## 第七章 输入系统应用编程

### 7.1 什么是输入系统

先来了解什么是输入设备？

常见的输入设备有键盘、鼠标、遥控杆、书写板、触摸屏等等,用户通过这些输入设备与Linux系统进行数据交换。

什么是输入系统？

输入设备种类繁多，能否统一它们的接口？既在驱动层面统一，也在应用程序层面统一？可以的。

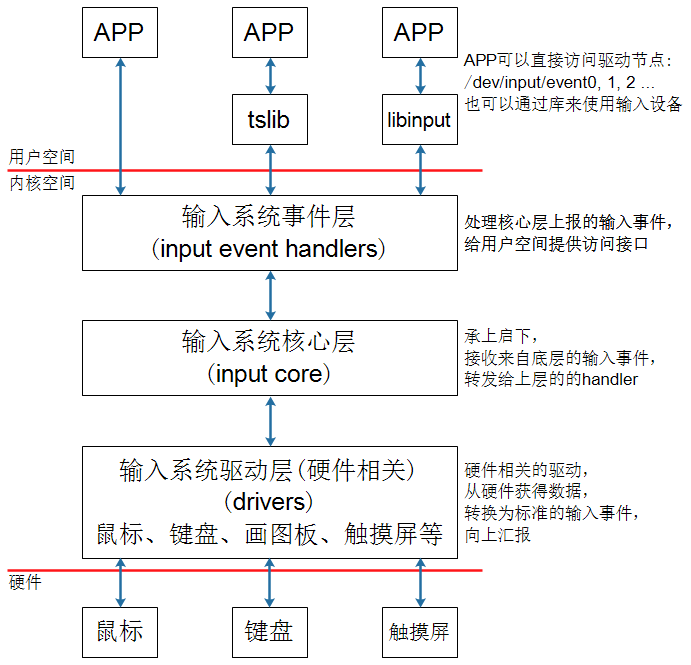
Linux系统为了统一管理这些输入设备，实现了一套能兼容所有输入设备的框架：输入系统。驱动开发人员基于这套框架开发出程序，应用开发人员就可以使用统一的API去使用设备。

### 7.2 输入系统框架及调试

#### 7.2.1 框架概述

作为应用开发人员，可以只基于API使用输入子系统。但是了解内核中输入子系统的框架、了解数据流程，有助于解决开发过程中碰到的硬件问题、驱动问题。

输入系统框架如下图所示：



假设用户程序直接访问/dev/input/event0设备节点，或者使用tslib访问设备节点，数据的流程如下：

① APP发起读操作，若无数据则休眠；

② 用户操作设备，硬件上产生中断；

③ 输入系统驱动层对应的驱动程序处理中断：

读取到数据，转换为标准的输入事件，向核心层汇报。

所谓输入事件就是一个“struct input\_event”结构体。

④ 核心层可以决定把输入事件转发给上面哪个handler来处理：

从handler的名字来看，它就是用来处输入操作的。有多种handler，比如：evdev\_handler、kbd\_handler、joydev\_handler等等。

最常用的是evdev\_handler：它只是把input\_event结构体保存在内核buffer等，APP来读取时就原原本本地返回。它支持多个APP同时访问输入设备，每个APP都可以获得同一份输入事件。

当APP正在等待数据时，evdev\_handler会把它唤醒，这样APP就可以返回数据。

⑤ APP对输入事件的处理：

APP获得数据的方法有2种：直接访问设备节点(比如/dev/input/event0,1,2,...)，或者通过tslib、libinput这类库来间接访问设备节点。这些库简化了对数据的处理。

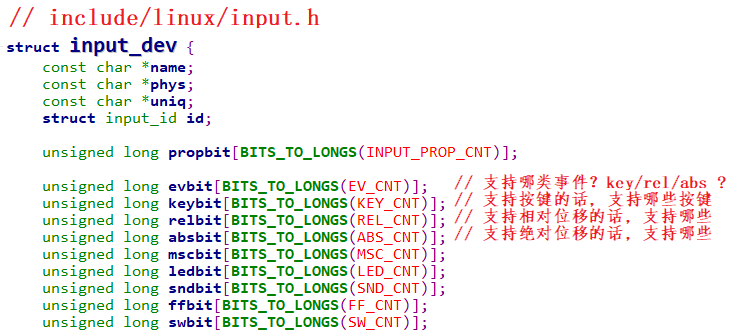
要想深入理解整个输入系统，就必须研究内核的输入系统，这在后续的“驱动大全”中会讲解。

#### 7.2.2 编写APP需要掌握的知识

基于编写应用程序的角度，只需要理解这些内容：

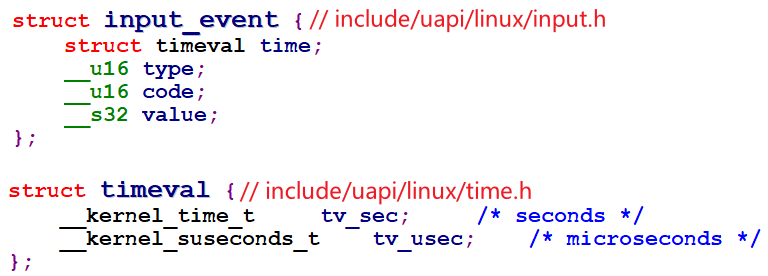
##### 1. 内核中怎么表示一个输入设备？

使用input\_dev结构体来表示输入设备，它的内容如下：



##### 2. APP可以得到什么数据？

可以得到一系列的输入事件，就是一个一个“struct input\_event”，它定义如下：

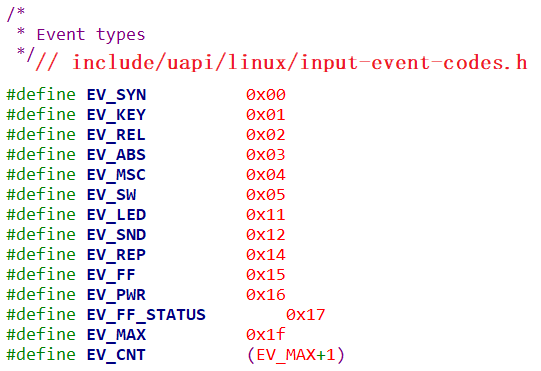


每个输入事件input\_event中都含有发生时间：timeval表示的是“自系统启动以来过了多少时间”，它是一个结构体，含有“tv\_sec、tv\_usec”两项(即秒、微秒)。

输入事件input\_event中更重要的是：type(哪类事件)、code(哪个事件)、value(事件值)，细讲如下：

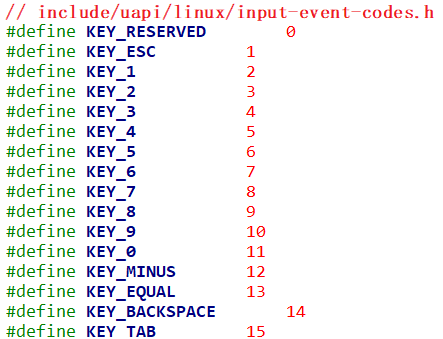
① type：表示哪类事件

比如EV\_KEY表示按键类、EV\_REL表示相对位移(比如鼠标)，EV\_ABS表示绝对位置(比如触摸屏)。有这几类事件(参考Linux内核头文件)：

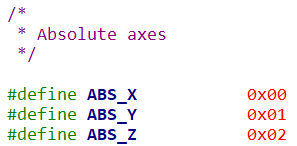


② code：表示该类事件下的哪一个事件

比如对于EV\_KEY(按键)类事件，它表示键盘。键盘上有很多按键，比如数字键1、2、3，字母键A、B、C里等。所以可以有这些事件：



对于触摸屏，它提供的是绝对位置信息，有X方向、Y方向，还有压力值。所以code值有这些：



③ value：表示事件值

对于按键，它的value可以是0(表示按键被按下)、1(表示按键被松开)、2(表示长按)；

对于触摸屏，它的value就是坐标值、压力值。

④ 事件之间的界线

APP读取数据时，可以得到一个或多个数据，比如一个触摸屏的一个触点会上报X、Y位置信息，也可能会上报压力值。

APP怎么知道它已经读到了完整的数据？

驱动程序上报完一系列的数据后，会上报一个“同步事件”，表示数据上报完毕。APP读到“同步事件”时，就知道已经读完了当前的数据。

同步事件也是一个input\_event结构体，它的type、code、value三项都是0。

##### 3 输入子系统支持完整的API操作

支持这些机制：阻塞、非阻塞、POLL/SELECT、异步通知。

注意：如果你想深入理解上述机制，需要学习以下内容：

《第5篇 嵌入式Linux驱动开发基础知识》

[《第十九章 驱动程序基石》](#_第十九章_驱动程序基石)

#### 7.2.3 调试技巧

##### 1. 确定设备信息

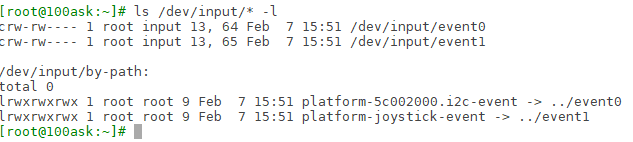
输入设备的设备节点名为/dev/input/eventX(也可能是/dev/eventX，X表示0、1、2等数字)。查看设备节点，可以执行以下命令：

ls /dev/input/\* -l

或

ls /dev/event\* -l

可以看到类似下面的信息：



怎么知道这些设备节点对应什么硬件呢？可以在板子上执行以下命令：

cat /proc/bus/input/devices

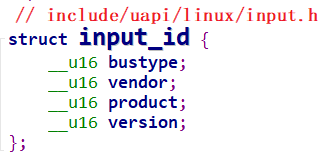
这条指令的含义就是获取与event对应的相关设备信息，可以看到类似以下的结果：



那么这里的I、N、P、S、U、H、B对应的每一行是什么含义呢？

**① I:id of the device(设备ID)**

该参数由结构体struct input\_id来进行描述，驱动程序中会定义这样的结构体：



**② N:name of the device**

设备名称

**③ P:physical path to the device in the system hierarchy**

系统层次结构中设备的物理路径。

**④ S:sysfs path**

位于sys文件系统的路径

**⑤ U:unique identification code for the device(if device has it)**

设备的唯一标识码

**⑥ H:list of input handles associated with the device.**

与设备关联的输入句柄列表。

**⑦ B:bitmaps(位图)**

PROP:device properties and quirks(设备属性)

EV:types of events supported by the device(设备支持的事件类型)

KEY:keys/buttons this device has(此设备具有的键/按钮)

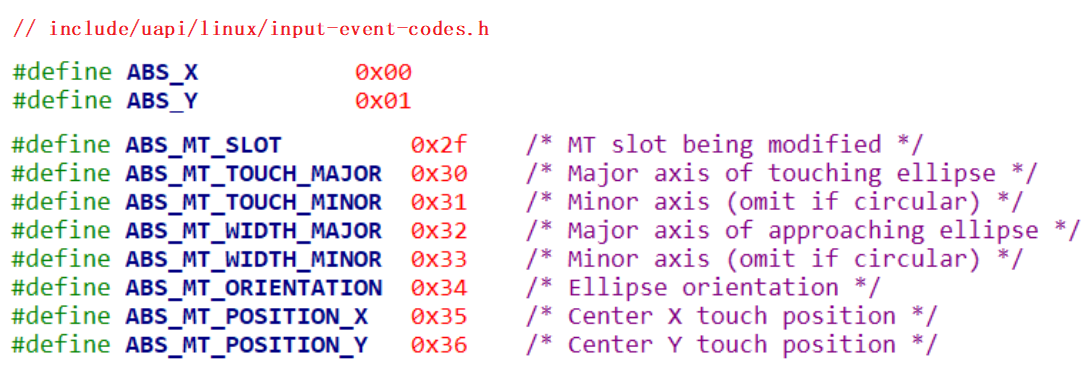
MSC:miscellaneous events supported by the device(设备支持的其他事件)

LED:leds present on the device(设备上的指示灯)

值得注意的是B位图，比如上图中“B: EV=b”用来表示该设备支持哪类输入事件。b的二进制是1011，bit0、1、3为1，表示该设备支持0、1、3这三类事件，即EV\_SYN、EV\_KEY、EV\_ABS。

再举一个例子，“B: ABS=2658000 3”如何理解？

它表示该设备支持EV\_ABS这一类事件中的哪一些事件。这是2个32位的数字：0x2658000、0x3，高位在前低位在后，组成一个64位的数字：“0x2658000,00000003”，数值为1的位有：0、1、47、48、50、53、54，即：0、1、0x2f、0x30、0x32、0x35、0x36，对应以下这些宏：



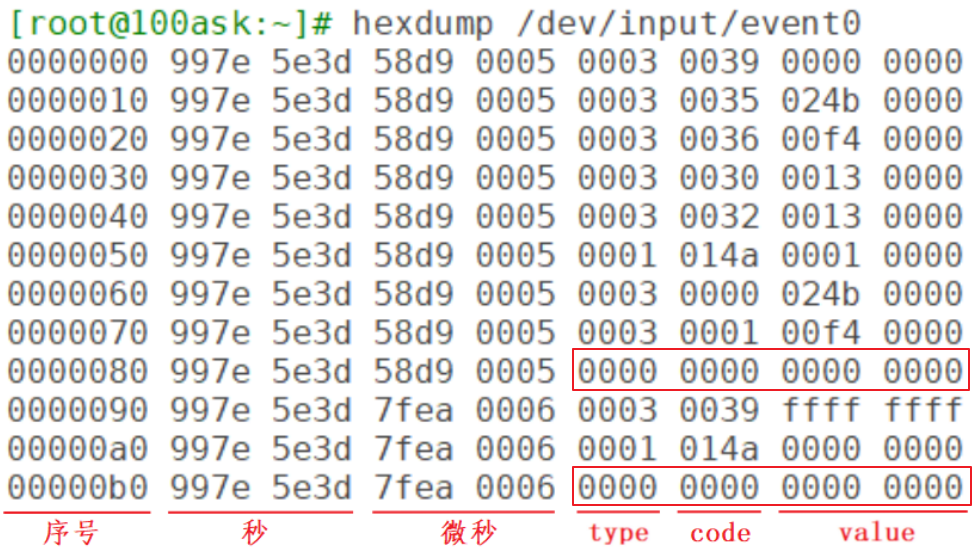
即这款输入设备支持上述的ABS\_X、ABS\_Y、ABS\_MT\_SLOT、ABS\_MT\_TOUCH\_MAJOR、ABS\_MT\_WIDTH\_MAJOR、ABS\_MT\_POSITION\_X、ABS\_MT\_POSITION\_Y这些绝对位置事件(它们的含义在后面讲解电容屏时再细讲)。

##### 2. 使用命令读取数据

调试输入系统时，直接执行类似下面的命令，然后操作对应的输入设备即可读出数据：

hexdump /dev/input/event0

在开发板上执行上述命令之后，点击按键或是点击触摸屏，就会打印以下信息：



上图中的type为3，对应EV\_ABS；code为0x35对应ABS\_MT\_POSITION\_X；code为0x36对应ABS\_MT\_POSITION\_Y。

上图中还发现有2个同步事件：它的type、code、value都为0。表示电容屏上报了2次完整的数据。

### 7.3 不使用库的应用程序示例

使用GIT下载所有源码后，本节源码位于如下目录：

01\_all\_series\_quickstart\

04\_嵌入式Linux应用开发基础知识\source\11\_input\

01\_app\_demo\

有些同学反馈：老师，你不从0写代码，我都不知道怎么写程序了。基础好的同学直接看这个文档，我录完这个视频后会补齐这个文档，你能看到这部分手册时肯定已经齐全了。

#### 7.3.1 输入系统支持完整的API操作

支持这些机制：阻塞、非阻塞、POLL/SELECT、异步通知。

注意：如果你想深入理解上述机制，需要学习以下内容：

《第5篇 嵌入式Linux驱动开发基础知识》

[《第十九章 驱动程序基石》](#_第十九章_驱动程序基石)

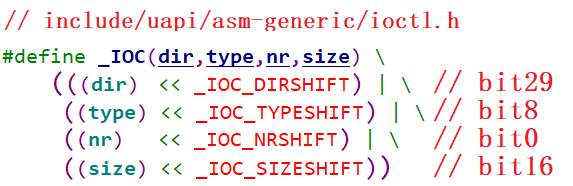
作为APP开发人员，即使没有深入理解这些机制，也是可以编写出程序的。

#### 7.3.3 获取设备信息

通过ioctl获取设备信息，ioctl的参数如下：

int ioctl(int fd, unsigned long request, ...);

有些驱动程序对request的格式有要求，它的格式如下：



比如dir为\_IOC\_READ(即2)时，表示APP要读数据；为\_IOC\_WRITE(即4)时，表示APP要写数据。

size表示这个ioctl能传输数据的最大字节数。

type、nr的含义由具体的驱动程序决定。

比如要读取输入设备的evbit时，ioctl的request要写为“EVIOCGBIT(0, size)”，size的大小可以由你决定：你想读多少字节就设置为多少。这个宏的定义如下：



### 电阻屏数据

Linux驱动程序中，会上报触点的X、Y数据，注意：这不是LCD的坐标值，需要APP再次处理才能转换为LCD坐标值。

对应的input\_event结构体中，“type、code、value”如下：

按下时：

EV\_KEY BTN\_TOUCH 1 /\* 按下 \*/

EV\_ABS ABS\_PRESSURE 1 /\* 压力值，可以上报，也可以不报，可以是其他压力值 \*/

EV\_ABS ABS\_X x\_value /\* X坐标 \*/

EV\_ABS ABS\_Y y\_value /\* Y坐标 \*/

EV\_SYNC 0 0 /\* 同步事件 \*/

松开时：

EV\_KEY BTN\_TOUCH 0 /\* 松开 \*/

EV\_ABS ABS\_PRESSURE 0 /\* 压力值，可以上报，也可以不报 \*/

EV\_SYNC 0 0 /\* 同步事件 \*/

### 电容屏数据

参考文档：Linux内核Documentation\input\multi-touch-protocol.rst。

电容屏可以支持多点触摸(Multi touch)，驱动程序上报的数据中怎么分辨触点？

这有两种方法：Type A、Type B，这也对应两种类型的触摸屏：

① Type A

该类型的触摸屏不能分辨是哪一个触点，它只是把所有触点的坐标一股脑地上报，由软件来分辨这些数据分别属于哪一个触点。

Type A已经过时，Linux内核中都没有Type A的源码了。

② Type B

该类型的触摸屏能分辨是哪一个触点，上报数据时会先上报触点ID，再上报它的数据。

具体例子如下，这是最简单的示例，使用场景分析来看看它上报的数据。

当有2个触点时(type, code, value)：

EV\_ABS ABS\_MT\_SLOT 0 // 这表示“我要上报一个触点信息了”，用来分隔触点信息

EV\_ABS ABS\_MT\_TRACKING\_ID 45 // 这个触点的ID是45

EV\_ABS ABS\_MT\_POSITION\_X x[0] // 触点X坐标

EV\_ABS ABS\_MT\_POSITION\_Y y[0] // 触点Y坐标

EV\_ABS ABS\_MT\_SLOT 1 // 这表示“我要上报一个触点信息了”，用来分隔触点信息

EV\_ABS ABS\_MT\_TRACKING\_ID 46 // 这个触点的ID是46

EV\_ABS ABS\_MT\_POSITION\_X x[1] // 触点X坐标

EV\_ABS ABS\_MT\_POSITION\_Y y[1] // 触点Y坐标

EV\_SYNC SYN\_REPORT 0 // 全部数据上报完毕

当ID为45的触点正在移动时：

EV\_ABS ABS\_MT\_SLOT 0 // 这表示“我要上报一个触点信息了”，之前上报过ID，就不用再上报ID了

EV\_ABS ABS\_MT\_POSITION\_X x[0] // 触点X坐标

EV\_SYNC SYN\_REPORT 0 // 全部数据上报完毕

松开ID为45的触点时(在前面slot已经被设置为0，这里这需要再重新设置slot，slot就像一个全局变量一样：如果它没变化的话，就无需再次设置)：

// 刚刚设置了ABS\_MT\_SLOT为0，它对应ID为45，这里设置ID为-1就表示ID为45的触点被松开了

EV\_ABS ABS\_MT\_TRACKING\_ID -1

EV\_SYNC SYN\_REPORT 0 // 全部数据上报完毕

最后，松开ID为46的触点：

EV\_ABS ABS\_MT\_SLOT 1 // 这表示“我要上报一个触点信息了”，在前面设置过slot 1的ID为46

EV\_ABS ABS\_MT\_TRACKING\_ID -1 // ID为-1，表示slot 1被松开，即ID为46的触点被松开

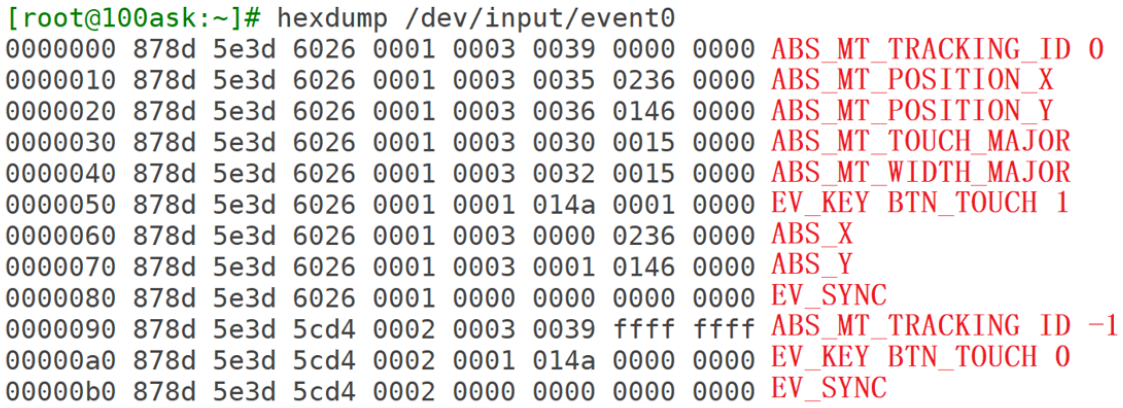
EV\_SYNC SYN\_REPORT // 全部数据上报完毕

**电容屏的实验数据**

假设你的开发板上电容屏对应的设备节点是/dev/input/event0，执行以下命令：

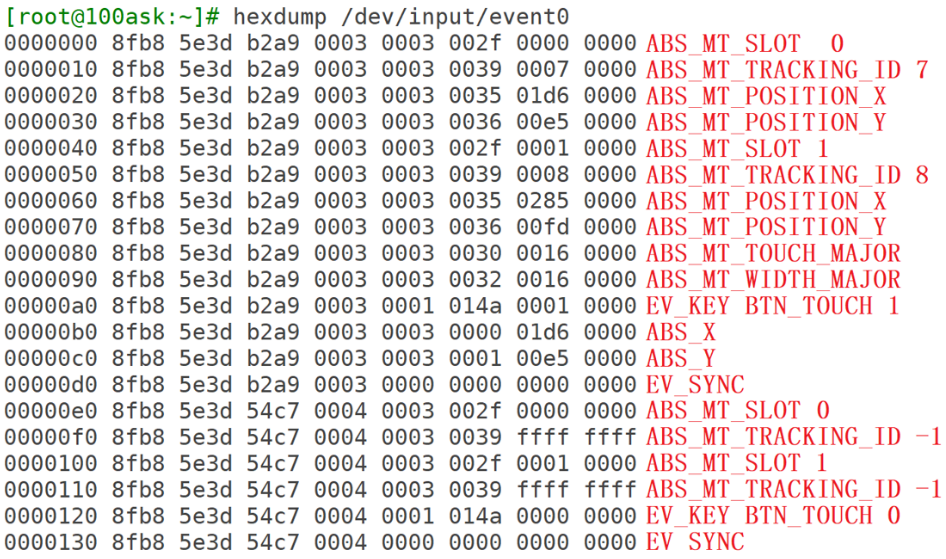
hexdump /dev/input/event0

然后用一个手指点击触摸屏，得到类似如下的数据：



在上面的数据中，为了兼容老程序，它也上报了ABS\_X、ABS\_Y数据，电阻触摸屏就是使用这类型的数据。所以基于电阻屏的程序，也可以用在电容屏上。

使用两个手指点击触摸屏时，得到类似如下的数据：



为了兼容老程序，它也上报了ABS\_X、ABS\_Y数据，但是只上报第1个触点的数据。

### 触屏驱动上报数据的解析

一般使用 tslib 库来读取/解析驱动上传的原始数据（input/eventx里的type\code\value），通过tslib来读取触屏数据写应用就好了

具体看 嵌入式Linux应用开发完全手册V4.0\_韦东山全系列视频文档-IMX6ULL开发板 里面 7.5 tslib 一节即可。