Tekkaman Ninja

tekkamanninja.blog.chinaunix.net

tekkamanninj

博客访问： 75897
博文数量： 263
博客积分： 15936
博客等级： 上将
技术积分： 13951
用 户 组： 普通用户
注册时间： 2007-03-27 11:22

加关注 短消息
论坛 加好友

个人简介

Fedora-ARM

文章分类

文章存档

Linux设备驱动程序学习（1）-字符设备驱动程序 2007-10-20 15:48:57

分类： LINUX

Linux设备驱动程序学习（1）
-字符设备驱动程序

今天进入《Linux设备驱动程序（第3版）》第三章字符设备驱动程序的学习。

这一章主要通过介绍字符设备scull（Simple Character Utility for Loading Localities，区域装载的简单字符工具）的驱动程序编写，来学习Linux设备驱动的基本知识。scull可以为真正的设备驱动程序提供样板。

一、主设备号和次设备号

主设备号表示设备对应的驱动程序；次设备号由内核使用，用于正确确定设备文件所指的设备。

内核用dev_t类型（<linux/types.h>）来保存设备编号，dev_t是一个32位的数，12位表示主设备号，20为表示次设备号。

在实际使用中，是通过<linux/kdev_t.h>中定义的宏来转换格式。

(dev_t)-->主设备号、次设备号	MAJOR(dev_t dev) MINOR(dev_t dev)
主设备号、次设备号-->(dev_t)	MKDEV(int major,int minor)

建立一个字符设备之前，驱动程序首先要做的事情就是获得设备编号。其这主要函数在<linux/fs.h>中声明：

```
int register_chrdev_region(dev_t first, unsigned int count,
char *name);   //指定设备编号

int alloc_chrdev_region(dev_t *dev, unsigned int firstminor,
unsigned int count, char *name);   //动态生成设备编号

void unregister_chrdev_region(dev_t first, unsigned int count);      //释放设备编号
```

分配之设备号的最佳方式是：默认采用动态分配，同时保留在加载甚至是编译时指定主设备号的余地。

以下是在scull.c中用来获取主设备好的代码：

```
if (scull_major) {
    dev = MKDEV(scull_major, scull_minor);
    result = register_chrdev_region(dev, scull_nr_devs, "scull");
} else {
    result = alloc_chrdev_region(&dev, scull_minor, scull_nr_devs,"scull");
    scull_major = MAJOR(dev);
}
if (result < 0) {
    printk(KERN_WARNING "scull: can't get major %d\n", scull_major);
    return result;
```

```
}
```

在这部分中，比较重要的是在用函数获取设备编号后，其中的参数name是和该编号范围关联的设备名称，它将出现在/proc/devices和sysfs中。

看到这里，就可以理解为什么mdev和udev可以动态、自动地生成当前系统需要的设备文件。udev就是通过读取sysfs下的信息来识别硬件设备的.
(请看《理解和认识udev》
URL：http://blog.chinaunix.net/u/6541/showart_396425.html)

二、一些重要的数据结构

大部分基本的驱动程序操作涉及及到三个重要的内核数据结构，分别是file_operations、file和inode，它们的定义都在<linux/fs.h>。

三、字符设备的注册

内核内部使用struct cdev结构来表示字符设备。在内核调用设备的操作之前，必须分配并注册一个或多个struct cdev。代码应包含<linux/cdev.h>，它定义了struct cdev以及与其相关的一些辅助函数。

注册一个独立的cdev设备的基本过程如下：

1、为struct cdev 分配空间(如果已经将struct cdev 嵌入到自己的设备的特定结构体中，并分配了空间，这步略过！)

```
struct cdev *my_cdev = cdev_alloc();
```

2、初始化struct cdev

```
void cdev_init(struct cdev *cdev, const struct file_operations *fops)
```

3、初始化cdev.owner

```
cdev.owner = THIS_MODULE;
```

4、cdev设置完成，通知内核struct cdev的信息（在执行这步之前必须确定你对struct cdev的以上设置已经完成！）

```
int cdev_add(struct cdev *p, dev_t dev, unsigned count)
```

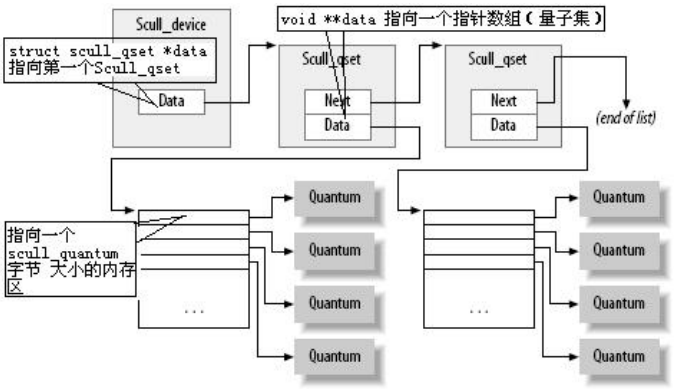
从系统中移除一个字符设备：`void cdev_del(struct cdev *p)`

以下是scull中的初始化代码（之前已经为struct scull_dev 分配了空间）：

```
/*
 * Set up the char_dev structure for this device.
 */
static void scull_setup_cdev(struct scull_dev *dev, int index)
{
    int err, devno = MKDEV(scull_major, scull_minor + index);

    cdev_init(&dev->cdev, &scull_fops);
    dev->cdev.owner = THIS_MODULE;
    dev->cdev.ops = &scull_fops;  //这句可以省略，在cdev_init中已经做过
    err = cdev_add (&dev->cdev, devno, 1);
    /* Fail gracefully if need be 这步值得注意*/
    if (err)
        printk(KERN_NOTICE "Error %d adding scull%d", err, index);
}
```

四、scull模型的内存使用



以下是scull模型的结构体：

```
/*
 * Representation of scull quantum sets.
 */
struct scull_qset {
    void **data;
    struct scull_qset *next;
};

struct scull_dev {
    struct scull_qset *data; /* Pointer to first quantum set */
    int quantum; /* the current quantum size */
    int qset; /* the current array size */
    unsigned long size; /* amount of data stored here */
    unsigned int access_key; /* used by sculluid and scullpriv */
    struct semaphore sem; /* mutual exclusion semaphore */
    struct cdev cdev;     /* Char device structure        */
};
```

scull驱动程序引入了两个Linux内核中用于内存管理的核心函数，它们的定义都在<linux/slab.h>:

```
void *kmalloc(size_t size, int flags);
void kfree(void *ptr);
```

以下是scull模块中的一个释放整个数据区的函数（类似清零），将在scull以写方式打开和scull_cleanup_module中被调用：

```
int scull_trim(struct scull_dev *dev)
{
  struct scull_qset *next, *dptr;
     int qset = dev->qset; /* 量子集中量子的个数*/
     int i;
     for (dptr = dev->data; dptr; dptr = next) { /* 循环scull_set个数次，直到dptr为NULL为止。*/
          if (dptr->data) {
                for (i = 0; i < qset; i++)/* 循环一个量子集中量子的个数次*/
                     kfree(dptr->data[i]);/* 释放其中一个量子的空间*/

                kfree(dptr->data);/* 释放当前的scull_set的量子集的空间*/
                dptr->data = NULL;/* 释放一个scull_set中的void **data指针*/
           }
      next = dptr->next; /* 准备下个scull_set的指针*/
      kfree(dptr);/* 释放当前的scull_set*/
      }
  dev->size = 0; /* 当前的scull_device所存的数据为0字节*/
  dev->quantum = scull_quantum;/* 初始化一个量子的大小*/
  dev->qset = scull_qset;/* 初始化一个量子集中量子的个数*/
  dev->data = NULL;/* 释放当前的scull_device的struct scull_qset *data指针*/
  return 0;
}
```

以下是scull模块中的一个沿链表前行得到正确scull_set指针的函数，将在read和write方法中被调用：

```
/*Follow the list*/
struct scull_qset *scull_follow(struct scull_dev *dev, int n)
{
    struct scull_qset *qs = dev->data;
        /* Allocate first qset explicitly if need be */
    if (! qs) {
        qs = dev->data = kmalloc(sizeof(struct scull_qset), GFP_KERNEL);
        if (qs == NULL)
            return NULL; /* Never mind */
        memset(qs, 0, sizeof(struct scull_qset));
    }
    /* Then follow the list */
    while (n--) {
        if (!qs->next) {
            qs->next = kmalloc(sizeof(struct scull_qset), GFP_KERNEL);
            if (qs->next == NULL)
                return NULL; /* Never mind */
            memset(qs->next, 0, sizeof(struct scull_qset));
        }
        qs = qs->next;
        continue;
    }
    return qs;
}
```

其实这个函数的实质是：如果已经存在这个scull_set，就返回这个scull_set的指针。如果不存在这个scull_set，一边沿链表为scull_set分配空间一边沿链表前行，直到所需要的scull_set被分配到空间并初始化为止，就返回这个scull_set的指针。

五、open和release

open方法提供给驱动程序以初始化的能力，为以后的操作作准备。应完成的工作如下：

(1)检查设备特定的错误（如设备未就绪或硬件问题）；

(2)如果设备是首次打开，则对其进行初始化；

(3)如有必要，更新f_op指针；

(4)分配并填写置于filp->private_data里的数据结构。

而根据scull的实际情况，他的open函数只要完成第四步（将初始化过的struct scull_dev dev的指针传递到filp->private_data里，以备后用）就好了，所以open函数很简单。但是其中用到了定义在<linux/kernel.h>中的container_of宏，源码如下：

```
#define container_of(ptr, type, member) ({            \
    const typeof( ((type *)0)->member ) *__mptr = (ptr);    \
    (type *)( (char *)__mptr - offsetof(type,member) );})
```

其实从源码可以看出，其作用就是：通过指针ptr，获得包含ptr所指向数据（是member结构体）的type结构体的指针。即是用指针得到另外一个指针。

release方法提供释放内存，关闭设备的功能。应完成的工作如下：

（1）释放由open分配的、保存在file->private_data中的所有内容；

（2）在最后一次关闭操作时关闭设备。

由于前面定义了scull是一个全局且持久的内存区，所以他的release什么都不做。

六、read和write

read和write方法的主要作用就是实现内核与用户空间之间的数据拷贝。因为Linux的内核空间和用户空间隔离的，所以要实现数据拷贝就必须使用在<asm/uaccess.h>中定义的：

```
unsigned long copy_to_user(void __user *to,
                           const void *from,
                           unsigned long count);
```

```
unsigned long copy_from_user(void *to,
                             const void __user *from,
                             unsigned long count);
```

而值得一提的是以上两个函数和

```
#define __copy_from_user(to,from,n)     (memcpy(to, (void __force *)from, n), 0)
#define __copy_to_user(to,from,n)     (memcpy((void __force *)to, from, n), 0)
```

之间的关系：通过源码可知，前者调用后者，但前者在调用前对用户空间指针进行了检查。

至于read和write 的具体函数比较简单，就在实验中验证好了。

七、模块实验

这次模块实验的使用是友善之臂SBC2440V4，使用Linux2.6.22.2内核。

模块程序链接：scull模块源程序

模块测试程序链接：模块测试程序

测试结果：

量子大小为6：

```
[Tekkaman2440@SBC2440V4]#cd /lib/modules/ [Tekkaman2440@SBC2440V4]#insmod scull.ko scull_quantum=6
[Tekkaman2440@SBC2440V4]#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
 14 sound
 81 video4linux
 89 i2c
 90 mtd
116 alsa
128 ptm
136 pts
180 usb
189 usb_device
204 s3c2410_serial
252 scull
253 usb_endpoint
254 rtc

Block devices:
  1 ramdisk
256 rfd
  7 loop
 31 mtdblock
 93 nftl
 96 inftl
179 mmc
[Tekkaman2440@SBC2440V4]#mknod -m 666 scull0 c  252 0
[Tekkaman2440@SBC2440V4]#mknod -m 666 scull1 c  252 1
[Tekkaman2440@SBC2440V4]#mknod -m 666 scull2 c  252 2
```

```
[Tekkaman2440@SBC2440V4]#mknod -m 666 scull3 c  252 3
```

启动测试程序

```
[Tekkaman2440@SBC2440V4]#./scull_test
write error! code=6
write error! code=6
write error! code=6
write ok! code=2
read error! code=6
read error! code=6
read error! code=6
read ok! code=2
[0]=0 [1]=1 [2]=2 [3]=3 [4]=4
[5]=5 [6]=6 [7]=7 [8]=8 [9]=9
[10]=10 [11]=11 [12]=12 [13]=13 [14]=14
[15]=15 [16]=16 [17]=17 [18]=18 [19]=19
```

改变量子大小为默认值4000：

```
[Tekkaman2440@SBC2440V4]#cd /lib/modules/
[Tekkaman2440@SBC2440V4]#rmmod scull
[Tekkaman2440@SBC2440V4]#insmod scull.ko
```

启动测试程序

```
[Tekkaman2440@SBC2440V4]#./scull_test
write ok! code=20
read ok! code=20
[0]=0 [1]=1 [2]=2 [3]=3 [4]=4
[5]=5 [6]=6 [7]=7 [8]=8 [9]=9
[10]=10 [11]=11 [12]=12 [13]=13 [14]=14
[15]=15 [16]=16 [17]=17 [18]=18 [19]=19

[Tekkaman2440@SBC2440V4]#
```

改变量子大小为6，量子集大小为2：

```
[Tekkaman2440@SBC2440V4]#cd /lib/modules/
[Tekkaman2440@SBC2440V4]#rmmod scull
[Tekkaman2440@SBC2440V4]#insmod scull.ko scull_quantum=6 scull_qset=2
```

启动测试程序

```
[Tekkaman2440@SBC2440V4]#./scull_test
write error! code=6
write error! code=6
write error! code=6
write ok! code=2
read error! code=6
read error! code=6
read error! code=6
read ok! code=2
[0]=0 [1]=1 [2]=2 [3]=3 [4]=4
[5]=5 [6]=6 [7]=7 [8]=8 [9]=9
[10]=10 [11]=11 [12]=12 [13]=13 [14]=14
[15]=15 [16]=16 [17]=17 [18]=18 [19]=19
```

实验不仅测试了模块的读写能力，还测试了量子读写是否有效。

上一篇：Linux设备驱动程序学习
下一篇：Linux设备驱动程序学习（0）—Hello, world模块

2

相关热门文章

云存储的优势有哪些	linux 常见服务端口	移植 ushare 到开发板
私有云到底有什么用	【ROOTFS搭建】busybox的httpd...	系统提供的库函数存在内存泄漏...
企业私有云存储有哪些功能...	xmanager 2.0 for linux配置	linux虚拟机 求教
常用MFC和API函数	什么是shell	初学UNIX环境高级编程的，关于...
mtd子系统-向block系统注册块...	linux socket的bug??	chinaunix博客什么时候可以设...

给主人留下些什么吧！~~

2013-01-03 17:52:15

您好，我正在按照你的博客学习ldd3这本书。您的实例代码链接出现了问题，能否发一份给我。邮箱sk_kai@163.com。麻烦了。

回复 | 举报

aCayF 2012-05-06 15:32:39

tek兄你好，我最近在学ldd3，遇到些棘手的问题，希望您能在方便，有空的时候做个解答，先在此谢过了，问题的详细描述我已经发到你的gmail邮箱

回复 | 举报

phoenixxyang 2011-05-08 19:47:54

int err, devno = MKDEV(scull_major, scull_minor + index);
初学linux驱动，请教下这句代码中的devno为什么定义成int啊？devno是设备号，不应该定义为dev_t么？

回复 | 举报

siyuantianxia 2011-04-18 12:53:12

博主您好，我在逛CU的时候，一不小心就进了您的空间，正好我也在学LDD3，不过只是刚刚开始学，看到您在2007年就开始看LDD3了，感觉自己晚了好多年啊！

有一个问题想问博主：驱动开发和嵌入式linux开发有什么区别啊？我在网上搜过，在CSDN上也找过，但是讨论的人太多了，各有各的说法，我都无法分清驱动和嵌入式linux到底区别在哪里。希望博主能在百忙之中抽点儿时间帮我解决我心中的疑惑。
万分谢谢！！！

回复 | 举报

yuyangwbc 2011-04-15 10:37:43

我也是刚刚踏入Linux驱动的门槛，正在努力
这里发现你写的很不错，对我来说资料丰富。
所以在这里说声谢谢！
同时向你学习！

回复 | 举报

1 2

评论热议

请登录后评论。
登录 注册