设备驱动中的misc(kernel-4.7)

原创

viewsky11 于 2017-01-05 21:49:17 发布 1214 收藏 1

分类专栏： linux设备驱动 kernel linux设备驱动分析（kernel-4.7） 文章标签： 内核 链表 linux 结构 misc

本文链接：https://blog.csdn.net/viewsky11/article/details/54098020

linux设备驱动 同时被 3 个专栏收录

47 篇文章 9 订阅

订阅专栏

kernel

101 篇文章 5 订阅

订阅专栏

linux设备驱动分析（kernel-4.7）

22 篇文章 21 订阅

订阅专栏

Linux里面的misc杂项设备是主设备号为10的驱动设备，它的注册跟使用比较的简单，所以比较适用于功能简单的设备。

miscdevice共享一个主设备号`MISC_MAJOR`(即10)，但次设备号不同。 所有的miscdevice设备形成了一个链表，对设备访问时内核根据次设备号查找对应的miscdevice设备，然后调用其`file_operations`结构中注册的文件操作接口进行操作。 在内核中用struct miscdevice表示miscdevice设备，然后调用其`file_operations`结构中注册的文件操作接口进行操作。miscdevice的API实现在drivers/char/misc.c中，misc装备的初始化，注册，注销都在这个文件中。在内核中，misc杂项装备驱动接口是对一些字符装备的简单封装，它们同享一个主装备号，有不同的次装备号，同享一个open调用，其他的操作函数在打开后应用linux驱动程序的方法重载进行装载。

misc的设备结构体定义在头文件linux/miscdevice.h里面

```
struct miscdevice  {
    int minor;            //misc设备的次设备号，misc设备主要依赖minor去区分
    const char *name;     //设备名
    const struct file_operations *fops;  //驱动主体处理函数入口指针
    struct list_head list;    //内核有misc list链表
    struct device *parent;
    struct device *this_device;  //当前设备
    const struct attribute_group **groups;
    const char *nodename;    //在/dev下面创建的设备驱动节点
    umode_t mode;
};
```

在misc.c中`misc_init`这个就是misc驱动模型的入口

```
static int __init misc_init(void)
{
    int err;
    struct proc_dir_entry *ret;

    ret = proc_create("misc", 0, NULL, &misc_proc_fops);//创建misc的proc入口
    misc_class = class_create(THIS_MODULE, "misc");//在/sys/class/目录下创建名为misc的类
    err = PTR_ERR(misc_class);
    if (IS_ERR(misc_class))
        goto fail_remove;

    err = -EIO;
    ///注册设备，其中设备的主设备号为MISC_MAJOR为10，设备名为misc，misc_fops是操作函数的集合
    if (register_chrdev(MISC_MAJOR,"misc",&misc_fops))
        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;
}
```

可以看出，这个初始化函数，最主要的功能就是注册字符设备，所用的注册函数是`register_chrdev`。它注册了主装备号为`MISC_MAJOR`，次装备号为0~255的256个设备。并且创建了一个misc类。

```
static inline int register_chrdev(unsigned int major, const char *name,
                  const struct file_operations *fops)
{
    return __register_chrdev(major, 0, 256, name, fops);
}
```

`register_chrdev`函数调用`__register_chrdev`函数来实现注册，`__register_chrdev`函数如下：

```
int __register_chrdev(unsigned int major, unsigned int baseminor,
              unsigned int count, const char *name,
              const struct file_operations *fops)
{
    struct char_device_struct *cd;
    struct cdev *cdev;
    int err = -ENOMEM;

    ///主设备号是10，次设备号为从0开始，分配256个设备
    cd = __register_chrdev_region(major, baseminor, count, name);
    if (IS_ERR(cd))
        return PTR_ERR(cd);

    cdev = cdev_alloc();//分配字符装备
    if (!cdev)
        goto out2;

    cdev->owner = fops->owner;
    cdev->ops = fops;
    kobject_set_name(&cdev->kobj, "%s", name);//设置kobject的名字

    err = cdev_add(cdev, MKDEV(cd->major, baseminor), count); //把字符设备增加到系统中
    if (err)
        goto out;

    cd->cdev = cdev;

    return major ? 0 : cd->major;
out:
    kobject_put(&cdev->kobj);
out2:
    kfree(__unregister_chrdev_region(cd->major, baseminor, count));
    return err;
}
```

misc设备的操作函数的集合

```
static const struct file_operations misc_fops = {
    .owner      = THIS_MODULE,
    .open       = misc_open,
    .llseek     = noop_llseek,
};
```

可以看到这里只有open和llseek函数，用户打开miscdevice装备是通过主装备号对应的打开函数，在这个函数中找到次装备号对应的相应的具体装备的open函数。它的实现以下：

```
static int misc_open(struct inode * inode, struct file * file)
{
    int minor = iminor(inode);
    struct miscdevice *c;
    int err = -ENODEV;
    const struct file_operations *new_fops = NULL;

    mutex_lock(&misc_mtx);
    /*找到次设备号对应的操作函数集合，让new_fops指向这个具体设备的操作函数集合*/
    list_for_each_entry(c, &misc_list, list) {
        if (c->minor == minor) {
            new_fops = fops_get(c->fops);
            break;
        }
    }

    if (!new_fops) {
        mutex_unlock(&misc_mtx);
          /*如果没有找到，则要求加载这个次设备号对应的模块*/
        request_module("char-major-%d-%d", MISC_MAJOR, minor);
        mutex_lock(&misc_mtx);
     /*重新遍历misc_list链表，如果没有找到就退出，否则让new_fops指向这个具体设备的操作函数集合*/
        list_for_each_entry(c, &misc_list, list) {
            if (c->minor == minor) {
                new_fops = fops_get(c->fops);
                break;
            }
        }
        if (!new_fops)
            goto fail;
    }

    /*
     * Place the miscdevice in the file's
     * private_data so it can be used by the
     * file operations, including f_op->open below
     */
    file->private_data = c;

    err = 0;
    /*让主设备号的操作函数集合指针指向具体设备的操作函数集合*/
    replace_fops(file, new_fops);
    if (file->f_op->open)
        /*使用具体设备的open函数open装备*/
        err = file->f_op->open(inode,file);
fail:
    mutex_unlock(&misc_mtx);
    return err;
}
```

misc子系统对外提供的两个重要的API，即`misc_register`和`misc_deregister`函数。

`misc_register()`函数在misc.c中，最主要的功能是基于`misc_class`构造一个设备，将miscdevice结构挂载到`misc_list`列表上，并初始化与linux装备模型相干的结构，它的参数是miscdevice结构体。

```
/**
 *  misc_register   -   register a miscellaneous device
```

```
 *  @misc: device structure
 *
 *  Register a miscellaneous device with the kernel. If the minor
 *  number is set to %MISC_DYNAMIC_MINOR a minor number is assigned
 *  and placed in the minor field of the structure. For other cases
 *  the minor number requested is used.
 *
 *  The structure passed is linked into the kernel and may not be
 *  destroyed until it has been unregistered. By default, an open()
 *  syscall to the device sets file->private_data to point to the
 *  structure. Drivers don't need open in fops for this.
 *
 *  A zero is returned on success and a negative errno code for
 *  failure.
 */

int misc_register(struct miscdevice * misc)
{
    dev_t dev;
    int err = 0;
    bool is_dynamic = (misc->minor == MISC_DYNAMIC_MINOR);

    INIT_LIST_HEAD(&misc->list); //链表项使用时必须初始化

    mutex_lock(&misc_mtx);

    if (is_dynamic) {
        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;
     /*遍历misc_list链表，看这个次设备号之前有无被用过，如果次设备号已被占有则退出*/
        list_for_each_entry(c, &misc_list, list) {
            if (c->minor == misc->minor) {
                err = -EBUSY;
                goto out;
            }
        }
    }

    dev = MKDEV(MISC_MAJOR, misc->minor);//技术处设备号

    misc->this_device =
        device_create_with_groups(misc_class, misc->parent, dev,
                      misc, misc->groups, "%s", misc->name); //创建设备节点
    if (IS_ERR(misc->this_device)) {
        if (is_dynamic) {
            int i = DYNAMIC_MINORS - misc->minor - 1;

            if (i < DYNAMIC_MINORS && i >= 0)
                clear_bit(i, misc_minors);
            misc->minor = MISC_DYNAMIC_MINOR;
        }
        err = PTR_ERR(misc->this_device);
        goto out;
```

```
        goto out;
    }

    /*
     * Add it to the front, so that later devices can "override"
     * earlier defaults
     */
    list_add(&misc->list, &misc_list);//将这个miscdevice添加到misc_list链表中
 out:
    mutex_unlock(&misc_mtx);
    return err;
}
```

这个函数首先遍历`misc_list`链表，查找所用的次设备号是不是已被注册，避免冲突。如果是动态次设备号则分配一个，然后调用MKDEV生成装备号,从这里可以看出所有的misc装备同享一个主装备号`MISC_MAJOR`，然后调用`device_create_with_groups`，生成装备文件。最后将设备加入到`misc_list`链表中。

miscdevice的卸载函数

```
/**
 *  misc_deregister - unregister a miscellaneous device
 *  @misc: device to unregister
 *
 *  Unregister a miscellaneous device that was previously
 *  successfully registered with misc_register().
 */

void misc_deregister(struct miscdevice *misc)
{
    int i = DYNAMIC_MINORS - misc->minor - 1;

    if (WARN_ON(list_empty(&misc->list)))
        return;

    mutex_lock(&misc_mtx);
    list_del(&misc->list);//在misc_list链表中删除miscdevice设备
    device_destroy(misc_class, MKDEV(MISC_MAJOR, misc->minor));//删除设备节点
    if (i < DYNAMIC_MINORS && i >= 0)
        clear_bit(i, misc_minors);
    mutex_unlock(&misc_mtx);
}
```

misc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，可以斟酌使用misc设备接口，方便简单，只需要初始化一个miscdevice的结构，调用`misc_register`就能够了。系统最多有255个杂项装备，由misc设备模块自己占用了一个次装备号。