字符设备驱动之misc驱动

原创

Jason Gel 于 2016-03-29 22:28:50 发布 3703 收藏 9
分类专栏： linux内核 linux驱动 文章标签： linux设备驱动 misc驱动 字符设备驱动

本文链接：https://blog.csdn.net/jin13277480598/article/details/51009616

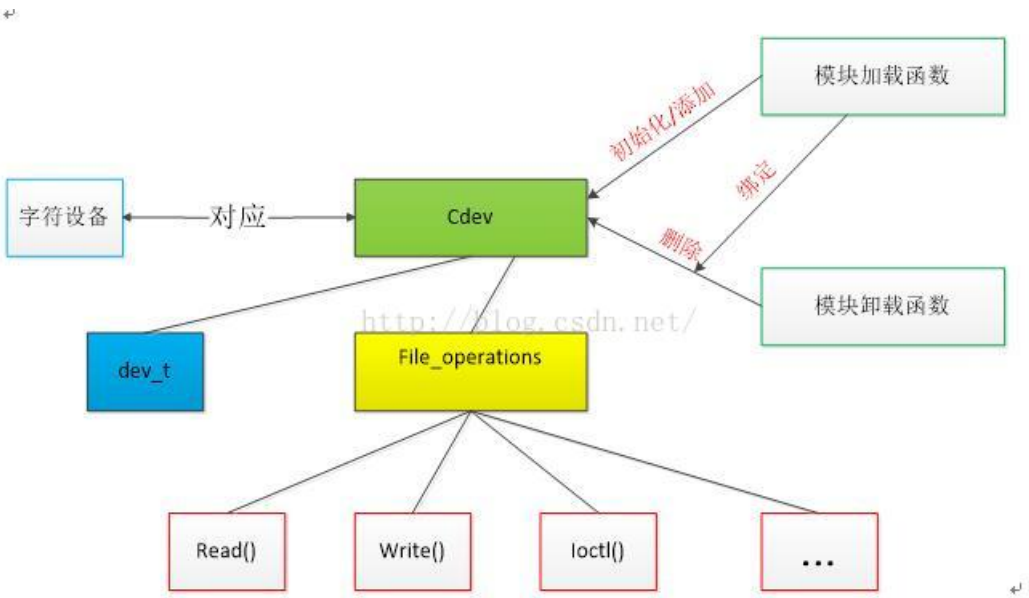
字符设备驱动之misc驱动

Misc驱动，在LDD3上面基本没讲，但由于其方便，现在使用已经相当广泛。本文首先介绍了普通字符驱动的结

如果对字符设备驱动是初次接触，不是特别了解驱动的框架，可以参考 字符设备驱动内核框架小结（一），这篇

1. 普通字符驱动结构及相应的流程

1.1结构图



1.2流程

1. 申请设备号，特别注意：对于一个新的驱动程序，由于使用驱动程序的人并不一定只有我们自己，其可能被广泛使用，随机选定的主设备号可能造成冲突和麻烦。最好不要
随机选择一个当前未使用的设备号，而应该用动态分配机制去获取主设备号。

即驱动程序应该使用alloc_chrdev_region而不是register_chrdev_region函数。

2. 完成cdv的注册，调用cdev_add()函数向系统注册字符设备

3. 调用cdev_de()函数从系统注销字符设备。

4. 用unregister_chrdev_region()函数释放原先申请的设备号。

2.misc驱动

Misc设备驱动与一般字符驱动的优点与联系：

优点：

1. 节省主设备号：使用普通字符设备驱动框架，不管该设备的主设备号是静态或者是动态分布的都会使用一个主设备号。而miscdevice结构体的**主设备号是固定的，MISC_MAJOR定义为10**，在linux内核中，大概可以找到200多处使用miscdevice框架结构的驱动。

2. 使用方便：Misc驱动不再直接采用registe_chrdev_region()或者alloc_chrdev_region()、cdev_add()之类的原始方法申请设备号、注册，而是才用miscdevice的注册方法misc_register(struct miscdevice * misc)。因为它已经封装和优化的很好，能很大程度上简化我们的工作量。

3. 利于linux驱动的分层设计思想：由于Linux驱动倾向于分层设计，各个具体的设备都可以找到它归属的类型，从而嵌套它相应的架构上面，并且只需要实现最底层的那一部分。因为有些设备不知道它属于什么类型，Misc驱动的引入，很好的解决了这个问题，同时它使用起来也更加的

联系：

1.miscdevice本质上也是字符设备，只是具体的miscdevice实例调用misc_register()的时候自动完成了为相应类添加设备、动态获取次设备号。

3.相关源代码

3.1 miscdevice结构体

```
include\linux\Miscdevice.h     内核版本：2.6.11.12

struct miscdevice  {
 int minor;
 const char *name;
 struct file_operations *fops;
 struct list_head list;
 struct device *dev;
 struct class_device *class;
 char devfs_name[64];
};

extern int misc_register(struct miscdevice * misc);
extern int misc_deregister(struct miscdevice * misc);
```

3.2 misc_register函数

```
/**
 * misc_register - register a miscellaneous device                <span style="font-family: monospace; whit
 * @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.
 *
 * A zero is returned on success and a negative errno code for
 * failure.
 */

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

 down(&misc_sem);
 list_for_each_entry(c, &misc_list, list) {
  if (c->minor == misc->minor) {
   up(&misc_sem);
   return -EBUSY;
  }
 }

 if (misc->minor == MISC_DYNAMIC_MINOR) {
  int i = DYNAMIC_MINORS;
  while (--i >= 0)
   if ( (misc_minors[i>>3] & (1 << (i&7))) == 0)
    break;
  if (i<0) {
   up(&misc_sem);
   return -EBUSY;
  }
  misc->minor = i;
 }

 if (misc->minor < DYNAMIC_MINORS)
  misc_minors[misc->minor >> 3] |= 1 << (misc->minor & 7);
 if (misc->devfs_name[0] == '\0') {
  snprintf(misc->devfs_name, sizeof(misc->devfs_name),
    "misc/%s", misc->name);
 }
 //由主、次设备号生产设备号
 dev = MKDEV(MISC_MAJOR, misc->minor);
*/
/**
 *struct class_device *class_simple_device_add(struct class_simple *cs, dev_t dev,
 * struct device *device, const char *fmt, ...)
 * 为一个简单设备类添加设备。
 *  cs:   将设备添加到此简单设备类。
 *  dev:     分配的设备号。
 *  device:  要添加的设备。
 *  fmt:  用于格式化名称。
*/
 misc->class = class_simple_device_add(misc_class, dev,
           misc->dev, misc->name);

 if (IS_ERR(misc->class)) {
```

```
 if (IS_ERR(misc->class)) {
  err = PTR_ERR(misc->class);
  goto out;
 }

 err = devfs_mk_cdev(dev, S_IFCHR|S_IRUSR|S_IWUSR|S_IRGRP,
       misc->devfs_name);
 if (err) {
  class_simple_device_remove(dev);
  goto out;
 }

 /*
  * Add it to the front, so that later devices can "override"
  * earlier defaults
  */
 list_add(&misc->list, &misc_list);
 out:
 up(&misc_sem);
 return err;
}
```

3.3 misc_deregister函数

```
/**
 * misc_deregister - unregister a miscellaneous device                <span style="font-family: monospace;
 * @misc: device to unregister
 *
 * Unregister a miscellaneous device that was previously
 * successfully registered with misc_register(). Success
 * is indicated by a zero return, a negative errno code
 * indicates an error.
 */

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

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

 down(&misc_sem);
 list_del(&misc->list);
*/
/**
 * 当拨除设备时，使用下面的class_simple_device_remove函数删除类入口。
 */
 class_simple_device_remove(MKDEV(MISC_MAJOR, misc->minor));
 devfs_remove(misc->devfs_name);
 if (i < DYNAMIC_MINORS && i>0) {
  misc_minors[i>>3] &= ~(1 << (misc->minor & 7));
 }
 up(&misc_sem);
 return 0;
}

EXPORT_SYMBOL(misc_register);
EXPORT_SYMBOL(misc_deregister);
```

参考文献：

《Linux设备驱动开发详解 》
《Linux设备驱动程序》