Linux内核之misc框架

原创

hwx1546 于 2022-02-26 12:27:51 发布 431 收藏

文章标签： linux

本文链接：https://blog.csdn.net/hwx1546/article/details/123147299

Linux内核为每种设备都抽象出了框架，开发人员只需往框架中填充信息即可进行设备的注册。下面来讲解一下Linux内核的misc框架。

misc又叫杂散类设备，在早期的内核中，向ADC、WATCHDOG、PWM等设备都没有一个明确的框架，于是这些设备就都归类到了misc框架。在后来内核版本中才逐步完善各种设备框架。下面就来简单了解一下misc设备的框架，以及使用misc框架来写一个测试驱动。

- misc框架代码

```
static int __init misc_init(void)
{
 int err;

#ifdef CONFIG_PROC_FS
 proc_create("misc", 0, NULL, &misc_proc_fops);
#endif
 misc_class = class_create(THIS_MODULE, "misc");  // 创建类，类名叫misc
 err = PTR_ERR(misc_class);
 if (IS_ERR(misc_class))
  goto fail_remove;

 err = -EIO;
 if (register_chrdev(MISC_MAJOR,"misc",&misc_fops))  // 注册misc字符设备，设备号是10
  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;
}
```

misc_init函数是初始化misc框架，首先会先创建一个叫misc的类，然后在这个类中注册一个叫misc的设备，设备号是10。

```
/ # cd /sys/class/
/sys/class # ls
ata_device    gpio          misc          regulator     tty
ata_link      graphics      mmc_host      rfkill        ubi
ata_port      i2c-dev       mtd           rtc           udc
backlight     ieee80211     net           scsi_device   vc
bdi           input         power_supply  scsi_disk     video4linux
block         lcd           pps           scsi_host     vtconsole
dma           leds          ptp           sound         watchdog
drm           mdio_bus      pwm           spi_master
firmware      mem           rc            thermal
/sys/class #
```



```
/sys/class # cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  7 vcs
 10 misc
 13 input
 29 fb
 81 video4linux
 89 i2c
 90 mtd
116 alsa
128 ptm
136 pts
180 usb
189 usb_device
207 ttymxc
226 drm
249 ttyLP
250 iio
251 watchdog
252 ptp
253 pps
254 rtc
```


可以看到在内核中，通过命令可以查看misc设备的主设备号是10，在class中有一个叫misc的类。

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

 INIT_LIST_HEAD(&misc->list);

 mutex_lock(&misc_mtx);

 if (misc->minor == MISC_DYNAMIC_MINOR) { // 判断次设备是否为255，如果是就自动分配次设备号，否则就使用开发人员传进来的次设备号
  int i = find_first_zero_bit(misc_minors, DYNAMIC_MINORS);
  if (i >= DYNAMIC_MINORS) {
   err = -EBUSY;
   goto out;
  }
  misc->minor = DYNAMIC_MINORS - i - 1;
  set_bit(i, misc_minors);
 } else {
  struct miscdevice *c;

  list_for_each_entry(c, &misc_list, list) {
   if (c->minor == misc->minor) {
    err = -EBUSY;
    goto out;
   }
  }
 }

 dev = MKDEV(MISC_MAJOR, misc->minor); // 将主设备号和次设备号进行合并得到设备号

 misc->this_device =
  device_create_with_groups(misc_class, misc->parent, dev,
       misc, misc->groups, "%s", misc->name); // 创建设备
 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是内核提供给开发人员的注册misc设备的接口函数。该函数主要做了两件事情，首先判断次设备号是否需要自动分配，然后调用device_create_with_groups注册设备。

```
struct miscdevice  {
 int minor;  // 次设备号
 const char *name;  // 设备名字
 const struct file_operations *fops;  // 字符设备操作函数
 struct list_head list;               // 链表头
 struct device *parent;               // 父设备
 struct device *this_device;
 const struct attribute_group **groups;
 const char *nodename;
 umode_t mode;
};
```

struct miscdevice结构体需要开发者对里面的成员变量进行填充，填充完之后调用misc_register注册设备。

下面来写一个简单的misc设备驱动。
驱动程序

```c
struct miscdevice led_misc;  // 定义misc结构体变量

ssize_t chrdev_read (struct file *file, char __user *usr, size_t size, loff_t *loff)
{
 printk("%s\r\n",__func__);
 return 0;
}
int chrdev_open (struct inode *inode, struct file *file)
{
 printk("%s\r\n",__func__);
 return 0;
}
int chrdev_release (struct inode *inode, struct file *file)
{
 printk("%s\r\n",__func__);
 return 0;
}
struct file_operations led_fops =
{
 .open    = chrdev_open,
 .read    = chrdev_read,
 .release = chrdev_release,
};


static int __init chrdev_init(void)
{
 int ret = 0;

 led_misc.minor = MISC_DYNAMIC_MINOR; // 自动分配次设备号
 led_misc.name = "led_misc_dev";      // 设备名
 led_misc.fops = &led_fops;           // 关联文件操作函数

 misc_register(&led_misc);           // 注册设备

 return 0;
}


static void __exit chrdev_exit(void)
{
 misc_deregister(&led_misc);

}

module_init(chrdev_init);
module_exit(chrdev_exit);

MODULE_DESCRIPTION("xxxxxx");
MODULE_AUTHOR("xxxxxx");
MODULE_LICENSE("GPL");
```

可以看到驱动程序使用misc框架注册设备非常简单，不用再关心主设备号，次设备号，不用再自己去注册类，然后再注册设备。因为misc框架已经帮我们把这些东西全部都搭建好了，我们需要做的就是填充struct miscdevice结构体，然后调用misc_register函数就可以完成注册。