Linux 下的 Input 子系统①

朱银瑞, 吴庆洪, 吴华玲

(辽宁科技大学 电子与信息工程学院, 鞍山)

摘 要: Input 子系统属于 Linux 系统下字符类驱动系统, 现在 Android、X-windows、Qt 等众多应用于 Linux 系统中键盘、鼠标、触摸屏等输入设备的支持都通过、或越来越多倾向于标准的 Input 子系统. 基于现况本文首先从 Input_dev 层, Input Core 层 和 Event Handler 层介绍 Input 输入子系统的实现框架, 然后通过 4x4 矩阵按键在 Input 子系统中的实现以及用户在应用层通过 Input 子系统提供的接口函数对按键操作来查看具体的键值和按键状态. 测试结果表明, Input 子系统里的按键驱动稳定高效通用性强.

关键词: 嵌入式系统; Linux; Input 系统; 矩阵键盘

Input Sub-System in Linux

ZHU Yin-Rui, WU Qing-Hong, WU Hua-Ling

(School of Electronic and Information Engineering, University of Science and Technology Liaoning, Anshan, China)

Abstract: In Linux system the Input sub-system is character driven system. Now android, X-windows, Qt, and many other applications in Linux system for input devices such as keyboard, mouse, touch screen support all through, or more and more inclined to standard Input subsystem. Based on current conditions, this article first from Input_dev layer, Input Core layer and the Event Handler layer introduced the implementation of Input sub-system framework, and then achieve it through the Input sub-system for 4x4 matrix keypad and the user in the application layer use the interface function provided by Input sub-system to operate the buttons in order to view the specific states and values of keys. The results shows that the keypad driver in Input sub-system are more stable and have a high efficiency and so it has a good common use.

Key words: embedded system; Linux; Input sub-system; matrix keyboard

随着科技的进步, 电子信息技术得到了飞速的发展, 嵌入式系统产品已经深入到人们日常的工作和生活当中了, 例如, 打印机、空调、PDA、洗衣机、智能手机等. 这些设备都需要用到或多或少的按键, 而按键作为人机交互的重要手段之一, 在嵌入式系统产品中起着越来越重要的作用. 而 Input 子系统作为实现输入设备驱动的特殊系统具有: a、统一了物理形态各异的相似的输入设备的处理功能; b、提供了用于分发输入报告给用户应用程序的简单的事件接口; c、抽取出了输入驱动程序的通用部分, 简化了驱动, 并引入了一致性; 其构建灵活, 只需要调用一些简单的函数, 就可以将一个输入设备的功能呈现给应用程序[1,2]. 在 Linux 系统中, 都可以利用 Input 接口函数来实现按键、触摸屏、鼠标、跟踪球、操纵杆、加速计和手写板等输入型设备驱动, 使这些驱动变得更易操作应用.

1 输入子系统结构

在 Linux 中, 输入子系统作为一个模块存在, 向上为用户层提供接口函数, 向下为驱动层程序提供统一的接口函数. 这样, 就能够使输入设备的事件通过输入子系统发送给用户层应用程序, 用户层应用程序也可以通过输入子系统通知驱动程序完成某项功能[2]. Input 子系统的分层结构主要有三层: 驱动层, 输入子系统核心层(Input Core)和事件处理层(Event Hander).

① 收稿时间:2013-05-14;收到修改稿时间:2013-06-24

© 中国科学院软件研究所 http://www.c-s-a.org.cn

驱动层设备用 Input_dev 结构体描述负责并与具体的硬件设备进行通讯, 将相应的硬件动作描述成(Event)进行上报. Input Core 层: 输入系统的的核心层, 由 driver/input/input.c 及其相关的头文件组成. 主要是向下提供设备驱动的接口, 向上提供事件处理层的编程接口. 事件处理层负责与用户的应用程序打交到, 将从驱动层传过来的数据报告给用户程序, 三者之间的关系如下图 1 所示. 其中输入子系统的核心层和事件处理层是 Linux 内核已经提供好的, 不用自己再编写文件结构体和 read()、write()等函数. 用户需要处理的主要在 Input_dev 层也就是驱动层. 例如 keyboad.c、mouse.c、touchscreen.c 等. 对于事件处理的同一类设备来说, 他们的处理方式大体相同, 因此内核已经在事件处理层做好了. 然而中断的产生和读取数值是因设备而异的, 需要根据具体的设备来编写驱动.

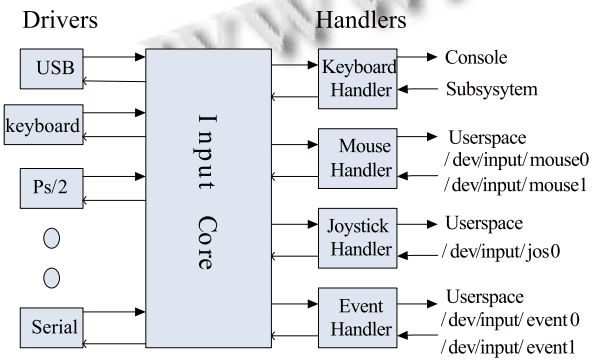

图 1 输入子系统关系图

一个大致的流程就是, Input_dev 向上层报告>>Input Core 接收报告, 并根据在注册 Input_dev 时建立好的连接选择哪一类 handler 来处理事件>>通过 handler 将数据存放在相应的 dev(evdev,mousedev...)实例的缓冲区中, 等待应用程序来读取.

2 矩阵按键在输入子系统中的应用

2.1 硬件平台:

S3c2416 核心板是一款高性能的 ARM9 追线系统板, 采用了三星公司推出的 S3c2416 ARM926EJ 核为控制器, 最高运行频率为 400MHz, 支持 DDR2、SDRAM, 集成 TFT-LCD 控制器和 2D 图形加速器. 本设计用此核心板对其进行外围设备电路扩展, 采用了它的 GPF3、GPF5、GPF6、GPF7 与 GPG0、GPG1、GPG2、GPF7 端口. 用矩阵式键盘, 它能用比较少的 I/O 端口驱动比较多的按键. 图(2)是嵌入式系统中最常采用的 4×4 矩阵键盘. 4×4 矩阵键盘电路由 4 根行线和 4 根列线组成, 按键位于行、列的交叉点上. 一个 4×4 的行列结构可以构成一个 16 个按键的键盘. 4×4 矩阵键盘的行和列分别和 S3c2416 处理器的 4 个 GPIO 端口相连. 硬件电路图如图(2)所示下.

2.2 软件平台:

本研究采用 Linux 内核 2.6.38 作为设计的软件平台, Input 子系统在 2.6 内核的版本中已经得到了很成熟的应用, 相对于 Linux2.4 内核 2.6 在可扩展性、吞吐率等方面有较大提升并且用了新的可调度算法由原先的 O(n)算法改进为 O(1)算法并引进了内存池技术.

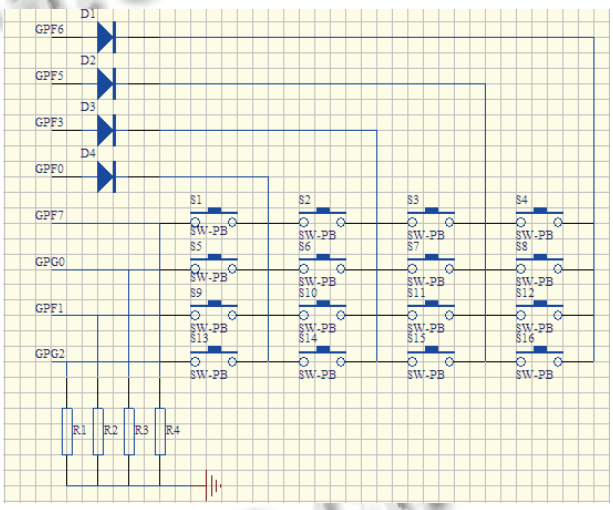

图 2 4x4 矩阵按键

Input_dev 层 keyboad 驱动设计:

S3c2416 支持多大 53 个中断源, 这些中断源可以来自内部功能模块, 也可以来自外部引脚信号(外部中断)这里选择 GPG0、GPG1、GPG2、GPF7 引脚作为外部中断. 根据 S3c2416 数据手册, 对 8 个 GPIO 口引脚进行初始化伪代码如下:

```
static void init_gpio(void)
{//设置 GPF0 为输出, 并置 GPF0 为低电平;
s3c2410_gpio_cfgpin(S3C2410_GPF0,
S3C2410_GPF0_OUTP);      // GPF0
s3c2410_gpio_setpin(S3C2410_GPF0, 0);
……
//设置 GPG0 外部中断, 并设置为下降沿产生中断;
s3c2410_gpio_cfgpin(S3C2410_GPG2,
S3C2410_GPG2_EINT10); // GPG2
```


© 中国科学院软件研究所 http://www.c-s-a.org.cn

```
set_irq_type(IRQ_EINT10, IRQT_FALLING);
…….
}
初始化键值信息
key_info_matrix key_info_matrix[COLS][ROWS] ={
//键值, 外部中断, 外部中断引脚, 列引脚
{1, IRQ_EINT7, S3C2410_GPF7, S3C2410_GPF0},
.....
{16, IRQ_EINT10, S3C2410_GPG2, S3C2410_GPF6},
}
```

2.3 按键扫描

在 keyboad_scan()函数里完成按键扫描利用中断方式进行扫描: 当中断信号差生, 说明某一行有按键按下, 记录中断号, 然后查询列的引脚状态, 若列的引脚状态发生改变则可确定为某列的按键, 行列确定了, 就得到了键值保存, 清除中断与列状态. 再次进行等待查询下一次的行列状态. 用中断方式可以大大节省 cpu 资源, 避免 cpu 的资源浪费. 当确定键值以后, 接下来就要会调用 input 子系统的 input_report_key()函数,

```
input_report_key(input_dev,
keypad->keycodes[code],
new_state[col] & (1 << row));
```

函数向子系统报告发生的事件, 在这里就是一个按键事件, 在上边的扫描中我们不需要考虑重复按键的重复点击情况, input_report_key()函数会自动检查这个问题, 并报告一次事件给输入子系统. 在此函数里起作用的是 input_event(dev, EV_KEY, code, !!value)函数, 其参数代表意义如下: dev 为差生事件的输入设备, 第二个为差生的事件可以是按键、光标、手写板差生的置等. Code 为按键的键值(当事件为按键时), value 表示按键状态 0 表示按键释放, 非 0 表示按键按下. 在此函数里做了以下工作:

a、锁定事件 spin_lock_irqsave();

b、input_handle_event()向输入子系统的相关模块发送数据, 在这里就是事件处理的 handler 层了,这里面有一个很重要的结构体 input_handler 结构, 用来连接 input_dev 和 handler 与事件处理函数 event().

2.4 按键驱动数据的初始化

Keyboard_init_probe()此函数首先对按键驱动的结构数据进行了定义与初始化, 紧接着就调用了 input_allocate_device()函数: input_dev = input_allocate_device(), 此函数在内存中为输入设备结构体分配了一个空间, 并对其主要成员进行了初始化. 例如初始化了设备的类型、初始化 device 结构、初始化互斥锁、初始化时间自旋锁等[3]. 接着通过 input 子系统核心提供的函数 input_register_device()将 input_dev 结构体注册到系统的核心中, 而 input_dev 结构体还必须有 input_allocate_device()函数来分配. 在按键驱动中如:

```
err = init_matrix_gpio(pdev, keypad);
err = input_register_device(keypad->input_dev);
驱动平台数据结构体:
struct platform_driver matrix_keypad_driver = {
    .probe          = keyboad_init_probe,
    .remove         =
__devexit_p(matrix_keypad_remove),
    .driver         = {
        .name       = "matrix-keypad",
        .owner      = THIS_MODULE,
    },
};
```

3 键盘驱动测试

在超级终端下通过命令 cat /proc/bus/input/devices 来查看事件号, 以便应用程序知道要打开那个文件.

```
struct input_event data;
fa=open("/dev/input/event1",O_RDONLY);//event1  表示按键事件
        if(!fa){
            printf("Error:cannot open Keypad device.\n");
            exit(1);
        }
        printf("The Keypad device was opened successfully.\n");
read(fa,&data,sizeof(struct input_event));
        if (data.type == EV_KEY)
                printf(" type: EV_KEY = %x, event = %d, value = %d\n",data.type,data.code, data.value);
```

将测试程序放入文件系统, 当有按键按下时就可以在终端看到事件类型、键值、和按键的状态. 从测试程序可以看到 value 值为按键状态, 当 value 值为 1

© 中国科学院软件研究所 http://www.c-s-a.org.cn

时表示键按下, 值为 0 时表示键抬起, 值为 2 时表示按键按下之后并没抬起. 这样用户就可以通过得到的键值以及状态来编写相应的按键应用程序.

测试结果如图 3、图 4 所示.

```
type: EV_KEY = 1, event = 3, value = 1
type: EV_KEY = 1, event = 3, value = 0
type: EV_KEY = 1, event = 3, value = 1
type: EV_KEY = 1, event = 3, value = 0
type: EV_KEY = 1, event = 3, value = 1
type: EV_KEY = 1, event = 3, value = 0
```

图 3 按键状态 1

```
type: EV_KEY = 1, event = 3, value = 2
type: EV_KEY = 1, event = 3, value = 2
type: EV_KEY = 1, event = 3, value = 2
type: EV_KEY = 1, event = 3, value = 2
type: EV_KEY = 1, event = 3, value = 2
type: EV_KEY = 1, event = 3, value = 2
type: EV_KEY = 1, event = 3, value = 0
```

图 4 长按状态 2

4 结束语

本文介绍了嵌入式 Linux 下 Input 输入子系统设备驱动的结构, 向上为用户提供接口函数, 向下为驱动层程序提供统一接口函数, 以及以 4x4 矩阵按键为例说明驱动层程序调用了 Input 子系统的那些接口函数. 最后以 ARM9 系列的芯片 S3c2416 为核心板的硬件基础上进行外围硬件扩展, 编写了简单的测 试程序实现了按键驱动和 Input 子系统为用户提供的接口函数的调用. 对诸如嵌入式中的 Input 子系统中输入设备的应用提供了一定的参考价值.

参考文献

1 李俊.嵌入式 Linux 设备驱动开发详解.北京:人民邮电出版社.2008.

2 郑强.驱动开发入门与实战.北京:清华大学出版社.2011.

3 Bovel DP, Cesati M. Understanding the Linux Kernel. 北京:中国电力出版社.2007.

(上接第 198 页)

参考文献

1 Document Object Model(DOM)Level 1 Specification, Version 1.0. W3C Recommendation, Sep.1998.

2 Document Object Model(DOM)Level 2 Core Specification, Version 1.0. W3C Recommendation, Nov.2000.

3 Document Object Model(DOM)Level 3 Core Specification, Version 1.0. W3C Recommendation, Apr.2004.

4 Seacord RC, Plakosh D, Lewis GA.梁海华译.遗留系统的现代化改造.北京:清华大学出版社.2004.

5 DHTML Reference, MSDN Library. Microsoft, 1996.

6 Fowler M.熊节译.重构:改善既有代码的设计.北京:人民邮电出版社.2010.

7 David Flanagn.李强等译 JavaScript 权威指南.北京:机械工业出版社.2007.

8 Aho AV. Compilers: Principles, Techniques & Tools. 北京:人民邮电出版社.2008.

9 Parr T. The Definitive ANTLR Reference. Pragmatic Bookshelf,2007.

10 Sambells J, Gustafson A.李松峰,李雅雯译. JavaScript DOM 高级程序设计.北京:人民邮电出版社.2008.

© 中国科学院软件研究所 http://www.c-s-a.org.cn