

实验 17 静态申请字符类设备号

17.1 本章导读

这里开始介绍的是纯粹的字符设备，前面学习的是杂项设备，主设备号已经固定为 10，。

但是考虑到大家学习之后，也会自己申请主设备号以及次设备号，就是标准的字符设备，所以从这一个实验开始给大家介绍相关的知识。

字符设备分为静态申请和动态申请，静态申请就是主设备号是程序员手动分配了，动态申请是系统给分配，本实验先介绍静态申请的方法。

17.1.1 工具

17.1.1.1 硬件工具

- 1) iTOP4412 开发板
- 2) U 盘或者 TF 卡
- 3) PC 机
- 4) 串口

17.1.1.2 软件工具

- 1) 虚拟机 Vmware
- 2) Ubuntu12.04.2
- 3) 超级终端 (串口助手)

4) 源码文件夹 “request_cdev_num”

17.1.2 预备课程

实验 16 驱动模块传参数

17.1.3 视频资源

本节配套视频为 “视频 17_静态申请字符类设备号”

17.2 学习目标

本章需要学习以下内容：

静态申请字符类设备号

进一步理解主设备号和次设备号

17.3 字符设备基本知识

前面已经提到过很多次，Linux 的设备主要分为三大类，字符设备、块设备、网络设备。

前面带大家写的驱动是杂项设备，它和字符设备唯一的区别就是主设备号已经搞定了，不需要像字符设备那样去手动申请。

这里先给大家介绍几个常用的申请字符类设备的函数。

如下图所示，在头文件 “include/linux/fs.h” 中，可以找到三个注册字符设备的函数。这三个分别是函数 `register_chrdev_region`、函数 `alloc_chrdev_region`、函数 `register_chrdev()`。

```
root@ubuntu: /home/topeet/android4.0/iTop4412_Kernel_3.0
}
static inline void bd_unlink_disk_holder(struct block_device *bdev,
                                         struct gendisk *disk)
{
}
#endif
#endif

/* fs/char_dev.c */
#define CHRDEV_MAJOR_HASH_SIZE 255
extern int alloc_chrdev_region(dev_t *, unsigned, unsigned, const char *);
extern int register_chrdev_region(dev_t, unsigned, const char *);
extern int __register_chrdev(unsigned int major, unsigned int baseminor,
                           unsigned int count, const char *name,
                           const struct file_operations *fops);
extern void __unregister_chrdev(unsigned int major, unsigned int baseminor,
                              unsigned int count, const char *name);
extern void unregister_chrdev_region(dev_t, unsigned);
extern void chrdev_show(struct seq_file *, off_t);

static inline int register_chrdev(unsigned int major, const char *name,
                                  const struct file_operations *fops)
{
}
2102,12      80%
```

如上图所示，三个函数的区别如下。

函数 `register_chrdev_region()` 是提前知道设备的主次设备号,再去申请设备号。

函数 `alloc_chrdev_region()` 是动态分配主次设备号。

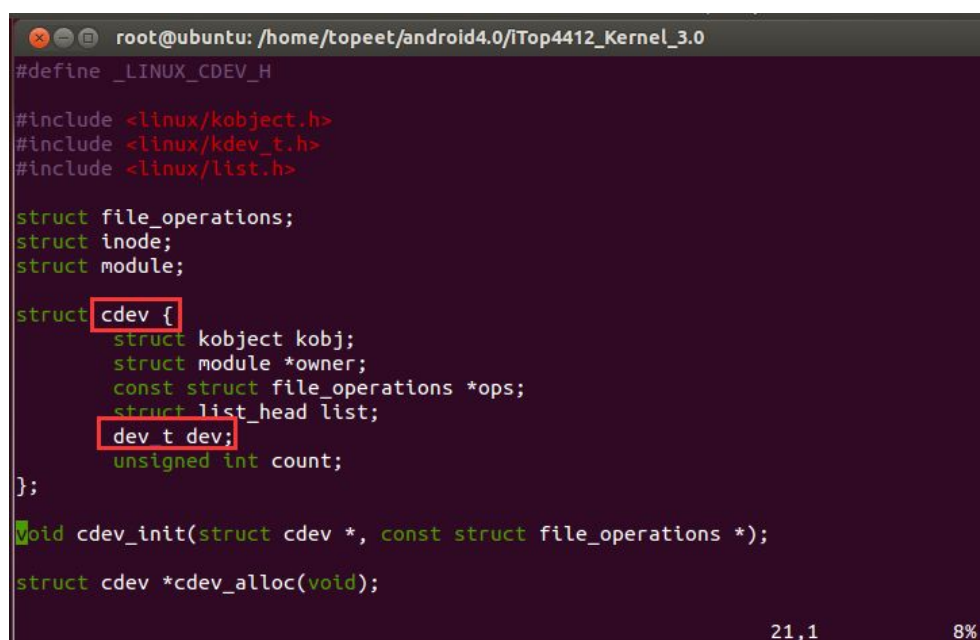
函数 `register_chrdev()`。是老版本的设备号注册方式,只分配主设备号。从设备号在 `mknod` 的时候指定。这个函数现在虽然仍然可以支持，但是已经不再使用了。

本节实验主要介绍的是动态申请函数 `register_chrdev_region()`。

在函数 `extern int register_chrdev_region(dev_t, unsigned, const char *)` 中，它的参数 `dev_t` 相关的知识会慢慢的介绍到，提到的部分大家需要理解并应用，而且 Linux 系统中还有几个函数和参数是专门为它服务的。

在头文件 “`include/linux/cdev.h`” 中，如下图所示。

在 `cdev` 中有专门一个参数 `dev_t dev`。



```
root@ubuntu: /home/topeet/android4.0/iTop4412_Kernel_3.0
#define _LINUX_CDEV_H

#include <linux/kobject.h>
#include <linux/kdev_t.h>
#include <linux/list.h>

struct file_operations;
struct inode;
struct module;

struct cdev {
    struct kobject kobj;
    struct module *owner;
    const struct file_operations *ops;
    struct list_head list;
    dev_t dev;
    unsigned int count;
};

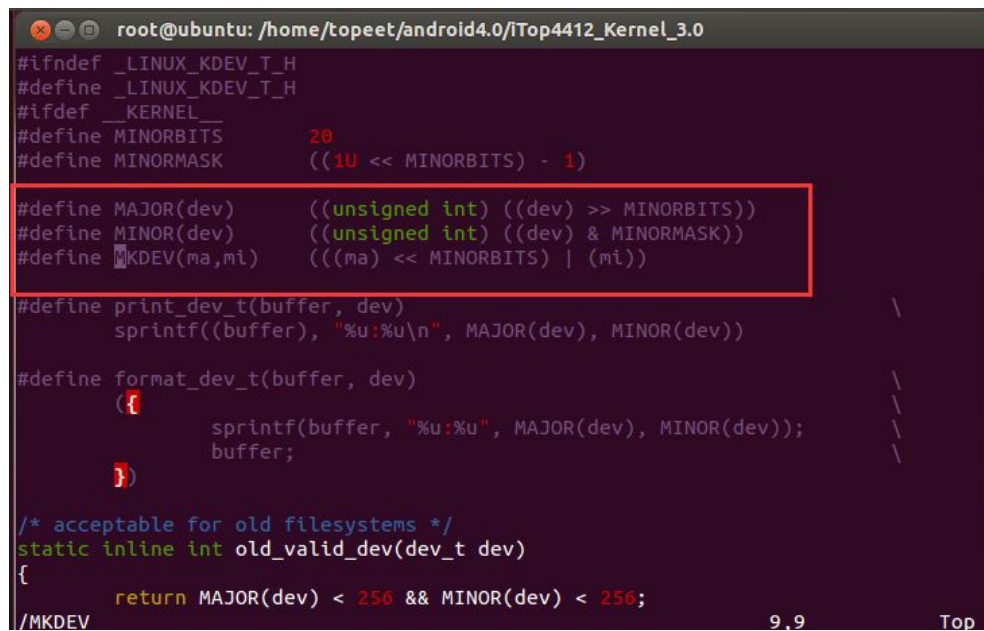
void cdev_init(struct cdev *, const struct file_operations *);
struct cdev *cdev_alloc(void);
```

21,1 8%

如上图所示，cdev 类型是字符设备描述的结构，其中的设备号必须用“dev_t”类型来描述，高 12 位为主设备号，低 20 位为次设备号。

把 dev 理解为二进制数，就很容易理解了，dev_t 是一个 32 位类型的数，前 12 位表示主设备，后 20 位表示次设备号。

如下图所示，在头文件“include/linux/kdev_t.h”中，有一些专门用来处理 dev_t 数据类型的宏定义。



```
root@ubuntu: /home/topeet/android4.0/iTop4412_Kernel_3.0
#ifndef _LINUX_KDEV_T_H
#define _LINUX_KDEV_T_H
#ifndef __KERNEL__
#define MINORBITS      20
#define MINORMASK      ((1U << MINORBITS) - 1)
#define MAJOR(dev)      ((unsigned int) ((dev) >> MINORBITS))
#define MINOR(dev)      ((unsigned int) ((dev) & MINORMASK))
#define MKDEV(ma,mi)     (((ma) << MINORBITS) | (mi))
#define print_dev_t(buffer, dev) \
    sprintf((buffer), "%u:%u\n", MAJOR(dev), MINOR(dev))
#define format_dev_t(buffer, dev) \
    ({ \
        sprintf(buffer, "%u:%u", MAJOR(dev), MINOR(dev)); \
        buffer; \
    })
/* acceptable for old filesystems */
static inline int old_valid_dev(dev_t dev)
{
    return MAJOR(dev) < 256 && MINOR(dev) < 256;
}
/MKDEV 9,9 Top
```

如上图所以，三个函数比较容易理解。

MAJOR(dev)，就是对 dev 操作，提取高 12 位主设备号；

MINOR(dev)，就是对 dev 操作，提取低 20 位数次设备号；

MKDEV(ma,mi)，就是对主设备号和低设备号操作，合并为 dev 类型。

17.4 实验操作

在视频教程“16_驱动模块传参数”的基础上做这个实验。

先修改一下 Makefile 文件，如下图所示，将 module_param 改为 request_cdev_num。

```
#!/bin/bash
#通知编译器我们要编译模块的哪些源码
#这里是编译itop4412_hello.c这个文件编译成中间文件mini_linux_module.o
obj-m += request_cdev_num.o

#源码目录变量，这里用户需要根据实际情况选择路径
#作者是将Linux的源码拷贝到目录/home/topeet/android4.0下并解压的
KDIR := /home/topeet/android4.0/iTop4412_Kernel_3.0

#当前目录变量
PWD ?= $(shell pwd)

#make命名默认寻找第一个目标
#make -C就是指调用执行的路径
#$(KDIR) Linux源码目录，作者这里指的是/home/topeet/android4.0/iTop4412_Kernel_3.0
#$(PWD) 当前目录变量
#modules要执行的操作
all:
    make -C $(KDIR) M=$(PWD) modules

#make clean执行的操作是删除后缀为o的文件
clean:
    rm -rf *.mod.c *.o *.order *.ko *.mod.o *.symvers
```

将视频教程“16_驱动模块传参数”中的文件“module_param.c 改写为

“request_cdev_num.c”。

如下图所示，先调用头文件，然后将主设备号和设备号通过模块参数传入，定义此设备号数。

```
/*包含初始化加载模块的头文件,代码中的MODULE_LICENSE在此头文件中*/

/*定义module_param module_param_array的头文件*/
#include <linux/moduleparam.h>
/*定义module_param module_param_array中perm的头文件*/
#include <linux/stat.h>

/*三个字符设备函数*/
#include <linux/fs.h>
/*MKDEV转换设备号数据类型的宏定义*/
#include <linux/kdev_t.h>
/*定义字符设备的结构体*/
#include <linux/cdev.h>

#define DEVICE_NAME "sscdev"
#define DEVICE_MINOR_NUM 2
#define DEV_MAJOR 0
#define DEV_MINOR 0

MODULE_LICENSE("Dual BSD/GPL");
/*声明是开源的,没有内核版本限制*/
MODULE_AUTHOR("iTOPEET_dz");
/*声明作者*/

int numdev_major = DEV_MAJOR;
int numdev_minor = DEV_MINOR;

/*输入主设备号*/
module_param(numdev_major,int,S_IRUSR);
/*输入次设备号*/
module_param(numdev_minor,int,S_IRUSR);
```

接着将入口函数和出口函数名称修改一下，“hello_init”和“hello_exit”改为“scdev_init”和“scdev_exit”。

```
module_init(scdev_init);
/*初始化函数*/
module_exit(scdev_exit);
/*卸载函数*/
```

如下图所示，在入口和出口函数中调用函数 register_chrdev_region 和函数 unregister_chrdev_region。


```
static int scdev_init(void)
{
    int ret = 0;
    dev_t num_dev;

    printk(KERN_EMERG "numdev_major is %d!\n", numdev_major);
    printk(KERN_EMERG "numdev_minor is %d!\n", numdev_minor);

    if(numdev_major){
        num_dev = MKDEV(numdev_major, numdev_minor);
        ret = register_chrdev_region(num_dev, DEVICE_MINOR_NUM, DEVICE_NAME);
    }
    else{
        printk(KERN_EMERG "numdev_major %d is failed!\n", numdev_major);
    }

    if(ret < 0){
        printk(KERN_EMERG "register_chrdev_region req %d is failed!\n", numdev_major);
    }

    printk(KERN_EMERG "scdev_init!\n");
    /*打印信息, KERN_EMERG表示紧急信息*/
    return 0;
}

static void scdev_exit(void)
{
    printk(KERN_EMERG "scdev exit!\n");
    unregister_chrdev_region(MKDEV(numdev_major, numdev_minor), DEVICE_MINOR_NUM);
}
```

如上图所示，先将主设备号和次设备号默认为 0，然后做一个简单的判断。如果没有参数传入，默认为零，就会提示注册失败；如果参数传入的话，和已有的主次设备号有重复，也会失败。

在 Ubuntu 系统下新建 request_cdev_num 文件夹，将写好的 request_cdev_num.c、编译脚本拷贝到 request_cdev_num 文件夹下，如下图所示。


```
root@ubuntu: /home/topeet/request_cdev_num
root@ubuntu:/home/topeet# mkdir request_cdev_num
root@ubuntu:/home/topeet# cd request_cdev_num/
root@ubuntu:/home/topeet/request_cdev_num# ls
Makefile request_cdev_num.c
root@ubuntu:/home/topeet/request_cdev_num#
```

使用 Makefile 命令编译驱动命令 “Make” 编译应用，如下图所示。

```
root@ubuntu: /home/topeet/request_cdev_num
root@ubuntu:/home/topeet# mkdir request_cdev_num
root@ubuntu:/home/topeet# cd request_cdev_num/
root@ubuntu:/home/topeet/request_cdev_num# ls
Makefile request_cdev_num.c
root@ubuntu:/home/topeet/request_cdev_num# make
make -C /home/topeet/android4.0/iTop4412_Kernel_3.0 M=/home/topeet/request_cdev_num modules
make[1]: Entering directory `/home/topeet/android4.0/iTop4412_Kernel_3.0'
  CC [M] /home/topeet/request_cdev_num/request_cdev_num.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC      /home/topeet/request_cdev_num/request_cdev_num.mod.o
  LD [M] /home/topeet/request_cdev_num/request_cdev_num.ko
make[1]: Leaving directory `/home/topeet/android4.0/iTop4412_Kernel_3.0'
root@ubuntu:/home/topeet/request_cdev_num# ls
Makefile request_cdev_num.c request_cdev_num.mod.o
modules.order request_cdev_num.ko request_cdev_num.o
Module.symvers request_cdev_num.mod.c
root@ubuntu:/home/topeet/request_cdev_num#
```

将上图中的文件 request_cdev_num.ko 拷贝到 U 盘。

启动开发板，将 U 盘插入开发板，使用命令 “mount /dev/sda1 /mnt/udisk/” 加载 U 盘，如下图所示。

```
[root@iTOP-4412]# mount /dev/sda1 /mnt/udisk/
[root@iTOP-4412]#
```

使用命令“cat /proc/devices”查看已经被注册的主设备号，发现设备号 9 没有被注册，也可以使用其它没有被使用的主设备号，如下图所示。


```
[root@iTOP-4412]# cat /proc/devices
Character devices:
1 mem
4 ttyS
5 /dev/tty
5 /dev/console
5 /dev/ptmx
10 misc
13 input
21 sg
29 fb
81 video4linux
89 i2c
108 ppp
116 alsa
128 ptm
136 pts
153 rc522_test
166 ttyACM
180 usb
188 ttyUSB
189 usb_device
204 ttySAC
216 rfcomm
250 roccat
251 BaseRemoteCtl
252 media
253 ttyGS
254 rtc
```

这里使用设备号 9，使用加载模块的命令 “insmod /mnt/udisk/request_cdev_num.ko numdev_major=9 numdev_minor=0” 加载驱动 request_cdev_num.ko，如下图所示，加载成功。

```
130 sd
131 sd
132 sd
133 sd
134 sd
135 sd
179 mmc
254 device-mapper
[root@iTOP-4412]# insmod /mnt/udisk/request_cdev_num.ko numdev_major=9 numdev_min
nor=0
[ 227.626029] numdev_major is 9!
[ 227.627628] numdev_minor is 0!
[ 227.630723] scdev_init!
```

加载之后可以再次使用命令 “cat /proc/devices” 查看，如下图所示，主设备号 9 已经被驱动所占用。

```
[root@iTOP-4412]# cat /proc/devices
Character devices:
 1 mem
 4 ttyS
 5 /dev/tty
 5 /dev/console
 5 /dev/ptmx
 9 sscdev
10 misc
13 input
21 sg
29 fb
81 video4linux
89 i2c
108 ppp
116 alsa
128 ptm
136 pts
153 rc522_test
166 ttyACM
180 usb
188 ttyUSB
189 usb device
```



接着如下图所示，使用命令“rmmod request_cdev_num.ko”卸载模块，如下图所示。

```
COM1
[root@iTOP-4412]#
[root@iTOP-4412]# lsmod
request_cdev_num 1429 0 - Live 0xbf000000
[root@iTOP-4412]# rmmod request_cdev_num
[ 379.431137] sscdev_exit!
[root@iTOP-4412]#
```

然后再使用命令“cat /proc/devices”查看主设备号，可以看到设备号 9 又处于空闲状态了。

```
request_cdev_num 1429 0 - Live 0xbf000000
[root@iTOP-4412]# rmmod request_cdev_num
[ 379.431137] scdev_exit!
[root@iTOP-4412]# cat /proc/devices
Character devices:
1 mem
4 ttyS
5 /dev/tty
5 /dev/console
5 /dev/ptmx
10 misc
13 input
21 sg
29 fb
81 video4linux
89 i2c
```