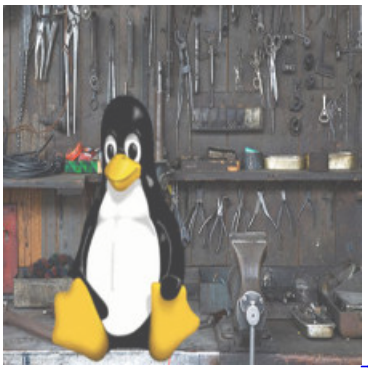
Linux misc设备（一）misc驱动框架

原创

JT同学 于 2019-08-23 15:33:48 发布 1929 收藏 20

分类专栏： Linux驱动 文章标签： Linux驱动 misc

本文链接：https://blog.csdn.net/weixin_42462202/article/details/100039448



Linux misc设备驱动

Linux misc设备（一）misc驱动框架

文章目录

一、misc简介

Linux的驱动设计是趋向于分层的，大多数设备都有自己归属的类型，例如按键、触摸屏属于输入设备，Linux有一个input子系统框架。但是对于adc、蜂鸣器等设备，无法明确其属于什么类型，对于这种设备一般推荐使用misc驱动框架编写驱动程序

misc设备也是一个字符设备，在misc的初始化函数中注册了一个字符设备，主设备号为MISC_MAJOR (10)

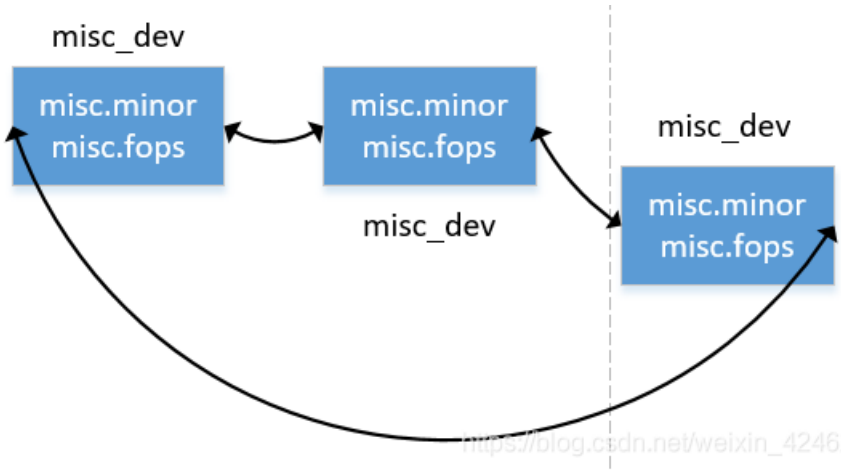
这个字符设备并不是具体的misc设备实例的实现，只是misc核心层用来将应用层的操作转发到具体的misc实例中

在编写misc设备驱动实例时，通过`misc_register()`向misc核心层注册一个字符设备，在此函数中完成了生成设备节点、动态获取次设备号的动作

二、misc驱动框架

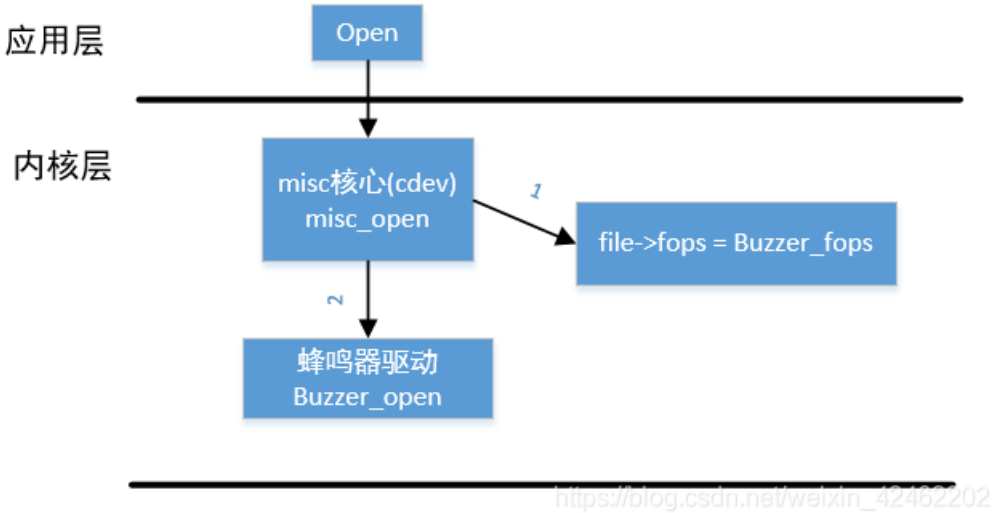
misc设备实例表明一个具体的驱动程序（例如adc、蜂鸣器），每一个misc设备实例都有对应得次设备号和相应驱动程序实现的文件操作集

misc设备实例驱动通过`misc_register`向misc核心层注册misc设备，所有的misc的主设备号都是(MISC_MAJOR)，misc核心为注册的misc设备维护一个双向链表，如下图所示

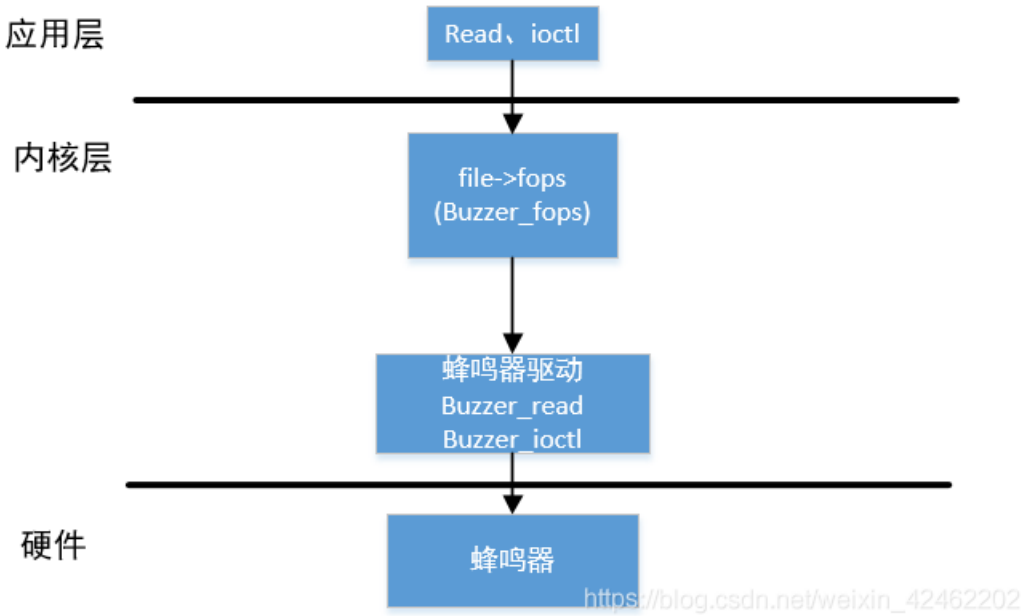


当应用层通过open打开设备节点时，内核会为打开的设备文件创建一个`struct file`对象

然后调用到misc核心层的`cdev的open函数`，在cdev的open函数中，从misc维护的`misc设备链表`中，根据`次设备号`找到对应的misc设备实例，将`struct file`对象中的文件操作集合更改为`misc设备实例的文件操作集合`，然后调用misc设备实例的open函数，如下图所示



之后应用层通过read、ioctl等来操作设备文件，都会调用对应的`struct file`对象中的文件操作集合，进而操作具体的misc实例，如下图所示



以上就是misc设备的驱动框架

三、misc源码剖析

在内核文件`drivers\char\misc.c`中时misc驱动核心的实现，下面详细剖析这个源码

首先看驱动的入口

```
static struct class *misc_class;
static const struct file_operations misc_fops = {
 .owner  = THIS_MODULE,
 .open  = misc_open,
};


static int __init misc_init(void)
{
    /* 创建misc类 */
    misc_class = class_create(THIS_MODULE, "misc");

    /* 注册字符设备 */
    register_chrdev(MISC_MAJOR,"misc",&misc_fops); //MISC_MAJOR主设备号（10）
}
```

首先创建misc类，这将给misc设备实例生成设备节点时使用

然后注册字符设备，可以看到`misc_fops`中只实现了open函数，此函数是用于转发系统调用到具体的misc设备实例中，是misc驱动框架的核心

再来看注册函数`misc_register`

```
int misc_register(struct miscdevice * misc)
{
    /* 找到空闲的次设备号 */
    int i = find_first_zero_bit(misc_minors, DYNAMIC_MINORS);

    /* 设备号 */
 dev = MKDEV(MISC_MAJOR, misc->minor);

    /* 生成设备节点 */
 misc->this_device = device_create(misc_class, misc->parent, dev,
       misc, "%s", misc->name);

 list_add(&misc->list, &misc_list); //将设备添加到链表中
}
```

可以看到注册函数会分配一个次设备号，然后生成设备节点，再将misc设备添加到链表中

接下来分析`misc_open`

```
static int misc_open(struct inode * inode, struct file * file)
{
    /* 遍历misc设备链表，根据次设备号找到对应misc设备实例的fops */
 list_for_each_entry(c, &misc_list, list) {
  if (c->minor == minor) {
   new_fops = fops_get(c->fops);
   break;
  }
 }

    /* 将file对象的fops设置为misc实例的fops */
    file->f_op = new_fops;
}
```

从上面代码中可以看到，首先从misc设备链表中找到misc设备实例，然后将file对应得fops设置为misc设备实例对应的fops，其中`file`是内核为每个打开文件创建的一个对象

之后应用层再进行系统调用，都会调用到misc设备实例

四、misc设备实例驱动编写模板

如何基于misc写一个设备驱动非常简单，下面给出一个模板

```c
static int buzzer_open(struct inode *inode, struct file *file)
{
    return 0;
}


static int buzzer_close(struct inode *inode, struct file *file)
{
 return 0;
}

static int buzzer_ioctl(struct inode *inode, struct file *file, unsigned int cmd, unsigned long arg)
{
    return 0;
}


static struct file_operations buzzer_fops = {
    .owner   =   THIS_MODULE,
    .open    =   buzzer_open,
    .release =   buzzer_close,
    .ioctl   =   buzzer_ioctl,
};

static struct miscdevice buzzer_dev = {
 .minor  = MISC_DYNAMIC_MINOR, //动态分配次设备号
 .name   = "buzzer",
 .fops   = &buzzer_fops, //文件操作集
};

static int __init buzzer_init(void)
{
    /* 注册杂项设备 */
 misc_register(&buzzer_dev);

 return 0;
}

static void __exit buzzer_exit(void)
{
    /* 注销杂项设备 */
 misc_deregister(&buzzer_dev);
}

module_init(buzzer_init);
module_exit(buzzer_exit);
MODULE_LICENSE("GPL");
```