183_驱动_杂项驱动misc.c文件解析

原创

HanLongXia 于 2021-11-16 16:57:19 发布 689 收藏

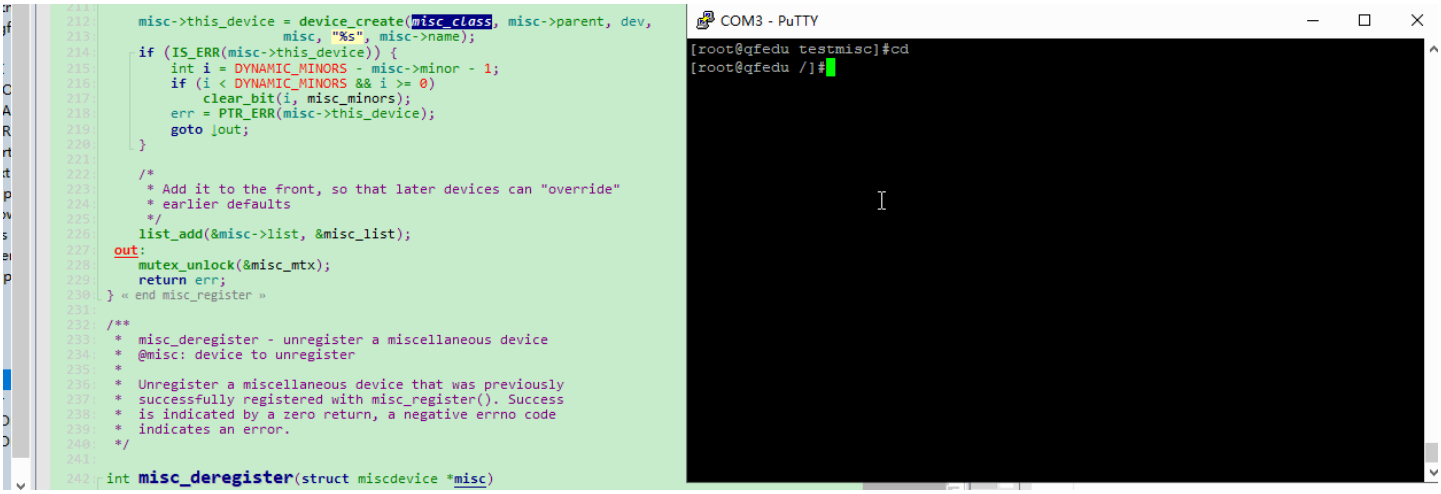
分类专栏： 驱动 物联网 文章标签： c语言 开发语言 后端

本文链接：https://blog.csdn.net/HanLongXia/article/details/121359753

class类文件下的misc文件集数据路径

/sys/class/misc/testmisc

动图：




```
[root@qfedu testmisc]#cat uevent
MAJOR=10 ：主设备号，固定
MINOR=41：次设备号，分配
DEVNAME=testmisc：我们创建的文件名
[root@qfedu testmisc]#
```

```
/*
 * linux/drivers/char/misc.c
 *
 * Generic misc open routine by Johan Myreen
 *
 * Based on code from Linus
 *
 * Teemu Rantanen's Microsoft Busmouse support and Derrick Cole's
```

```
 *   changes incorporated into 0.97pl4
 *   by Peter Cervasio (pete%q106fm.uucp@wupost.wustl.edu) (08SEP92)
 *   See busmouse.c for particulars.
 *
 * Made things a lot mode modular - easy to compile in just one or two
 * of the misc drivers, as they are now completely independent. Linus.
 *
 * Support for loadable modules. 8-Sep-95 Philip Blundell <pjb27@cam.ac.uk>
 *
 * Fixed a failing symbol register to free the device registration
 *  Alan Cox <alan@lxorguk.ukuu.org.uk> 21-Jan-96
 *
 * Dynamic minors and /proc/mice by Alessandro Rubini. 26-Mar-96
 *
 * Renamed to misc and miscdevice to be more accurate. Alan Cox 26-Mar-96
 *
 * Handling of mouse minor numbers for kerneld:
 *  Idea by Jacques Gelinas <jack@solucorp.qc.ca>,
 *  adapted by Bjorn Ekwall <bj0rn@blox.se>
 *  corrected by Alan Cox <alan@lxorguk.ukuu.org.uk>
 *
 * Changes for kmod (from kerneld):
 * Cyrus Durgin <cider@speakeasy.org>
 *
 * Added devfs support. Richard Gooch <rgooch@atnf.csiro.au>  10-Jan-1998
 */

#include <linux/module.h>

#include <linux/fs.h>
#include <linux/errno.h>
#include <linux/miscdevice.h>
#include <linux/kernel.h>
#include <linux/major.h>
#include <linux/mutex.h>
#include <linux/proc_fs.h>
#include <linux/seq_file.h>
#include <linux/stat.h>
#include <linux/init.h>
#include <linux/device.h>
#include <linux/tty.h>
#include <linux/kmod.h>
#include <linux/gfp.h>

/*
 * Head entry for the doubly linked miscdevice list
 */
static LIST_HEAD(misc_list);
static DEFINE_MUTEX(misc_mtx);

/*
 * Assigned numbers, used for dynamic minors
 */
#define DYNAMIC_MINORS 64 /* like dynamic majors */
static DECLARE_BITMAP(misc_minors, DYNAMIC_MINORS);

#ifdef CONFIG_PROC_FS
static void *misc_seq_start(struct seq_file *seq, loff_t *pos)
{
 mutex_lock(&misc_mtx);
```

```
 return seq_list_start(&misc_list, *pos);
}

static void *misc_seq_next(struct seq_file *seq, void *v, loff_t *pos)
{
 return seq_list_next(v, &misc_list, pos);
}

static void misc_seq_stop(struct seq_file *seq, void *v)
{
 mutex_unlock(&misc_mtx);
}

static int misc_seq_show(struct seq_file *seq, void *v)
{
 const struct miscdevice *p = list_entry(v, struct miscdevice, list);

 seq_printf(seq, "%3i %s\n", p->minor, p->name ? p->name : "");
 return 0;
}


static const struct seq_operations misc_seq_ops = {
 .start = misc_seq_start,
 .next  = misc_seq_next,
 .stop  = misc_seq_stop,
 .show  = misc_seq_show,
};

static int misc_seq_open(struct inode *inode, struct file *file)
{
 return seq_open(file, &misc_seq_ops);
}

static const struct file_operations misc_proc_fops = {
 .owner   = THIS_MODULE,
 .open    = misc_seq_open,
 .read    = seq_read,
 .llseek  = seq_lseek,
 .release = seq_release,
};
#endif

static int misc_open(struct inode * inode, struct file * file)
{//当应用层打开杂项设备文件，用系统调用Open，驱动层就会进入到这里


//inode携带了，当前杂项设备驱动struct cdev  *i_cdev;

 int minor = iminor(inode);//获得当前设备文件的次设备号 41
 struct miscdevice *c;
 int err = -ENODEV;
 const struct file_operations *old_fops, *new_fops = NULL;

 mutex_lock(&misc_mtx);


 //遍历双向链表
 list_for_each_entry(c, &misc_list, list) {
```

```
  if (c->minor == minor) {

    //如果CPU进入到这里，那么c 里面存就是当前正打开的设备文件的  杂项结构体
   new_fops = fops_get(c->fops);//获得目标结构体的fops
   break;
  }
 }

 if (!new_fops) {
  mutex_unlock(&misc_mtx);
  request_module("char-major-%d-%d", MISC_MAJOR, minor);
  mutex_lock(&misc_mtx);

  list_for_each_entry(c, &misc_list, list) {
   if (c->minor == minor) {
    new_fops = fops_get(c->fops);
    break;
   }
  }
  if (!new_fops)
   goto fail;
 }

 err = 0;
 old_fops = file->f_op;//把原来的f_op 跟换
 file->f_op = new_fops;//当CPU执行这句话，那么就真正把 当前设备文件的f_ops更换成，我们自己注册那个f_ops



 //还要调用我们自己定义fops里面的那个open
 if (file->f_op->open) {
  file->private_data = c;
  err=file->f_op->open(inode,file);
  if (err) {
   fops_put(file->f_op);
   file->f_op = fops_get(old_fops);
  }
 }


 fops_put(old_fops);
fail:
 mutex_unlock(&misc_mtx);
 return err;
}

static struct class *misc_class;


//当杂项设备设备文件没有被应用层的open打开，当前杂项设备的fileopts确实是misc_fops
//但是一旦应用层调用open
static const struct file_operations misc_fops = {
 .owner  = THIS_MODULE,
 .open  = misc_open,
 .llseek  = noop_llseek,
};

/**
 * 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.
 *
 * 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 = 0;

 INIT_LIST_HEAD(&misc->list);

 mutex_lock(&misc_mtx);//互斥锁


 //for循环去遍历，由miscdevice 沟通双向链表，这条链表就是当前杂项这一类驱动的所有的驱动集合，每一个驱动占据
 //一个结构体，遍历目的是为了判断当前传进来这个 杂项结构体的次设备号有没有在当前的注册驱动里面有相同的次设备
 list_for_each_entry(c, &misc_list, list) {
  if (c->minor == misc->minor) {
   mutex_unlock(&misc_mtx);
   return -EBUSY;//忙错误退出，不把当前的杂项设备结构体注册到链表
  }
 }

 //如果能够正常退出，说明 当前的杂项设备结构体，要么次设备号为255 （动态分配），或者指定的这个次设备号
 //与当前的链表里，任何一个结构体的次设备号都不相同

 if (misc->minor == MISC_DYNAMIC_MINOR) {
  //动态分配
  int i = find_first_zero_bit(misc_minors, DYNAMIC_MINORS);
  if (i >= DYNAMIC_MINORS) {
   mutex_unlock(&misc_mtx);
   return -EBUSY;
  }
  misc->minor = DYNAMIC_MINORS - i - 1;//小于64，次设备号，主设备内部肯定是10
  set_bit(i, misc_minors);
 }

 dev = MKDEV(MISC_MAJOR, misc->minor);//把申请到的次设备号和主设备

    //创建设备文件，第一步 注册主设备号，和创建类工作，内核里面已经帮我做了
    //系统帮我注册10作为主设备号，帮我创建一个类“misc”
    //我们只需要，创建一个设备文件
 misc->this_device = device_create(misc_class, misc->parent, dev,
       misc, "%s", misc->name);//"%s", misc->name变参，我们的name字符串，就是设备文件名字
 if (IS_ERR(misc->this_device)) {
  int i = DYNAMIC_MINORS - misc->minor - 1;
  if (i < DYNAMIC_MINORS && i >= 0)
   clear_bit(i, misc_minors);
  err = PTR_ERR(misc->this_device);
```

```
  goto out;
 }

 /*
  * Add it to the front, so that later devices can "override"
  * earlier defaults
  */

 //把当前这个杂项设备结构体添加到双向链表里，表面已经注册
 list_add(&misc->list, &misc_list);
 out:
 mutex_unlock(&misc_mtx);
 return err;
}

/**
 * misc_deregister - unregister a miscellaneous device
 * @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 = DYNAMIC_MINORS - misc->minor - 1;

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

 mutex_lock(&misc_mtx);
 list_del(&misc->list);
 device_destroy(misc_class, MKDEV(MISC_MAJOR, misc->minor));
 if (i < DYNAMIC_MINORS && i >= 0)
  clear_bit(i, misc_minors);
 mutex_unlock(&misc_mtx);
 return 0;
}

EXPORT_SYMBOL(misc_register);
EXPORT_SYMBOL(misc_deregister);

static char *misc_devnode(struct device *dev, umode_t *mode)
{
 struct miscdevice *c = dev_get_drvdata(dev);

 if (mode && c->mode)
  *mode = c->mode;
 if (c->nodename)
  return kstrdup(c->nodename, GFP_KERNEL);
 return NULL;
}

static int __init misc_init(void)
{
 int err;

#ifdef CONFIG_PROC_FS
```

```
#ifdef CONFIG_PROC_FS
 proc_create("misc", 0, NULL, &misc_proc_fops);
#endif
 misc_class = class_create(THIS_MODULE, "misc");
 err = PTR_ERR(misc_class);
 if (IS_ERR(misc_class))
  goto fail_remove;


 err = -EIO;
 //&misc_fops: 杂项驱动的自己的文件操作集
 //-lgq 固定主设备号10    、文件名"misc"
 if (register_chrdev(MISC_MAJOR,"misc",&misc_fops))//杂项的misc_fops 有自己的文件操作集
  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;
}
subsys_initcall(misc_init);
```