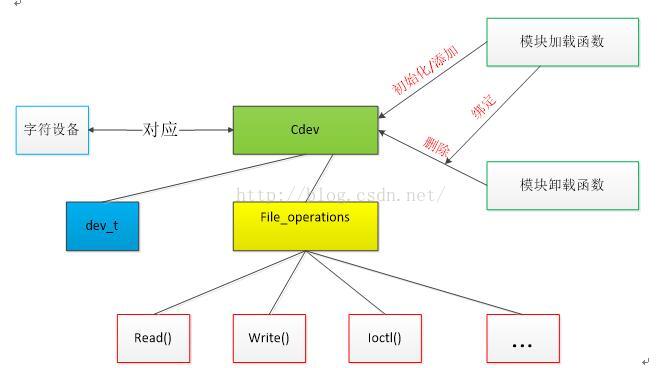
# 2. misc杂项设备驱动

在Linux驱动中把无法归类的五花八门的设备定义为混杂设备(用miscdevice结构体表述)。misc设备其实也就是特殊的字符设备。miscdevice共享一个主设备号MISC\_MAJOR(即10)，但次设备号不同。 所有的miscdevice设备形成了一个链表，对设备访问时内核根据次设备号查找对应的miscdevice设备，然后调用其file\_operations结构中注册的文件操作接口进行操作。 在内核中用struct miscdevice表示miscdevice设备，然后调用其file\_operations结构中注册的文件操作接口进行操作。miscdevice的API实现在drivers/char/misc.c中。



## 2.1misc使用的结构体和函数

misc设备其实也是字符设备，只不过misc设备驱动在字符设备的基础上又进行了一次封装，使用户可以更方便的使用。

struct miscdevice {

int minor;

const charchar \*name;

const struct file\_operations \*fops;//还是字符设备中的文件操作结构体，只不过misc结构体对其又封装了一次

struct list\_head list;

struct device \*parent;

struct device \*this\_device;

const charchar \*nodename;

mode\_t mode;

};

int misc\_register(struct miscdevice \* misc)

用户在注册misc设备的时候只需要：初始化file\_operations结构体->初始化miscdevice结构体->调用misc\_register将miscdevice注册到系统中就ok了。

static const struct file\_operations mmtimer\_fops = {

.owner = THIS\_MODULE,

.mmap =mmtimer\_mmap,

.unlocked\_ioctl = mmtimer\_ioctl,

.llseek = noop\_llseek,

};

static struct miscdevice mmtimer\_miscdev = {

SGI\_MMTIMER,

MMTIMER\_NAME,

&mmtimer\_fops

};

static int \_\_init mmtimer\_init(void)

{

cnodeid\_t node, maxn = -1;

......

if (request\_irq(SGI\_MMTIMER\_VECTOR, mmtimer\_interrupt, IRQF\_PERCPU, MMTIMER\_NAME, NULL)) {

printk(KERN\_WARNING "%s: unable to allocate interrupt.",

MMTIMER\_NAME);

goto out1;

}

if (misc\_register(&mmtimer\_miscdev)){

printk(KERN\_ERR "%s: failed to register device\n",

MMTIMER\_NAME);

goto out2;

}

......

return 0;

out3:

kfree(timers);

misc\_deregister(&mmtimer\_miscdev);

out2:

free\_irq(SGI\_MMTIMER\_VECTOR, NULL);

out1:

return -1;

}

那么到了这里就会有疑问，为什么linux还费劲的又造了一个misc设备呢？为什么不直接都使用字符设备驱动呢？

## 2.2为什么要有misc设备

优点：

1节省主设备号：使用普通字符设备驱动框架，不管该设备的主设备号是静态或者是动态分布的都会使用一个主设备号。而miscdevice结构体的**主设备号是固定的，MISC\_MAJOR定义为10**，在linux内核中，大概可以找到200多处使用miscdevice框架结构的驱动。

2使用方便：Misc驱动不再直接采用registe\_chrdev\_region()或者alloc\_chrdev\_region()、cdev\_add()之类的原始方法申请设备号、注册，而是才用miscdevice的注册方法misc\_register(struct miscdevice \* misc)。因为它已经封装和优化的很好，能很大程度上简化我们的工作量。

3利于linux驱动的分层设计思想：由于Linux驱动倾向于分层设计，各个具体的设备都可以找到它归属的类型，从而嵌套它相应的架构上面，并且只需要实现最底层的那一部 分。因为有些设备不知道它属于什么类型，Misc驱动的引入，很好的解决了这个问题，同时它使用起来也更加的方便。

总的来讲，如果使用misc驱动可以满足要求的话，那么这可以为开发人员剩下不少麻烦。

主要使用到的函数有：

int misc\_register(struct miscdevice \* misc);  
int misc\_deregister(struct miscdevice \*misc);

## 2.3 内核源码

### 1) miscdevice结构体

include\linux\Miscdevice.h 内核版本：2.6.11.12

struct miscdevice {

int minor;

const char \*name;

struct file\_operations \*fops;

struct list\_head list;

struct device \*dev;

struct class\_device \*class;

char devfs\_name[64];

};

extern int misc\_register(struct miscdevice \* misc);

extern int misc\_deregister(struct miscdevice \* misc);

### 2) misc\_register函数

/\*\*

\* misc\_register - register a miscellaneous device <span style="font-family: monospace; white-space: pre; background-color: rgb(240, 240, 240);">linux/drivers/char/misc.c</span>

\* @misc: device structure

\*

\* Register a miscellaneous device with the kernel. If the minor

\* number is set to %MISC\_DYNAMIC\_MINOR a minor number is assigned

\* and placed in the minor field of the structure. For other cases

\* the minor number requested is used.

\*

\* The structure passed is linked into the kernel and may not be

\* destroyed until it has been unregistered.

\*

\* A zero is returned on success and a negative errno code for

\* failure.

\*/

int misc\_register(struct miscdevice \* misc)

{

struct miscdevice \*c;

dev\_t dev;

int err;

down(&misc\_sem);

list\_for\_each\_entry(c, &misc\_list, list) {

if (c->minor == misc->minor) {

up(&misc\_sem);

return -EBUSY;

}

}

if (misc->minor == MISC\_DYNAMIC\_MINOR) {

int i = DYNAMIC\_MINORS;

while (--i >= 0)

if ( (misc\_minors[i>>3] & (1 << (i&7))) == 0)

break;

if (i<0) {

up(&misc\_sem);

return -EBUSY;

}

misc->minor = i;

}

if (misc->minor < DYNAMIC\_MINORS)

misc\_minors[misc->minor >> 3] |= 1 << (misc->minor & 7);

if (misc->devfs\_name[0] == '\0') {

snprintf(misc->devfs\_name, sizeof(misc->devfs\_name),

"misc/%s", misc->name);

}

//由主、次设备号生产设备号

dev = MKDEV(MISC\_MAJOR, misc->minor);

\*/

/\*\*

\*struct class\_device \*class\_simple\_device\_add(struct class\_simple \*cs, dev\_t dev,

\* struct device \*device, const char \*fmt, ...)

\* 为一个简单设备类添加设备。

\* cs: 将设备添加到此简单设备类。

\* dev: 分配的设备号。

\* device: 要添加的设备。

\* fmt: 用于格式化名称。

\*/

misc->class = class\_simple\_device\_add(misc\_class, dev,

misc->dev, misc->name);

if (IS\_ERR(misc->class)) {

err = PTR\_ERR(misc->class);

goto out;

}

err = devfs\_mk\_cdev(dev, S\_IFCHR|S\_IRUSR|S\_IWUSR|S\_IRGRP,

misc->devfs\_name);

if (err) {

class\_simple\_device\_remove(dev);

goto out;

}

/\*

\* Add it to the front, so that later devices can "override"

\* earlier defaults

\*/

list\_add(&misc->list, &misc\_list);

out:

up(&misc\_sem);

return err;

}

### 3）misc\_deregister函数

/\*\*

\* misc\_deregister - unregister a miscellaneous device <span style="font-family: monospace; white-space: pre; background-color: rgb(240, 240, 240);">linux/drivers/char/misc.c</span>

\* @misc: device to unregister

\*

\* Unregister a miscellaneous device that was previously

\* successfully registered with misc\_register(). Success

\* is indicated by a zero return, a negative errno code

\* indicates an error.

\*/

int misc\_deregister(struct miscdevice \* misc)

{

int i = misc->minor;

if (list\_empty(&misc->list))

return -EINVAL;

down(&misc\_sem);

list\_del(&misc->list);

\*/

/\*\*

\* 当拨除设备时，使用下面的class\_simple\_device\_remove函数删除类入口。

\*/

class\_simple\_device\_remove(MKDEV(MISC\_MAJOR, misc->minor));

devfs\_remove(misc->devfs\_name);

if (i < DYNAMIC\_MINORS && i>0) {

misc\_minors[i>>3] &= ~(1 << (misc->minor & 7));

}

up(&misc\_sem);

return 0;

}

EXPORT\_SYMBOL(misc\_register);

EXPORT\_SYMBOL(misc\_deregister);

## 2.4 具体实例

/\*miscdemo.c\*/

#include <linux/init.h>

#include <linux/module.h>

#include <linux/fs.h>

#include <linux/miscdevice.h>

int open\_state = 0;

/////////////////////////////////////////////////

int miscdemo\_open(struct inode \*inode, struct file \*filp)

{

if (open\_state == 0)

{

open\_state = 1;

printk("miscdemo open!\n");

return 0;

}

printk("miscdemo has been open!\n");

return -1;

}

int miscdemo\_release(struct inode \*inode, struct file \*filp)

{

if (open\_state == 1)

{

open\_state = 0;

printk("miscdemo release!\n");

return 0;

}

printk("miscdemo has not been open yet!\n");

return -1;

}

ssize\_t miscdemo\_read(struct file \*filp, char \*buf,

size\_t count, loff\_t fpos)

{

printk("miscdemo read!\n");

return 0;

}

ssize\_t miscdemo\_write(struct file \*filp, char \*buf,

size\_t count, loff\_t fpos)

{

printk("miscdemo write!\n");

return 0;

}

int miscdemo\_ioctl(struct inode \*inode, struct file \*filp,

unsigned int cmd, unsigned long arg)

{

printk("ioctl is called!\n");

printk("cmd:%d arg:%d\n", cmd, arg);

return 0;

}

/////////////////////////////////////////////////

struct file\_operations fops =

{

.owner = THIS\_MODULE,

.open = miscdemo\_open,

.release = miscdemo\_release,

.write = miscdemo\_write,

.read = miscdemo\_read,

. unlocked\_ioctl = miscdemo\_ioctl

};

struct miscdevice dev =

{

.minor = MISC\_DYNAMIC\_MINOR,

.fops = &fops,

.name = "miscdemo",

.nodename = "miscdemo\_node"

};

int setup\_miscdemo(void)

{

return misc\_register(&dev);

}

/////////////////////////////////////////////////

static int \_\_init miscdemo\_init(void)

{

printk("miscdemo init!\n");

return setup\_miscdemo();

}

static void \_\_exit miscdemo\_exit(void)

{

printk("miscdemo exit!\n");

misc\_deregister(&dev);

}

MODULE\_AUTHOR("sundm75");

MODULE\_LICENSE("GPL");

module\_init(miscdemo\_init);

module\_exit(miscdemo\_exit);

测试函数

#include <stdio.h>

#include <sys/stat.h>

#include <sys/ioctl.h>

#include <fcntl.h>

#include <errno.h>

/////////////////////////////////////////////////

int main(int argc, char \*\*argv)

{

int fd;

fd = open("/dev/miscdemo\_node", O\_RDONLY);

if (fd < 0)

{

printf("open /dev/miscdemo\_node failed!\n");

printf("%s\n", strerror(errno));

return -1;

}

printf("open /dev/miscdemo\_node ok!\n");

if (ioctl(fd, 6) != 0)

{

printf("ioctl failed!\n");

printf("%s\n", strerror(errno));

}

else

printf("ioctl ok!\n");

close(fd);

return 0;

}

运行结果：

[root@Loongson:/]#insmod miscdemo.ko

miscdemo init!

[root@Loongson:/]#lsmod

Module Size Used by Not tainted

miscdemo 1481 0

[root@Loongson:/]#./test\_miscdemo

miscdemo open!

open /dev/miscdioctl is called!

emo\_node ok!

cmd:30 arg:30

ioctl ok!miscdemo release!

[root@Loongson:/]#cat /dev/miscdemo\_node

miscdemo open!

miscdemo read!

miscdemo release!