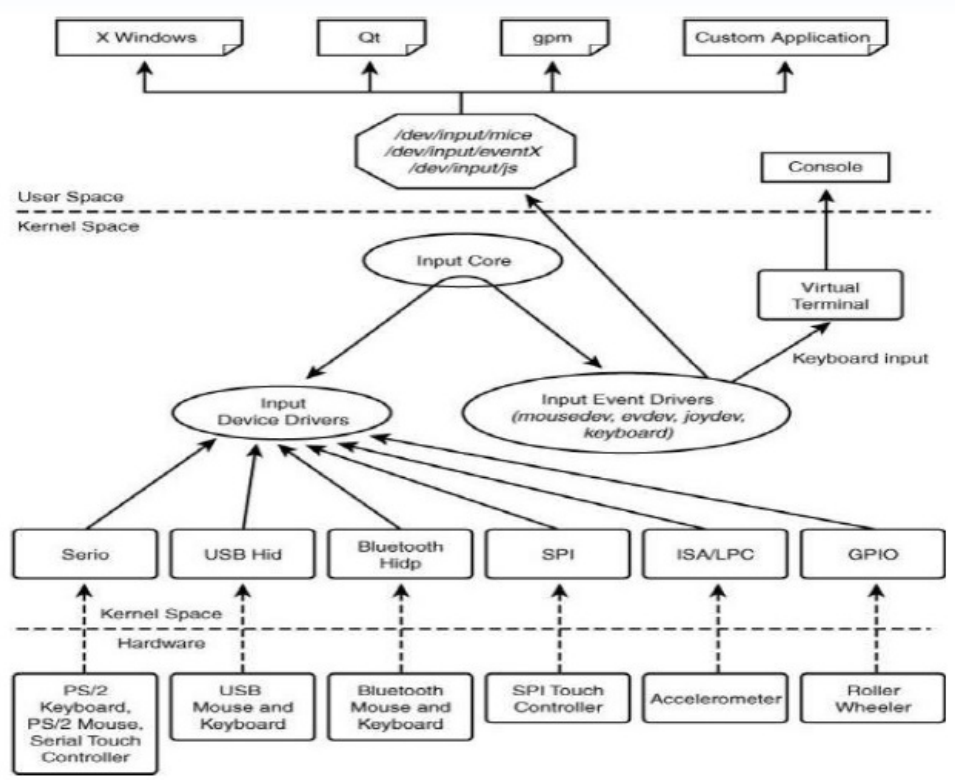
专题16-触摸屏驱动程序设计

一、输入子系统模型分析

1.1、为什么需要输入子系统

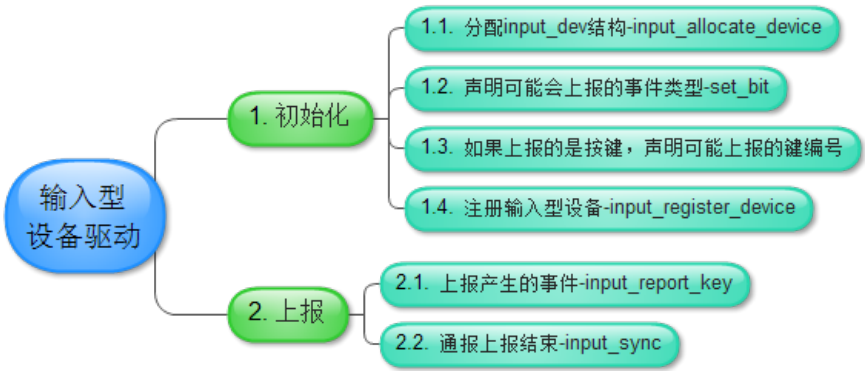


输入子系统由设备驱动层(input device driver)， 核心层(input core)和事件驱动层(event driver) 三部份组成。任何一次输入事件，如鼠标移动，按 键按下，都需要通过
InputDeviceDriver->InputCore->EventDrive
才能到达用户空间的应用程序。

设备驱动层：
将底层的硬件输入转化为统一事件型式，向输入核心
（InputCore）汇报。
输入核心层：
为设备驱动层提供输入设备注册与操作接口，如：
input_register_device；通知事件处理层对事件进行处理；
事件驱动层：
主要作用是和用户空间交互，如提供read,open等设备方 法，创建设备文件等。

附录
事件类型：
EV_RST Reset
EV_KEY 按键
EV_REL 相对坐标
EV_ABS 绝对坐标
EV_MSC 其它
EV_LED LED
EV_SND 声音
EV_REP Repeat
EV_FF 力反馈
当事件类型为EV_KEY时，还需指明按键类型：
BTN_ LEFT:鼠标左键
BTN_0:数字0键
BTN_RIGHT:鼠标右键
BTN_1:数字1键

1.2、输入子系统案例介绍-按键鹋驭动



1.3、输入子系统案例分析-按键驱动

1.3.1、初始化

1.3.1.1、分配input_dev结构

```
button_dev = input_allocate_device();
```

1.3.1.2、声明可能会上报的事件类型

```
set_bit(EV_KEY,button_dev->evbit);
```

1.3.1.3、如果上报的是按键，声明可能上报的键编号

```
set_bit(KEY_3,button_dev->keybit);
set_bit(KEY_4,button_dev->keybit);
```

1.3.1.4、注册输入型设备

```
input_register_device(button_dev);
```

1.3.2、上报

1.3.2.1、上报产生的事件

```
input_report_key(button_dev,KEY_4,1);
```

1表示按下，0表示弹起。

1.3.2.2、通报上报结束

```
input_sync(button_dev);
```

代码：

key.c:

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/miscdevice.h>
#include <linux/interrupt.h>
#include <linux/fs.h>
#include <linux/io.h>
#include <linux/workqueue.h>
#include <linux/slab.h>
#include <linux/timer.h>
#include <linux/uaccess.h>
#include <linux/wait.h>
#include <linux/sched.h>
#include <linux/input.h>

#define GPNCON    0x7f008830
#define GPNDAT    0x7f008834

struct work_struct * work1;

struct timer_list key_timer;

unsigned int * gpio_data;
unsigned int key_num = 0;

wait_queue_head_t key_q;
```

```
struct input_dev * button_dev;

void work1_func(struct work_struct * work)
{
    mod_timer(&key_timer, jiffies + HZ/10);
}

void key_timer_func(unsigned long data)
{
    unsigned int key_val;

    key_val = readl(gpio_data) & 0b11;

    if (key_val == 0b10) {
        key_num = 1;
        input_report_key(button_dev,KEY_2,1);
    }

    if (key_val == 0b01) {
        key_num = 2;
        input_report_key(button_dev,KEY_3,1);
    }

    //wake_up(&key_q);
    input_sync(button_dev);
}

static irqreturn_t key_int(int irq, void *dev_id)
{
    /*Check if a key interrupt has occurred */

    /*Clear key interrupts that have occurred(If it is a CPU internal interrupt (non-peripheral), the system will help clear) */

    /*Submit the bottom half */
        /*queue work*/
    schedule_work(work1);

    return IRQ_HANDLED;
}

void key_hw_init(void)
{
    unsigned int data;
    unsigned int * gpio_config;

    gpio_config = ioremap(GPNCON, 4);
    data = readl(gpio_config);
    data &= ~0b1111;                    //set key1 and key2
    data |= 0b1010;
    writel(data, gpio_config);

    gpio_data = ioremap(GPNDAT, 4);
}


static int __init ok6410_key_init(void)
{
    printk(KERN_WARNING"key init\n");

    /*Assign input device structure*/
    button_dev = input_allocate_device();

    /*Declare the type of event that may be reported*/
    set_bit(EV_KEY, button_dev->evbit);

    /*Declare the button number that may be reported*/
    set_bit(KEY_2, button_dev->keybit);
```

```
    set_bit(KEY_3, button_dev->keybit);

    /*register input device*/
    input_register_device(button_dev);

    request_irq(S3C_EINT(0), key_int, IRQF_TRIGGER_FALLING, "key", (void *)0);
    request_irq(S3C_EINT(1), key_int, IRQF_TRIGGER_FALLING, "key", (void *)1);

    key_hw_init();

    /*init work*/
    work1 = kmalloc(sizeof(struct work_struct), GFP_KERNEL);

    INIT_WORK(work1, work1_func);

    /*init timer */
    init_timer(&key_timer);
    key_timer.function = key_timer_func;

    /*register timer */
    add_timer(&key_timer);

    /*init wait queue*/
    init_waitqueue_head(&key_q);

    return 0;
}

static void __exit ok6410_key_exit(void)
{
    printk(KERN_WARNING"key exit\n");

    input_unregister_device(button_dev);
}


module_init(ok6410_key_init);
module_exit(ok6410_key_exit);
MODULE_LICENSE("GPL");
```

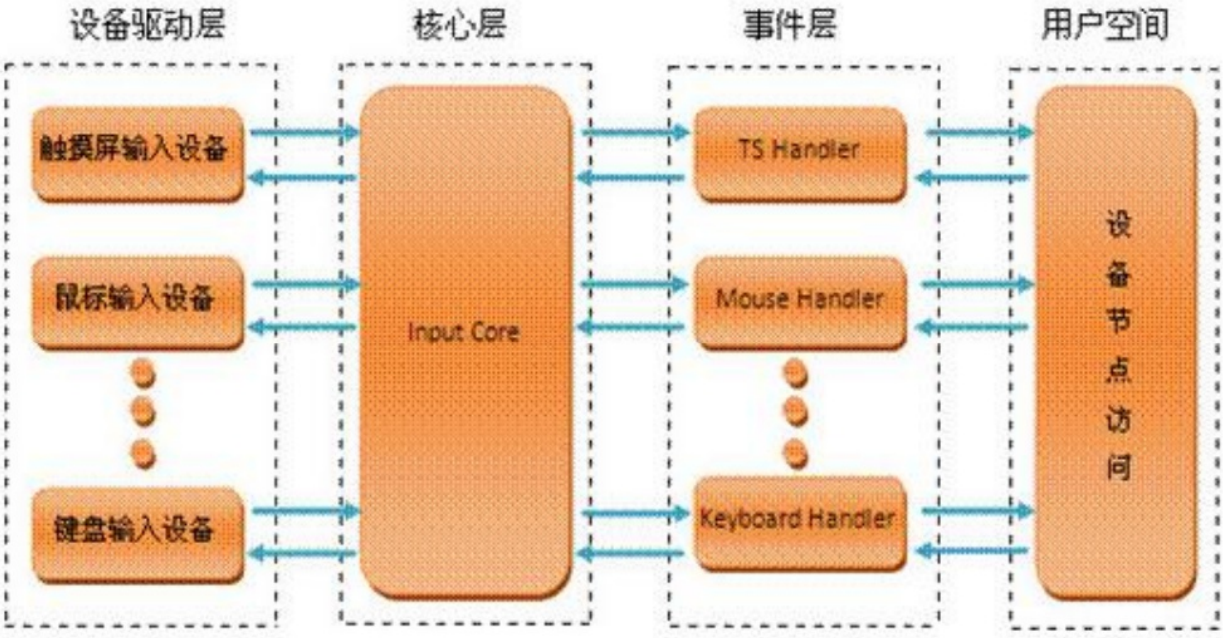
key_app.c:

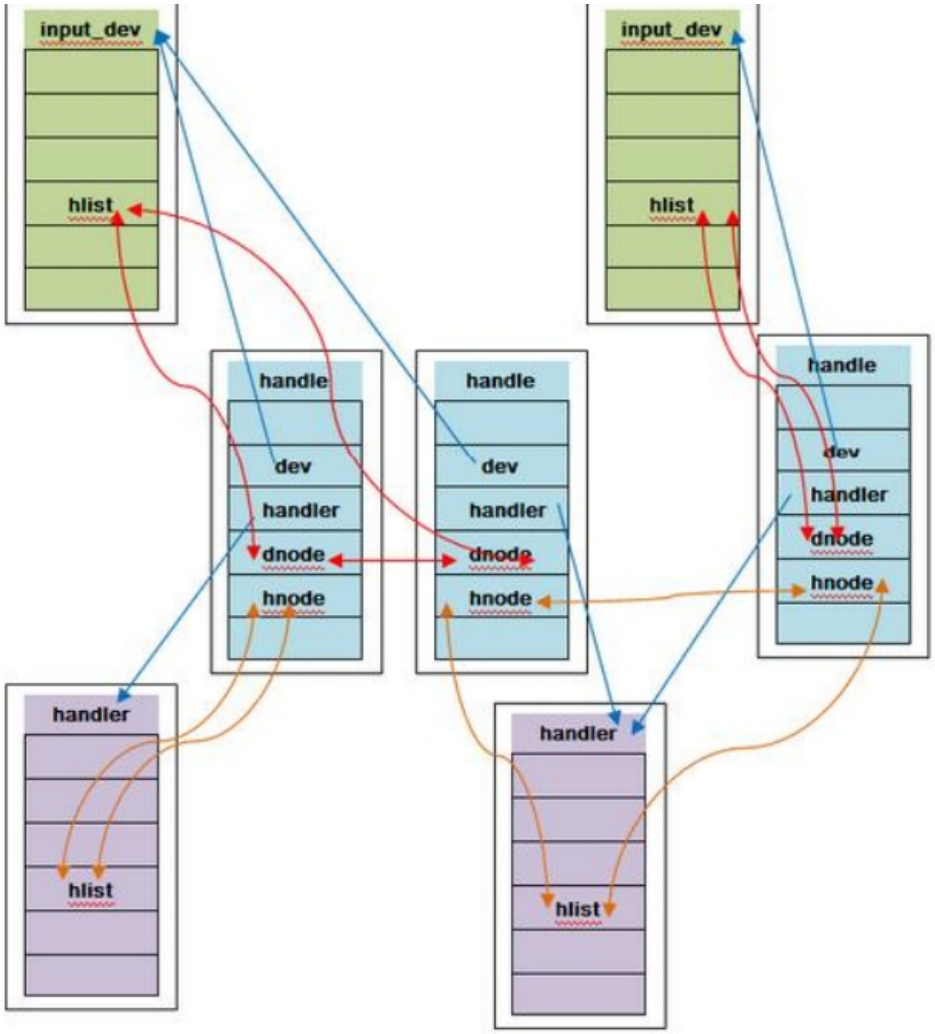
```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/select.h>
#include <sys/time.h>
#include <errno.h>
#include <linux/input.h>

int main(void)
{
    int buttons_fd;
    int key_value, i = 0, count;

    struct input_event ev_key;

    /*open device*/
    buttons_fd = open("/dev/event1", O_RDWR);

    if (buttons_fd < 0) {
        printf("open device fail!\n");
        exit(1);
    }
```

```
    for(;;) {
        count = read(buttons_fd, &ev_key, sizeof(struct input_event));
        for (i = 0; i < (int)count / sizeof(struct input_event); i++)
            if (EV_KEY == ev_key.type)
                printf("type:%d, code:%d, value:%d\n", ev_key.type, ev_key.code - 1, ev_key.value);
            if (EV_SYN == ev_key.type)
                printf("syn event\n");
    }


    /*close device*/
    close(buttons_fd);

    return 0;
}
```

二、输入子系统原理分析

2.1、输入子系统核心架构





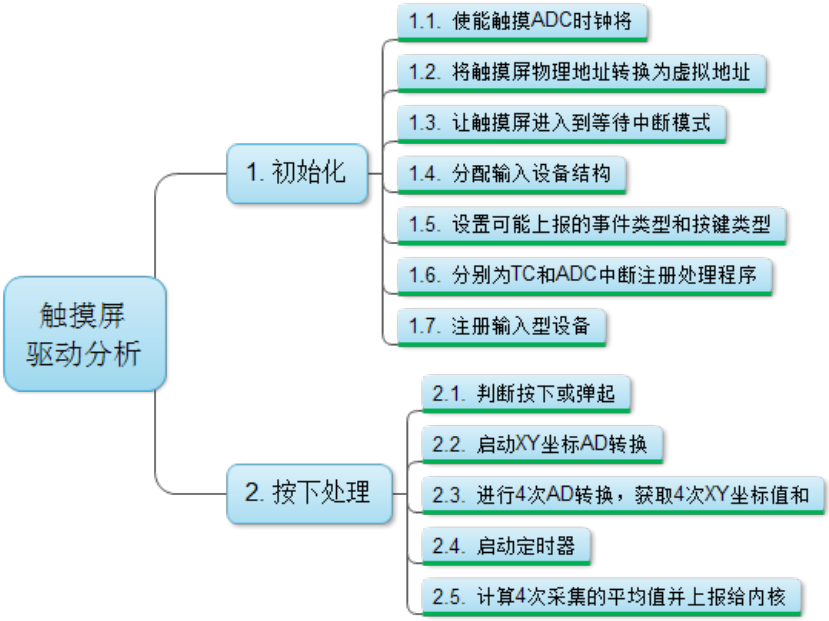
2.2、输入设备注册

2.2.1、拿设备ID去匹配handler ID，找到对应的handler（处理者）。

2.2.2、调用handler_connect函数注册字符设备，创建设备文件。

2.3、事件上报

三、触摸屏驱动分析



四、触摸屏驱动编程