重点
1、驱动和应用是分开的
2、Ubuntu中关闭中间复制功能
3、第一个驱动
4、字符设备驱动
1、驱动框架编写用的是printk
2、编写应用程序用的是printf，多注意要加入的头文件（man是个很好用的工具）
3、如何使用应用程序
6、驱动程序的完善

重点

1、驱动和应用是分开的

> ① 这种思想就是，我先写好底层代码，然后通过编写上层应用，调用底层驱动，达到控制外设的目的。
>
> ② 要有一个内核态和用户态的概念（用户空间和内核空间）
>
> Linux操作系统内核和驱动程序运行在内核空间，应用程序运行在用户空间
>
> ③ Linux内核的镜像在/home/linux/IMX6ULL/linux/alientek_linux/arch/arm/boot 下面的zImage
>
> ④ 设备树文件也一样

2、Ubuntu中关闭中间复制功能

关闭命令

```
xmodmap -e "pointer = 1 25 3 4 5 6 7 2"
```

使能中间复制命令

```
xmodmap -e "pointer = 1 2 3 4 5 6 7 8"
```

3、第一个驱动

代码

```

#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>


static int __init chrdevbase_init(void)//加载时，初始化函数
{
    printk("my name is liutao\r\n");
    return 0;
}

static void __exit chrdevbase_exit(void)//退出函数
```

```
{
    printk("exit liutao\r\n");
    return;
}

module_init(chrdevbase_init);//定义的模块加载函数
module_exit(chrdevbase_exit);//定义的模块卸载函数

MODULE_LICENSE("GPL");//许可协议
MODULE_AUTHOR("liutao");
```

makefile文件

```
KERNELDIR := /home/local/linux/IMX6ULL/linux/alientek_linux

CURRENT_PATH := $(shell pwd)

obj-m := chrdevbase.o

build: kernel_modules

kernel_modules:
    $(MAKE) -C $(KERNELDIR) M=$(CURRENT_PATH) modules
clean:
    $(MAKE) -C $(KERNELDIR) M=$(CURRENT_PATH) clean

```

4、字符设备驱动

1、驱动框架编写用的是printk

```
//多去抄抄kernel的框架
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/fs.h>
#include <linux/kernel.h>

//主设备号和名字
#define CHARDEVBASE_MAJOR   200
#define CHRDEVBASE_NAME     "chrdevbase"

static int chrdevbase_open(struct inode *inode,struct file *file)
{
    printk("chrdevbase open\r\n");
    return 0;
}
static int chrdevbase_release(struct inode *inode,struct file *filp)
{
    printk("chrdevbase release\r\n");
    return 0;
}
```

```c
static ssize_t chrdevbase_reade(struct file *filp,__user char *buf,size_t count,loff_t *ppos)
{
    printk("chrdevbase read\r\n");
    return 0;
}
static ssize_t chrdevbase_write(struct file *filp,const char __user *buf,size_t count,loff_t *ppos)
{
    printk("chrdevbase write\r\n");
    return 0;
}

//字符设备框架
static struct file_operations chrdevbase_fops={
    .owner = THIS_MODULE,
    .open = chrdevbase_open,
    .release = chrdevbase_release,
    .read = chrdevbase_reade,
    .write = chrdevbase_write,
};

static int __init chrdevbase_init(void)
{
    //device
    int ret = 0;
    // 注册字符设备
    ret = register_chrdev(CHARDEVBASE_MAJOR,CHRDEVBASE_NAME,&chrdevbase_fops);
    if(ret < 0)
    {
        printk("my name is liutao init failed\r\n");
    }
    printk("my name is liutao is ok\r\n");
    return 0;
}

static void __exit chrdevbase_exit(void)
{
    printk("exit liutao\r\n");
    //卸载字符设备
    unregister_chrdev(CHARDEVBASE_MAJOR,CHRDEVBASE_NAME);
    return;
}

module_init(chrdevbase_init);
module_exit(chrdevbase_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("liutao");
```

2、编写应用程序用的是printf, 多注意要加入的头文件（man是个很好用的工具）

主要是open（返回文件描述符（fd)，read，write，release函数的调用

```c
fd = open(filename,O_RDWR);//参数：文件名，读写类型
```

```
//返回值就是返回读取的字节数，具体再去看看man手册
ret = read(fd,readbuf,100);//参数：从那个文件读，存到哪里，从驱动读取的字节数
```

```
ret = write(fd,writebuf,100);//参数：写到那个文件，写的内容存的地方，写多少字节
```

完整代码

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>

/**
 *argc:应用程序参数个数（比如 ls -l ,包括命令本身，个数就是2，后面再加参数，个数也得加。
 * *argv[]:保存这几个参数的内容，字符串形式
 *具体参数内容 ./chrdevbaseAPP <filename> 第一个是我们应用程序本身，第二个就是要操作的文件名
 *
 **/
int main(int argc,char *arcv[])
{
    int ret = 0;
    int fd;
    char *filename;

    char readbuf[100],writebuf[100];

    filename = arcv[1];
    /*open*/
    fd = open(filename, O_RDWR);
    if(fd < 0) {
        printf("can't open file %s\r\n",filename);
        return -1;
    }
    /*read*/
    ret = read(fd,readbuf,50);
    if(ret < 0){
        printf("read file %s failed\r\n",filename);

    }
    else {
        printf("read is ok\r\n");
    }
    /*write*/
    ret = write(fd,writebuf,50);
    if(ret < 0) {
        printf("write file %s faild\r\n",filename);
    }
    else {
        printf("write is ok\r\n");
    }
    /*close*/
    ret = close(fd);
    if (ret < 0) {
        printf("close file %s faild\r\n",filename);
```

```
    }
    else {
        printf("close is ok\r\n");
    }

    return 0;
}
```

3、如何使用应用程序

1、首先需要加载驱动，然后将应用程序拷贝到开发板和驱动同一目录下

2、加载驱动后，查看开发板的dev/目录，查看设备文件，chrdevbase,/dev/chrdevbase。但是实际是没有的，因为还没有创建设备节点

创建节点 mknod chrdevbase c 200 0 （c：代表字符设备，200：主设备号。0：次设备号）

执行命令后有个chrdevbase设备节点

```
/dev # mknod chrdevbase c 200 0
/dev # ls
ap3216c             ram11               tty36
apm_bios            ram12               tty37
autofs              ram13               tty38
bus                 ram14               tty39
chrdevbase          ram15               tty4
console             ram2                tty40
```

创建节点后查看

```
/dev # ls chrdevbase -l
crw-r--r--    1 0        0         200,   0 Jan  1 01:04 chrdevbase
/dev #
```

可以使用 cat /proc/devices查看设备号

```
/lib/modules/4.1.15 # cat /proc/devices
Character devices:
  1 mem
  4 /dev/vc/0
  4 tty
  5 /dev/tty
  5 /dev/console
  5 /dev/ptmx
  7 vcs
 10 misc
 13 input
 29 fb
 81 video4linux
 89 i2c
 90 mtd
108 ppp
116 alsa
128 ptm
136 pts
153 spi
180 usb
189 usb_device
200 chrdevbase
207 ttymxc
```

5、创建节点完成后就可以开始测试了。

怎么测试：1、执行应用程序 ./chardevbaseAPP /dev/chrdevbase

```
/lib/modules/4.1.15 # ./chardevbaseAPP /dev/chrdevbase
[ 4258.688072] chrdevbase open
[ 4258.691007] chrdevbase read
[ 4258.694291] chrdevbase write

[ 4258.698207] chrdevbase release
write is ok
close is ok
```

6、驱动程序的完善

要求：应用程序可以对驱动读写操作，读的话就是从驱动里面读取字符串，写的话就是应用向驱动写字符串。

1、chrdevbase_read驱动函数编写

去查看驱动程序中read驱动的参数有哪些

```
static ssize_t chrdevbase_reade(struct file *filp,__user char *buf,size_t count,loff_t *ppos)
{
    printk("chrdevbase read\r\n");
    return 0;
}
```

```
memcpy(readbuf,kerneldata,sizeof(kerneldata));//内存拷贝函数
```

```
copy_to_user(buf,readbuf,count);//驱动程序给应用程序传递数据的时候需要用到
```

2、chrdevbase_write驱动函数的编写

```
atoi();//字符串转换为整型变量
```

```
copy_from_user();//将buf里面的数据复制到应用程序里面
```

结果（有些被覆盖掉了，可以把驱动里面的release,open的打印注释掉）

```
/lib/modules/4.1.15 # mknod /dev/chrdevbase c 200 0
/lib/modules/4.1.15 # ./chrdevbaseAPP /dev/chrdevbase 1
[ 2262.028649] chrdevbase open
[ 2262.031872] chrdevbase read

[ 2262.035782] copy_to_user is ok
[ 2262.039079] chrdevbase release

APP read data:kernel data
close is ok
/lib/modules/4.1.15 # ./chrdevbaseAPP /dev/chrdevbase 2
[ 2301.943421] chrdevbase open
[ 2301.946612] chrdevbase write

[ 2301.950460] kernel recevdata:usr data
[ 2301.954547] chrdevbase release

close is ok
/lib/modules/4.1.15 #
```

修改后的驱动程序

```
#include <linux/module.h>
#include <linux/kernel.h>
```

```c
#include <linux/init.h>
#include <linux/fs.h>
#include <linux/kernel.h>
#include <linux/slab.h>
#include <linux/delay.h>
#include <linux/uaccess.h>
#include <linux/io.h>

#define CHARDEVBASE_MAJOR   200
#define CHRDEVBASE_NAME     "chrdevbase"

static char readbuf[100]; /*readbuf*/
static char writebuf[100];/*writebuf*/
static char kerneldata[] = {"kernel data"};


static int chrdevbase_open(struct inode *inode,struct file *file)
{
    printk("chrdevbase open\r\n");
    return 0;
}
static int chrdevbase_release(struct inode *inode,struct file *filp)
{
    printk("chrdevbase release\r\n");
    return 0;
}
static ssize_t chrdevbase_reade(struct file *filp,__user char *buf,size_t
count,loff_t *ppos)
{
    int ret = 0;
    printk("chrdevbase read\r\n");
    memcpy(readbuf,kerneldata,sizeof(kerneldata));
    ret = copy_to_user(buf,readbuf,count);
    if (ret < 0) {
        printk("copy_to_user is error\r\n");
    }
    else {
        printk("copy_to_user is ok\r\n");
    }


    return 0;
}
static ssize_t chrdevbase_write(struct file *filp,const char __user
*buf,size_t count,loff_t *ppos)
{

    int ret = 0;
    printk("chrdevbase write\r\n");

    ret = copy_from_user(writebuf,buf,count);
    if (ret < 0) {
        printk("write is error\r\n");
    }
    else {
        printk("kernel recevdata:%s\r\n",writebuf);
    }
    return 0;
```

```
}


static struct file_operations chrdevbase_fops={
    .owner = THIS_MODULE,
    .open = chrdevbase_open,
    .release = chrdevbase_release,
    .read = chrdevbase_reade,
    .write = chrdevbase_write,
};

static int __init chrdevbase_init(void)
{
    //device
    int ret = 0;
    ret =
register_chrdev(CHARDEVBASE_MAJOR,CHRDEVBASE_NAME,&chrdevbase_fops);
    if(ret < 0)
    {
        printk("my name is liutao init failed\r\n");
    }
    printk("my name is liutao is ok\r\n");
    return 0;
}

static void __exit chrdevbase_exit(void)
{
    printk("exit liutao\r\n");
    unregister_chrdev(CHARDEVBASE_MAJOR,CHRDEVBASE_NAME);
    return;
}

module_init(chrdevbase_init);
module_exit(chrdevbase_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("liutao");
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>

/**
 *argc:应用程序参数个数（比如 ls -l ,包括命令本身，个数就是2，后面再加参数，个数也得加。
 * *argv[]:保存这几个参数的内容，字符串形式
 *具体参数内容 ./chrdevbaseAPP <filename> <1:2> 第一个是我们应用程序本身，第二个就是
要操作的文件名,1表示读，2表示写
 *./chrdevbaseAPP /dev/chrdevbase 1 表示读数据
 *./chrdevbaseAPP /dev/chrdevbase 2 表示写数据
 **/
int main(int argc,char *argv[])
```

```
{
    int ret = 0;
    int fd;
    char *filename;

    char readbuf[100],writebuf[100];
    static char usrdata[] = {"usr data"};
    if(argc != 3) {
    printf("error usag!\r\n");
    return -1;
    }

    filename = argv[1];
    /*open*/
    fd = open(filename, O_RDWR);
    if(fd < 0) {
        printf("can't open file %s\r\n",filename);
        return -1;
    }
    else {
        printf("open is ok\r\n");
    }

    /*read value is 1*/
    if (atoi(argv[2]) == 1){
        /*read*/
        ret = read(fd,readbuf,50);
        if(ret < 0){
            printf("read file %s failed\r\n",filename);

        }
        else {
            printf("read is ok\r\n");
            printf("APP read data:%s\r\n",readbuf);
        }
    }

    /*write value is 2*/
    if(atoi(argv[2]) == 2) {
        memcpy(writebuf,usrdata,sizeof(usrdata));
        /*write*/
        ret = write(fd,writebuf,50);
        if(ret < 0) {
            printf("write file %s faild\r\n",filename);
        }
        else {
            printf("write is ok\r\n");
        }
    }

    /*close*/
    ret = close(fd);
    if (ret < 0) {
        printf("close file %s faild\r\n",filename);
    }
    else {
        printf("close is ok\r\n");
    }
```

```
75
76      return 0;
77  }
```