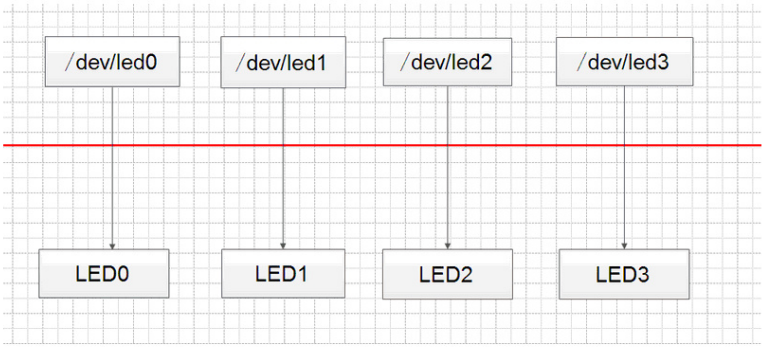
Linux 设备驱动之LED 驱动(二)

——编写者:草根老师(程姚根)

前面我们提出了一个问题，如何让我们的驱动可以支持对单个LED灯进行控制?
这里给出一种解决方案，当然不是唯一的，我们使用这种解决方案的目的是让大家理解"如何让一个驱动来驱动多个同类型的设备"。

一、 解决方案的思想



我们在用户空间创建四个设备文件，这四个文件的主设备号一样，次设备号分别为0,1,2,3。当用户空间打开对应的设备文件时，我在驱动层获取设备文件的次设备号。我们根据不同的次设备号，对不同的LED灯进行控制。

二、修改驱动代码

(1)修改register_led_device函数

```
int register_led_device(struct led_device *led)
{
    int ret;
    int i;
    struct device *device;

    //初始化led_device
    init_led_device(led);

    //动态申请设备号
    ret = alloc_chrdev_region(&led->devno,0, 4,"s5pc100-led");
    if(ret < 0){
        printk("Cannot alloc dev num");
        return ret;
    }

    //添加字符设备
    ret = cdev_add(&led->cdev,led->devno, 4);
    if(ret < 0){
        printk("Failed to add cdev");
        goto err_cdev_add;
    }

    //创建led类
    led->cls = class_create(THIS_MODULE, "led");
    if(IS_ERR(led->cls)){
        printk("Failed to class_create\n");
        ret = PTR_ERR(led->cls);

    for(i = 0;i < 4;i ++)
    {
        //导出设备信息到用户空间,由udev来创建设备节点
        device = device_create(led->cls,NULL,MKDEV(MAJOR(led->devno),i),NULL, "led%d",i);
        if(IS_ERR(device)){
            printk("Failed to device_create");
            ret = PTR_ERR(device);
            goto err_device_create;
        }
```

```
    }
    return 0;

err_device_create:
    for(;i > 0;i --)
    {
        device_destroy(led->cls, MKDEV(MAJOR(led->devno),i));
    }
    class_destroy(led->cls);

err_class_create:
    cdev_del(&led->cdev);

err_cdev_add:
    unregister_chrdev_region(led->devno,4);
    return ret;
}
```

(2)修改字符设备驱动的xxx_open接口

```
int s5pc100led_open(struct inode *inode, struct file *file)
{
    int n;

    //获取设备文件对应的此设备号
    n = iminor(inode);
    file->private_data = (void *) (unsigned long) n;

    return 0;
}
```

为什么要这样做呢?思考.....

(3)修改xxx_ioctl接口

```
void s5pc100_led_opt(int mode,int num)
{
    unsigned long leddat;

    leddat = readl(gled->regbase + S5PC100_LEDDAT);
    switch(mode)
    {
        case S5PC100_LED_ON:
            leddat |= (1 << num);
            break;

        case S5PC100_LED_OFF:
            leddat &=~(1 << num);
            break;
    }

    writel(leddat,gled->regbase  + S5PC100_LEDDAT);

    return;
}

long s5pc100led_ioctl(struct file *file, unsigned int cmd, unsigned long arg)
{

    unsigned long  led_num = (unsigned long)file->private_data;

    switch(cmd)
    {
        case DEV_LED_ON:
            s5pc100_led_opt(S5PC100_LED_ON, led_num);
            break;

        case DEV_LED_OFF:
            s5pc100_led_opt(S5PC100_LED_OFF, led_num);
            break;
```

```
        default:
            return -EINVAL;
    }

    return 0;
}
```

前面我们在s5pc100_open函数中把次设备号保存在了file->private_data中，这样当应用层调用 ioctl函数操作的时候，我们只需要从file->private_data读取前面我们保存的值，就可以知道对哪个LED灯进行控制了。

三、实现平台驱动