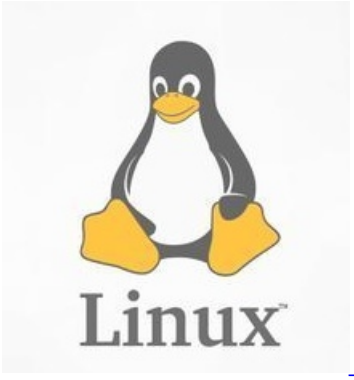
【Linux驱动编程】Linux字符驱动之misc device

原创

Acuity. 于 2019-11-29 23:50:45 发布 507 收藏 2

分类专栏： Linux驱动编程 文章标签： misc device Linux字符驱动

本文链接：https://blog.csdn.net/qq_20553613/article/details/103285204



Linux驱动编程 专栏收录该内容

21 篇文章 26 订阅

订阅专栏

1. misc device

1.1 什么是杂项设备

Linux常用的驱动设备分为字符设备、块设备、网络设备。字符设备是嵌入式Linux中常见的设备。misc device中文翻译称为“杂项设备”，杂项设备本质就是字符设备。嵌入式硬件上存在各类设备，如ADC、DAC、按键、蜂鸣器等，一方面不便于单独分类，另一方面驱动设备号分配有限，因此Linux系统引入“杂项设备”，驱动工程师将这些没有明显区分的设备统一归类为杂项设备。

杂项设备不是新的设备类型，只是是字符设备的一类，是最简单的字符设备。同时，关于设备归类也是由驱动工程决策，并无严格规定。比如，一个ADC驱动，可以归类为普通字符设备，也可以归类为杂项设备。

1.2杂项设备特点

杂项设备的主设备号（MISC_MAJOR）为10，通过次设备号区分不同杂项设备。杂项设备注册时，所有的杂项设备以链表形式存在，系统应用访问设备时，内核会根据次设备号匹配到对应的杂项设备，然后调用“file_operations”结构中注册的文件操作接口（open/read/ioctl等）进行操作。

> - 杂项设备是字符设备的一个子类，是最简单的字符设备。
> - 杂项设备的主设备号（MISC_MAJOR）为10。
> - Linux内核提供杂项设备注册、释放框架，简化字符设备注册过程。

1.3 使用杂项设备优点

- 便于实现，简化设备驱动实现过程。
- 整个系统更规范化，多样化设备统一由内核杂项设备管理。
- 节约系统资源，如主设备号。

2. 杂项设备分析

2.1 杂项设备抽象

杂项设备将字符设备进一步封装一层，有专门的驱动框架，对于驱动工程师来说可以简化字符设备注册过程。Linux内核把杂项设备抽象为一个结构体，位于“include/linux/miscdevice.h”中。

```
struct miscdevice  {
 int minor;
 const char *name;
 const struct file_operations *fops;
 struct list_head list;
 struct device *parent;
 struct device *this_device;
 const char *nodename;
 umode_t mode;
 int (*suspend)(struct miscdevice *, pm_message_t);
 int (*resume)(struct miscdevice *);
};
```

- minor：次设备号。
- name：设备名称，“/dev”目录下显示。
- fops：设备文件操作接口，open/read/write等。
- list：misc设备链表节点
- parent：当前设备父设备，一般为NULL。
- this_device：当前设备，即是linux基本设备驱动框架。

编写杂项设备时，我们主要实现前三个变量。杂项设备的主设备号是10，次设备号需程序员指定，用以区分不同杂项设备。Linux内核在“include/linux/miscdevice.h”预定义了部分次设备号。编写程序时，可以使用预定义的次设备号，也可以自定义次设备号，前提该次设备号没有被其他杂项设备使用，否则设备会注册失败。当分配的次设备号为255（MISC_DYNAMIC_MINOR），则表示由内核自动分配次设备号。

```
/*
 * These allocations are managed by device@lanana.org. If you use an
 * entry that is not in assigned your entry may well be moved and
 * reassigned, or set dynamic if a fixed value is not justified.
 */
#define PSMOUSE_MINOR  1
#define MS_BUSMOUSE_MINOR 2
#define ATIXL_BUSMOUSE_MINOR 3
/*#define AMIGAMOUSE_MINOR 4 FIXME OBSOLETE */
#define ATARIMOUSE_MINOR 5
......
#define MISC_DYNAMIC_MINOR 255
```

设备名称，设备注册成功后，在“/dev”目录生成一个该名称的设备文件。应用层调用杂项设备，最终是调驱动程序注册的fops文件操作集合。

2.2 杂项设备注册

传统字符设备注册过程步骤一般为：

```
[1] alloc_chrdev_region(); /* 申请设备号 */
[2] cdev_init();   /* 初始化字符设备 */
[3] class_create();   /* 创建类 */
[4] device_create();       /* 创建基本驱动设备 */
```

Linux把杂项设备注册封装为一个接口函数“misc_register”，简化了传统字符设备注册过程，注册成功返回0，失败返回负数。关于注册函数原型和分析，其实也是创建传统字符设备的步骤过程。

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

    /* 初始化一个链表头，所有杂项设备挂在这个链表上 */
    INIT_LIST_HEAD(&misc->list);

/* 互斥锁上锁 */
    mutex_lock(&misc_mtx);
    /*先遍历已经注册的设备，通过对比次设备号，如果待注册设备已存在，返回设备忙错误 */
    list_for_each_entry(c, &misc_list, list) {
        if (c->minor == misc->minor) {
            mutex_unlock(&misc_mtx);
            return -EBUSY;
        }
    }

    /* 检查是否需要动态分配次设备号 */
    if (misc->minor == MISC_DYNAMIC_MINOR) {
        int i = find_first_zero_bit(misc_minors, DYNAMIC_MINORS);
        if (i >= DYNAMIC_MINORS) {
            mutex_unlock(&misc_mtx);   /* 无可用次设备号？ */
            return -EBUSY;
        }
        misc->minor = DYNAMIC_MINORS - i - 1;
        set_bit(i, misc_minors);  /* 标识当前设备号已使用 */
    }

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

    /* 创建驱动设备，linux基本设备驱动模型相关 */
    misc->this_device = device_create(misc_class, misc->parent, dev,
                      misc, "%s", misc->name);
    if (IS_ERR(misc->this_device)) {    /* 创建失败 */
        int i = DYNAMIC_MINORS - misc->minor - 1;
        if (i < DYNAMIC_MINORS && i >= 0)
            clear_bit(i, misc_minors);   /* 清除设备号“已使用”标识 */
        err = PTR_ERR(misc->this_device);
        goto out;
    }

    /*
     * Add it to the front, so that later devices can "override"
     * earlier defaults
     */
     /* 将新注册的杂项设备加入到内核维护的misc_list链表中 */
    list_add(&misc->list, &misc_list);
 out:
    mutex_unlock(&misc_mtx);
    return err;
}
```

2.3 杂项设备注销

与注册过程一样，杂项设备注销也只需要执行一个函数即可。传统字符串设备注销过程步骤一般为：

```
[1] device_destory();  /* 删除基本驱动设备*/
[2] class_destory();   /* 删除类 */
[3] cdev_del();   /* 删除字符设备 */
[4] unregister_chrdev_region(); /* 注销设备号 */
```

杂项设备注销函原型和注释，注销成功返回0，失败返回负数：

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

 if (WARN_ON(list_empty(&misc->list))) /* 检查待注销设备是否存在链表中 */
  return -EINVAL;

 mutex_lock(&misc_mtx); /* 互斥锁上锁 */
 list_del(&misc->list);  /* 删除设备链表 */
 device_destroy(misc_class, MKDEV(MISC_MAJOR, misc->minor)); /*删除设备 */
 if (i < DYNAMIC_MINORS && i >= 0)
  clear_bit(i, misc_minors); /* 释放次设备号 */
 mutex_unlock(&misc_mtx); /* 解锁 */
 return 0;
```

3. 杂项设备例子

在之前文章中，编写了一个“软驱动”，注册为字符设备。现将其注册为杂项设备。

3.1 实现过程

第一步：驱动抽象结构体

设备号、类、设备等杂项设备在注册过程已实现，故只需驱动的设备只需保留本身需要的信息（私有数据）。

```
struct memory_device
{
    //struct  cdev   dev;
 //dev_t     devno;
 //struct class  *devclass;
 //struct device   *device;
 char   *mem_buf;
 uint32_t  mem_size;
};
```

第二步：文件操作接口集合

文件操作接口集合保持不变，即read、write、ioctl等实例化。

```
static const struct file_operations memory_fops =
{
    .owner    = THIS_MODULE,
    .open     = memory_drv_open,
    .read     = memory_drv_read,
    .write    = memory_drv_write,
    .release = memory_drv_close,
    .ioctl    = memory_drv_ioctl,
    .llseek = memory_drv_llseek,
};
```

第三步：杂项设备（misc device）实例化

```
static struct miscdevice memory_miscdrv =
{
 .minor = 23,   /* 子设备号 */
 .name  = "dev_mem",  /* 设备名称 */
 .fops  = &memory_fops, /* 操作文件接口 */
};
```

第四步：设备注册

杂项设备基于platform框架实现，通过“探测”函数实现注册。

```
static int  memory_drv_probe(struct platform_device *dev)
{
    int ret = -1;

    pmemory_dev = kmalloc(sizeof(struct memory_device), GFP_KERNEL);
    if (NULL == pmemory_dev)
    {
        ret = -ENOMEM;
        printk("kmalloc request memory falied.\n");
  return ret;
    }
    memset(pmemory_dev, 0, sizeof(struct memory_device));

 ret = misc_register(&memory_miscdrv); /* 注册杂项设备 */
 if(ret < 0)
 {
  printk("memory miscdrv register failed.\n");
  kfree(pmemory_dev);
  pmemory_dev = NULL;
  ret = -EFAULT;
  return ret;
 }

    return 0;
}
```

第五步：设备卸载（注销）

```
static int  memory_drv_remove(struct platform_devie *dev)
{
 misc_deregister(&memory_miscdrv); /* 注销设备 */

 kfree(pmemory_dev->mem_buf);
 pmemory_dev->mem_buf = NULL;
 pmemory_dev->mem_size = 0;
 kfree(pmemory_dev);
 pmemory_dev = NULL;

 return 0;
}
```

余下步骤则是platform驱动框架的注册过程。

3.2 实现结果

在Ubuntu下编译dev_mem.c和drv_mem.c，分别生成dev_mem.ko和drv_mem.ko,分别手动insmod到内核中。然后执行app测试文件。

```
root@ubuntu:/mnt/hgfs/LSW/STHB/drivers/devmem_misc# insmod drv_mem.ko
root@ubuntu:/mnt/hgfs/LSW/STHB/drivers/devmem_misc# insmod dev_mem.ko
root@ubuntu:/mnt/hgfs/LSW/STHB/drivers/devmem_misc# ./app
open "dev_mem" success
Get dev memory size [128]
Set dev memory size [256]
Get dev memory size [256]
read mem:Hello word ABCD
```

加载成功后会在“/dev” 目录生成“dev_mem”设备文件，在misc设备类“/sys/class/misc”中生成“dev_mem”类（所有杂项设备都同属一个类）。

```
root@ubuntu:/mnt/hgfs/LSW/STHB/drivers/devmem_misc# ls /sys/class/misc/
agpgart          ecryptfs  network_latency     rfkill       vmci
cpu_dma_latency  fuse      network_throughput  snapshot     vsock
dev_mem          hpet      pktcdvd             tun
device-mapper    mcelog    psaux               vga_arbiter
```

通过“ls /dev/dev_mem -l”可查看设备的主、次设备号；或者可以通过“cat /proc/misc”查看次设备号。如下图dev_mem主设备号为10，次设备号为23。

```
root@ubuntu:/mnt/hgfs/LSW/STHB/drivers/devmem_misc# ls /dev/dev_mem  -l
crw-rw---- 1 root root 10, 23 Nov 28 23:07 /dev/dev_mem
```

3.3 源码

【1】https://github.com/Prry/linux-drivers/tree/master/devmem_misc

4. 参考

【1】https://blog.csdn.net/armwind/article/details/52166139

【2】https://www.cnblogs.com/deng-tao/p/6042965.html