

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

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

使能中间复制命令

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

3、第一个驱动

代码

```
1
2 #include <linux/module.h>
3 #include <linux/kernel.h>
4 #include <linux/init.h>
5
6
7 static int __init chrdevbase_init(void)//加载时，初始化函数
8 {
9     printk("my name is liutao\r\n");
10    return 0;
11 }
12
13 static void __exit chrdevbase_exit(void)//退出函数
```

```

14 {
15     printk("exit liutao\r\n");
16     return;
17 }
18
19 module_init(chrdevbase_init); //定义的模块加载函数
20 module_exit(chrdevbase_exit); //定义的模块卸载函数
21
22 MODULE_LICENSE("GPL"); //许可协议
23 MODULE_AUTHOR("liutao");

```

makefile文件

```

1  KERNELDIR := /home/local/linux/IMX6ULL/linux/alientek_linux
2
3  CURRENT_PATH := $(shell pwd)
4
5  obj-m := chrdevbase.o
6
7  build: kernel_modules
8
9  kernel_modules:
10     $(MAKE) -C $(KERNELDIR) M=$(CURRENT_PATH) modules
11  clean:
12     $(MAKE) -C $(KERNELDIR) M=$(CURRENT_PATH) clean
13

```

4、字符设备驱动

1、驱动框架编写用的是printk

```

1  //多去抄抄kernel的框架
2  #include <linux/module.h>
3  #include <linux/kernel.h>
4  #include <linux/init.h>
5  #include <linux/fs.h>
6  #include <linux/kernel.h>
7
8  //主设备号和名字
9  #define CHARDEVBASE_MAJOR    200
10 #define CHRDEVBASE_NAME      "chrdevbase"
11
12 static int chrdevbase_open(struct inode *inode, struct file *file)
13 {
14     printk("chrdevbase open\r\n");
15     return 0;
16 }
17 static int chrdevbase_release(struct inode *inode, struct file *filp)
18 {
19     printk("chrdevbase release\r\n");
20     return 0;
21 }

```

```

22 static ssize_t chrdevbase_read(struct file *filp, __user char *buf, size_t
count, loff_t *ppos)
23 {
24     printk("chrdevbase read\r\n");
25     return 0;
26 }
27 static ssize_t chrdevbase_write(struct file *filp, const char __user
*buf, size_t count, loff_t *ppos)
28 {
29     printk("chrdevbase write\r\n");
30     return 0;
31 }
32
33 //字符设备框架
34 static struct file_operations chrdevbase_fops={
35     .owner = THIS_MODULE,
36     .open = chrdevbase_open,
37     .release = chrdevbase_release,
38     .read = chrdevbase_read,
39     .write = chrdevbase_write,
40 };
41
42 static int __init chrdevbase_init(void)
43 {
44     //device
45     int ret = 0;
46     // 注册字符设备
47     ret =
register_chrdev(CHARDEVBASE_MAJOR, CHRDEVBASE_NAME, &chrdevbase_fops);
48     if(ret < 0)
49     {
50         printk("my name is liutao init failed\r\n");
51     }
52     printk("my name is liutao is ok\r\n");
53     return 0;
54 }
55
56 static void __exit chrdevbase_exit(void)
57 {
58     printk("exit liutao\r\n");
59     //卸载字符设备
60     unregister_chrdev(CHARDEVBASE_MAJOR, CHRDEVBASE_NAME);
61     return;
62 }
63
64 module_init(chrdevbase_init);
65 module_exit(chrdevbase_exit);
66
67 MODULE_LICENSE("GPL");
68 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

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

```
1 //返回值就是返回读取的字节数，具体再去看看man手册
2 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
```

```
1 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

```
1 #include <sys/types.h>
2 #include <sys/stat.h>
3 #include <fcntl.h>
4 #include <stdio.h>
5 #include <unistd.h>
6
7 /**
8  *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。
9  * *argv[]:保存这几个参数的内容，字符串形式
10  *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
11  *
12  **/
13 int main(int argc,char *arvc[])
14 {
15     int ret = 0;
16     int fd;
17     char *filename;
18
19     char readbuf[100],writebuf[100];
20
21     filename = arvc[1];
22     /*open*/
23     fd = open(filename, O_RDWR);
24     if(fd < 0) {
25         printf("can't open file %s\r\n",filename);
26         return -1;
27     }
28     /*read*/
29     ret = read(fd,readbuf,50);
30     if(ret < 0){
31         printf("read file %s failed\r\n",filename);
32     }
33
34     else {
35         printf("read is ok\r\n");
36     }
37     /*write*/
38     ret = write(fd,writebuf,50);
39     if(ret < 0) {
40         printf("write file %s faild\r\n",filename);
41     }
42     else {
43         printf("write is ok\r\n");
44     }
45     /*close*/
46     ret = close(fd);
47     if (ret < 0) {
48         printf("close file %s faild\r\n",filename);
```

```

49     }
50     else {
51         printf("close is ok\r\n");
52     }
53
54     return 0;
55 }

```

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

```

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

```

s /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 # ./chrdevbaseAPP /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

```

1 static ssize_t chrdevbase_read(struct file *filp, __user char *buf, size_t
  count, loff_t *ppos)
2 {
3     printk("chrdevbase read\r\n");
4     return 0;
5 }

```

```

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

```

```

1 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

```

1 atoi(); //字符串转换为整型变量

```

```

1 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

```

1 #include <linux/module.h>
2 #include <linux/kernel.h>

```

```

3  #include <linux/init.h>
4  #include <linux/fs.h>
5  #include <linux/kernel.h>
6  #include <linux/slab.h>
7  #include <linux/delay.h>
8  #include <linux/uaccess.h>
9  #include <linux/io.h>
10
11 #define CHARDEVBASE_MAJOR    200
12 #define CHRDEVBASE_NAME      "chrdevbase"
13
14 static char readbuf[100]; /*readbuf*/
15 static char writebuf[100]; /*writebuf*/
16 static char kerneldata[] = {"kernel data"};
17
18
19 static int chrdevbase_open(struct inode *inode, struct file *file)
20 {
21     printk("chrdevbase open\r\n");
22     return 0;
23 }
24 static int chrdevbase_release(struct inode *inode, struct file *filp)
25 {
26     printk("chrdevbase release\r\n");
27     return 0;
28 }
29 static ssize_t chrdevbase_read(struct file *filp, __user char *buf, size_t
count, loff_t *ppos)
30 {
31     int ret = 0;
32     printk("chrdevbase read\r\n");
33     memcpy(readbuf, kerneldata, sizeof(kerneldata));
34     ret = copy_to_user(buf, readbuf, count);
35     if (ret < 0) {
36         printk("copy_to_user is error\r\n");
37     }
38     else {
39         printk("copy_to_user is ok\r\n");
40     }
41
42
43     return 0;
44 }
45 static ssize_t chrdevbase_write(struct file *filp, const char __user
*buf, size_t count, loff_t *ppos)
46 {
47
48     int ret = 0;
49     printk("chrdevbase write\r\n");
50
51     ret = copy_from_user(writebuf, buf, count);
52     if (ret < 0) {
53         printk("write is error\r\n");
54     }
55     else {
56         printk("kernel recevdata:%s\r\n", writebuf);
57     }
58     return 0;

```

```

59 }
60
61
62 static struct file_operations chrdevbase_fops={
63     .owner = THIS_MODULE,
64     .open = chrdevbase_open,
65     .release = chrdevbase_release,
66     .read = chrdevbase_read,
67     .write = chrdevbase_write,
68 };
69
70 static int __init chrdevbase_init(void)
71 {
72     //device
73     int ret = 0;
74     ret =
register_chrdev(CHARDEVBASE_MAJOR,CHRDEVBASE_NAME,&chrdevbase_fops);
75     if(ret < 0)
76     {
77         printk("my name is liutao init failed\r\n");
78     }
79     printk("my name is liutao is ok\r\n");
80     return 0;
81 }
82
83 static void __exit chrdevbase_exit(void)
84 {
85     printk("exit liutao\r\n");
86     unregister_chrdev(CHARDEVBASE_MAJOR,CHRDEVBASE_NAME);
87     return;
88 }
89
90 module_init(chrdevbase_init);
91 module_exit(chrdevbase_exit);
92
93 MODULE_LICENSE("GPL");
94 MODULE_AUTHOR("liutao");

```

```

1  #include <sys/types.h>
2  #include <sys/stat.h>
3  #include <fcntl.h>
4  #include <stdio.h>
5  #include <unistd.h>
6  #include <stdlib.h>
7  #include <string.h>
8
9  /**
10   *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。
11   * argv[]:保存这几个参数的内容，字符串形式
12   *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
13   *./chrdevbaseAPP /dev/chrdevbase 1 表示读数据
14   *./chrdevbaseAPP /dev/chrdevbase 2 表示写数据
15   */
16 int main(int argc,char *argv[])

```



```

17 {
18     int ret = 0;
19     int fd;
20     char *filename;
21
22     char readbuf[100],writebuf[100];
23     static char usrdata[] = {"usr data"};
24     if(argc != 3) {
25         printf("error usag!\r\n");
26         return -1;
27     }
28
29     filename = argv[1];
30     /*open*/
31     fd = open(filename, O_RDWR);
32     if(fd < 0) {
33         printf("can't open file %s\r\n",filename);
34         return -1;
35     }
36     else {
37         printf("open is ok\r\n");
38     }
39
40     /*read value is 1*/
41     if (atoi(argv[2]) == 1){
42         /*read*/
43         ret = read(fd,readbuf,50);
44         if(ret < 0){
45             printf("read file %s failed\r\n",filename);
46
47         }
48         else {
49             printf("read is ok\r\n");
50             printf("APP read data:%s\r\n",readbuf);
51         }
52     }
53
54     /*write value is 2*/
55     if(atoi(argv[2]) == 2) {
56         memcpy(writebuf,usrdata,sizeof(usrdata));
57         /*write*/
58         ret = write(fd,writebuf,50);
59         if(ret < 0) {
60             printf("write file %s faild\r\n",filename);
61         }
62         else {
63             printf("write is ok\r\n");
64         }
65     }
66
67     /*close*/
68     ret = close(fd);
69     if (ret < 0) {
70         printf("close file %s faild\r\n",filename);
71     }
72     else {
73         printf("close is ok\r\n");
74     }

```

75

76

77

```
    return 0;  
}
```