# 框架介绍

linux使用input子系统去管理诸如键盘、鼠标、触屏、按键等输入设备。

input子系统主要分为三层：设备驱动层、核心层、事件处理层，整体框架如下：

输入设备

核心

事件处理

用户空间

userspace

evdev.c

joydev.c

mousedev.c

keyboard.c

input.c

ads7846.c

gpio\_keys.c

usbmouse.c

usbkbd.c

1. 设备驱动层完成各种输入设备的操作与事件上报，通过input\_register\_device完成输入设备的注册，然后通过input\_event上报输入事件。
2. 事件处理层完成对输入事件的处理，完成与用户空间的交互，可通过input\_register\_handler向input子系统注册事件处理方法。
3. 核心层则完成设备驱动层和事件处理层的协调处理，上面的无论是输入设备的注册方法还是事件处理的注册方法都是在核心层实现。

# 设备驱动层

## 数据结构

输入设备通过input\_dev结构进行描述，结构如下：

struct input\_dev {

const char \*name;

const char \*phys;

const char \*uniq;

struct input\_id id;

unsigned long propbit[BITS\_TO\_LONGS(INPUT\_PROP\_CNT)];

unsigned long evbit[BITS\_TO\_LONGS(EV\_CNT)];

unsigned long keybit[BITS\_TO\_LONGS(KEY\_CNT)];

unsigned long relbit[BITS\_TO\_LONGS(REL\_CNT)];

unsigned long absbit[BITS\_TO\_LONGS(ABS\_CNT)];

unsigned long mscbit[BITS\_TO\_LONGS(MSC\_CNT)];

unsigned long ledbit[BITS\_TO\_LONGS(LED\_CNT)];

unsigned long sndbit[BITS\_TO\_LONGS(SND\_CNT)];

unsigned long ffbit[BITS\_TO\_LONGS(FF\_CNT)];

unsigned long swbit[BITS\_TO\_LONGS(SW\_CNT)];

};

evbit：支持的事件类型

#define EV\_SYN 0x00 //同步事件，每个输入设备都会产生同步事件，所以不用我们自己设置，输入设备支持时会自动设置

#define EV\_KEY 0x01 //按键事件

#define EV\_REL 0x02 //相对坐标，如鼠标的坐标

#define EV\_ABS 0x03 //绝对坐标，如触摸屏的坐标

#define EV\_MSC 0x04 //其他事件

#define EV\_SW 0x05 //开关事件

#define EV\_LED 0x11 //led灯事件

#define EV\_SND 0x12 //声音事件

#define EV\_REP 0x14 //重复事件

#define EV\_FF 0x15 //力反馈事件

#define EV\_PWR 0x16 //电源事件

#define EV\_FF\_STATUS 0x17 //力反馈状态

keybit：按键事件编码

## 注册方法

我们想将一个新的输入设备注册到input系统中：

首先需要完成input\_dev的申请：

struct input\_dev \*input\_allocate\_device(void)

设备的释放：

void input\_free\_device(struct input\_dev \*dev)

初始化设备，主要需要完成设备支持的事件以及事件的编码，设置方法如下：

1. 直接赋值 input\_dev->evbit[0] = BIT\_MASK(EV\_KEY);
2. 使用\_\_set\_bit()，如\_\_set\_bit(EV\_KEY, input\_dev->evbit);

清除事件使用\_\_clear\_bit();

1. 使用input\_set\_capability()方法，这个方法内部还是调用\_\_set\_bit()，它不仅可以设置事件编码，同时也会将事件设置到evbit中。如input\_set\_capability(input\_dev,EV\_KEY,KEY\_1)。

初始化完毕后，就可以向input子系统中注册输入设备了：

int input\_register\_device(struct input\_dev \*dev)

设备的注销：

void input\_unregister\_device(struct input\_dev \*dev)

此后，如果设备产生了输入事件，可将事件进行上报：

void input\_event(struct input\_dev \*dev, unsigned int type, unsigned int code, int value)

dev：input\_dev指针

type：事件类型，在介绍数据结构的时候已说明

code：事件编码，定义在input-event-codes.h中，可以根据事件类型进行选择

value：事件值

当然可以直接使用内核已经封装好的上报函数：

void input\_report\_key(struct input\_dev \*dev, unsigned int code, int value)

上报同步事件

void input\_report\_rel(struct input\_dev \*dev, unsigned int code, int value)

void input\_report\_abs(struct input\_dev \*dev, unsigned int code, int value)

上报坐标事件

void input\_sync(struct input\_dev \*dev)

上报同步事件

上报事件后，还需要提交到input子系统中：

void input\_sync(struct input\_dev \*dev)

## 示例说明

示例中还是以按键为例，向系统注册输出设备后就可以在应用层读取输入事件了，执行结果如下：

