linux驱动开发（十）——misc杂散设备

转载

weixin_30415801 于 2017-04-07 14:38:00 发布 139 收藏 2
文章标签： 操作系统 驱动开发 数据结构与算法
原文链接：http://www.cnblogs.com/biaohc/p/6676499.html

1：什么是misc驱动模型？

2：为什么要有misc驱动模型？

3：misc驱动模型的代码实现

4：misc驱动模型实战

参考：

http://blog.csdn.net/yicao821/article/details/6785738

http://www.thinksaas.cn/topics/0/507/507168.html

http://www.cnblogs.com/fellow1988/p/6235080.html

https://www.zhihu.com/question/21508904

http://www.cnblogs.com/snake-hand/p/3212483.html

http://blog.csdn.net/chenlong12580/article/details/7339127

1：什么是misc驱动模型

Linux包含了许多的设备驱动类型，而不管分类有多细，总会有些漏网的，这就是我们经常说到的“其他的”等等。
在Linux里面，把无法归类的五花八门的设备定义为混杂设备（用miscdevice结构体来描述）。Linux/内核所提供的miscdevice有很强的包容性。如NVRAM，看门狗，DS1286等实时时钟，字符LCD,AMD 768随机数发生器。
miscdevice共享一个主设备号MISC_MAJOR（10），但此设备号不同，所有的miscdevice设备形成一个链表，对设备访问时内核根据次设备号查找对应的 miscdevice设备，然后调用其中的file_operations结构体中注册的文件操作接口进程操作。

2：为什么要有misc驱动模型

作者：智多芯
链接：https://www.zhihu.com/question/21508904/answer/19623523

第一，节省主设备号：
使用普通字符设备，不管该驱动的主设备号是静态还是动态分配，都会消耗一个主设备号，这太浪费了。而且如果你的这个驱动最终会提交到内核主线版本上的话，需要申请一个专门的主设备号，这也麻烦。
如果使用misc驱动的话就好多了。因为内核中已经为misc驱动分配了一个主设备号。当系统中拥有多个misc设备驱动时，那么它们的主设备号相同，而用子设备号来区分它们。

第二，使用简单：
有时候驱动开发人员需要开发一个功能较简单的字符设备驱动，导出接口让用户空间程序方便地控制硬件，只需要使用misc子系统提供的接口即可快速地创建一个misc设备驱动。
当使用普通的字符设备驱动时，如果开发人员需要导出操作接口给用户空间的话，需要自己去注册字符驱动，并创建字符设备class以自动在/dev下生成设备节点，相对麻烦一点。而misc驱动则无需考虑这些，基本上只需要把一些基本信息通过struct miscdevice交给misc_register()去处理即可。

本质上misc驱动也是一个字符设备驱动，可能相对特殊一点而已。在drivers/char/misc.c的misc驱动初始化函数misc_init()中实际上使用了MISC_MAJOR（主设备号为10）并调用register_chrdev()去注册了一个字符设备驱动。同时也创建了一个misc_class，使得最后可自动在/dev下自动生成一个主设备号为10的字符设备。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。
所以说misc驱动模型让我们很简单的在底层实现了字符设备驱动，并且在在应用层给予了一定的接口，节省了主设备号；其实就相当于一个杂货铺，乱七八糟的字符设备驱动模型都可以往里面

堆。

3：驱动模型代码实现：
misc驱动的实现代码在driver/char/misc.c目录下，

misc_init函数：

```
 1 static int __init misc_init(void)
 2 {
 3     int err;
 4
 5 #ifdef CONFIG_PROC_FS
 6     proc_create("misc", 0, NULL, &misc_proc_fops);
 7 #endif
 8     misc_class = class_create(THIS_MODULE, "misc");
 9     err = PTR_ERR(misc_class);
10     if (IS_ERR(misc_class))
11         goto fail_remove;
12
13     err = -EIO;
14     if (register_chrdev(MISC_MAJOR,"misc",&misc_fops))
15         goto fail_printk;
16     misc_class->devnode = misc_devnode;
17     return 0;
18
19 fail_printk:
20     printk("unable to get major %d for misc devices\n", MISC_MAJOR);
21     class_destroy(misc_class);
22 fail_remove:
23     remove_proc_entry("misc", NULL);
24     return err;
25 }
26 subsys_initcall(misc_init);
```

misc_init

class_create 创建了一个名为misc的类

register_chrdev(MISC_MAJOR,"misc",&misc_fops) 使用register_chrdev注册了一个字符设备驱动，主设备号为MISC_MAJOR（10）；

```
static const struct file_operations misc_fops = {
    .owner        = THIS_MODULE,
    .open        = misc_open,
};
```

misc类型驱动提供了一个统一.open函数misc_open函数；

misc_open 这个函数的实质是通过inode找到misc类的次设备号minor，然后在通过次设备号和misc链表的次设备号进行匹配，匹配好以后取出

```c
static int misc_open(struct inode * inode, struct file * file)
{
    int minor = iminor(inode);
    struct miscdevice *c;
    int err = -ENODEV;
    const struct file_operations *old_fops, *new_fops = NULL;

    mutex_lock(&misc_mtx);

    list_for_each_entry(c, &misc_list, list) {
        if (c->minor == minor) {
            new_fops = fops_get(c->fops);
            break;
        }
    }

    if (!new_fops) {
        mutex_unlock(&misc_mtx);
        request_module("char-major-%d-%d", MISC_MAJOR, minor);
        mutex_lock(&misc_mtx);

        list_for_each_entry(c, &misc_list, list) {
            if (c->minor == minor) {
                new_fops = fops_get(c->fops);
                break;
            }
        }
        if (!new_fops)
            goto fail;
    }

    err = 0;
    old_fops = file->f_op;
    file->f_op = new_fops;
    if (file->f_op->open) {
        file->private_data = c;
        err=file->f_op->open(inode,file);
        if (err) {
            fops_put(file->f_op);
            file->f_op = fops_get(old_fops);
        }
    }
    fops_put(old_fops);
fail:
    mutex_unlock(&misc_mtx);
    return err;
}
```

```c
int misc_register(struct miscdevice * misc)
{
    struct miscdevice *c;
    dev_t dev;
    int err = 0;

    INIT_LIST_HEAD(&misc->list);

    mutex_lock(&misc_mtx);
    list_for_each_entry(c, &misc_list, list) {
        if (c->minor == misc->minor) {
            mutex_unlock(&misc_mtx);
            return -EBUSY;
        }
    }

    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);
    }

    dev = MKDEV(MISC_MAJOR, misc->minor);

    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
     */
    list_add(&misc->list, &misc_list);
 out:
    mutex_unlock(&misc_mtx);
    return err;
}
```

在include/linux/miscdevice.h中定义了miscdevice 结构体，所有的misc模型驱动设备；都在内核围护的一个misc_list链表中；

内核维护一个misc_list链表，misc设备在misc_register注册的时候链接到这个链表，在misc_deregister中解除链接。

```
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;
};
```

misc_register函数

```c
int misc_register(struct miscdevice * misc)
{
    struct miscdevice *c;
    dev_t dev;
    int err = 0;

    INIT_LIST_HEAD(&misc->list);

    mutex_lock(&misc_mtx);
    list_for_each_entry(c, &misc_list, list) {
        if (c->minor == misc->minor) {
            mutex_unlock(&misc_mtx);
            return -EBUSY;
        }
    }

    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);
    }

    dev = MKDEV(MISC_MAJOR, misc->minor);

    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
     */
    list_add(&misc->list, &misc_list);
 out:
    mutex_unlock(&misc_mtx);
    return err;
}
```

misc_register

misc->this_device = device_create(misc_class, misc->parent, dev, misc, "%s", misc->name);

调用这个函数来初创建设备；

misc_deregister函数来取消注册；

```
int misc_deregister(struct miscdevice *misc)
{
    int i = DYNAMIC_MINORS - misc->minor - 1;

    if (list_empty(&misc->list))
        return -EINVAL;

    mutex_lock(&misc_mtx);
    list_del(&misc->list);
    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;
}
```

4：代码实战：

拿一段x210_buzzer的代码进行分析

```
module_init(dev_init);
module_exit(dev_exit);
```

看一下dev_init函数(首先初始化好dev_fops结构体、misc结构体)

```
static struct file_operations dev_fops = {
    .owner   =   THIS_MODULE,
    .open    =   x210_pwm_open,
    .release =   x210_pwm_close,
    .ioctl   =   x210_pwm_ioctl,
};

static struct miscdevice misc = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = DEVICE_NAME,
    .fops = &dev_fops,
};
```

```c
static int __init dev_init(void)
{
    int ret;

    init_MUTEX(&lock);
    ret = misc_register(&misc);

    /* GPD0_2 (PWMTOUT2) */
    ret = gpio_request(S5PV210_GPD0(2), "GPD0");
    if(ret)
        printk("buzzer-x210: request gpio GPD0(2) fail");

    s3c_gpio_setpull(S5PV210_GPD0(2), S3C_GPIO_PULL_UP);
    s3c_gpio_cfgpin(S5PV210_GPD0(2), S3C_GPIO_SFN(1));
    gpio_set_value(S5PV210_GPD0(2), 0);

    printk ("x210 "DEVICE_NAME" initialized\n");
        return ret;
}
```

这个函数中做了三件事：

init_MUTEX	初始化信号量
misc_register	注册驱动
gpio_request	申请gpio

这样misc设备驱动已经写好了，在补充一下具体fops中的硬件的操作方法即可；

三个函数分别为：x210_pwm_close、x210_pwm_open、x210_pwm_ioctl

x210_pwm_open：尝试lock如果成功则返回0，表示可以使用，如果不成功则返回EBUSY

```c
static int x210_pwm_open(struct inode *inode, struct file *file)
{
    if (!down_trylock(&lock))
        return 0;
    else
        return -EBUSY;

}
```

x210_pwm_close，解锁返回0

```c
static int x210_pwm_close(struct inode *inode, struct file *file)
{
    up(&lock);
    return 0;
}
```

最关键的是*x210_pwm_ioctl函数*

这个函数是真正的提供给应用层操作buzzer的函数；

函数原型：

*int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);*

使用内核的ioctl函数可以对很多驱动程序的参数进行设置，如串口波特率、buzzer的频率等等；

这个函数主要的两个参数是：*unsigned int, unsigned long*

unsigned int传的是cmd，unsigned long 传的是参数；

*当命令为*PWM_IOCTL_SET_FREQ时，调用PWM_Set_Freq函数设置频率

当命令为PWM_IOCTL_STOP时，调用PWM_Stop函数；

```
 1 static int x210_pwm_ioctl(struct inode *inode, struct file *file, unsigned int cmd,
unsigned long arg)
 2 {
 3     switch (cmd)
 4     {
 5         case PWM_IOCTL_SET_FREQ:
 6             printk("PWM_IOCTL_SET_FREQ:\r\n");
 7             if (arg == 0)
 8                 return -EINVAL;
 9             PWM_Set_Freq(arg);
10             break;
11 
12         case PWM_IOCTL_STOP:
13         default:
14             printk("PWM_IOCTL_STOP:\r\n");
15             PWM_Stop();
16             break;
17     }
18 
19     return 0;
20 }
```

```
1 void PWM_Stop( void )
2 {
3     //将GPD0_2设置为input
4     s3c_gpio_cfgpin(S5PV210_GPD0(2), S3C_GPIO_SFN(0));
5 }
```

pwm_set_freq函数是真正的操作硬件的函数

```c
static void PWM_Set_Freq( unsigned long freq )
{
    unsigned long tcon;
    unsigned long tcnt;
    unsigned long tcfg1;

    struct clk *clk_p;
    unsigned long pclk;

    //unsigned tmp;

    //设置GPD0_2为PWM输出
    s3c_gpio_cfgpin(S5PV210_GPD0(2), S3C_GPIO_SFN(2));

    tcon = __raw_readl(S3C2410_TCON);
    tcfg1 = __raw_readl(S3C2410_TCFG1);

    //mux = 1/16
    tcfg1 &= ~(0xf<<8);
    tcfg1 |= (0x4<<8);
    __raw_writel(tcfg1, S3C2410_TCFG1);

    clk_p = clk_get(NULL, "pclk");
    pclk  = clk_get_rate(clk_p);

    tcnt  = (pclk/16/16)/freq;

    __raw_writel(tcnt, S3C2410_TCNTB(2));
    __raw_writel(tcnt/2, S3C2410_TCMPB(2));//占空比为50%

    tcon &= ~(0xf<<12);
    tcon |= (0xb<<12);        //disable deadzone, auto-reload, inv-off, update TCNTB0&TCMPB0, start timer 0
    __raw_writel(tcon, S3C2410_TCON);

    tcon &= ~(2<<12);            //clear manual update bit
    __raw_writel(tcon, S3C2410_TCON);
}
```

所以说不通的硬件会根据他不同的特点来写驱动，这需要更多的经验；

---

x210_pwm_open

转载于:https://www.cnblogs.com/biaohc/p/6676499.html