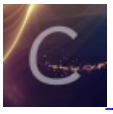
linux misc device字符杂项设备驱动

原创

hello_courage 于 2018-10-07 20:38:05 发布 658 收藏 2
分类专栏： Embeded Software Development Linux 文章标签： linux 字符设备 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。miscdevice共享一个主设备号MISC_MAJOR(即10)，但次设备号不同。misc设备其实就是特殊的字符设备，主设备编号采用10，并且可自动生成设备节点。

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，misc_register最终也是通过调用register_chrdev()来注册设备。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。

1. 实现struct file_operations

```
static struct file_operations leds_ops = {

.owner = THIS_MODULE,
.open = leds_open,
.release = leds_release,
.unlocked_ioctl = leds_ioctl,

};
```

2. 初始化struct miscdevice，定义一个misc设备

```
static struct miscdevice leds_dev = {

.minor = MISC_DYNAMIC_MINOR,
.fops = &leds_ops,
.name = "leds", //此名称将显示在/dev目录下面

};
```

注：其中minor如果填充MISC_DYNAMIC_MINOR，则是动态动态次设备号。

3. 注册和释放misc设备

1）注册

int misc_register(struct miscdevice * misc);

misc_register(&leds_dev);

注：此函数中会自动创建设备节点，即设备文件。无需mknod指令创建设备文件，因为misc_register()会调用device_create()创建设备节点。

2）释放

int misc_deregister(struct miscdevice *misc);

misc_deregister(&leds_dev);

4. 其它

1）头文件 #include <linux/miscdevice.h>

2）miscdevice 结构体

struct miscdevice { int minor; //次设备号 通常为MISC_DYNAMIC_MINOR 动态分配 const char *name; //设备的名字 const struct file_operations *fops;//函数操作集 struct list_head list; struct device *parent; struct device *this_device; const char *nodename; umode_t mode; };

通常 miscdevice的 minor 、name 和 fops是需要实现的。

5. A33下一个GPIO驱动实例

```
#include <linux/init.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/fs.h>
#include <mach/gpio.h>
#include <linux/gpio.h>
#include <linux/miscdevice.h>



static int led_gpios[] = {

GPIOH(7),

};



#define LED_NUM ARRAY_SIZE(led_gpios)
```

```
int leds_open(struct inode *inode,struct file *filp)
{

 printk("leds device opened success!\n");
 return nonseekable_open(inode,filp); //通知内核你的设备不支持llseek

}



int leds_release(struct inode *inode,struct file *filp)
{

 printk("leds device closed success!\n");
 return 0;

}


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

 printk("debug: leds_ioctl cmd is %d\n" , cmd);

 switch(cmd)
 {

  case 0: //不加break，执行case1
  case 1:
  if (arg > LED_NUM) {

  return -EINVAL;

  }

  gpio_set_value(led_gpios[arg], cmd);
  break;

  default:
  return -EINVAL;

 }

 return 0;

}



static struct file_operations leds_ops = {
 .owner = THIS_MODULE,
 .open = leds_open,
 .release = leds_release,
 .unlocked_ioctl = leds_ioctl,

};


```

```
static struct miscdevice leds_dev = {
 .minor = MISC_DYNAMIC_MINOR,
 .fops = &leds_ops,
 .name = "leds", //此名称将显示在/dev目录下面

};


static int __init leds_init(void)
{

 int ret, i;
 char *banner = "leds Initialize\n";

 printk(banner);

 for(i=0; i<LED_NUM; i++)
 {

  //申请gpio，设置为输出，高电平

  ret = gpio_request_one(led_gpios[i], GPIOF_OUT_INIT_HIGH,"LED");

  if (ret) {
   printk("leds: request GPIO %d for LED failed, ret = %d\n", led_gpios[i], ret);
   return ret;

  }

 }

 ret = misc_register(&leds_dev);

 if(ret<0)
 {
  printk("leds: register device failed!\n");
  goto exit;

 }

 return 0;
 exit:
 misc_deregister(&leds_dev);
 return ret;

}



static void __exit leds_exit(void)
{

 misc_deregister(&leds_dev);

}



module_init(leds_init);
module_exit(leds_exit);
```

```
MODULE_LICENSE("Dual BSD/GPL");
```