

实验 20 生成字符类设备节点

20.1 本章导读

本实验介绍一下如何生成字符设备的设备节点，这部分和前面注册杂项设备类似，不过在调用生成设备节点的时候需要额外的添加一个设备类。

20.1.1 工具

20.1.1.1 硬件工具

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

20.1.1.2 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 “create_cnode”

20.1.2 预备课程

实验 19 注册字符类设备

20.1.3 视频资源

本节配套视频为“视频 20_生成字符类设备节点”

20.2 学习目标

本章需要学习以下内容：

了解设备类的概念

初始化代码中创建设备节点

手动创建设备节点

20.3 创建设备类

在前面介绍的设备中的模型,例如总线 bus、设备 device、驱动 driver 都是有明确的定义。。

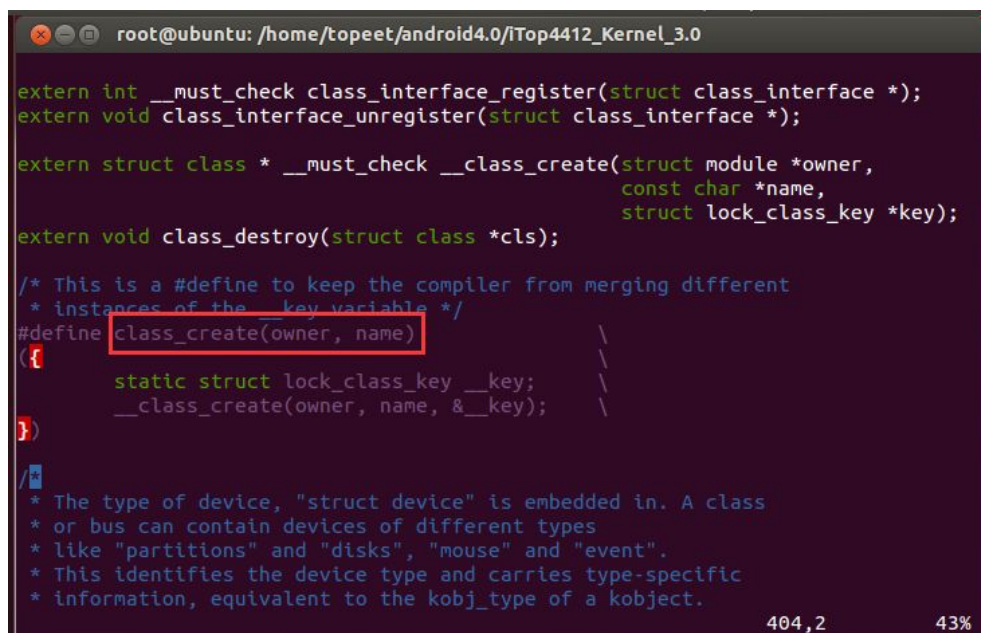
bus 代表总线，device 代表实际的设备和接口，driver 代表驱动。

Linux 中的 class 是设备类，它是一个抽象的概念，没有对应的实体。它是提供给用户接口相似的一类设备的集合。常见的有输入子系统 input、usb、串口 tty、块设备 block 等。

以 4412 的串口为例，它有四个串口，不可能为每一个串口都重复申请设备以及设备节点，因为它们有类似的地方，而且很多代码都是重复的地方，所以引入了一个抽象的类，将其打包为 ttySACX，在实际调用串口的时候，只需要修改 X 值，就可以调用不同的串口。

对于本实验中的设备，它有两个子设备，将对应两个设备节点，所以需要用到 class 类这样一个概念。

创建设备类的函数 `class_create` 在头文件 “`include/linux/device.h`” 中，使用命令 “`vim include/linux/device.h`” 打开头文件，如下图所示。



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

extern int __must_check class_interface_register(struct class_interface *);
extern void class_interface_unregister(struct class_interface *);

extern struct class * __must_check __class_create(struct module *owner,
                                                const char *name,
                                                struct lock_class_key *key);

extern void class_destroy(struct class *cls);

/* This is a #define to keep the compiler from merging different
 * instances of the __key variable */
#define class_create(owner, name) \
({ \
    static struct lock_class_key __key; \
    __class_create(owner, name, &__key); \
})

/*
 * The type of device, "struct device" is embedded in. A class
 * or bus can contain devices of different types
 * like "partitions" and "disks", "mouse" and "event".
 * This identifies the device type and carries type-specific
 * information, equivalent to the kobj_type of a kobject.
 */
```

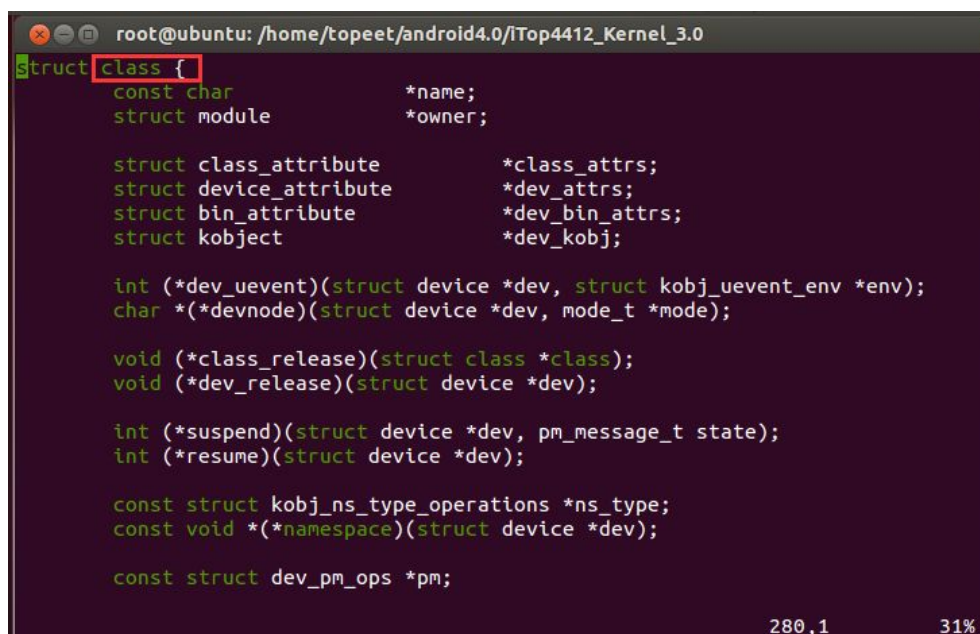
如上图所示，函数 `class_create(owner, name)` 只有两个参数

参数 `owner`：一般是 `THIS_MODULE`

参数 `name`：设备名称

调用函数 `class_create` 会返回一个 `class` 结构体变量

`class` 结构体变量在头文件 `include/linux/device.h` 的 280 行，如下图所示。



```
root@ubuntu: /home/topeet/android4.0/iTop4412_Kernel_3.0
struct class {
    const char          *name;
    struct module       *owner;

    struct class_attribute *class_attrs;
    struct device_attribute *dev_attrs;
    struct bin_attribute *dev_bin_attrs;
    struct kobject *dev_kobj;

    int (*dev_uevent)(struct device *dev, struct kobj_uevent_env *env);
    char *(*devnode)(struct device *dev, mode_t *mode);

    void (*class_release)(struct class *class);
    void (*dev_release)(struct device *dev);

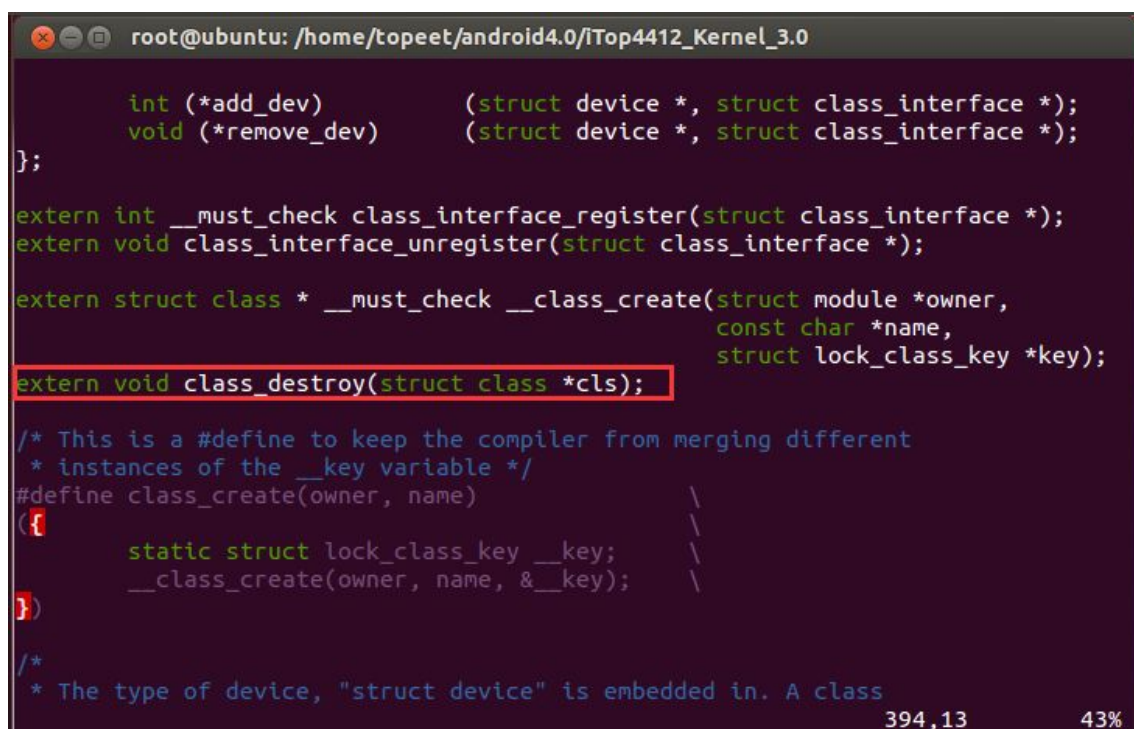
    int (*suspend)(struct device *dev, pm_message_t state);
    int (*resume)(struct device *dev);

    const struct kobj_ns_type_operations *ns_type;
    const void *(*namespace)(struct device *dev);

    const struct dev_pm_ops *pm;
}
```

如上图所示，看着复杂，其实不用管，在实际应用中就是给创建设备节点用的。在代码中，只需要定义一个 class 变量，然后在创建设备节点的时候作为一个参数赋值使用即可。

还有释放设备类 class 的函数 class_destroy，就只有一个参数 class。这个函数也是在头文件 “include/linux/device.h” 中，如下图所示。



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

    int (*add_dev)      (struct device *, struct class_interface *);
    void (*remove_dev)   (struct device *, struct class_interface *);
};

extern int __must_check class_interface_register(struct class_interface *);
extern void class_interface_unregister(struct class_interface *);

extern struct class * __must_check __class_create(struct module *owner,
                                                const char *name,
                                                struct lock_class_key *key);
extern void class_destroy(struct class *cls);

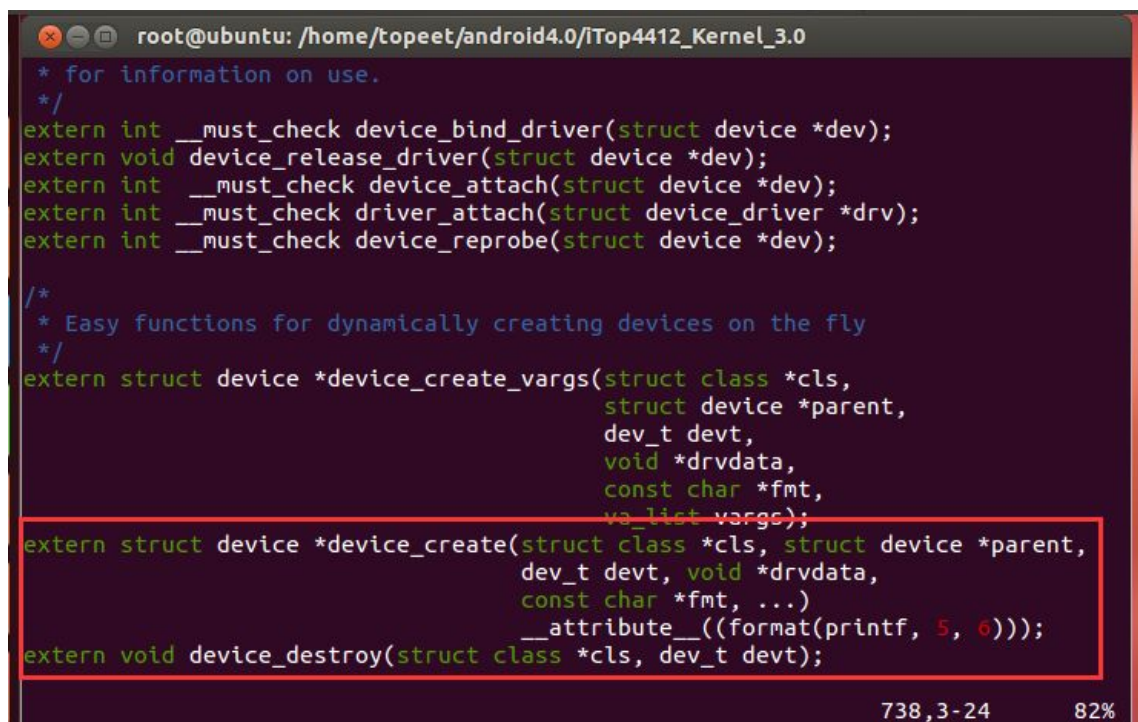
/* This is a #define to keep the compiler from merging different
 * instances of the __key variable */
#define class_create(owner, name) \
({ \
    static struct lock_class_key __key; \
    __class_create(owner, name, &__key); \
})

/*
 * The type of device, "struct device" is embedded in. A class
```

394,13 43%

20.4 创建字符设备节点

创建设备节点的函数 `device_create` 在头文件 `"include/linux/device.h"`，如下图所示。



```
root@ubuntu: /home/topeet/android4.0/iTop4412_Kernel_3.0
/* for information on use.
*/
extern int __must_check device_bind_driver(struct device *dev);
extern void device_release_driver(struct device *dev);
extern int __must_check device_attach(struct device *dev);
extern int __must_check driver_attach(struct device_driver *drv);
extern int __must_check device_reprobe(struct device *dev);

/*
 * Easy functions for dynamically creating devices on the fly
 */
extern struct device *device_create_vargs(struct class *cls,
                                         struct device *parent,
                                         dev_t devt,
                                         void *drvdata,
                                         const char *fmt,
                                         va_list vargs);
extern struct device *device_create(struct class *cls, struct device *parent,
                                   dev_t devt, void *drvdata,
                                   const char *fmt, ...)
    __attribute__((format(printf, 5, 6)));
extern void device_destroy(struct class *cls, dev_t devt);
```

函数 `extern struct device *device_create(struct class *cls, struct device *parent, dev_t devt, void *drvdata, const char *fmt, ...)` 中的参数比较多。

参数 `struct class *cls`：设备所属于的类，前面创建类的返回值

参数 `struct device *parent`：设备的父设备，NULL

参数 `dev_t devt`：设备号

参数 `void *drvdata`：设备数据，NULL

参数 `const char *fmt`：设备名称

如上图所示，还有一个摧毁设备节点的函数 `extern void device_destroy(struct class *cls, dev_t devt)`；只有两个参数，分别是设备类和设备号。

20.5 实验操作

将“19_注册字符类设备”中的“register_cdev.c”文件改为“create_cnode.c”，然后添加注册类以及创建设备节点的代码。

如下图所示，首先添加头文件“linux/device.h”，然后将设备名称改为 chardevnode。

```
/*定义字符设备的结构体*/
#include <linux/cdev.h>
/*分配内存空间函数头文件*/
#include <linux/slab.h>

/*包含函数device_create 结构体class等头文件*/
#include <linux/device.h>

#define DEVICE_NAME "chardevnode"
#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*/
```

然后定义一个设备类，如下图所示。

```

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);

static struct class *myclass;

struct reg_dev
{

```

如下图所示，在初始化函数中添加申请设备类以及创建设备节点的代码。

```

}
myclass = class_create(THIS_MODULE,DEVICE_NAME);

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);

    /*创建设备节点*/
    device_create(myclass,NULL,MKDEV(numdev_major,numdev_minor+i),NULL,DEVICE_NAME"%d",i);
}

```

最后在退出函数中添加释放设备以及释放内存的代码，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));
        /*摧毁设备节点函数*/
        device_destroy(myclass,MKDEV(numdev_major,numdev_minor+i));
    }
    /*释放设备class*/
    class_destroy(myclass);
    /*释放内存*/
    kfree(my_devices);

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

然后修改一下 Makefile 编译文件，如下图所示。

```
#!/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 += create_cnode.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
```

修改完成之后，在 Ubuntu 系统下使用命令 “mkdir create_cnode” 新建文件夹 “create_cnode”，然后将修改好的驱动文件 “create_cnode.c” 和 Makefile 文件拷贝到 “create_cnode” 中，如下图所示。

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

使用编译命令“make”编译驱动，如下图所示。

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

将生成的驱动模块“create_cnode.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/create_cnode.ko” 加载模块，如下图所示。

```
COM1
[root@iTOP-4412]# mount /dev/sda1 /mnt/udisk/
[root@iTOP-4412]# insmod /mnt/udisk/create_cnode.ko
[ 76.258420] numdev_major is 0!
[ 76.260142] numdev_minor is 0!
[ 76.263112] adev_region req 249 !
[ 76.267643] cdev_add 0 is success!
[ 76.271529] cdev_add 1 is success!
[ 76.274974] scdev_init!
[root@iTOP-4412]#
```

使用命令 “ls /sys/class/” 可以查看到生成的 class，如下图所示。

```
[root@iTOP-4412]# ls /sys/class/
android_usb  firmware  koneplus  mmc_host  regulator  spi_master
arvo         graphics  kovaplug  net       rtc        switch
backlight   i2c-adapter  lcd      power_supply  scsi_device  tty
bdi         i2c-dev   lirc     ppp        scsi_disk   usb_device
block       ieee80211 mdio_bus  pyra       scsi_generic video4linux
bluetooth   input     mem      rc         scsi_host
chardevnode kone      misc     rc522      sound
```

使用命令 “ls /dev” 可以查看到生成的两个设备节点，如下图所示。

```

[root@iTOP-4412]# ls /dev
AGPS          kmsg          ram1          srp_ctrl
adc           log           ram10         tty
alarm         loop0         ram11         tty1
android_adb   loop1         ram12         tty2
ashmem        loop2         ram13         tty3
bus           loop3         ram14         tty4
chardevnode0  loop4         ram15         ttyGS0
chardevnode1  loop5         ram2          ttyGS1
console       loop6         ram3          ttyGS2
cpu_dma_latency loop7         ram4          ttyGS3
exynos-mem    mapper        ram5          ttyS0
fb0           max485_ctl_pin ram6          ttyS1
fb1           mem           ram7          ttyS2
fb2           mmcblk0       ram8          ttyS3
fb3           mmcblk0p1     ram9          ttySAC0
fb4           mmcblk0p2     random        ttySAC1
full          mmcblk0p3     rc522         ttySAC2
fuse          mmcblk0p4     relay_ctl     ttySAC3
i2c-0         mtp_usb       root          uinput
i2c-1         network_latency rtc0          urandom
i2c-3         network_throughput rtc1          usb_accessory
i2c-4         null          s3c-mem       usbdev1.1
i2c-5         pmem          sda           usbdev1.2
i2c-7         pmem_gpu1     sda1          usbdev1.3
input         ppp           sg0           usbdev1.4
ion           ptmx          shm           watchdog
keychord      pts           snd           xt_qtaguid
kmem          ram0          srp           zero

```

这里再介绍一点额外的知识，也是可以通过命令来创建设备节点的，如下图所示，使用命令 “mknod dev/test0 c 249 0” 和 “mknod dev/test1 c 249 1” 。

```

[root@iTOP-4412]# mknod dev/test0 c 249 0
[root@iTOP-4412]# mknod dev/test1 c 249 1
[root@iTOP-4412]# ls /dev/
AGPS          log           ram11         tty
adc           loop0         ram12         tty1
alarm         loop1         ram13         tty2
android_adb   loop2         ram14         tty3
ashmem        loop3         ram15         tty4
bus           loop4         ram2          ttyGS0
chardevnode0  loop5         ram3          ttyGS1
chardevnode1  loop6         ram4          ttyGS2
console       loop7         ram5          ttyGS3
cpu_dma_latency mapper        ram6          ttyS0
exynos-mem    max485_ctl_pin ram7          ttyS1
fb0           mem           ram8          ttyS2
fb1           mmcblk0       ram9          ttyS3
fb2           mmcblk0p1     random        ttySAC0
fb3           mmcblk0p2     rc522         ttySAC1
fb4           mmcblk0p3     relay_ctl     ttySAC2
full         mmcblk0p4     root          ttySAC3
fuse         mtp_usb       rtc0          uinput
i2c-0        network_latency rtc1          urandom
i2c-1        network_throughput s3c-mem       usb_accessory
i2c-3        null          sda           usbdev1.1
i2c-4        pmem          sda1          usbdev1.2
i2c-5        pmem_gpu1     sg0           usbdev1.3
i2c-7        ppp           shm           usbdev1.4
input        ptmx          snd           watchdog
ion          pts           srp           xt_qtaguid
keychord     ram0          srp_ctrl     zero
kmem         ram1
kmsg         ram10
test0
test1

```

如上图所示，这里给设备号 249 申请了两个设备节点，如果设备号 249 有对应的驱动，用命令创建的设备节点和用代码创建的设备节点有一样的效果，都可以给提供给应用程序调用和操作。