Linux MISC 驱动

作者：zhang-ke-wei
原文链接：https://ld246.com/article/1597915323634
来源网站：链滴

misc 的意思是混合、杂项的，因此 MISC 驱动也叫做杂项驱动，其实就是最简单的字符设备驱动，通常嵌套在 platform 总线驱动中，实现复杂的驱动。

所有的 MISC 设备驱动的主设备号都为 10，不同的设备使用不同的从设备号。随着 Linux字符设备动的不断增加，设备号变得越来越紧张，尤其是主设备号， MISC 设备驱动就用于解决此问题。 MISC 设备会自动创建 cdev，不需要手动创建，因此采用 MISC 设备驱动可以简化字符设备驱动的编写。们需要向 Linux 注册一个 miscdevice 设备， miscdevice是一个结构体，定义在文件include/linux/mscdevice.h 中，内容如下：

```
struct miscdevice {
    int minor;  /* 子设备号 */
    const char *name;  /* 设备名字 */
    const struct file_operations *fops;  /* 设备操作集 */
    struct list_head list;
    struct device *parent;
    struct device *this_device;
    const struct attribute_group **groups;
    const char *nodename;
    umode_t mode;
};
```

定义一个 MISC 设备(miscdevice 类型)以后我们需要设置 minor、 name 和 fops 这三个成员变量。minor 表示子设备号， MISC 设备的主设备号为 10，这个是固定的，需要用户指定子设备号， Linux 系统已经预定义了一些 MISC 设备的子设备号，这些预定义的子设备号定义在include/linux/miscdevie.h 文件中，如下所示：

```
#define PSMOUSE_MINOR 1
#define MS_BUSMOUSE_MINOR 2 /* unused */
#define ATIXL_BUSMOUSE_MINOR 3 /* unused */
/*#define AMIGAMOUSE_MINOR 4 FIXME OBSOLETE */
#define ATARIMOUSE_MINOR 5 /* unused */
#define SUN_MOUSE_MINOR 6 /* unused */
......
#define MISC_DYNAMIC_MINOR 255
```

我们在使用的时候可以从这些预定义的子设备号中挑选一个，当然也可以自己定义，只要这个子设备没有被其他设备使用接口。

name 就是此 MISC 设备名字，当此设备注册成功以后就会在/dev 目录下生成一个名为 name的设文件。 fops 就是字符设备的操作集合， MISC 设备驱动最终是需要使用用户提供的 fops操作集合。设置好 miscdevice 以后就需要使用 misc_register 函数向系统中注册一个 MISC 设备，此函数原型下：

```
int misc_register(struct miscdevice * misc)
```

misc：要注册的 MISC 设备。

返回值： 负数，失败； 0，成功。

ps：之前创建设备的过程：

```
alloc_chrdev_region();  /* 申请设备号 */
cdev_init();  /* 初始化 cdev */
cdev_add();  /* 添加 cdev */
```

```
class_create();  /* 创建类 */
device_create();  /* 创建设备 */
```

杂项设备之需要调用misc_register即可完成上述繁杂的操作，misc_register函数原型如下：

```
int misc_register(struct miscdevice *misc)
```

misc：杂项设备

返回值：负数，失败； 0，成功。

ps：同样的注销设备驱动的时候，我们需要调用一堆的函数去删除此前创建的 cdev、设备等等内容如下所示：

```
cdev_del(); /* 删除 cdev */
unregister_chrdev_region(); /* 注销设备号 */
device_destroy(); /* 删除设备 */
class_destroy(); /* 删除类 */
```

杂项只需要调用misc_deregister即可，函数原型如下：

```
int misc_deregister(struct miscdevice *misc)
```

misc：要注销的 MISC 设备。

返回值： 负数，失败； 0，成功。

一般杂项设备和platform配合使用，利用platform完成设备和驱动的分离，杂项设备简化字符设备的种操作，一般用于一些简单驱动的编写。

example：

```
struct miscxxx_dev misctest_dev;        /* misc设备 */

/*
 * @description       : 打开设备
 * @param - inode     : 传递给驱动的inode
 * @param - filp      : 设备文件，file结构体有个叫做private_data的成员变量
 *                      一般在open的时候将private_data指向设备结构体。
 * @return            : 0 成功;其他 失败
 */
static int miscxxx_open(struct inode *inode, struct file *filp)
{
    filp->private_data = &miscxxx; /* 设置私有数据 */
    return 0;
}

/*
 * @description       : 向设备写数据
 * @param - filp      : 设备文件，表示打开的文件描述符
 * @param - buf       : 要写给设备写入的数据
 * @param - cnt       : 要写入的数据长度
 * @param - offt      : 相对于文件首地址的偏移
 * @return            : 写入的字节数，如果为负值，表示写入失败
 */
static ssize_t miscxxx_write(struct file *filp, const char __user *buf, size_t cnt, loff_t *offt)
```

```
{
    ......
    return 0;
}

/* 设备操作函数 */
static struct file_operations miscxxx_fops = {
    .owner = THIS_MODULE,
    .open = miscxxx_open,
    .write = miscxxx_write,
};

/* MISC设备结构体 */
static struct miscdevice xxx_miscdev = {
    .minor = MISCBEEP_MINOR,
    .name = MISCBEEP_NAME,
    .fops = &miscxxx_fops,
};

 /*
  * @description     : flatform驱动的probe函数，当驱动与
  *                    设备匹配以后此函数就会执行
  * @param - dev     : platform设备
  * @return          : 0，成功;其他负值,失败
  */
static int miscxxx_probe(struct platform_device *dev)
{
    ......

    /* 一般情况下会注册对应的字符设备，但是这里我们使用MISC设备
       * 所以我们不需要自己注册字符设备驱动，只需要注册misc设备驱动即可
     */
    ret = misc_register(&xxx_miscdev);
    if(ret < 0){
        printk("misc device register failed!\r\n");
        return -EFAULT;
    }

    return 0;
}

/*
 * @description     : platform驱动的remove函数，移除platform驱动的时候此函数会执行
 * @param - dev     : platform设备
 * @return          : 0，成功;其他负值,失败
 */
static int miscxxx_remove(struct platform_device *dev)
{

    /* 注销misc设备 */
    misc_deregister(&xxx_miscdev);
    return 0;
}
```

```
 /* 匹配列表 */
 static const struct of_device_id xxx_of_match[] = {
     { .compatible = "xxxx" },
     { /* Sentinel */ }
 };

 /* platform驱动结构体 */
static struct platform_driver xxx_driver = {
     .driver     = {
         .name   = "xxxx",         /* 驱动名字，用于和设备匹配 */
         .of_match_table = xxx_of_match, /* 设备树匹配表          */
     },
     .probe      = miscxxx_probe,
     .remove     = miscxxx_remove,
};

/*
 * @description    : 驱动出口函数
 * @param          : 无
 * @return         : 无
 */
static int __init xxx_init(void)
{
    return platform_driver_register(&xxx_driver);
}

/*
 * @description    : 驱动出口函数
 * @param          : 无
 * @return         : 无
 */
static void __exit xxx_exit(void)
{
    platform_driver_unregister(&xxx_driver);
}

module_init(xxx_init);
module_exit(xxx_exit);
MODULE_LICENSE("GPL");
MODULE_AUTHOR("xxx");
```