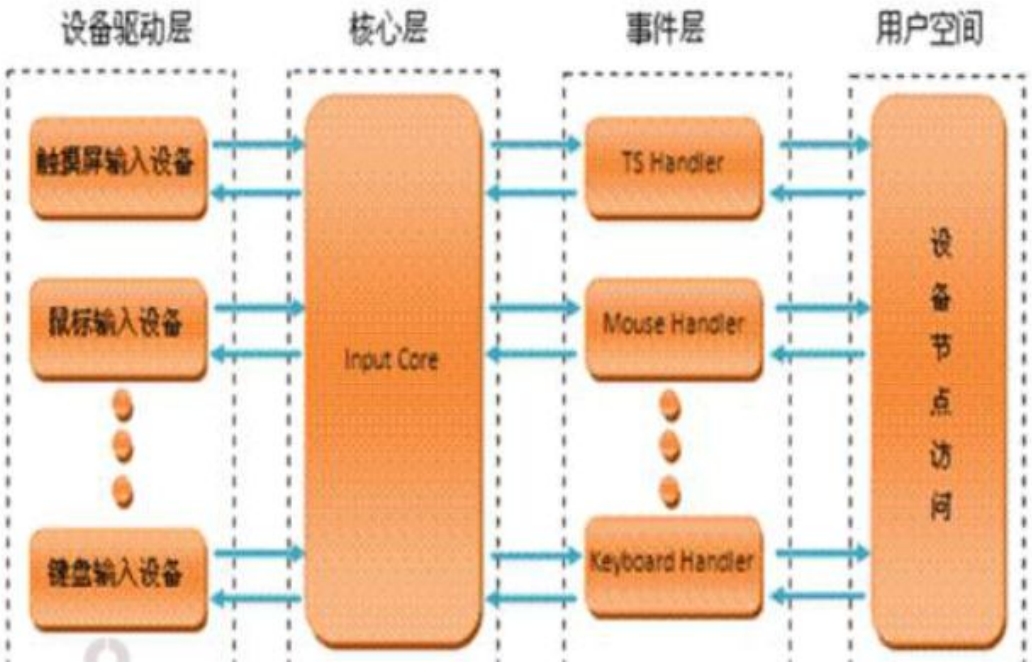
iTOP-4412-驱动-输入子系统之按键

前面的教程中，介绍过通过按键和上层通信的方式：轮询读，poll 机制，异步通信机制等等。在 linux 中，有一套标准的处理用户输入的子系统叫输入子系统，输入子系统通常用来实现用户的输入，如下图所示（下面这段取自网络），这是输入子系统的简易模型。



主要分为三个层次：设备驱动层、核心层、事件层。

设备驱动层：主要实现对具体硬件的驱动以后获得硬件产生的数据，并且把硬件设备产生的数据封装成事件的形式提交给核心层。这些是通过申请一些 io、中断等资源完成的来读写硬件设备的，用核心层定义的规范来注册到核心层的。

核心层：核心层处于事件处理层和设备驱动层的中间，它的作用就是承上启下的作用。设备驱动层调用核心层提供的接口，把具体数据提交给核心层，核心层就自动把数据交给事件处理层。

事件处理层：这里就是对提交的数据做相应的处理了，同时这里也是用户的编程接口。

到这里，大家有没有一个疑惑，我们写一个简单的按键驱动程序，还要遵循这么多东西，我们直接定义一个字符设备写不就得了吗？的确如果仅仅是写一个按键程序，其实怎么样写都一样，但是如果要写的是触摸屏、电脑的键盘以及鼠标等，那在驱动中要做的事就很多了，而且你写的驱动，在应用层调用时，没有一个统一的标准，那就必须做一个文档介绍给应用层提

供的接口了，而且不见得能写的比较好，再一个 Android、X windows、qt 等众多应用对于 linux 系统中键盘、鼠标、触摸屏等输入设备的支持都通过、或越来越倾向于标准的 input 输入子系统。如果按照 linux 提供的 input 子系统的规范来完成驱动，在设备驱动中仅仅需要把硬件产生的数据提交给核心层，而核心层就会自动提交给事件处理层，在事件处理层中已经完成了对字符设备的文件操作接口，同时也完成了所提交数据的处理，这样写设备驱动的工作量就不是很大了，而且写的驱动能更规范、更稳定，这就是咱们要研究 input 子系统的原因。当然老话告诉我们，什么都是一把双刃剑，要按照 input 子系统写驱动，你首先得了解 input 子系统，这就为写设备驱动提高了门槛，而且如果要灵活应用，还得进一步研究 input 子系统。

本文档将设计一个输入子系统的驱动例程，使用 HOME 按键，在 HOME 按键按下之后，向输入子系统中传输字母“l”。通过例程来学习输入子系统驱动的框架和用法。

1 输入子系统驱动的实现步骤

输入子系统驱动一般分为以下几个步骤：

第一步：定义 input_dev 结构体，向内核申请一个 input_dev 事件，标准内核函数接口 input_allocate_device（该函数返回输入子系统事件）和 input_free_device（释放申请的输入子系统事件）。

第二步：配置输入子系统的某一类“事件类型”，例如：按键事件，鼠标事件，触摸事件等等。在 input.h 头文件中，定义了 10 多种事件类型,这里定义按键事件。

第三步：配置能产生这类事件中的那些具体操作，例如：按键按键事件中，可以输入键盘 a，键盘 b 等，我们例程中定义产生 l 按键值。

第四步：注册输入子系统，标准内核函数接口 input_register_device 和 input_unregister_device。

下一节结合具体的代码来分析这几个步骤。

2 驱动代码分析

驱动例程源码 keys_input_test.c 在和文档一起的压缩包中。

代码如下，其中的注释已经写得很详细了。

```
#include <linux/init.h>
#include <linux/module.h>

#include <linux/kernel.h>
#include <linux/fs.h>

#include <linux/interrupt.h>
#include <linux/irq.h>
#include <mach/gpio.h>
#include <plat/gpio-cfg.h>
#include <linux/miscdevice.h>
#include <linux/platform_device.h>
#include <mach/regs-gpio.h>
#include <asm/io.h>
#include <linux/regulator/consumer.h>
#include <linux/delay.h>
#include <linux/input.h>

#define  MY_KEY_HOME    EXYNOS4_GPX1(1)
#define  MY_KEY_BACK    EXYNOS4_GPX1(2)
#define  MY_KEY_SLEEP   EXYNOS4_GPX3(3)
#define MY_KEY_VOL_U    EXYNOS4_GPX2(1)
#define MY_KEY_VOL_D    EXYNOS4_GPX2(0)

//定义一个输入子系统结构体，它是输入子系统的核心，后面所有工作都是围绕这个结构体还做的！
//第一步：定义之后，向内核申请一个 input_dev 事件，标准内核函数接口 input_allocate_device 和
input_free_device
//第二步：配置输入子系统的某一类“事件类型”，例如：按键事件，鼠标事件，触摸事件等等
//第三步：配置能产生这类事件中的那些具体操作，例如：按键按键事件中，可以输入键盘 a，键盘 b 等
//第四步：注册输入子系统，标准内核函数接口 input_register_device 和 input_unregister_device
static struct input_dev *buttons_dev;

static irqreturn_t ehome_interrupt(int irq, void *dev_id)
{
```

```
	//printk("%s(%d)\n", __FUNCTION__, __LINE__);

	//char *ps = dev_id;
	//printk("%s\n",ps);

	//input_event(buttons_dev, EV_KEY, KEY_L, 1);
	//input_sync(buttons_dev);

	//最后一个参数: 0-松开, 非 0-按下
	input_report_key(buttons_dev, KEY_L, 1);
	//通知上层，本次事件结束
	input_sync(buttons_dev);

	input_report_key(buttons_dev, KEY_L, 0);
	//通知上层，本次事件结束
	input_sync(buttons_dev);

	return IRQ_HANDLED;
}

static void __exit keys_input_exit(void)
{
	printk("%s(%d)\n", __FUNCTION__, __LINE__);
	gpio_free(MY_KEY_HOME);
	free_irq(gpio_to_irq(MY_KEY_HOME),(void *)"my_key_home");

//释放申请的 input_dev 事件以及释放输入子系统
	input_unregister_device(buttons_dev);
	input_free_device(buttons_dev);
}

static int __init keys_input_init(void)
{
	int ret;
	printk("%s(%d)\n", __FUNCTION__, __LINE__);

//第一步：定义之后，向内核申请一个 input_dev 事件，标准内核函数接口 input_allocate_device 和
//input_free_device
	buttons_dev = input_allocate_device();
```

```
//第二步：配置输入子系统的某一类“事件类型”，例如：按键事件，鼠标事件，触摸事件等等
//在 input.h 头文件中，定义了 10 多种事件类型,这里定义按键事件
	set_bit(EV_KEY, buttons_dev->evbit);
	//set_bit(EV_REP, buttons_dev->evbit);  //重复扫描

//第三步：配置能产生这类事件中的那些具体操作，例如：按键按键事件中，可以输入键盘 a，键盘 b 等
//这里定义产生 l 按键值
	set_bit(KEY_L, buttons_dev->keybit);

//第四步：注册输入子系统，标准内核函数接口 input_register_device 和 input_unregister_device
	ret = input_register_device(buttons_dev);
	if(ret <0){
		printk("%s(%d)\n", __FUNCTION__, __LINE__);
		goto exit;
	}

	gpio_request(MY_KEY_HOME, "my_key_home");
	s3c_gpio_cfgpin(MY_KEY_HOME, S3C_GPIO_OUTPUT);
	gpio_set_value(MY_KEY_HOME, 0);
	gpio_free(MY_KEY_HOME);
	ret = request_irq(gpio_to_irq(MY_KEY_HOME),ehome_interrupt,IRQ_TYPE_EDGE_FALLING,
"my_key_home", (void *)"my_key_home");
	if(ret<0){
		printk("Request IRQ %s failed, %d\n","key_home" , ret);
		goto exit;
	}

	return 0;

exit:
	return ret;
}

module_init(keys_input_init);
module_exit(keys_input_exit);

MODULE_LICENSE("Dual BSD/GPL");
```

注释写得已经很详细，下面分析重要的部分。

```
#define  MY_KEY_HOME    EXYNOS4_GPX1(1)
```

以上代码，定义 5 个按键的宏，方便操作，例程中只使用了其中一个 MY_KEY_HOME。

```
static struct input_dev *buttons_dev;
```

以上代码，定义输入子系统的事件“buttons_dev”，后面的代码几乎全部是围绕这个事件来操作。

```
buttons_dev = input_allocate_device();
```

以上代码，向内核申请一个输入子系统的事件， input_allocate_device()函数返回一个输入子系统的事件。

```
set_bit(EV_KEY, buttons_dev->evbit);
```

以上代码，将输入子系统事件的类型设置为按键事件。申请输入子系统事件之后，需要配置输入子系统的某一类“事件类型”，例如：按键事件，鼠标事件，触摸事件等等。在 input.h 头文件中，定义了 10 多种事件类型,我们这里定义的是按键事件。

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

以上代码，定义按键事件中允许产生输入 l 按键值事件的操作。配置能产生这类事件中的那些具体操作，例如：按键按键事件中，可以输入键盘值 a，键盘值 b 等。

```
ret = input_register_device(buttons_dev);
```

以上代码，注册输入子系统，标准内核函数接口注册和撤销：input_register_device 和 input_unregister_device。

接着看一下，中断处理函数 ehome_interrupt 部分，如下图所示。

```
    //最后一个参数: 0-松开, 非 0-按下
    input_report_key(buttons_dev, KEY_L, 1);
    //通知上层，本次事件结束
    input_sync(buttons_dev);

    input_report_key(buttons_dev, KEY_L, 0);
    //通知上层，本次事件结束
    input_sync(buttons_dev);
```

中断产生之后，需要向系统上报产生具体哪个事件，使用 input_report_key 可以上报按键事件。这里需要注意一下，在网上的例程中，有很多例程中，需要使用定时器来防抖，4412 是不需要在代码中进行按键防抖的，因为 4412 芯片内部可以实现硬件防抖。所以，在上面代码中，直接上报按键事件两次。input_report_key 的第三个参数，如果为 0，则是松开

按键 l 事件；如果为非 0，则是按下按键 l 事件。input_sync 函数则是通知系统，本次上报事件结束。

3 测试

注册按键的输入子系统之后，会产生“/dev/input/event*”节点。

在加载驱动之前，使用命令“ls /dev/input/event*” ，如下图所示。

```
[root@iTOP-4412]# ls /dev/input/event*
/dev/input/event0
```

加载驱动之后，如下图所示，可以看到产生了新的“/dev/input/event1”节点。

```
[root@iTOP-4412]# ls /dev/input/event*
/dev/input/event0
[root@iTOP-4412]# insmod keys_input_test.ko
[10657.911228] keys_input_init(70)
[10657.920675] input: Unspecified device as /devices/virtual/input/input6
[root@iTOP-4412]# ls /dev/input/event*
/dev/input/event0  /dev/input/event1
[root@iTOP-4412]#
```

接着需要使用到一个工具“hexdump” ，这个工具一般 linux 系统都自带，如下图所示，使用命令“hexdump /dev/input/event1” ，然后按一下开发板上“HOME”键。

```
[root@iTOP-4412]# ls /dev/input/event*
/dev/input/event0
[root@iTOP-4412]# insmod keys_input_test.ko
[10657.911228] keys_input_init(70)
[10657.920675] input: Unspecified device as /devices/virtual/input/input6
[root@iTOP-4412]# ls /dev/input/event*
/dev/input/event0  /dev/input/event1
[root@iTOP-4412]# hexdump /dev/input/event1
0000000 2a5f 0000 a95d 0004 0001 0026 0001 0000
0000010 2a5f 0000 a972 0004 0000 0000 0000 0000
0000020 2a5f 0000 a98d 0004 0001 0026 0000 0000
0000030 2a5f 0000 a994 0004 0000 0000 0000 0000
```

如上图所示，可以看到有九列数据，比较重要的是 6，7，8 列。第 6 列表示上报的事件，第 7 列表示上报的键值，第 8 列表示按键事件是按下还是释放。

先看第一行的 6,7,8 列，分别是 0001,0026,0001，

第一行第 6 列-0001，如下图所示，是和 EV_KEY 宏的值对应。

```
00145:  * Event types
00146:  */
00147:
00148: #define EV_SYN          0x00
00149: #define EV_KEY          0x01
00150: #define EV_REL          0x02
00151: #define EV_ABS          0x03
00152: #define EV_MSC          0x04
00153: #define EV_SW           0x05
00154: #define EV_LED          0x11
00155: #define EV_SND          0x12
00156: #define EV_REP          0x14
00157: #define EV_FF           0x15
00158: #define EV_PWR          0x16
00159: #define EV_FF_STATUS        0x17
```

第一行第 7 列-0026，如下图所示，对应宏 KEY_L，宏定义的值是 38，是个 10 进制数，将其转换为 16 进制数就是 0x26。

```
00213: #define KEY_A           30
00214: #define KEY_S           31
00215: #define KEY_D           32
00216: #define KEY_F           33
00217: #define KEY_G           34
00218: #define KEY_H           35
00219: #define KEY_J           36
00220: #define KEY_K           37
00221: #define KEY_L           38
00222: #define KEY_SEMICOLON       39
00223: #define KEY_APOSTROPHE      40
00224: #define KEY_GRAVE       41
00225: #define KEY_LEFTSHIFT       42
00226: #define KEY_BACKSLASH       43
00227: #define KEY_Z           44
00228: #define KEY_X           45
```

第一行第 7 列-0001，可以看到按键产生之后，第一个上报事件是有按键按下。

```
    //最后一个参数：0-松开，非0-按下
    input_report_key(buttons_dev, KEY_L, 1);
    //通知上层，本次事件结束
    input_sync(buttons_dev);

    input_report_key(buttons_dev, KEY_L, 0);
    //通知上层，本次事件结束
    input_sync(buttons_dev);

    return IRQ_HANDLED;
}
```

如上图所示，代码“input_sync(buttons_dev);”，接着通知上层本次事件结束，所以第二行 6,7,8 列都是 0。

接着上报按键事件，有按键松开，所以第三行的 6,7 列和第一行相同，第 8 列就是 0 了，表示有按键释放。

第四行和第二行一样，表示事件终止。

到此，按键的输入子系统就分析完毕。

联系方式

北京迅为电子有限公司致力于嵌入式软硬件设计，是高端开发平台以及移动设备方案提供商；基于多年的技术积累，在工控、仪表、教育、医疗、车载等领域通过 OEM/ODM 方式为客户创造价值。

iTOP-4412 开发板是迅为电子基于三星最新四核处理器 Exynos4412 研制的一款实验开发平台，可以通过该产品评估 Exynos 4412 处理器相关性能，并以此为基础开发出用户需要的特定产品。

本手册主要介绍 iTOP-4412 开发板的使用方法，旨在帮助用户快速掌握该产品的应用特点，通过对开发板进行后续软硬件开发，衍生出符合特定需求的应用系统。

如需平板电脑案支持，请访问迅为平板方案网“http://www.topeet.com”，我司将有能力为您提供全方位的技术服务，保证您产品设计无忧！

本手册将持续更新，并通过多种方式发布给新老用户，希望迅为电子的努力能给您的学习和开发带来帮助。

迅为电子

2018 年 1 月