一：如何写输入子系统框架驱动程序代码？

1. 分配一个input\_dev结构体//input\_dev是描述一个硬件设备
2. 设置
3. 注册
4. 硬件相关的代码比如在中断服务程序里上报事件

二：我们自己实现的驱动的时候：

App: open read write

驱动：drv\_open drv\_write drv\_read

硬件：

构建file\_operations{

.open

.write

.write

.Poll

.fasync

}

register\_chardev(magior,name, file\_operations)

入口

出口

三：自己实现驱动：

同样的在输入子系统的核心层input.c中也有类似的处理过程。

实现自己的驱动程序：//参考Gpio\_keys.c 从init()函数开始

测试：

ls -l /dev/event\*

crw-rw---- 1 0 0 13, 64 Jan 1 00:00 /dev/event0

crw-rw---- 1 0 0 13, 65 Jan 1 00:05 /dev/event1//对应我们们的Buttos.ko 主设备13 次设备65

/dev/event1以前我们自己写驱动的时候：

class\_register//注册类

class\_device\_create()//udev 或者mdev帮助我们创建设备节点

但是在输入子系统的时候：

input\_init（）//

class\_register（）//只有注册类

evdev\_connect（）//才去创建设备节点

class\_device\_create（）

1： hexdump /dev/event1 // hexdump 打开设备event1并以16进制显示 打开设备并去读取

当我们调用open的时候input\_table[65/32] 也就是打开input\_table[2]

我们调用input\_register\_handler（）时 input\_table[2] = &edev\_handler; 我们调用read时候—>>evdev\_read（）->>evdev\_event\_to\_user()-->copy\_to\_user() 将input\_event（）传递给用户层。

struct input\_event {

struct timeval time;//按键的时间

\_\_u16 type;//哪类按键

\_\_u16 code;//key\_L 或者其他的

\_\_s32 value;//值

};

本身输出的数据 秒 微妙 类 code value

0000000 02a1 0000 cd49 000e 0001 0026 0001（低位）0000//1 按键类 code 38(26) 1按下

0000010 02a1 0000 cd52 000e 0000 0000 0000 0000

0000020 02a2 0000 4a29 0002 0001 0026 0000 0000

0000030 02a2 0000 4a31 0002 0000 0000 0000 0000

2 ：cat /dev/tty1(没有启动qt)

按下按键对应的ls，再按下回车。就可以得到ls。但是不会像我们修改标准输入，ls那样的效果

3：/dev/tty1 tty1主设备号是4 ,次设备号是1

通过tty\_io.c 驱动访问keyboard.c tty\_io 非常复杂

exec 0</dev/tty1//在我们的串口上shell等待我们输入命令解析。改变将shell程序的0号文件 也就是输入文件 改为从/dev/tty1输入

ps ：如下

770 0 3096 S -sh

ls -l /proc/770/fd：可以发现我们的标准输入，标准输出，标准错误都是串口：

lrwx------ 1 0 0 64 Jan 1 00:00 0 -> /dev/console

lrwx------ 1 0 0 64 Jan 1 00:00 1 -> /dev/console

lrwx------ 1 0 0 64 Jan 1 00:00 10 -> /dev/tty

lrwx------ 1 0 0 64 Jan 1 00:00 2 -> /dev/console

通过：exec 0</dev/tty1 从dev/tty1输入。tty1通过tty\_io.c 驱动访问keyboard.c tty\_io 非常复杂。

这样就可以达到我们通过键盘输入的ls效果。

4.启动了QT

取消Qt才能观察。

环形缓冲区：

buf[4] = {0,1,2,3}

0

最开始的时候 R W 在起始位置R = W 空

写buf[w] =val w= (w+)%LEN

读val= buf[R] R = (R+1)%LEN

满的情况：R == (w+)%LEN