Linux驱动学习记录-18.MISC的LED驱动

原创

不良高须 于 2021-11-12 11:09:10 发布 512 收藏

分类专栏： Linux Linux驱动 文章标签： linux

本文链接：https://blog.csdn.net/weixin_41903358/article/details/121282807

misc的意思是混杂的，因此叫做混杂驱动。MISC驱动是最简单的字符设备驱动，通常嵌套在platform总线驱动里。

目录标题

一、MISC驱动简介

1.结构体

所有MISC驱动的主设备号都是10，不同的设备使用不同的次设备号。MISC会自动创建cdev，不用像之前一样手动创建。因此只要向Linux注册一个miscdevice设备，定义在miscdevice.h。结构体如下：

```
struct miscdevice  {
 int minor;
 const char *name;
 const struct file_operations *fops;
 struct list_head list;
 struct device *parent;
 struct device *this_device;
 const struct attribute_group **groups;
 const char *nodename;
 umode_t mode;
};
```

我们关心的成员主要是前三个：

- minor：测设备号
- *name：设备名字
- *fops：设备函数操作集
  预定义的次设备号在结构体上面，使用MISC的时候从上面已定义的次号选择一个，自己也可添加设备号。

2.使用

当设置好miscdevice后，需要使用misc_register向内核注册MISC设备。移除时，用misc_deregister来注销设备。

```
int misc_register(struct miscdevice *misc)
//成功返回0，失败返回负数
int misc_deregister(struct miscdevice *misc)
//成功返回0，失败返回负数
```

而不用像以前一样，调用一大堆函数来注册注销。

二、LED驱动程序

采用设备树platform总线方式下的MISC的led驱动。程序框架如下：

1.MISC设备结构体

相比于从前，MISC驱动少了主、此设备号的成员，因为主设备号默认10，次设备号在预定义里。

```
struct miscled_dev{
 dev_t devid;
 struct cdev cdev;
 struct calss *calss;
 struct device *device;
 struct device_node *nd;
 int led_gpio;
};
struct miscled_dev miscled;
```

2.开、写等操作函数，以及函数集

```
static int miscled_open(struct inode *inode, struct file *filp)
{
 filp->peivate_data = &miscled;
 return 0;
}

static ssize_t miscled_write(struce file *filp, const char __user *buf, size_t cnt, loff_t *offt)
{
 int retvalue;
 unsigned cahr databuf[1];
 unsigned char ledstate;
 struct miscled_dev *dev = filp->peivate_data;

 retvalue = copy_from_user(databuf, buf, cnt);
 if(retvalue < 0){
  printk("kernel write failed!\r\n");
  return -EFAULT;
 }
 ledstate = databuf[0];

 if(ledstate == LEDON){
  gpio_set_value(dev->led_gpio, 0);
 } else if(ledstate == LEDOFF){
  gpio_set_value(dev->led_gpio, 1);
 }
 return 0;
}

static sturt file_operations miscled_fops = {
 .owner = THINS_MODULE,
 .open = miscled_open,
 .write = miscled_write,
};
```

3.MISC驱动结构体

```
#define MISCLED_NAME "miscled"
#define MISCLED_MINOR 144

static struct miscdevice miscled_dev = {
 .minor = MISCLED_MINOR,
 .name = MISCLED_NAME,
 .fops = miscled_fops,
};
```

4.probe和remov函数、匹配表以及platform结构体

```
static int miscled_probe(struct platform_device *dev)
{
 int ret = 0;
 ret = misc_register(&miscled_dev); //注册misc设备，输入结构体地址

 miscled.nd = of_find_node_by_path("/led");

 miscled.led_gpio = of_get_named_gpio(miscled.nd, "led-gpio", 0);

 gpio_request(miscled.led_gpio, "miscled");
 gpio_direction_output(miscled.led_gpio, 1);

 return 0;
}

static int miscled_remove(struct platform_device *dev)
{
 gpio_set_value(miscled.led_gpio, 1);
 misc_deregister(&miscled_dev); //注销misc设备
 return 0;
}
/*匹配列表*/
static const struct of_device_id led_of_match[] = {
 { .compatible = "gpio-led" },
 { /*Sentinel*/ }
};
static struct platform_device miscled_driver = {
 .driver = {
  .name = "imx6ul-led", //驱动名字，加载成功会在/sys/bus/platform/drivers目录下存一个“imx6ul-led”的文件，用于和platfor
m设备的name匹配。
  .of_math_table = led_of_match,
 },
 .probe = miscled_probe,
 .remove = miscled_remove,
};
```

5.出口入口函数以及最后的定义

```
static int __init miscled_init(void)
{
 return platform_driver_register(&miscled_driver);
}
static void __exit miscled_exit(void)
{
 platform_driver_unregister(&miscled_driver);
}

module_init(miscled_init);
module_exit(led_eximiscled_exitt);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("Gordon");
```

三、测试编译

进入/sys/class/misc/可以看到对应的MISC设备文件。
驱动和设备匹配成功后会生成/dev/miscled这个设备驱动文件。
测试驱动直接用设备树led的应用程序即可。