

实验 19 注册字符类设备

19.1 本章导读

前面几期详细介绍了如何申请设备号，这一期实验介绍如何注册字符类设备。

前面介绍的注册杂项设备，在这一步的处理非常简单，只要在平台文件中添加一个结构体和一个指针调用即可。字符设备和前面的杂项设备唯一的区别就是多了这一步注册设备，难度并不大。

19.1.1 工具

19.1.1.1 硬件工具

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

19.1.1.2 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 “register_cdev”

19.1.2 预备课程

实验 17 静态申请字符类设备号

实验 18 动态申请字符类设备号

19.1.3 视频资源

本节配套视频为“视频 19_注册字符类设备”

19.2 学习目标

本章需要学习以下内容：

给设备分配内存空间

注册字符类设备的整个过程

19.3 分配内存空间

前面介绍的杂项设备并没有分配内存空间这个过程，是因为系统自带的代码已经给杂项设备分配好了。

Linux 中注册字符类设备需要首先申请内存空间，有一个专门分配小内存空间的函数 `kmalloc`，这个函数在头文件“`include/linux/slab.h`”中，如下图所示，使用命令“`vim include/linux/slab.h`”打开头文件。

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

#endif /* CONFIG_NUMA */

/*
 * Shortcuts
 */
static inline void *kmem_cache_zalloc(struct kmem_cache *k, gfp_t flags)
{
    return kmem_cache_alloc(k, flags | __GFP_ZERO);
}

/**
 * kcalloc - allocate memory. The memory is set to zero.
 * @size: how many bytes of memory are required.
 * @flags: the type of memory to allocate (see kmalloc).
 */
static inline void *kcalloc(size_t size, gfp_t flags)
{
    return kmalloc(size, flags | __GFP_ZERO);
}

/**
 * kcalloc_node - allocate zeroed memory from a particular memory node.
```

如上图所示，函数 `static inline void *kcalloc(size_t size, gfp_t flags)` 有两个参数，

参数 `size_t size`：申请的内存大小(最大 128K)，

参数 `gfp_t flags`：常用参数 `GFP_KERNEL`，代表优先权，内存不够可以延迟分配。

和申请内存的函数 `kcalloc` 对应的是 `kfree` 释放函数，如下图所示。

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

#define KMALLOC_SHIFT_HIGH  ((MAX_ORDER + PAGE_SHIFT - 1) <= 25 ? \
                             (MAX_ORDER + PAGE_SHIFT - 1) : 25)

#define KMALLOC_MAX_SIZE    (1UL << KMALLOC_SHIFT_HIGH)
#define KMALLOC_MAX_ORDER   (KMALLOC_SHIFT_HIGH - PAGE_SHIFT)

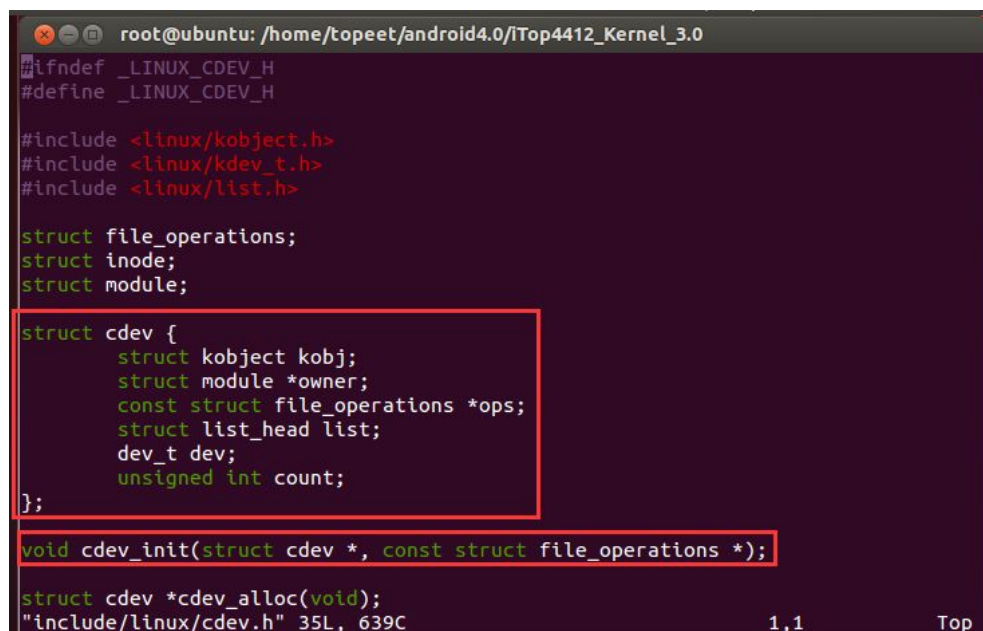
/*
 * Common kmalloc functions provided by all allocators
 */
void * __must_check __krealloc(const void *, size_t, gfp_t);
void * __must_check krealloc(const void *, size_t, gfp_t);
void kfree(const void *);
void kzfree(const void *);
size_t ksize(const void *);

/*
 * Allocator specific definitions. These are mainly used to establish optimized
 * ways to convert kmalloc() calls to kmem_cache_alloc() invocations by
 * selecting the appropriate general cache at compile time.
 *
 * Allocators must define at least:
 *
 *     kmem_cache_alloc()
```

void kfree(const void *)函数只有一个参数，就是内存的指针，这个指针由申请内存的函数 kzalloc 返回。

19.4 注册字符类设备的函数

注册字符类设备的初始化函数为 cdev_init，这个函数在头文件 “include/linux/cdev.h” 中，使用命令 “vim include/linux/cdev.h” 打开这个头文件如下图所示。



```
root@ubuntu: /home/topeet/android4.0/iTop4412_Kernel_3.0
#ifndef _LINUX_CDEV_H
#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);
"include/linux/cdev.h" 35L, 639C                                1,1                                Top
```

如上图所示红框中的函数 “

void cdev_init(struct cdev *, const struct file_operations *)”

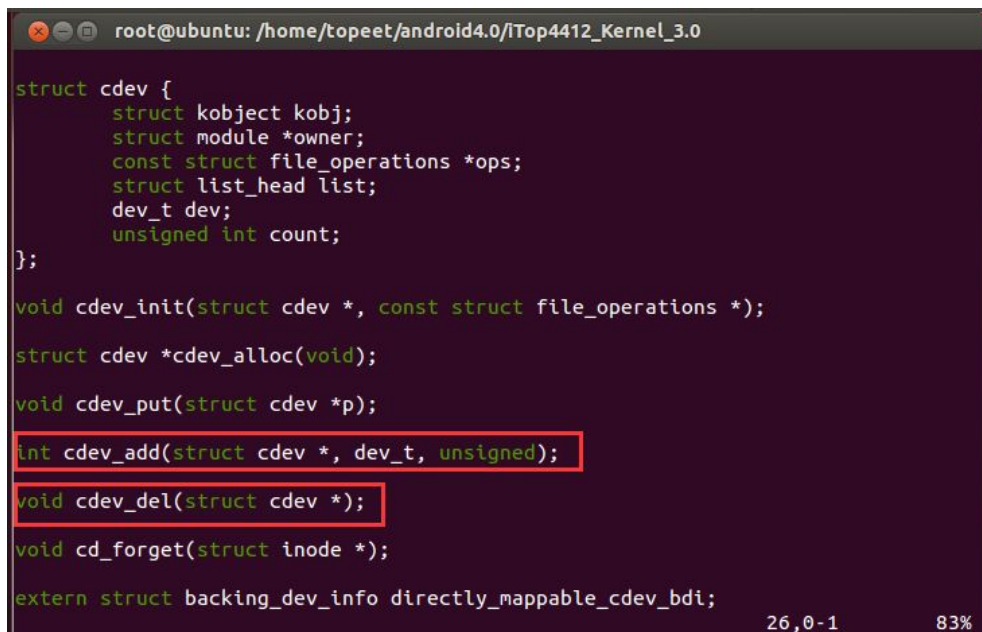
和结构体 “cdev” 。

cdev_init 函数有两个参数，

参数 struct cdev *：cdev 字符设备文件结构体

参数 `const struct file_operations *` : `file_operations` 结构体，这个结构体已经用过很多次了，就不再介绍了。

注册字符设备的函数为 `cdev_add`，这个函数也是在头文件 “`include/linux/cdev.h`” 中，如下图所示。



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

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);

void cdev_put(struct cdev *p);

int cdev_add(struct cdev *, dev_t, unsigned);

void cdev_del(struct cdev *);

void cd_forget(struct inode *);

extern struct backing_dev_info directly_mappable_cdev_bdi;
```

如上图所示，和注册驱动函数 `cdev_add` 对应的还有卸载驱动函数 `cdev_del`。

注册驱动的函数 `int cdev_add(struct cdev *, dev_t, unsigned)` 有三个参数：

参数 `struct cdev *` : `cdev` 字符设备文件结构体

参数 `dev_t` : 设备号 `dev`, 前面已经介绍和使用过了

参数 `unsigned` : 设备范围大小

卸载驱动的函数 `void cdev_del(struct cdev *)` 只有一个参数，`cdev` 字符设备结构体

19.5 实验操作

将“18_动态申请字符类设备号”中的文件“request_ascdev_num.c”改为“register_cdev.c”。

首先修改一下 Makefile 文件，如下图所示，将“request_ascdev_num”改为“register_cdev”。

```
#!/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 += register_cdev.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
```

如下图所示，添加申请内存空间函数的头文件“linux/slab.h”，然后定义内存空间大小为3000。


```
#include <linux/cdev.h>

/*分配内存空间函数头文件*/
#include <linux/slab.h>

#define DEVICE_NAME "ascdev"
#define DEVICE_MINOR_NUM 2
#define DEV_MAJOR 0
#define DEV_MINOR 0
#define REGDEV_SIZE 3000

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*/
```

接着将申请的内存数据控件打个包为结构体 reg_dev，因为是有两个子设备，所以接着定义一个结构体数组*my_devices。

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

struct reg_dev
{
    char *data;
    unsigned long size;

    struct cdev cdev;
};

struct reg_dev *my_devices;
```

然后申请内存空间，并将内存空间先清零，接着对设备进行初始化，如下图所示。

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

my_devices = kmalloc(DEVICE_MINOR_NUM * sizeof(struct reg_dev),GFP_KERNEL);
if(!my_devices){
    ret = -ENOMEM;
    goto fail;
}
memset(my_devices,0,DEVICE_MINOR_NUM * sizeof(struct reg_dev));

/*设备初始化*/
for(i=0;i<DEVICE_MINOR_NUM;i++){
    my_devices[i].data = kmalloc(REGDEV_SIZE,GFP_KERNEL);
    memset(my_devices[i].data,0,REGDEV_SIZE);
    /*设备注册到系统*/
    reg_init_cdev(&my_devices[i],i);
}
```

如上图所示，用到多次 memset 函数，第一次由于没有规定 my_devices[i].data 的大小，所以只是对默认大小的数据赋值为 0，在设备初始化的循环中，又重新对 my_devices[i].data 申请了 REGDEV_SIZE 大小的数据，所以需要重新赋值为 0。

然后如下图所示，是自定义的函数 reg_init_cdev，在这个函数中会将 my_devices 和编号传过去。


```
struct file_operations my_fops = {  
    .owner = THIS_MODULE,  
};  
  
/*设备注册到系统*/  
static void reg_init_cdev(struct reg_dev *dev,int index){  
    int err;  
    int devno = MKDEV(numdev_major,numdev_minor+index);  
  
    /*数据初始化*/  
    cdev_init(&dev->cdev,&my_fops);  
    dev->cdev.owner = THIS_MODULE;  
    dev->cdev.ops = &my_fops;  
  
    /*注册到系统*/  
    err = cdev_add(&dev->cdev,devno,1);  
    if(err){  
        printk(KERN_EMERG "cdev_add %d is fail! %d\n",index,err);  
    }  
    else{  
        printk(KERN_EMERG "cdev_add %d is success!\n",index);  
    }  
}
```

如上图所示，首先需要将设备号使用 MKDEV 转化成 dev_t 类型，然后初始化，接着给 dev->cdev 赋值。

这里只给 cdev 添加了两个参数 owner 和 ops，

参数 owner 默认为 THIS_MODULE，

参数 ops 这里只定义了其中的参数 owner。

接着添加一下 fail 部分，如下图所示。

```
printk(KERN_EMERG "scdev_init!\n");  
/*打印信息，KERN_EMERG表示紧急信息*/  
return 0;  
  
fail:  
/*注销设备号*/  
unregister_chrdev_region(MKDEV(numdev_major,numdev_minor),DEVICE_MINOR_NUM);  
printk(KERN_EMERG "kmallocc is fail!\n");  
  
return ret;  
}
```

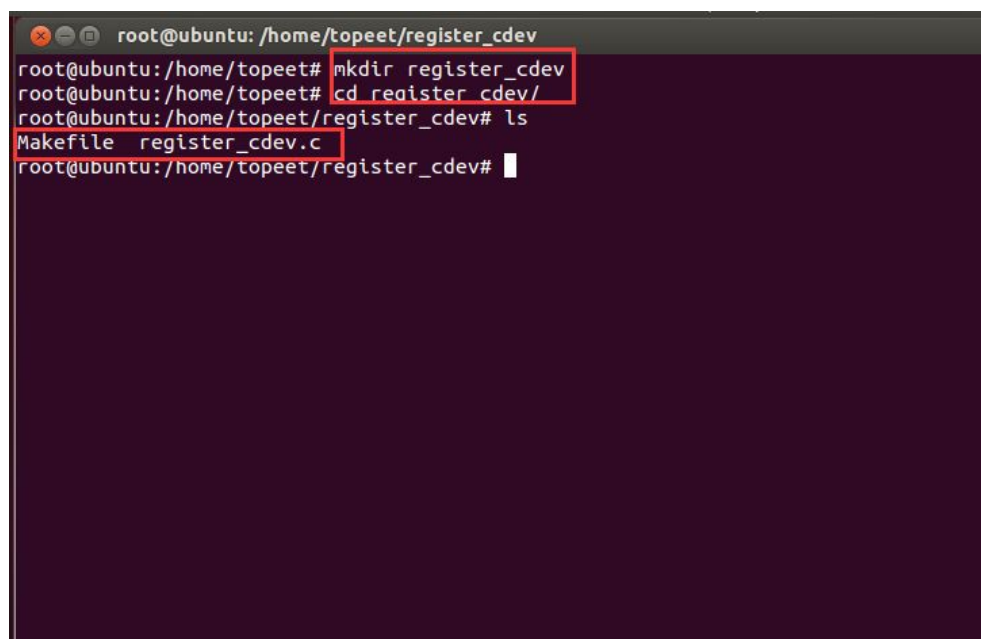
最后添加模块出口函数的代码，如下图所示。

```
static void scdev_exit(void)
{
    int i;
    printk(KERN_EMERG "scdev_exit!\n");

    /*除去字符设备*/
    for(i=0;i<DEVICE_MINOR_NUM;i++){
        cdev_del(&(my_devices[i].cdev));
    }

    unregister_chrdev_region(MKDEV(numdev_major,numdev_minor),DEVICE_MINOR_NUM);
}
```

修改完之后，在 Ubuntu 系统下使用命令 “mkdir register_cdev” 新建 “register_cdev” 文件夹，将写好的 register_cdev.c、编译脚本拷贝到 register_cdev 文件夹下，如下图所示。



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

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

将上图中的文件 register_cdev.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]#
[root@iTOP-4412]#
[root@iTOP-4412]#
[root@iTOP-4412]#
[root@iTOP-4412]#
[root@iTOP-4412]# mount /dev/sda1 /mnt/udisk/
```

如下图所示，使用命令 “insmod /mnt/udisk/register_cdev.ko” 加载驱动。

```
COM1
[root@iTOP-4412]# insmod /mnt/udisk/register_cdev.ko
[ 96.725769] numdev_major is 0!
[ 96.727381] numdev_minor is 0!
[ 96.749430] adev_region req 249 !
[ 96.751398] cdev_add 0 is success!
[ 96.754695] cdev_add 1 is success!
[ 96.800112] scdev_init!
[root@iTOP-4412]#
```

可以看到根据上图中的打印信息判断，设备注册已经成功。