IOCTSIZE _IO(SCULL_IOC_MAGIC,   13)
172#define SCULL_P_IOCQSIZE _IO(SCULL_IOC_MAGIC,   14)
173/* ... more to come */
174
175#define SCULL_IOC_MAXNR 14
```

140行，定义scull的幻数是字符'k'

146行，'S'代表通过参数arg指向的内容设置。

147行，'T'代表直接通过参数arg的值设置。

148行，'G'代表通过参数arg指向的地址返回请求的值。

149行，'Q'代表通过ioctl函数的返回值返回请求的值。

150行，'X'代表通过参数arg指向的内容设置，再把原来的值通过arg指向的地址返回。即'S'与'G'两个操作合为一

2014年01月 (1)
2013年12月 (6)
展开

文章搜索

推荐文章

步。

151行，’H’代表通过参数arg的值直接设置，再通过ioctl函数的返回值将原来的值返回。即’T’和’Q’两个操作合为一步。

153行，定义命令SCULL_IOCSQUANTUM，该命令表示通过参数arg指向的内容设置quantum。

154行，定义命令SCULL_IOCSQSET，该命令表示通过参数arg指向的内容设置qset。

155行，定义命令SCULL_IOCTQUANTUM，该命令表示通过参数arg的值直接设置quantum。

156行，定义命令SCULL_IOCTQSET，该命令表示通过参数arg的值直接设置qset。

157行，定义命令SCULL_IOCGQUANTUM，该命令表示通过参数arg指向的地址返回quantum。

158行，定义命令SCULL_IOCGQSET，该命令表示通过参数arg指向的地址返回qset。

159行，定义命令SCULL_IOCQQUANTUM，该命令表示通过ioctl的返回值返回quantum。

160行，定义命令SCULL_IOCQQSET，该命令表示通过ioctl的返回值返回qset。

161行，定义命令SCULL_IOCXQUANTUM，该命令表示通过参数arg指向的内容设置quantum，然后，再把quantum原来的值写入arg指向的地址返回。

162行，定义命令SCULL_IOCXQSET，该命令表示通过参数arg指向的内容设置qset，然后，再把qset原来的值写入arg指向的地址返回。

163行，定义命令SCULL_IOCHQUANTUM，该命令表示通过参数arg的值直接设置quantum，然后，再通过ioctl的返回值返回quantum原来的值。

164行，定义命令SCULL_IOCHQSET，该命令表示通过参数arg的值直接设置qset，然后，再通过ioctl的返回值返回qset原来的值。

171行，定义命令SCULL_P_IOCTSIZE，该命令表示通过参数arg的值直接设置scull_p_buffer。

172行，定义命令SCULL_P_IOCQSIZE，该命令表示通过ioctl的返回值返回scull_p_buffer。

175定义SCULL_IOC_MAXNR为14，代表一共有14个命令。

理解了scull的ioctl命令的含义，我们就可以看ioctl的代码了，下面列出scull的ioctl函数代码如下：

[cpp]

```
389/*
390 * The ioctl() implementation
391 */
392
393int scull_ioctl(struct inode *inode, struct file *filp,
394                 unsigned int cmd, unsigned long arg)
395{
396
397    int err = 0, tmp;
398    int retval = 0;
399
400    /*
401     * extract the type and number bitfields, and don't decode
402     * wrong cmds: return ENOTTY (inappropriate ioctl) before access_ok()
403     */
404    if (_IOC_TYPE(cmd) != SCULL_IOC_MAGIC) return -ENOTTY;
405    if (_IOC_NR(cmd) > SCULL_IOC_MAXNR) return -ENOTTY;
406
407    /*
408     * the direction is a bitmask, and VERIFY_WRITE catches R/W
409     * transfers. `Type' is user-oriented, while
410     * access_ok is kernel-oriented, so the concept of "read" and
411     * "write" is reversed
412     */
413    if (_IOC_DIR(cmd) & _IOC_READ)
414        err = !access_ok(VERIFY_WRITE, (void __user *)arg, _IOC_SIZE(cmd));
415    else if (_IOC_DIR(cmd) & _IOC_WRITE)
416        err =  !access_ok(VERIFY_READ, (void __user *)arg, _IOC_SIZE(cmd));
417    if (err) return -EFAULT;
```

```
418
419    switch(cmd) {
420
421      case SCULL_IOCRESET:
422        scull_quantum = SCULL_QUANTUM;
423        scull_qset = SCULL_QSET;
424        break;
425
426      case SCULL_IOCSQUANTUM: /* Set: arg points to the value */
427        if (! capable (CAP_SYS_ADMIN))
428            return -EPERM;
429        retval = __get_user(scull_quantum, (int __user *)arg);
430        break;
431
432      case SCULL_IOCTQUANTUM: /* Tell: arg is the value */
433        if (! capable (CAP_SYS_ADMIN))
434            return -EPERM;
435        scull_quantum = arg;
436        break;
437
438      case SCULL_IOCGQUANTUM: /* Get: arg is pointer to result */
439        retval = __put_user(scull_quantum, (int __user *)arg);
440        break;
441
442      case SCULL_IOCQQUANTUM: /* Query: return it (it's positive) */
443        return scull_quantum;
444
445      case SCULL_IOCXQUANTUM: /* eXchange: use arg as pointer */
446        if (! capable (CAP_SYS_ADMIN))
447            return -EPERM;
448        tmp = scull_quantum;
449        retval = __get_user(scull_quantum, (int __user *)arg);
450        if (retval == 0)
451            retval = __put_user(tmp, (int __user *)arg);
452        break;
453
454      case SCULL_IOCHQUANTUM: /* sHift: like Tell + Query */
455        if (! capable (CAP_SYS_ADMIN))
456            return -EPERM;
457        tmp = scull_quantum;
458        scull_quantum = arg;
459        return tmp;
460
461      case SCULL_IOCSQSET:
462        if (! capable (CAP_SYS_ADMIN))
463            return -EPERM;
464        retval = __get_user(scull_qset, (int __user *)arg);
465        break;
466
467      case SCULL_IOCTQSET:
468        if (! capable (CAP_SYS_ADMIN))
469            return -EPERM;
470        scull_qset = arg;
471        break;
472
473      case SCULL_IOCGQSET:
474        retval = __put_user(scull_qset, (int __user *)arg);
475        break;
476
477      case SCULL_IOCQQSET:
478        return scull_qset;
479
480      case SCULL_IOCXQSET:
481        if (! capable (CAP_SYS_ADMIN))
482            return -EPERM;
483        tmp = scull_qset;
484        retval = __get_user(scull_qset, (int __user *)arg);
485        if (retval == 0)
486            retval = put_user(tmp, (int __user *)arg);
487        break;
488
489      case SCULL_IOCHQSET:
490        if (! capable (CAP_SYS_ADMIN))
491            return -EPERM;
492        tmp = scull_qset;
493        scull_qset = arg;
494        return tmp;
495
496        /*
497         * The following two change the buffer size for scullpipe.
498         * The scullpipe device uses this same ioctl method, just to
499         * write less code. Actually, it's the same driver, isn't it?
500         */
501
502       case SCULL_P_IOCTSIZE:
503         scull_p_buffer = arg;
504         break;
505
506       case SCULL_P_IOCQSIZE:
507         return scull_p_buffer;
508
509
510       default:  /* redundant, as cmd was checked against MAXNR */
511         return -ENOTTY;
512     }
513     return retval;
514
515}
```

404行，如果_IOC_TYPE(cmd) != SCULL_IOC_MAGIC，即cmd的幻数不是'k'，则退出。

405行，如果_IOC_NR(cmd) > SCULL_IOC_MAXNR，即cmd的序数大于14，则退出。

413 - 417行，如果要使用arg指向的地址进行数据的读或写，必须保证对该地址的访问是合法的，这可通过access_ok函数来验证，如果访问不合法，则退出。

419行，进入switch语句块。根据传入的cmd值，进入不同的分支执行。

420 - 512行，是个各cmd的处理分支，只要我们理解了各个cmd的含义，就很容易实现这些命令要求完成的工作。如果有不理解的地方，回到前面的各个cmd的定义处再研究一下。值得一提的是，驱动程序与用户空间传递数据，采用的是__put_user和__get_user函数，相比copy_to_user和copy_from_user来说，这些函数在处理1、2、4、8个字节的数据传输时，效率更高。另外，scull允许任何用户查询quantum和qset的大小，但只允许被授权的用户修改quantum和qset的值。这种权能的检查是通过capable()函数实现的。

二、测试ioctl

要测试scull驱动中ioctl函数是否实现了我们要求的功能，需要编写用户空间程序对scull模块进行测试。下面是我写的一个比较简单的测试程序：

首先是头文件scull_ioctl.h：

[cpp]

```
#ifndef _SCULL_IOCTL_H_
#define _SCULL_IOCTL_H_

#include <linux/ioctl.h> /* needed for the _IOW etc stuff used later */

/*
 * Ioctl definitions
 */

/* Use 'k' as magic number */
#define SCULL_IOC_MAGIC  'k'
/* Please use a different 8-bit number in your code */

#define SCULL_IOCRESET    _IO(SCULL_IOC_MAGIC, 0)

/*
 * S means "Set" through a ptr,
 * T means "Tell" directly with the argument value
 * G means "Get": reply by setting through a pointer
 * Q means "Query": response is on the return value
 * X means "eXchange": switch G and S atomically
 * H means "sHift": switch T and Q atomically
 */
#define SCULL_IOCSQUANTUM _IOW(SCULL_IOC_MAGIC,  1, int)
#define SCULL_IOCSQSET    _IOW(SCULL_IOC_MAGIC,  2, int)
#define SCULL_IOCTQUANTUM _IO(SCULL_IOC_MAGIC,   3)
#define SCULL_IOCTQSET    _IO(SCULL_IOC_MAGIC,   4)
```

```
#define SCULL_IOCGQUANTUM _IOR(SCULL_IOC_MAGIC,  5, int)
#define SCULL_IOCGQSET    _IOR(SCULL_IOC_MAGIC,  6, int)
#define SCULL_IOCQQUANTUM _IO(SCULL_IOC_MAGIC,   7)
#define SCULL_IOCQQSET    _IO(SCULL_IOC_MAGIC,   8)
#define SCULL_IOCXQUANTUM _IOWR(SCULL_IOC_MAGIC, 9, int)
#define SCULL_IOCXQSET    _IOWR(SCULL_IOC_MAGIC,10, int)
#define SCULL_IOCHQUANTUM _IO(SCULL_IOC_MAGIC,  11)
#define SCULL_IOCHQSET    _IO(SCULL_IOC_MAGIC,  12)

/*
 * The other entities only have "Tell" and "Query", because they're
 * not printed in the book, and there's no need to have all six.
 * (The previous stuff was only there to show different ways to do it.
 */
#define SCULL_P_IOCTSIZE _IO(SCULL_IOC_MAGIC,   13)
#define SCULL_P_IOCQSIZE _IO(SCULL_IOC_MAGIC,   14)
/* ... more to come */

#define SCULL_IOC_MAXNR 14

#endif /* _SCULL_IOCTL_H_ */
```

下面是测试程序scull_ioctl_test.c的代码:

[cpp]

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

#define SCULL_DEVICE "/dev/scull0"

int main(int argc, char *argv[])
{
    int fd = 0;
    int quantum = 8000;
    int quantum_old = 0;
    int qset = 2000;
    int qset_old = 0;

    fd = open(SCULL_DEVICE, O_RDWR);
    if(fd < 0)
    {
        printf("open scull device error!\n");
        return 0;
    }

    printf("SCULL_IOCSQUANTUM: quantum = %d\n", quantum);
    ioctl(fd, SCULL_IOCSQUANTUM, &quantum);
    quantum -= 500;
    printf("SCULL_IOCTQUANTUM: quantum = %d\n", quantum);
    ioctl(fd, SCULL_IOCTQUANTUM, quantum);

    ioctl(fd, SCULL_IOCGQUANTUM, &quantum);
    printf("SCULL_IOCGQUANTUM: quantum = %d\n", quantum);
    quantum = ioctl(fd, SCULL_IOCQQUANTUM);
    printf("SCULL_IOCQQUANTUM: quantum = %d\n", quantum);

    quantum -= 500;
    quantum_old = ioctl(fd, SCULL_IOCHQUANTUM, quantum);
    printf("SCULL_IOCHQUANTUM: quantum = %d, quantum_old = %d\n", quantum, quantum_old);
    quantum -= 500;
    printf("SCULL_IOCXQUANTUM: quantum = %d\n", quantum);
    ioctl(fd, SCULL_IOCXQUANTUM, &quantum);
    printf("SCULL_IOCXQUANTUM: old quantum = %d\n", quantum);

    printf("SCULL_IOCSQSET: qset = %d\n", qset);
    ioctl(fd, SCULL_IOCSQSET, &qset);
    qset += 500;
    printf("SCULL_IOCTQSET: qset = %d\n", qset);
    ioctl(fd, SCULL_IOCTQSET, qset);

    ioctl(fd, SCULL_IOCGQSET, &qset);
    printf("SCULL_IOCGQSET: qset = %d\n", qset);
    qset = ioctl(fd, SCULL_IOCQQSET);
    printf("SCULL_IOCQQSET: qset = %d\n", qset);

    qset += 500;
    qset_old = ioctl(fd, SCULL_IOCHQSET, qset);
    printf("SCULL_IOCHQSET: qset = %d, qset_old = %d\n", qset, qset_old);
    qset += 500;
    printf("SCULL_IOCXQSET: qset = %d\n", qset);
    ioctl(fd, SCULL_IOCXQSET, &qset);
    printf("SCULL_IOCHQSET: old qset = %d\n", qset);

    return 0;
}
```

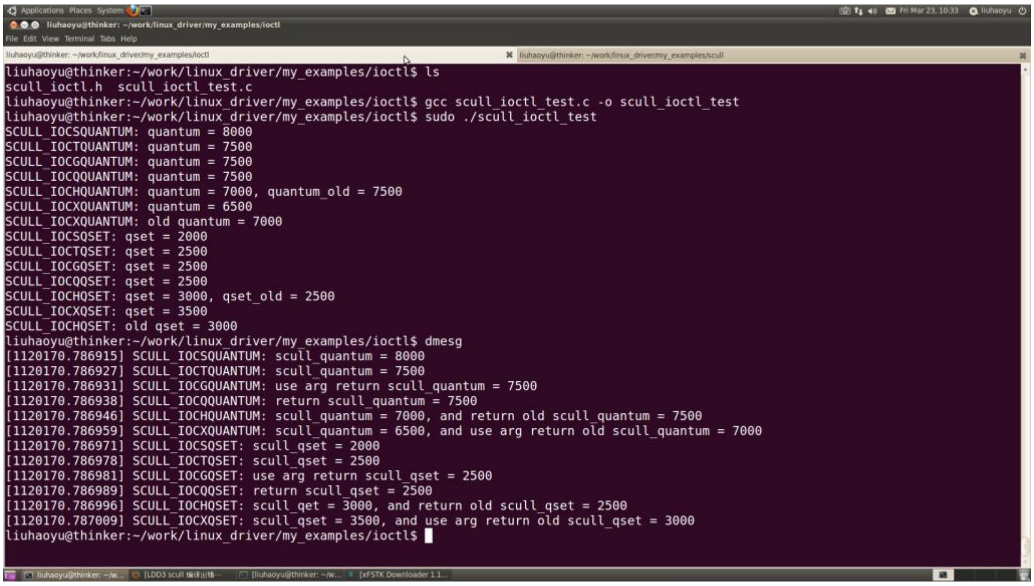
为了能看到测试效果，在修改驱动程序中的ioctl函数，打印一些语句。下面直接列出修改后的ioctl函数的实现：

[cpp]

```
/*
 * The ioctl() implementation
 */

int scull_ioctl(struct inode *inode, struct file *filp,
                 unsigned int cmd, unsigned long arg)
{

    int err = 0, tmp;
    int retval = 0;

    /*
     * extract the type and number bitfields, and don't decode
     * wrong cmds: return ENOTTY (inappropriate ioctl) before access_ok()
     */
    if (_IOC_TYPE(cmd) != SCULL_IOC_MAGIC) return -ENOTTY;
    if (_IOC_NR(cmd) > SCULL_IOC_MAXNR) return -ENOTTY;

    /*
     * the direction is a bitmask, and VERIFY_WRITE catches R/W
     * transfers. `Type' is user-oriented, while
     * access_ok is kernel-oriented, so the concept of "read" and
     * "write" is reversed
     */
    if (_IOC_DIR(cmd) & _IOC_READ)
        err = !access_ok(VERIFY_WRITE, (void __user *)arg, _IOC_SIZE(cmd));
    else if (_IOC_DIR(cmd) & _IOC_WRITE)
        err =  !access_ok(VERIFY_READ, (void __user *)arg, _IOC_SIZE(cmd));
    if (err) return -EFAULT;

    switch(cmd) {

      case SCULL_IOCRESET:
        scull_quantum = SCULL_QUANTUM;
        scull_qset = SCULL_QSET;
        printk("SCULL_IOCRESET: scull_quantum = %d, scull_qset = %d\n", scull_quantum, scull
        break;

      case SCULL_IOCSQUANTUM: /* Set: arg points to the value */
        if (! capable (CAP_SYS_ADMIN))
            return -EPERM;
        retval = __get_user(scull_quantum, (int __user *)arg);
        printk("SCULL_IOCSQUANTUM: scull_quantum = %d\n", scull_quantum);
        break;

      case SCULL_IOCTQUANTUM: /* Tell: arg is the value */
        if (! capable (CAP_SYS_ADMIN))
            return -EPERM;
        scull_quantum = arg;
        printk("SCULL_IOCTQUANTUM: scull_quantum = %d\n", scull_quantum);
        break;

      case SCULL_IOCGQUANTUM: /* Get: arg is pointer to result */
        retval = __put_user(scull_quantum, (int __user *)arg);
        printk("SCULL_IOCGQUANTUM: use arg return scull_quantum = %d\n", scull_quantum);
        break;

      case SCULL_IOCQQUANTUM: /* Query: return it (it's positive) */
        printk("SCULL_IOCQQUANTUM: return scull_quantum = %d\n", scull_quantum);
        return scull_quantum;

      case SCULL_IOCXQUANTUM: /* eXchange: use arg as pointer */
        if (! capable (CAP_SYS_ADMIN))
            return -EPERM;
        tmp = scull_quantum;
        retval = __get_user(scull_quantum, (int __user *)arg);
        if (retval == 0)
            retval = __put_user(tmp, (int __user *)arg);
        printk("SCULL_IOCXQUANTUM: scull_quantum = %d, and use arg return old scull_quantum
        break;

      case SCULL_IOCHQUANTUM: /* sHift: like Tell + Query */
        if (! capable (CAP_SYS_ADMIN))
            return -EPERM;
        tmp = scull_quantum;
        scull_quantum = arg;
        printk("SCULL_IOCHQUANTUM: scull_quantum = %d, and return old scull_quantum = %d\n",
        return tmp;

      case SCULL_IOCSQSET:
        if (! capable (CAP_SYS_ADMIN))
            return -EPERM;
        retval = __get_user(scull_qset, (int __user *)arg);
        printk("SCULL_IOCSQSET: scull_qset = %d\n", scull_qset);
        break;

      case SCULL_IOCTQSET:
        if (! capable (CAP_SYS_ADMIN))
            return -EPERM;
        scull_qset = arg;
        printk("SCULL_IOCTQSET: scull_qset = %d\n", scull_qset);
        break;

      case SCULL_IOCGQSET:
        retval = __put_user(scull_qset, (int __user *)arg);
        printk("SCULL_IOCGQSET: use arg return scull_qset = %d\n", scull_qset);
        break;

      case SCULL_IOCQQSET:
        printk("SCULL_IOCQQSET: return scull_qset = %d\n", scull_qset);
        return scull_qset;

      case SCULL_IOCXQSET:
        if (! capable (CAP_SYS_ADMIN))
            return -EPERM;
        tmp = scull_qset;
        retval = __get_user(scull_qset, (int __user *)arg);
        if (retval == 0)
            retval = put_user(tmp, (int __user *)arg);
        printk("SCULL_IOCXQSET: scull_qset = %d, and use arg return old scull_qset = %d\n",
        break;

      case SCULL_IOCHQSET:
        if (! capable (CAP_SYS_ADMIN))
            return -EPERM;
        tmp = scull_qset;
        scull_qset = arg;
        printk("SCULL_IOCHQSET: scull_qet = %d, and return old scull_qset = %d\n", scull_qse
        return tmp;

        /*
         * The following two change the buffer size for scullpipe.
         * The scullpipe device uses this same ioctl method, just to
         * write less code. Actually, it's the same driver, isn't it?
         */

      case SCULL_P_IOCTSIZE:
        scull_p_buffer = arg;
        break;

      case SCULL_P_IOCQSIZE:
        return scull_p_buffer;


      default:  /* redundant, as cmd was checked against MAXNR */
        return -ENOTTY;
    }
    return retval;
```

```
}
```

在我的系统上，测试过程如图所示。需要注意的是测试程序必须以root权限运行，因为普通用户只能读quantum和qset的值，只有root用户才能修改。



Applications Places System
liuhaoyu@thinker: ~/work/linux_driver/my_examples/ioctl
File Edit View Terminal Tabs Help
liuhaoyu@thinker: ~/work/linux_driver/my_examples/ioctl
liuhaoyu@thinker: ~/work/linux_driver/my_examples/scull
Fri Mar 23, 10:33
liuhaoyu

```
liuhaoyu@thinker:~/work/linux_driver/my_examples/ioctl$ ls
scull_ioctl.h  scull_ioctl_test.c
liuhaoyu@thinker:~/work/linux_driver/my_examples/ioctl$ gcc scull_ioctl_test.c -o scull_ioctl_test
liuhaoyu@thinker:~/work/linux_driver/my_examples/ioctl$ sudo ./scull_ioctl_test
SCULL_IOCSQUANTUM: quantum = 8000
SCULL_IOCTQUANTUM: quantum = 7500
SCULL_IOCGQUANTUM: quantum = 7500
SCULL_IOCQQUANTUM: quantum = 7500
SCULL_IOCHQUANTUM: quantum = 7000, quantum_old = 7500
SCULL_IOCXQUANTUM: quantum = 6500
SCULL_IOCXQUANTUM: old quantum = 7000
SCULL_IOCSQSET: qset = 2000
SCULL_IOCTQSET: qset = 2500
SCULL_IOCGQSET: qset = 2500
SCULL_IOCQQSET: qset = 2500
SCULL_IOCHQSET: qset = 3000, qset_old = 2500
SCULL_IOCXQSET: qset = 3500
SCULL_IOCHQSET: old qset = 3000
liuhaoyu@thinker:~/work/linux_driver/my_examples/ioctl$ dmesg
[1120170.786915] SCULL_IOCSQUANTUM: scull_quantum = 8000
[1120170.786927] SCULL_IOCTQUANTUM: scull_quantum = 7500
[1120170.786931] SCULL_IOCGQUANTUM: use arg return scull_quantum = 7500
[1120170.786938] SCULL_IOCQQUANTUM: return scull_quantum = 7500
[1120170.786946] SCULL_IOCHQUANTUM: scull_quantum = 7000, and return old scull_quantum = 7500
[1120170.786959] SCULL_IOCXQUANTUM: scull_quantum = 6500, and use arg return old scull_quantum = 7000
[1120170.786971] SCULL_IOCSQSET: scull_qset = 2000
[1120170.786978] SCULL_IOCTQSET: scull_qset = 2500
[1120170.786981] SCULL_IOCGQSET: use arg return scull_qset = 2500
[1120170.786989] SCULL_IOCQQSET: return scull_qset = 2500
[1120170.786996] SCULL_IOCHQSET: scull_qet = 3000, and return old scull_qset = 2500
[1120170.787009] SCULL_IOCXQSET: scull_qset = 3500, and use arg return old scull_qset = 3000
liuhaoyu@thinker:~/work/linux_driver/my_examples/ioctl$ 
```

更多 0

上一篇 LDD3源码分析之并发与竞态

下一篇 LDD3源码分析之简单休眠

顶
2

踩
0

主题推荐 源码 应用程序 kernel 工作 内核

猜你在找

get_free_page 和其友	[LeetCode]Word Break
ioctl的实现	ioctl 以及socket
百度地图使用之基本功能	STM32 对内部FLASH读写接口函数
linux下ALSA音频驱动软件开发	wifi流程详细分析
android开发之socket通信 向PC机发信息 获取本机IP	DDR，DDR2與DDR3的區別

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

核心技术类目

全部主题 Java VPN Android iOS ERP IE10 Eclipse CRM JavaScript Ubuntu NFC WAP jQuery 数据库 BI HTML5 Spring Apache Hadoop .NET API HTML SDK IIS Fedora XML LBS Unity Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack FTC coremail OPhone CouchBase 云计算 iOS6 Rackspace

Web App SpringSide Maemo Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP Spark HBase Pure Solr Angular Cloud Foundry Redis Scala Django Bootstrap