全志科技
Allwinner Technology

Sunxi input 子系统

使用说明文档

1.0

2017.05.14

文档履历

版本号	日期	制/修订人	内容描述
1.0	2017.05.14		

目录

1. 前言

1.1 编写目的

介绍 Linux 内核中 input 子系统的接口及使用方法，为 input 设备驱动的编写者提供参考。

1.2 适用范围

适用于 Allwinnertech 的 AW1718、AW1728、AW1721 系列平台。

1.3 相关人员

input 设备驱动的开发/维护人员。

2. 子系统介绍

2.1 整体框架介绍

Linux 输入设备种类繁杂，常见的包括有按键、键盘、触摸屏、鼠标、摇杆等等，Linux 内核为了统一处理不同的输入设备，设计实现了一个对于上层应用统一的试图抽象层，即输入子系统。在 linux 中，输入子系统作为一个模块存在，将输入子设备的共同性抽象出来，向上提供统一的接口函数，这样，就能够使输入子设备的事件通过输入子系统发给用户层应用程序，用户层应用程序也可以通过输入子系统通知驱动程序实现某项功能。

input 子系统属于内核空间，共分为三层，分别为：事件处理层（EventHandler）、核心层（InputCore）和设备驱动层（InputDriver）。如图所示：

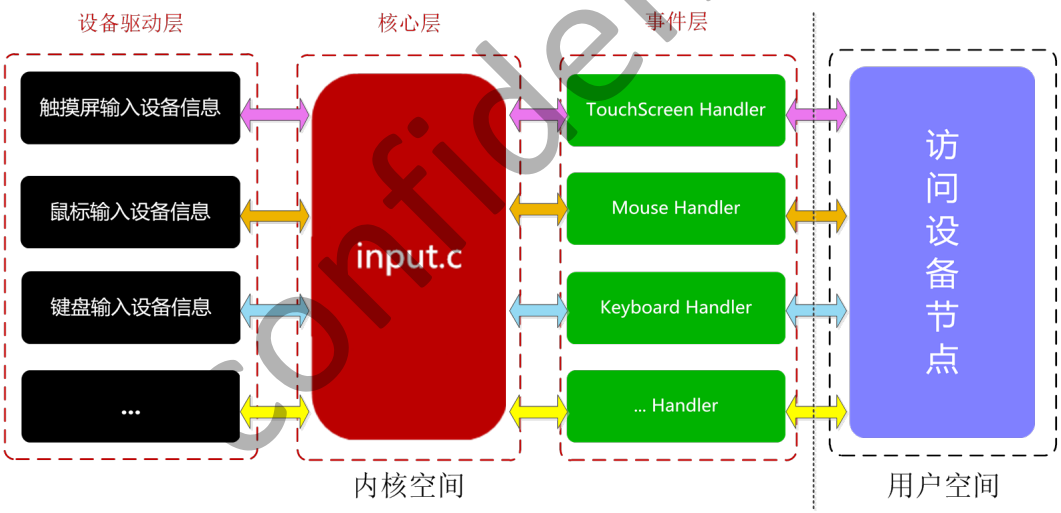

图 1: 输入子系统框架图

其中每个层对应的意义如下:

- 驱动层：主要实现是驱动开发人员，主要负责的任务是针对不同的设备创建相应的 input_dev，然后通过调用通用接口上报事件。

- 核心层：由内核实现，主要负责实现驱动所需要的各种调用接口，驱动层上报数据将经过核心层，核心层再将数据上报给事件层。
- 事件层：接收来自核心层的事件，并选择对应的 handler（处理程序）去处理，目的是将数据复制到用户空间，即在/dev/input/下生成相应的设备节点（设备文件）。
- 用户空间：Linux 系统中，用户空间将所有的设备都当文件来处理，即在用户空间访问设备节点，输入子设备节点为/dev/input/device/event*。

input 子系统的三个层中，核心层和事件处理层都是内核已经完成的，对于设备驱动开发都集中在驱动层。

2.2 工作流程流程介绍

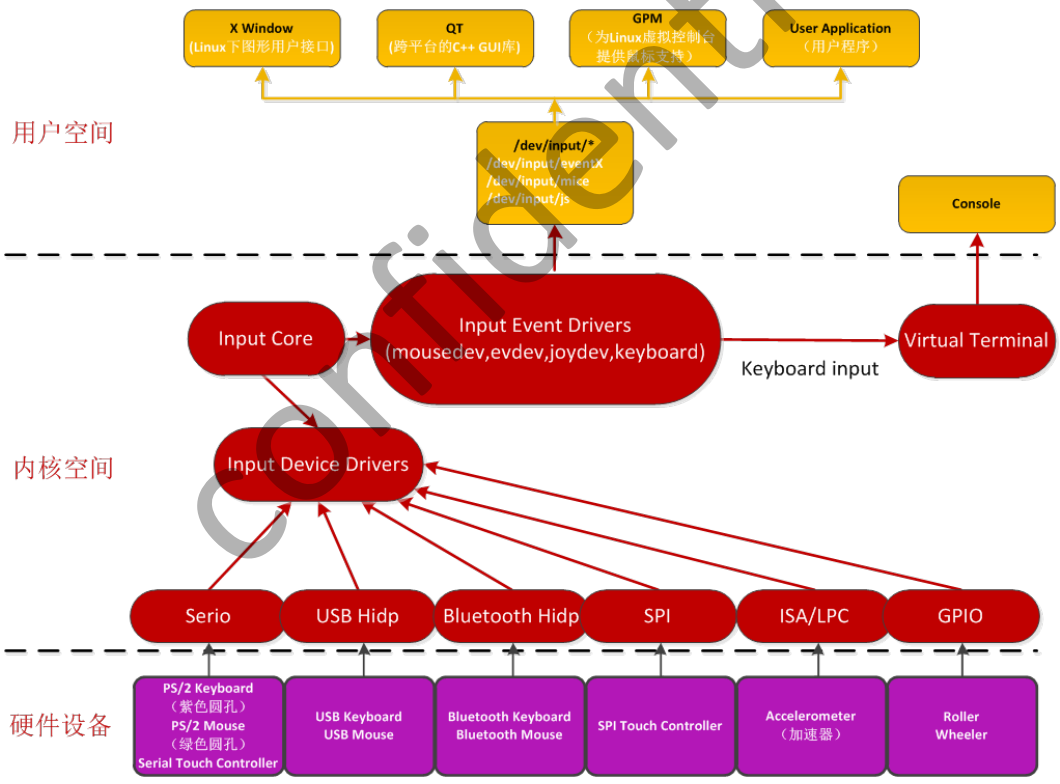

图 2: 工作流程介绍图

上图以 USB 键盘为例：当 USB 键盘连接上来后，USB 键盘的驱动程序（驱动层）把相应的事件交给核心层，然后核心层再把事件交给输入事件驱动（事件处理层），输入事

件驱动又把事件反馈到相应的设备文件中（/dev/input/*），最后应用程序就可以通过标准读写函数去操作这些设备文件了。

3. driver 编写流程

3.1 menuconfig 配置

打开输入子系统，menuconfig 需要进入 Device Driver->Input device support 里面选中大概如下配置选项：

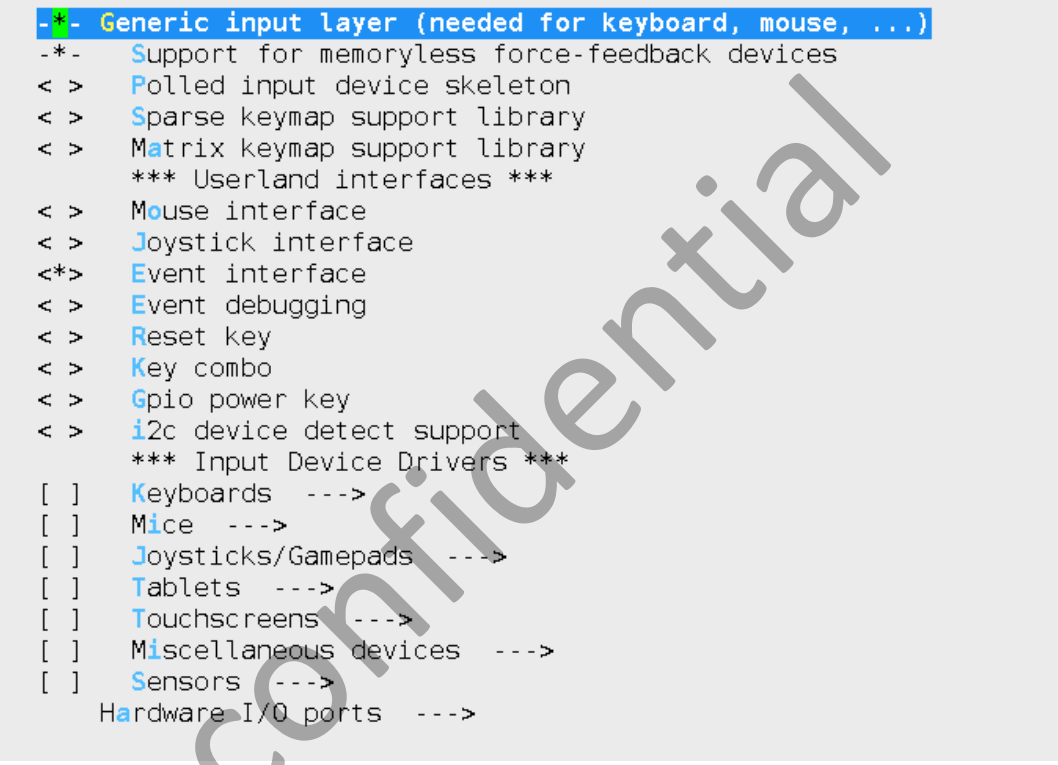

图 3: menuconfig 配置图

3.2 驱动编写步骤

注册输入设备驱动共分五步，且均是在驱动层中完成，如下所示：

1. 调用 input_allocate_device 函数：分配一个 input_dev 结构体。

2. 得到 input_dev 结构体指针后还需要为这个结构体增加对应设备的特性：主要是事

件类型和事件键值。

- 事件类型：表示该输入设备能上报的事件的具体类型，例如 TP 能上报绝对坐标类型事件，按键能上报按键类型的事件等，另外输入设备都要支持同步事件类型，用于当某个事件上报完毕后，告诉系统事件上报完毕。具体的事件类型可以参照内核 linux/include/input.h，以下是系统支持的事件类型：

```
#define EV_SYN          0x00  //同步事件类型
#define EV_KEY          0x01   //按键事件类型
#define EV_REL          0x02   //相对坐标事件类型
#define EV_ABS          0x03   //绝对坐标事件类型
#define EV_MSC          0x04   //杂项事件类型
#define EV_SW           0x05   //开关事件类型
#define EV_LED          0x11   //按键/设备灯事件类型
#define EV_SND          0x12   //声音/警报事件类型
#define EV_REP          0x14   //重复事件类型
#define EV_FF           0x15   //力反馈事件类型
#define EV_PWR          0x16   //电源事件类型
#define EV_FF_STATUS        0x17   //力反馈状态事件类型
#define EV_MAX          0x1f   //事件类型最大个数和提供位掩码支持
#define EV_CNT          (EV_MAX+1)
```

- 事件键值：表示具体的事件类型支持的具体键值，比如事件类型为 EV_KEY，可以支持上报 BINT_0、KEY_A 等按键键值，如果事件类型为 EV_ABS，可以支持 ABS_X、ABS_Y 坐标键值等，这些都需要初始化的时候告知系统，具体请参考 include/uap1/linux/input.h。

3. 调用 input_register_device 函数：注册此输入设备到 input 子系统中；

4. 在 input 设备发生输入操作时，提交锁发生的时间以及对应的键值/坐标等状态。

5. 在驱动退出函数里面卸载 input 设备。

input 子系统中输入设备驱动的核心工作就是：向系统申请要上报的输入子事件类型和输入子事件键值，并正确报告这个事件，而不用关心事件如何处理，因为事件处理由事件处理层来完成。

3.3 内核常用 API 介绍

包含头文件：#include <linux/input.h>

3.3.1 input_allocate_device

- 【函数原型】：struct input_dev *input_allocate_device(void)
- 【功能描述】：在内存中为输入设备结构体分配空间，并对其主要成员进行初始化。
- 【参数说明】：NULL
- 【返回值】：成功返回一个指向 input_dev 类型的指针，失败返回 null。

3.3.2 input_free_device

- 【函数原型】：void input_free_device(struct input_dev *dev)
- 【功能描述】：对 input_allocate_device 申请的 input_dev 结构体进行释放，一般用于 input_register_device 调用失败;
- 【参数说明】：dev：input_dev 结构体指针
- 【返回值】：无

3.3.3 input_register_device

- 【函数原型】：int input_register_device(struct input_dev *dev)
- 【功能描述】：将 input_dev 结构体注册到输入子系统中
- 【参数说明】：dev：input_dev 结构体指针，由 input_allocate_device 分配
- 【返回值】：成功返回 0，失败返回非 0 值

3.3.4 void input_unregister_device(struct input_dev *dev)

- 【函数原型】: int input_unregister_device(struct input_dev *dev)
- 【功能描述】: 将注册到输入子系统中的 input_dev 结构体从系统中卸载
- 【参数说明】: dev: 通过 input_register_device 注册到系统中的 input_dev 结构体指针
- 【返回值】: 无

3.3.5 常用的用于上报事件的函数

- 【函数原型】: void input_event(struct input_dev *dev,unsigned int type, unsigned int code, int value)
- 【功能描述】: 上报一个 input 事件，上报事件通用的函数
- 【参数说明】: dev: 上报的输入设备 type: 上报的事件类型 code: 上报的事件键值 value: 上报的事件数据
- 【返回值】: 无

- 【函数原型】: void input_report_key(struct input_dev *dev, unsigned int code, int value)
- 【功能描述】: 提交按键事件的函数
- 【参数说明】: dev: 上报的输入设备 code: 上报的事件键值 value: 上报的事件数据
- 【返回值】: 无

- 【函数原型】: void input_report_rel(struct input_dev *dev, unsigned int code, int value)
- 【功能描述】: 提交相对坐标事件的函数
- 【参数说明】: dev: 上报的输入设备 code: 上报的事件键值 value: 上报的事件数据
- 【返回值】: 无

- 【函数原型】: void input_report_abs(struct input_dev *dev, unsigned int code, int value)
- 【功能描述】: 提交绝对坐标事件的函数
- 【参数说明】: dev: 上报的输入设备 code: 上报的事件键值 value: 上报的事件数据
- 【返回值】: 无

应该注意的是，在提交输入设备的事件后必须用下列方法使事件同步，让它告知 input 系统，设备驱动已经发出了一个完整的报告:

- 【函数原型】：static inline void input_sync(struct input_dev *dev)
- 【功能描述】：提交绝对坐标事件的函数
- 【参数说明】：dev：要上报的输入设备
- 【返回值】：无

注：其他的 API 请参照内核文件，以 linux-3.10 为例，请参照 include/linux/input.h 和 drivers/input/input.c。

3.4 驱动示例介绍

以下驱动示例，支持上报的事件类型为按键事件，支持上报的事件键值为 KEY_L、KEY_S、KEY_ENTER，并通过定时器间隔一秒重复上报上面的事件键值。

```
#include <linux/module.h>
#include <linux/init.h>
#include <linux/input.h>
#include <linux/delay.h>
#include <linux/slab.h>
#include <linux/interrupt.h>
#include <linux/keyboard.h>
#include <linux/ioport.h>
#include <asm/irq.h>
#include <asm/io.h>
#include <linux/timer.h>
#include <linux/clk.h>
#include <linux/irq.h>
#include <linux/of_platform.h>
#include <linux/of_irq.h>
#include <linux/of_address.h>

static struct input_dev *buttons_dev;
static struct timer_list buttons_timer;
```



```
static void buttons_timer_function(unsigned long data)
{
    input_event(buttons_dev, EV_KEY, KEY_L, 0);
    input_sync(buttons_dev);
    input_event(buttons_dev, EV_KEY, KEY_L, 1);
    input_sync(buttons_dev);

    input_event(buttons_dev, EV_KEY, KEY_S, 0);
    input_sync(buttons_dev);
    input_event(buttons_dev, EV_KEY, KEY_S, 1);
    input_sync(buttons_dev);

    input_event(buttons_dev, EV_KEY, KEY_ENTER, 0);
    input_sync(buttons_dev);
    input_event(buttons_dev, EV_KEY, KEY_ENTER, 1);
    input_sync(buttons_dev);

    mod_timer(&buttons_timer, jiffies+HZ);
}

static int buttons_init(void)
{
    /* 1. 分配一个 input_dev 结构体 */
    buttons_dev = input_allocate_device();;

    /* 2. 增加设备特性 */
    buttons_dev->name = "keys";
    /* 2.1 支持哪种类型的事件，即事件类型 */
    set_bit(EV_KEY, buttons_dev->evbit);
    set_bit(EV_REP, buttons_dev->evbit);

    /* 2.2 支持哪些按键，即事件键值 */
    set_bit(KEY_L, buttons_dev->keybit);
    set_bit(KEY_S, buttons_dev->keybit);
```



```
    set_bit(KEY_ENTER, buttons_dev->keybit);

    /* 3. 注册 */
    input_register_device(buttons_dev);

    /* 4. 硬件相关的操作 */
    init_timer(&buttons_timer);
    buttons_timer.function = buttons_timer_function;
    add_timer(&buttons_timer);

    printk(">>>>vedic>> timeout=1s call buttons_timer_function\n");
    mod_timer(&buttons_timer, jiffies+5*HZ);

    return 0;
}

static void buttons_exit(void)
{
    del_timer(&buttons_timer);
    input_unregister_device(buttons_dev);
}

module_init(buttons_init);
module_exit(buttons_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("fuzhaoke");
MODULE_DESCRIPTION("test input key_board");
```


4. 日常调试手段

4.1 /proc/bus/input/devices

我们怎样才能知道哪个设备节点对应的是我们自己写的那个 input? 因为与这些设备节点对应的设备信息记录在/proc/bus/input/devices 中，所以可通过 cat /proc/bus/input/devices 来查看，结果如下:

```
# cat /proc/bus/input/devices
I: Bus=0000 Vendor=0000 Product=0000 Version=0000
N: Name="keys"              /* 输入子设备名字 */
P: Phys=
S: Sysfs=/devices/virtual/input/input6
U: Uniq=
H: Handlers=kbd event6   /* 输入子设备对应 event 节点 */
B: PROP=0
B: EV=100003
B: KEY=4090000000
.....
```


4.2 查看上报的事件

通过上面的手段已经可以知道具体的 input 设备是输入是属于哪个文件节点，即对应/dev/input/里面的哪个 event 节点，即可用以下 hexdump 工具对上报事件进行观察:

```
# hexdump /dev/input/event*
```

hexdump 列出来的值一般有 9 列:

1. 前五列为事件时间;
2. 第六列为事件类型，即 type;
3. 第七列为事件键值，即 code;

4. 第八和第九列为事件数据，即 value。

下面以按键类型和绝对坐标类型的输入设备讲解如何查看上报的事件：

- 观察按键类型的输入设备

```
# hexdump /dev/input/event
0000000 f6a6 4e15 154b 0006 0001 0004 0001 0000
0000010 f6a6 4e15 1557 0006 0000 0000 0000 0000
0000020 f6a6 4e15 8510 0008 0001 0004 0000 0000
0000030 f6a6 4e15 8517 0008 0000 0000 0000 0000
```

输入事件信息如下：

1. 第六列为事件类型，即 type，1 表示 EV_KEY，0 表示 EV_SYN;
2. 第七列为事件键值，即 code，0004 表示 KEY_3，同步的时候为 0000;
3. 第八列 + 第九列为事件数据，即 value。

- 观察绝对坐标类型的输入设备

```
# hexdump /dev/input/event
0000250 f832 4e15 c502 0006 0003 0039 0020 0000
0000260 f832 4e15 c50f 0006 0003 0030 0004 0000
0000270 f832 4e15 c514 0006 0003 0035 0263 0000
0000280 f832 4e15 c519 0006 0003 0036 01fd 0000
0000290 f832 4e15 c520 0006 0001 014a 0001 0000
00002a0 f832 4e15 c525 0006 0003 0000 0263 0000
00002b0 f832 4e15 c52b 0006 0003 0001 01fd 0000
00002c0 f832 4e15 c530 0006 0000 0000 0000 0000
00002d0 f832 4e15 be99 0007 0003 0039 ffff ffff
00002e0 f832 4e15 bea5 0007 0001 014a 0000 0000
00002f0 f832 4e15 bea8 0007 0000 0000 0000 0000
```

输入事件信息如下：

1. 第六列为事件类型，即 type，3 表示 EV_ABS，绝对坐标事件;

2. 第七列为事件键值，即 code，其中数值意义如下：
0039 --> ABS_MT_TRACKING_ID;
0030 --> ABS_MT_TOUCH_MAJOR;
0035 --> ABS_MT_POSITION_X;
0036 --> ABS_MT_POSITION_Y
014a --> BTN_TOUCH
3. 第八列 + 第九列为事件数据，即 value。

5. FAQ

5.1 一个输入设备可否同时支持不同的事件类型?

答：可以。

5.2 input_sync 的作用是什么?

答：用于告诉系统一个输入事件上报完毕，起到同步作用。

5.3 一个输入事件的上报的数据中，存有 value 值，请问 value 的值有什么　义?

答：可以简单的理解为私有数据，可以跟应用层协议上报 value 值的意义，例如上面的例子中，0 表示按下，1 表示松开。

6. Declaration

This document is the original work and copyrighted property of Allwinner Technology (“Allwinner”). Reproduction in whole or in part must obtain the written approval of Allwinner and give clear acknowledgement to the copyright owner. The information furnished by Allwinner is believed to be accurate and reliable. Allwinner reserves the right to make changes in circuit design and/or specifications at any time without notice. Allwinner does not assume any responsibility and liability for its use. Nor for any infringements of patents or other rights of the third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Allwinner. This datasheet neither states nor implies warranty of any kind, including fitness for any particular application.tates nor implies warranty of any kind, including fitness for any particular application.

