### 杂项设备（misc device）

杂项设备也是在嵌入式系统中用得比较多的一种设备驱动。在 Linux 内核的include/linux目录下有Miscdevice.h文件，要把自己定义的misc device从设备定义在这里。其实是因为这些字符设备不符合预先确定的字符设备范畴，**所有这些设备采用主编号10** ，一起归于misc device，**其实misc\_register就是用主标号10调用register\_chrdev()的。**

也就是说，misc设备其实也就是特殊的字符设备，可自动生成设备节点。

### 字符设备(char device)

使用register\_chrdev(LED\_MAJOR,DEVICE\_NAME,&dev\_fops)注册字符设备驱动程序时，如果有多个设 备使用该函数注册驱动程序，LED\_MAJOR不能相同，否则几个设备都无法注册(我已验证)。如果模块使用该方式注册并且 **LED\_MAJOR为0(自动分配主设备号 )，**使用insmod命令加载模块时会在终端显示分配的主设备号和次设备号，在/dev目录下建立该节点，比如 设备leds，如果加载该模块时分配的主设备号和次设备号为253和0，则建立节点:mknod leds c 253 0。使用**register\_chrdev (LED\_MAJOR,DEVICE\_NAME,&dev\_fops)注册字符设备驱动程序时都要手动建立节点 ，否则在应用程序无法打开该设备。**

阅读led驱动程序的代码的时候，没有发现ldd3中提到的各种字符设备注册函数，而是发现了一个misc\_register函数，这说明led设备是作为杂项设备出现在内核中的，在内核中，**misc杂项设备驱动接口是对一些字符设备的简单封装，他们共享一个主设备号，有不同的次设备号，共享一个open调用，其他的操作函数在打开后运用linux驱动程序的方法重载进行装载。**

<http://blog.csdn.net/lanmanck/article/details/4713978>

阅读led驱动程序的代码的时候，没有发现ldd3中提到的各种字符设备注册函数，而是发现了一个misc\_register函数，这说明led设备是作为杂项设备出现在内核中的，在内核中，**misc杂项设备驱动接口是对一些字符设备的简单封装，他们共享一个主设备号，有不同的次设备号，共享一个open调用，其他的操作函数在打开后运用linux驱动程序的方法重载进行装载。**

#### misc设备驱动模型

本节我们来看一下misc设备驱动模型的有关内容，首先是看看它的设备结构体，定义在include/linux/miscdevice.h中：

struct miscdevice {

int minor; //次设备号，若为 MISC\_DYNAMIC\_MINOR 自动分配

const char \*name; //设备名

const struct file\_operations \*fops; //设备文件操作结构体

struct list\_head list; //misc\_list链表头

struct device \*parent;

struct device \*this\_device;

const char \*nodename;

mode\_t mode;

};

1、为什么只有次设备号呢？一个设备不是有主、次设备号吗？

其实，我想大家应该能够想到了，此时没有明确指定，那就说明应该是使用默认值。

2、主设备号的默认值是多少呢？难道所有注册为misc的设备都有相同的主设备号？怎么区分各个设备呢？

这个主设备号是10.的确，所有注册为misc的设备都有相同的主设备号：10.在使用过程中我们主要是通过次设备号来区分各个设备。这一点不难理解，内核将所有注册为misc的设备都归为一大类。

3、结构体中的list\_head结构体类型的list成员的作用是什么呢？

内核自己会维护一个misc\_list链表，所有注册为misc的设备都必须挂在这个链表上，这个list就是该链表的链表头。

4、结构体中的两个device结构体类型指针作用是什么呢？

作用就是创建设备文件，稍候就可以看到了！

5、我们如何定义自己的misc类型的设备呢？

可如下定义：

static struct miscdevice misc = {

.minor = MISC\_DYNAMIC\_MINOR,

.name = DEVICE\_NAME,

.fops = &dev\_fops,

};

其中的设备文件操作结构体和字符设备类似，这里就不再细讲。

6、定义了自己的misc设备，那么我们如何向内核注册/注销设备呢？

使用如下两个函数：

int misc\_register(struct miscdevice \* misc); //在加载模块时会自动创建设备文件，是主设备号为10的字符设备

int misc\_deregister(struct miscdevice \*misc); //在卸载模块时会自动

好了，至此，整个设备驱动的流程就完了，接下来深入了解一下misc设备模型的工作原理。

首先看看misc初始化函数：

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

{

int err;

#ifdef CONFIG\_PROC\_FS

/\*如果使用proc文件系统，则创建misc项\*/

proc\_create("misc", 0, NULL, &misc\_proc\_fops);

#endif

/\*在/sys/class/目录下创建一个名为misc的类\*/

misc\_class = class\_create(THIS\_MODULE, "misc");

err = PTR\_ERR(misc\_class);

if (IS\_ERR(misc\_class))

goto fail\_remove;

err = -EIO;

/\*咦，怎么misc设备驱动调用字符驱动的注册函数呢？设备的主设备号为MISC\_MAJOR，为10\*/

if (register\_chrdev(MISC\_MAJOR,"misc",&misc\_fops))

goto fail\_printk;

misc\_class->devnode = misc\_devnode;

return 0;

fail\_printk:

printk("unable to get major %d for misc devices\n", MISC\_MAJOR);

class\_destroy(misc\_class);

fail\_remove:

remove\_proc\_entry("misc", NULL);

return err;

}

/\*向内核注册misc子系统\*/

subsys\_initcall(misc\_init);

接下来看看misc设备驱动的注册与注销函数：

注册函数：

int misc\_register(struct miscdevice \* misc)

{

struct miscdevice \*c;

dev\_t dev;

int err = 0;

/\*内核初始化一个链表头\*/

INIT\_LIST\_HEAD(&misc->list);

mutex\_lock(&misc\_mtx);

/\*遍历已经注册的misc，如果和当前准备注册的相同(依据次设备号来判断),就返回设备忙\*/

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

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

mutex\_unlock(&misc\_mtx);

return -EBUSY;

}

}

/\*动态分配设备的次设备号\*/

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

int i = find\_first\_zero\_bit(misc\_minors, DYNAMIC\_MINORS);

if (i >= DYNAMIC\_MINORS) {

mutex\_unlock(&misc\_mtx);

return -EBUSY;

}

misc->minor = DYNAMIC\_MINORS - i - 1;

set\_bit(i, misc\_minors);

}

/\*使用固定的主设备号，动态分配的次设备号构造设备号\*/

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

/\*创建设备文件，这里就是使用miscdevice结构体中两个device类型指针的地方，

当然，这是和linux设备驱动模型相关的\*/

misc->this\_device = device\_create(misc\_class, misc->parent, dev,

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

if (IS\_ERR(misc->this\_device)) {

int i = DYNAMIC\_MINORS - misc->minor - 1;

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

clear\_bit(i, misc\_minors);

err = PTR\_ERR(misc->this\_device);

goto out;

}

/\*

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

\* earlier defaults

\*/

/\*到这一步也就注册成功了，将新注册的misc设备加入到内核维护的misc\_list链表中\*/

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

out:

mutex\_unlock(&misc\_mtx);

return err;

}

注销函数：

int misc\_deregister(struct miscdevice \*misc)

{

int i = DYNAMIC\_MINORS - misc->minor - 1;

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

return -EINVAL;

mutex\_lock(&misc\_mtx);

/\*删除链表节点\*/

list\_del(&misc->list);

/\*销毁设备文件\*/

device\_destroy(misc\_class, MKDEV(MISC\_MAJOR, misc->minor));

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

clear\_bit(i, misc\_minors);

mutex\_unlock(&misc\_mtx);

return 0;

}

到这里，差不多misc设备驱动模型就差不多了。

2、misc设备驱动实例

这里贴一个简单的misc设备驱动程序，方便大家对照上面的理论部分进行分析，此驱动程序是友善之臂6410开发板的LED驱动程序，可以看看：

#include <linux/miscdevice.h>

#include <linux/delay.h>

#include <asm/irq.h>

//#include <mach/regs-gpio.h>

#include <mach/hardware.h>

#include <linux/kernel.h>

#include <linux/module.h>

#include <linux/init.h>

#include <linux/mm.h>

#include <linux/fs.h>

#include <linux/types.h>

#include <linux/delay.h>

#include <linux/moduleparam.h>

#include <linux/slab.h>

#include <linux/errno.h>

#include <linux/ioctl.h>

#include <linux/cdev.h>

#include <linux/string.h>

#include <linux/list.h>

#include <linux/pci.h>

#include <asm/uaccess.h>

#include <asm/atomic.h>

#include <asm/unistd.h>

#include <mach/map.h>

#include <mach/regs-clock.h>

#include <mach/regs-gpio.h>

#include <plat/gpio-cfg.h>

#include <mach/gpio-bank-e.h>

#include <mach/gpio-bank-k.h>

#define DEVICE\_NAME "leds"

static long sbc2440\_leds\_ioctl(struct file \*filp, unsigned int cmd, unsigned long arg)

{

switch(cmd) {

unsigned tmp;

case 0:

case 1:

if (arg > 4) {

return -EINVAL;

}

tmp = readl(S3C64XX\_GPKDAT);

tmp &= ~(1 << (4 + arg));

tmp |= ( (!cmd) << (4 + arg) );

writel(tmp, S3C64XX\_GPKDAT);

return 0;

default:

return -EINVAL;

}

}

static struct file\_operations dev\_fops = {

.owner = THIS\_MODULE,

.unlocked\_ioctl = sbc2440\_leds\_ioctl,

};

static struct miscdevice misc = {

.minor = MISC\_DYNAMIC\_MINOR,

.name = DEVICE\_NAME,

.fops = &dev\_fops,

};

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

{

int ret;

{

unsigned tmp;

tmp = readl(S3C64XX\_GPKCON);

tmp = (tmp & ~(0xffffU<<16))|(0x1111U<<16);

writel(tmp, S3C64XX\_GPKCON);

tmp = readl(S3C64XX\_GPKDAT);

tmp |= (0xF << 4);

writel(tmp, S3C64XX\_GPKDAT);

}

ret = misc\_register(&misc);

printk (DEVICE\_NAME"\tinitialized\n");

return ret;

}

static void \_\_exit dev\_exit(void)

{

misc\_deregister(&misc);

}

module\_init(dev\_init);

module\_exit(dev\_exit);

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("FriendlyARM Inc.");

好了，今天就到此结束，呵呵！

总结：

杂项设备作为字符设备的封装，为字符设备提供的简单的编程接口，如果编写新的字符驱动，可以考虑使用杂项设备接口，方便简单，只需要初始化一个miscdevice的结构，调用misc\_register就可以了。系统最多有255个杂项设备，因为杂项设备模块自己占用了一个次设备号。