

jscese

知其白 守其黑 為天下式 \_Read The Fucking Source Code

个人资料

南、烟

关注

原创100

粉丝252

喜欢240

评论135

等级：博客6

访问量：54万+

积分：6553

排名：4486

二手车玛莎拉蒂

文章搜索

博客专栏

Android——4.2 Vold

文章：8篇

阅读：33768

Android——4.2\_3G移植之路\_RIL

文章：6篇

阅读：38783

Linux/Android - Input 系统

文章：8篇

阅读：48627

文章分类

【Android — 应用】(25)

【Android — 框架】(13)

【Android — 机制】(24)

【Android — 驱动】(8)

【Android — 编译】(5)

【Linux — Driver】(7)

【Embedded】(25)

【Ubuntu】(10)

【Workplace】(3)

【Network】(2)

【C/C++】(6)

Linux/Android——input子系统核心 (三)

标签：inputinput\_registerinput\_handlerdevlist

2014年12月26日 15:10:075103人阅读评论(0)收藏

分类：【Android — 驱动】(7)

版权声明：免责声明：本人在此发文（包括但不限于汉字、拼音、拉丁字母）均为随意敲击键盘所出，用于检验本人电脑键盘录入、屏幕机械、光电性能，并不代表本人局部或全部同意、支持或者反对观点。如需要详查请直接与键盘生产厂商法人代表联系。挖井挑水无水表，不快莫怪。https://blog.csdn.net/jscese/article/details/42123673

之前的博客有涉及到linux的input子系统，这里学习记录一下input模块。

input子系统，作为管理输入设备与系统进行交互的中枢，任何的输入设备驱动都要通过input向内核注册，常用的输入设备也就是鼠标，键盘，触摸屏。

稍微细分一点整个输入体系，就是 硬件驱动层，input核心中转层，事件处理层。层次之间传递都以event形式，这其中input连接上下层，分别提供接口。

之前有分析usbtouchscreen的驱动，也就是硬件驱动部分，这里简单记录一下input核心中转处理 input。

撰写不易，转载需注明！

<http://blog.csdn.net/jscese/article/details/42123673>

input\_init：

源码位于/kernel/drivers/input/input.c，模块初始调用口subsys\_initcall(input\_init)，由kernel启动的时候由kernel\_init——>do\_basic\_setup();——>do\_initcalls调用到，这个启动逻辑，后面会去学习一下，这里首先调用到初始函数：

```
[objc]
1. static int __init input_init(void)
2. {
3.     int err;
4.
5.     err = class_register(&input_class); //注册input class，可在/sys/class下看到对应节点文件
6.     if (err) {
7.         pr_err("unable to register input_dev class\n");
8.         return err;
9.     }
10.
11.     err = input_proc_init(); //proc fs的下的一些初始操作，函数原型在input.c，可查看/proc/bus/input
12.     if (err)
13.         goto fail1;
14.
15.     err = register_chrdev(INPUT_MAJOR, "input", &input_fops); // 注册input字符设备，主节点为INPUT_MAJOR==
16.     if (err) { //以input_fops里看注册函数，注册到/dev/input
17.         pr_err("unable to register char major %d", INPUT_MAJOR);
18.         goto fail2;
19.     }
20.
21.     return 0;
22.
23. fail2: input_proc_exit();
24. fail1: class_unregister(&input_class);
25.     return err;
26. }
```

https://blog.csdn.net/jscese/article/details/42123673

1/8

- 【Video-Display】 (1)
- 【Unity3D】 (3)
- 【自我修养】 (3)
- 【视觉-OpenGL-CL-CV】 (9)

文章存档

2018年3月 (1)

2017年10月 (1)

2017年3月 (1)

2017年1月 (1)

2016年10月 (3)

展开

阅读排行

Ubuntu——grub rescue 主...	(23036)
Android——systrace使用分析	(16254)
Unity3D——android device...	(15225)
Android——编译体系中的【...	(13981)
Linux/Android——输入子系...	(11914)
Android——内存管理-lowm...	(11668)
Android——TV真机调试apk...	(11213)
OpenGL ES 入门 (一)	(10502)
Linux trace使用入门	(10272)
Android——4.2 - 3G移植之...	(10209)

最新评论

Android——RIL 机制源码...  
zisuchen : 你好问下，如果我是在mtk平台上移植要怎么移植，mtk有改动rild改成mtk的代码

Android——build.pr...  
LosingCarryJie : 虽然是篇老博客，但是帮助我弄懂了源码！感谢！

extern “C” 在C/C+...  
谗阿星 : 看了几篇博客还是比较晕，看了你的明白了，谢谢楼主分享，向你学习

OpenCV&OpenCL...  
xavier19841016 : 看到你这个免责声明，我笑了

Android—— 4.2 Vol...  
Winston\_jory : 楼主好⑥，就连版权声明都写得这么形象。

OpenGL ES 入门 (一)  
imwoohan : 期待下一期

OpenGL ES 入门 (一)  
imwoohan : 赞一个。

4月跳槽路  
UPON--知道个P : 唉！我也有过类似的遭遇！没法，打铁还需自身硬！练技术中。。。练完准备出山。

Unity3D——MonoBeha...  
南丶烟 : [reply]javi9111[/reply] 我也没在TV大厂待过，回答不了你的问题，在天朝有政府...

Unity3D——MonoBeha...  
南丶烟 : [reply]javi9111[/reply] 看你前面写了一大堆。。你想说啥--。uni...

1点点停止加盟

联系我们

请扫描二维码联系客服

这就是最开始的初始化过程了。

可以看下注册方法函数：

```
[objc]
1. static const struct file_operations input_fops = {
2.     .owner = THIS_MODULE,
3.     .open = input_open_file,
4.     .llseek = noop_llseek,
5. };
```

这里面关注open file方法即可，后面分析。

input.c中还有很多其它的接口以及全局数据，后面陆续联通，先从设备驱动最先调用到input\_register\_device

input\_register\_device:

```
[objc]
1. /**
2.  * input_register_device - register device with input core
3.  * @dev: device to be registered
4.  *
5.  * This function registers device with input core. The device must be
6.  * allocated with input_allocate_device() and all it's capabilities
7.  * set up before registering.
8.  * If function fails the device must be freed with input_free_device().
9.  * Once device has been successfully registered it can be unregistered
10.  * with input_unregister_device(); input_free_device() should not be
11.  * called in this case.
12.  */
13.
14. int input_register_device(struct input_dev *dev)
15. {
16.     static atomic_t input_no = ATOMIC_INIT(0);
17.     //这个原子变量，代表总共注册的input设备，每注册一个加1，因为是静态变量，所以每次调用都不会清零的
18.     struct input_handler *handler;
19.     const char *path;
20.     int error;
21.
22.     __set_bit(EV_SYN, dev->evbit); //EN_SYN 这个是设备都要支持的事件类型，所以要设置
23.
24.     /*
25.      * If delay and period are pre-set by the driver, then autorepeating
26.      * is handled by the driver itself and we don't do it in input.c.
27.      */
28.     // 这个内核定时器是为了重复按键而设置的
29.     init_timer(&dev->timer);
30.     if (!dev->rep[REP_DELAY] && !dev->rep[REP_PERIOD]) {
31.         dev->timer.data = (long) dev;
32.         dev->timer.function = input_repeat_key;
33.         dev->rep[REP_DELAY] = 250;
34.         dev->rep[REP_PERIOD] = 33;
35.         //如果没有定义有关重复按键的相关值，就用内核默认的
36.     }
37.
38.     if (!dev->getkeycode)
39.         dev->getkeycode = input_default_getkeycode;
40.     if (!dev->setkeycode)
41.         dev->setkeycode = input_default_setkeycode;
42.     //以上设置的默认函数由input核心提供
43.     dev_set_name(&dev->dev, "input%d",
44.                 (unsigned long) atomic_inc_return(&input_no) - 1);
45.     //设置input_dev中device的名字，这个名字会在/class/input中出现
46.     error = device_add(&dev->dev);
47.     //将device加入到linux设备模型中去
48.     if (error)
49.         return error;
50.
51.     path = kobject_get_path(&dev->dev.kobj, GFP_KERNEL);
52.     printk(KERN_INFO "input: %s as %s\n",
53.            dev->name ? dev->name : "Unspecified device", path ? path : "N/A");
54.     kfree(path);
```



webmaster@csdn.net

400-660-0108

QQ客服 客服论坛

关于 招聘 广告服务 百度

©1999-2018 CSDN版权所有

京ICP证09002463号

经营性网站备案信息

网络110报警服务

中国互联网举报中心

北京互联网违法和不良信息举报中心

```

55.     //这个得到路径名称，并打印出来
56.     error = mutex_lock_interruptible(&input_mutex);
57.     if (error) {
58.         device_del(&dev->dev);
59.         return error;
60.     }
61.
62.     list_add_tail(&dev->node, &input_dev_list);
63.     // 将新分配的input设备连接到input_dev_list链表上
64.     list_for_each_entry(handler, &input_handler_list, node)
65.         input_attach_handler(dev, handler);
66.     //遍历input_handler_list链表，配对 input_dev 和 input_handler
67.     //input_attach_handler 这个函数是配对的关键，下面将详细分析
68.     input_wakeup_procfs_readers();
69.     // 和proc文件系统有关，暂时不考虑
70.     mutex_unlock(&input_mutex);
71.
72.     return 0;
73. }

```

可以看到前面都是一些初始设置，加入到input.c 的全局input\_dev 链表里面，同时下面就行匹配对应handler，时候需要遍历 handler 链表：

[objc]

```

1. static LIST_HEAD(input_dev_list);
2. static LIST_HEAD(input_handler_list);

```

可以看到用到了一个list\_for\_each\_entry，刚开始看到还没看懂，这是一个宏定义，原型是在/kernel/include/linux :

[objc]

```

1. /**
2.  * list_for_each_entry - iterate over list of given type
3.  * @pos:    the type * to use as a loop cursor.
4.  * @head:   the head for your list.
5.  * @member: the name of the list_struct within the struct.
6.  */
7. #define list_for_each_entry(pos, head, member) \
8.     for (pos = list_entry((head)->next, typeof(*pos), member); \
9.          &pos->member != (head); \
10.          pos = list_entry(pos->member.next, typeof(*pos), member))

```

input\_attach\_handler(dev, handler)则是匹配这个要注册dev的handler：

[objc]

```

1. static int input_attach_handler(struct input_dev *dev, struct input_handler *handler)
2. {
3.     const struct input_device_id *id;
4.     int error;
5.
6.     id = input_match_device(handler, dev); //返回匹配的id，类型是struct input_device_id
7.     if (!id)
8.         return -ENODEV;
9.
10.    error = handler->connect(handler, dev, id); //配对成功调用handler的connect函数，这个函数在事件处理器中定义，主要生成一个input_handler结构，并初始化，还生成一个事件处理器相关的设备结构
11.    if (error && error != -ENODEV)
12.        pr_err("failed to attach handler %s to device %s, error: %d\n",
13.               handler->name, kobject_name(&dev->dev.kobj), error);
14.
15.    return error;
16. }

```

可以看下匹配 id 的结构：

[objc]

```

1. struct input_device_id {
2.
3.     kernel_ulong_t flags;
4.
5.     __u16 bustype;
6.     __u16 vendor;
7.     __u16 product;
8.     __u16 version;
9.
10.    kernel_ulong_t evbit[INPUT_DEVICE_ID_EV_MAX / BITS_PER_LONG + 1];
11.    kernel_ulong_t keybit[INPUT_DEVICE_ID_KEY_MAX / BITS_PER_LONG + 1];
12.    kernel_ulong_t relbit[INPUT_DEVICE_ID_REL_MAX / BITS_PER_LONG + 1];
13.    kernel_ulong_t absbit[INPUT_DEVICE_ID_ABS_MAX / BITS_PER_LONG + 1];
14.    kernel_ulong_t mscbit[INPUT_DEVICE_ID_MSC_MAX / BITS_PER_LONG + 1];
15.    kernel_ulong_t ledbit[INPUT_DEVICE_ID_LED_MAX / BITS_PER_LONG + 1];
16.    kernel_ulong_t sndbit[INPUT_DEVICE_ID_SND_MAX / BITS_PER_LONG + 1];
17.    kernel_ulong_t ffbbit[INPUT_DEVICE_ID_FF_MAX / BITS_PER_LONG + 1];
18.    kernel_ulong_t swbit[INPUT_DEVICE_ID_SW_MAX / BITS_PER_LONG + 1];
19.
20.    kernel_ulong_t driver_info;
21. };

```

有两个函数input\_match\_device 以及 下面的 connect需要了解：

## input\_match\_device:

```

[objc]
1. static const struct input_device_id *input_match_device(struct input_handler *handler,
2.                                                         struct input_dev *dev)
3. {
4.     const struct input_device_id *id;
5.     int i;
6.
7.     for (id = handler->id_table; id->flags || id->driver_info; id++) {
8.
9.         if (id->flags & INPUT_DEVICE_ID_MATCH_BUS) //匹配总线id
10.            if (id->bustype != dev->id.bustype)
11.                continue;
12.
13.         if (id->flags & INPUT_DEVICE_ID_MATCH_VENDOR) //匹配生产商id
14.            if (id->vendor != dev->id.vendor)
15.                continue;
16.
17.         if (id->flags & INPUT_DEVICE_ID_MATCH_PRODUCT) //匹配产品id
18.            if (id->product != dev->id.product)
19.                continue;
20.
21.         if (id->flags & INPUT_DEVICE_ID_MATCH_VERSION) //匹配版本
22.            if (id->version != dev->id.version)
23.                continue;
24.
25.         MATCH_BIT(evbit, EV_MAX); //匹配id的evbit和input_dev中evbit的各个位，如果不匹配则continue，数组中
设备
26.         MATCH_BIT(keybit, KEY_MAX);
27.         MATCH_BIT(relbit, REL_MAX);
28.         MATCH_BIT(absbit, ABS_MAX);
29.         MATCH_BIT(mscbit, MSC_MAX);
30.         MATCH_BIT(ledbit, LED_MAX);
31.         MATCH_BIT(sndbit, SND_MAX);
32.         MATCH_BIT(ffbit, FF_MAX);
33.         MATCH_BIT(swbit, SW_MAX);
34.
35.         if (!handler->match || handler->match(handler, dev))
36.             return id;
37.     }
38.
39.     return NULL;
40. }

```

MATCH\_bit 原型：

```

[objc]
1. #define MATCH_BIT(bit, max) \

```

```

2.         for (i = 0; i < BITS_TO_LONGS(max); i++) \
3.             if ((id->bit[i] & dev->bit[i]) != id->bit[i]) \
4.                 break; \
5.         if (i != BITS_TO_LONGS(max)) \
6.             continue;

```

可以看到这么多步的目的除了初始以及添加input\_dev到链表，就是为了去匹配 **input\_handler\_list** 中 **handler**，

匹配的最终是需要比对handler以及input\_dev中的id，其中input\_dev 中的id类型为 **input\_id**：

```

[objc]
1. struct input_id {
2.     __u16 bustype;
3.     __u16 vendor;
4.     __u16 product;
5.     __u16 version;
6. };

```

这跟上面 input\_handler 结构里面的 input\_device\_id 匹配id 变量，来确认 handler！

在最开始的时候就有提到，整个input输入体系，分三个层次，现在的input核心层做的事就是：

**在硬件驱动层调用 input\_register\_device时，往内核注册驱动的同时，根据硬件的相关id去匹配 适用层(input\_handler)！**

这里匹配上之后就会调用对应 input\_handler 的connect 函数。

## input\_handler：

input\_dev 变量代表的是硬件设备，前文[Linux/Android——输入子系统input\\_event传递 \(二\)](#)中有介绍

input\_handler 变量代表的是事件处理器

同样在input.h 中定义：

```

[objc]
1. /**
2.  * struct input_handler - implements one of interfaces for input devices
3.  * @private: driver-specific data
4.  * @event: event handler. This method is being called by input core with
5.  * interrupts disabled and dev->event_lock spinlock held and so
6.  * it may not sleep
7.  * @filter: similar to @event; separates normal event handlers from
8.  * "filters".
9.  * @match: called after comparing device's id with handler's id_table
10. * to perform fine-grained matching between device and handler
11. * @connect: called when attaching a handler to an input device
12. * @disconnect: disconnects a handler from input device
13. * @start: starts handler for given handle. This function is called by
14. * input core right after connect() method and also when a process
15. * that "grabbed" a device releases it
16. * @fops: file operations this driver implements
17. * @minor: beginning of range of 32 minors for devices this driver
18. * can provide
19. * @name: name of the handler, to be shown in /proc/bus/input/handlers
20. * @id_table: pointer to a table of input_device_ids this driver can
21. * handle
22. * @h_list: list of input handles associated with the handler
23. * @node: for placing the driver onto input_handler_list
24. *
25. * Input handlers attach to input devices and create input handles. There
26. * are likely several handlers attached to any given input device at the
27. * same time. All of them will get their copy of input event generated by
28. * the device.
29. *
30. * The very same structure is used to implement input filters. Input core
31. * allows filters to run first and will not pass event to regular handlers
32. * if any of the filters indicate that the event should be filtered (by
33. * returning %true from their filter() method).
34. *
35. * Note that input core serializes calls to connect() and disconnect()
36. * methods.

```

```
37.  */
38. struct input_handler {
39.
40.     void *private;
41.
42.     void (*event)(struct input_handle *handle, unsigned int type, unsigned int code, int value);
43.     bool (*filter)(struct input_handle *handle, unsigned int type, unsigned int code, int value);
44.     bool (*match)(struct input_handler *handler, struct input_dev *dev);
45.     int (*connect)(struct input_handler *handler, struct input_dev *dev, const struct input_device_id *
//上面就是调用这个函数指针
46.     void (*disconnect)(struct input_handle *handle);
47.     void (*start)(struct input_handle *handle);
48.
49.     const struct file_operations *fops;
50.     int minor;
51.     const char *name;
52.
53.     const struct input_device_id *id_table; //这个就是上面说到的 会跟input_dev中的input_id 比对 id项的
54.
55.     struct list_head h_list;
56.     struct list_head node;
57. };
```

这个结构详细的含义，注释有。

这个结构里面暂时只需要理解的：

注册input\_dev，在事件处理数据链表里面匹配上input\_handler,就会调用其\*connect函数指针进行将input\_dev跟input\_handler进行绑定，后续的运作事件的handler处理将会走这个input\_handler\*event！

在上篇input\_event传递中最后调用到event阶段。

这里简单记录到这里，下篇介绍input\_handler的处理机制~

- 上一篇 Linux/Android——输入子系统input\_event传递 (二)
- 下一篇 Linux/Android——input\_handler之evdev (四)




北大青鸟 北大青鸟学费一览 一点点加盟费 车展 婚纱照前十名 婚纱摄影排名 孕妇照

写下你的评论...

**Android Framework-----之Input子系统** wangkaiblog 2013-09-27 11:25:05   
<http://www.cnblogs.com/haiming/p/3318614.html> 下面这是基于Android4.2代码的关于Input子系统的笔记。在这篇中，只涉及Android相...

**Android之Input子系统事件分发流程** tankai19880619 2013-11-29 08:49:08   
一、Android4.2系统服务侧 1.服务端注册过程 frameworks/base/core/java/android/view/ViewRootImpl.java public t...

input子系统整体框架

 zjngogo 2015-06-04 16:24:24


2.模块结构 下图是input输入子系统框架，输入子系统linux层由输入子系统核心层（ Core层 ），驱动层和事件处理层（ ndler ）三部份组成。Android层操作input子...

android input子系统之三：事件层

 lixuehui848 2016-01-22 16:04:25

四 事件层 struct input\_dev物理输入设备的基本数据结构,包含设备相关的一些信息 structinput\_handler 事件处理结构体,怎么处理事件的逻辑 struct in...

android input系统

 bsxiaomage 2015-04-06 21:50:50

linux内核的input子系统是对分散的，多种不同类别的输入设备(如键盘，鼠标，跟踪球，操纵杆，触摸屏，加速计和手写符号设备进行统一处理的一层抽象，就是在字符设备驱动上抽象出的一层。input...

知网论文查重入口

在知网上查重论文一般需要多长时间

百度广告



Andriod Input子系统框架

 hongwazi\_2010 2015-01-25 20:27:09


原文地址：http://www.cnblogs.com/haiming/p/3318614.html 下面这是基于Android4.2代码的关于Input子系统的原创文章，这篇笔记中，只涉及And...

Android Input子系统浅谈

 tiantangniaochao 2016-01-11 15:30:34

Android Input子系统浅谈本文主要讲解[Android Input 子系统][6]，我会从一下几个方面讲解： linux kernel的input子以触摸屏驱动为例讲解内核input...

input输入子系统整体流程

 mike8825 2016-03-02 19:37:25

input输入子系统整体流程 本节分析input子系统在内核中的实现，包括输入子系统（ Input Core ），事件处理层（ Event r ）和设备驱动层。由于上节代码讲解了设备驱动层的写法...

《Android系统学习》第一章：Input子系统驱动部分

=====

 tankai19880619 2012-10-24 18:03:13 1323

熟悉笔记—数据结构（c语言版）之 顺序表

 maimang1001 2011-01-02 22:33:00


#include typedef int ElemType; #define INITSIZE 100 typedef struct { ElemType \*data; ...

程序员不会英语怎么行？

老司机教你一个数学公式秒懂天下英语



内核中的锁机制

 ruanjianruanjianruan 2017-02-11 23:00:19

atomic(原子操作): 原型：atomic\_t数据类型，atomic\_inc(atomic\_t \*v)将v加1 1，原子操作是不可分割的，在执行完毕何其它任务或事件中中断。 在单处理器系统(...

android4.0 input子系统分析(kernel部分)

 lin364812726 2014-04-26 16:32:08

一、前言 前面我们分析了android的input子系统的android部分的代码，下面我们继续来分析kernel部分的，对于这个nel部分和标准linux差别不大， goo...



Linux/Android——输入子系统input\_event传递 (二)

版权声明：免责声明： 本人在此发文（包括但不限于汉字、拼音、拉丁字母）均为随意敲击键盘所出，用于检验本人电脑入、屏幕显示的机械、光电性能，并不代表本人局部或全部同意、支持或者反对观点。如需要详查...

 u013491946 2017-05-23 13:28:12  227

Android4.2 Input子系统

 tankai19880619 2012-12-03 17:39:14 

=====

知网论文查重入口

知网论文检测入口在哪

百度广告



android Input子系统分析

 xiaoxiaoyu1107 2014-09-19 16:39:06

Input Technical Information Android 输入子系统支持许多不同的设备类，包括键盘，摇杆，轨迹球，鼠标和触摸屏。它描述了上层如何配置，校...

Android 5.0(Lollipop)事件输入系统(Input System) 《-- 推荐阅读这篇

<http://blog.csdn.net/jinzhuojun/article/details/41909159> 其实Android 5.0中事件输入子系统的框架和流程没有本质区别...

 thinkinwm 2015-10-19 21:53:38  1752


[Linux]input 子系统学习笔记（简单范例和四个基本函数）

输入子系统是为了将输入设备的功能呈现给应用程序。它支持 鼠标、键盘、蜂鸣器、触摸屏、传感器等需要不断上报数据。分析了四个函数： 1. input\_allocate\_device 在内存中...

 dearsq 2016-05-19 14:54:51  5983

Linux Input子系统浅析（二）-- 模拟tp上报键值

通过前一节的分析得到，linux Input子系统上传数据本质上是将input\_dev的数据，上报给input\_handler，当用户读入时，驱动层只需要利用copy\_to\_user将数...


 xiaopangzi313 2016-08-31 12:36:04  1365

程序员不会英语怎么行？

老司机教你一个数学公式秒懂天下英语



Linux input子系统

 bianyuke 2015-12-03 09:45:48

一、Input子系统分层思想 input子系统是典型的字符设备。首先分析输入子系统的工作机理。底层设备(按键、触摸等)工作时，产生一个事件(抽象)，CPU读取事件数据放入缓冲区，字符设备驱...

全网络对Linux input子系统最清晰、详尽的分析

 yueqian\_scut 2015-08-27 14:27:40 

本文应是全网对linux input子系统分析最有系统逻辑性和最清晰的分析文章了，主要结构input-core, input-handler和input-core三者的关系以及应用open和rea...