iTOP-4412-驱动-usb 文档 09-鼠标驱动详解 03-urb 请求块的使用

本文档，主要介绍在 probe 中，urb 的初始化。USB 中所有的对底层的通信，都是围绕 urb 来做的，所有的工作都是一个套路。

文档中，先带大家分析下内核自带 USB 鼠标驱动部分代码，然后再改写为简单模式，对 urb 进行监测。

1 USB 鼠标驱动部分代码分析

先结合前面介绍理论部分，对内核中自带的 USB 鼠标驱动代码进行分析，后面我们的例程会进行一些简化，让大家对其更容易理解。

1.1 初始化和 usb_interface 分析

下面是进入 probe 之后的一段匹配的代码，用于验证接入的设备是否和驱动匹配。

```
interface = intf->cur_altsetting;

if (interface->desc.bNumEndpoints != 1)
        return -ENODEV;

endpoint = &interface->endpoint[0].desc;
if (!usb_endpoint_is_int_in(endpoint))
        return -ENODEV;
```

在请求块理论部分，我们了解到鼠标驱动是“一个中断 IN 端点给带有特定端点号的特定 USB 设备”。在 USB 设备描述符的文档中，我们了解到“设备描述符→配置描述符→接口描述符→端点描述符”这几个概念，用户可以直接看下“usb_interface”这个结构体，里面内嵌了接口描述符信息和端点描述符等。

在上面的代码“interface = intf->cur_altsetting;”中，获取鼠标硬件传输过来的描述符信息。

在“if (interface->desc.bNumEndpoints != 1) return -ENODEV;”中，判断端点是否只有一个，鼠标设备只有一个端点，如果不是，则报错。

关于 static int usb_mouse_probe(struct usb_interface *intf, const struct usb_device_id *id)中的变量“struct usb_interface *intf”，看到这里很多人可能不太理解。前面我们介绍描述符的时候，并没有 usb_interface 这个结构体，实际上是这样的，它本质上是个“配置描述符”，它是从从机（鼠标）中传递过来的信息，配置描述符的组织方式可能有点不一样。可以这样类比，当我们要到大学报到的时候，我们要填写学籍档案，假设学籍档案中信息都在户口簿上，例如：姓名、籍贯以及身份证号等等，这些信息从户口簿传递到学籍档案中，信息其实是一样的，只是在户口簿和学籍档案中组织形式不一样，名称不一样，实际包含的信息是一样的。我们也可以看下“usb_interface”这个结构体，部分代码如下所示。

```
struct usb_interface {
    /* array of alternate settings for this interface,
     * stored in no particular order */
    struct usb_host_interface *altsetting;
```

接着看下“usb_host_interface”结构体，如下所示，可以看到结构体中包含了接口描述符和端点描述符。

```
struct usb_host_interface {
    struct usb_interface_descriptor desc;

    /* array of desc.bNumEndpoint endpoints associated with this
     * interface setting.  these will be in no particular order.
     */
    struct usb_host_endpoint *endpoint;
```

在“endpoint = &interface->endpoint[0].desc;”代码中，我们的驱动获取了端点描述符信息。

在我们的例程中，可以简化为“interface = intf->cur_altsetting;”和“endpoint = &interface->endpoint[0].desc;”，不做判断。

1.2 管道和管道数据包

代码如下所示。

```
    pipe = usb_rcvintpipe(dev, endpoint->bEndpointAddress);
```

```
    maxp = usb_maxpacket(dev, pipe, usb_pipeout(pipe));
```

前面介绍过，要和 USB 设备通信，唯一的方式是通过管道，每个驱动和 USB 设备通信，都必须创建管道，使用管道进行通信，而且管道必须使用内核统一的函数来申请！

1.3 初始化 USB 设备结构体和输入子系统设备结构体

如下所示，代码有详细的注释。

```
//为 mouse 分配内存，申请输入子系统
    mouse = kzalloc(sizeof(struct usb_mouse), GFP_KERNEL);
    input_dev = input_allocate_device();
    if (!mouse || !input_dev)
        goto fail1;

//申请内存空间用于数据传输，data 为指向该空间的地址，data_dma 则是这块内存空间的 dma 映射，
 即这块内存空间对应的 dma 地址。在使用 dma 传输的情况下，则使用 data_dma 指向的 dma 区域，
否则使用 data 指向的普通内存区域进行传输。 GFP_ATOMIC 表示不等待，GFP_KERNEL 是普通的优先级，
可以睡眠等待，由于鼠标使用中断传输方式，不允许睡眠状态，data 又是周期性获取鼠标事件的存储区，因此使
用 GFP_ATOMIC 优先级，如果不能 分配到内存则立即返回 0
    mouse->data = usb_alloc_coherent(dev, 8, GFP_ATOMIC, &mouse->data_dma);
    if (!mouse->data)
        goto fail1;

//为 urb 结构体申请内存空间，第一个参数表示等时传输时需要传送包的数量，其它传输方式则为 0。申请的内
存将通过下面即将见到的 usb_fill_int_urb 函数进行填充。
    mouse->irq = usb_alloc_urb(0, GFP_KERNEL);
    if (!mouse->irq)
        goto fail2;

//填充 usb 设备结构体和输入设备结构体
    mouse->usbdev = dev;
    mouse->dev = input_dev;

//填充鼠标设备的名称
    if (dev->manufacturer)
        strlcpy(mouse->name, dev->manufacturer, sizeof(mouse->name));
    if (dev->product) {
        if (dev->manufacturer)
            strlcat(mouse->name, " ", sizeof(mouse->name));
```

```
        strlcat(mouse->name, dev->product, sizeof(mouse->name));
    }

    if (!strlen(mouse->name))
        snprintf(mouse->name, sizeof(mouse->name),
             "USB HIDBP Mouse %04x:%04x",
             le16_to_cpu(dev->descriptor.idVendor),
             le16_to_cpu(dev->descriptor.idProduct));

//填充鼠标设备结构体中的节点名。usb_make_path 用来获取 USB 设备在 Sysfs 中的路径，格式为：usb-usb 总线号-路径名
    usb_make_path(dev, mouse->phys, sizeof(mouse->phys));
    strlcat(mouse->phys, "/input0", sizeof(mouse->phys));

//将鼠标设备的名称赋给鼠标设备内嵌的输入子系统结构体
    input_dev->name = mouse->name;
    input_dev->phys = mouse->phys;

//input_dev 中的 input_id 结构体，用来存储厂商、设备类型和设备的编号，这个函数是将设备描述符中的编号赋给内嵌的输入子系统结构体
    usb_to_input_id(dev, &input_dev->id);
    input_dev->dev.parent = &intf->dev;

//input_dev 进行初始化。EV_KEY 是按键事件，EV_REL 是相对坐标事件；keybit 表示键值，包括左键、右键和中键；relbit 用于表示相对坐标值；有的鼠标还有其它按键；中键滚轮的滚动值。
    input_dev->evbit[0] = BIT_MASK(EV_KEY) | BIT_MASK(EV_REL);
    input_dev->keybit[BIT_WORD(BTN_MOUSE)] = BIT_MASK(BTN_LEFT) |
        BIT_MASK(BTN_RIGHT) | BIT_MASK(BTN_MIDDLE);
    input_dev->relbit[0] = BIT_MASK(REL_X) | BIT_MASK(REL_Y);
    input_dev->keybit[BIT_WORD(BTN_MOUSE)] |= BIT_MASK(BTN_SIDE) |
        BIT_MASK(BTN_EXTRA);
    input_dev->relbit[0] |= BIT_MASK(REL_WHEEL);

//设置当前输入设备的种类
    input_set_drvdata(input_dev, mouse);

//为输入子系统设备添加打开，关闭等
    input_dev->open = usb_mouse_open;
    input_dev->close = usb_mouse_close;
```

1.4 设置 urb 等

如下所示，可以看注释来理解。

```
//填充构建 urb，将刚才填充好的 mouse 结构体的数据填充进 urb 结构体中，在 open 中递交 urb。当 urb
包含一个即将传输的 DMA 缓冲区时应该设置 URB_NO_TRANSFER_DMA_MAP。USB 核心使用
transfer_dma 变量所指向的缓冲区，而不是 transfer_buffer 变量所指向的。
URB_NO_SETUP_DMA_MAP 用于 Setup 包，URB_NO_TRANSFER_DMA_MAP 用于所有 Data 包。
    usb_fill_int_urb(mouse->irq, dev, pipe, mouse->data,
                (maxp > 8 ? 8 : maxp),
                usb_mouse_irq, mouse, endpoint->bInterval);
    mouse->irq->transfer_dma = mouse->data_dma;
    mouse->irq->transfer_flags |= URB_NO_TRANSFER_DMA_MAP;

//注册输入子系统
    error = input_register_device(mouse->dev);
    if (error)
        goto fail3;

//一般在 probe 函数中，都需要将设备相关信息保存在一个 usb_interface 结构体中，以便以后通过
usb_get_intfdata 获取使用。这里鼠标设备结构体信息将保存在 intf 接口结构体内嵌的设备结构体中
的 driver_data 数据成员中，即 intf->dev->dirver_data = mouse。
    usb_set_intfdata(intf, mouse);
    return 0;
```

1.5 usb_mouse_disconnect 操作

代码如下，拔出之后会进行以下操作。现获取鼠标设备结构体，然后清空；kill 掉 urb，删掉输入子系统；删掉 urb；释放内存等等。

```
static void usb_mouse_disconnect(struct usb_interface *intf)
{
    struct usb_mouse *mouse = usb_get_intfdata (intf);

    usb_set_intfdata(intf, NULL);
    if (mouse) {
        usb_kill_urb(mouse->irq);
        input_unregister_device(mouse->dev);
        usb_free_urb(mouse->irq);
        usb_free_coherent(interface_to_usbdev(intf), 8, mouse->data, mouse->data_dma);
```

```
        kfree(mouse);
    }
}
```

2 请求块 URB 的使用

在通信原理中，有信源、信道和信宿的概念。虽然这个概念更多的是指代硬件方面，不过我觉得放在这里来理解 USB 的几个概念，非常好。

类比生活中的例子，小 A 对小 B 说了一句“您好”，小 B 听到了。小 A 就是信源，空气就是信道，小 B 就是信宿。当然，也有数据源、传输通道和数据终端的说法，实质是差不多的。

那么我给大家说这几个概念有什么用呢？不知道大家还记不记的前面的文档中介绍过的几个概念。主机只能和 USB 设备的“端点”通信；通信是通过“管道”实现；USB 通信是通过请求块实现。最后数据终端就是我们的驱动程序了，我们要在驱动程序中获得 USB 传输来的数据，这个实验就成功了一多半。

接着，我们来看一下将要在 probe 中添加的代码，其中去掉了输入子系统的代码，简化了 urb 部分。

```
static int usb_mouse_probe(struct usb_interface *intf, const struct usb_device_id *id)
{
    struct usb_device *dev = interface_to_usbdev(intf);
    struct usb_endpoint_descriptor *endpoint;

    //鉴于我们在中断处理函数中，需要监听是否有数据传输，这几个变量设置为 static 变量
    /*int pipe, maxp;
    signed char *data;
    dma_addr_t data_dma;
    struct urb *mouse_urb;*/

    printk("usb mouse probe!\n");
    //check_usb_device_descriptor(dev);
```

```
    //在 usb 的数据传输中，"数据源"是端点， “传输通道"是管道，"数据终端"是内核 “USB 驱动” 中的
buffer
    //数据源，数据通道，数据终端全部是在 urb 中设置
    endpoint = &intf->cur_altsetting->endpoint[0].desc;      //数据源

    pipe = usb_rcvintpipe(dev, endpoint->bEndpointAddress);    //传输通道
    maxp = usb_maxpacket(dev, pipe, usb_pipeout(pipe));

    data = usb_alloc_coherent(dev, 8, GFP_ATOMIC, &data_dma);//数据终端

    //urb 必须使用内核函数申请，以便内核同一管理
    mouse_urb = usb_alloc_urb(0, GFP_KERNEL);
    //通过 urb 设置数据源，传输通道，数据终端
    usb_fill_int_urb(mouse_urb, dev, pipe, data,(maxp > 8 ? 8 : maxp),check_usb_data, NULL, endpoint-
>bInterval);
    mouse_urb->transfer_dma = data_dma;
    mouse_urb->transfer_flags |= URB_NO_TRANSFER_DMA_MAP;

    //提交 urb
    usb_submit_urb(mouse_urb, GFP_KERNEL);

    return 0;
}
```

然后看一下，中断处理函数。

```
//监测是否有数据传入
static void check_usb_data(struct urb *urb)
{
    int i = 0;

    printk("count is %d time!\n",++count);

    //重新提交 urb
    usb_submit_urb(mouse_urb, GFP_KERNEL);
}
```

完整的代码，请参考代码文件。

驱动编译加载之后，插上 USB 鼠标，如下图所示。

```
[root@iTOP-4412]# [ 2363.430386] usb 1-3.1: new low speed USB device number 42 using s5p-ehci
[ 2363.545161] usb 1-3.1: New USB device found, idVendor=046d, idProduct=c077, bcdDevice=7200
[ 2363.552155] usb 1-3.1: New USB device strings: Mfr=1, Product=2, SerialNumber=0
[ 2363.559349] usb 1-3.1: New USB device Class: Class=0, SubClass=0, Protocol=0
[ 2363.566369] usb 1-3.1: Product: USB Optical Mouse
[ 2363.571053] usb 1-3.1: Manufacturer: Logitech
[ 2363.586268] usb mouse probe!
```

接着，尝试按鼠标左键，右键，滚轮，滑动操作等等，如下图所示。USB 鼠标进行操作之后，驱动进入了中断。

```
[root@iTOP-4412]# [ 2408.851268] count is 1 time!
[ 2408.859253] count is 2 time!
[ 2408.867271] count is 3 time!
[ 2408.875270] count is 4 time!
[ 2408.883266] count is 5 time!
[ 2408.891262] count is 6 time!
[ 2408.899270] count is 7 time!
[ 2408.907272] count is 8 time!
[ 2408.915270] count is 9 time!
[ 2408.923262] count is 10 time!
[ 2408.931262] count is 11 time!
[ 2408.939256] count is 12 time!
[ 2408.947271] count is 13 time!
[ 2408.955258] count is 14 time!
[ 2408.971252] count is 15 time!
[ 2408.995261] count is 16 time!
[ 2409.011260] count is 17 time!
[ 2409.019244] count is 18 time!
[ 2409.027250] count is 19 time!
[ 2409.035255] count is 20 time!
[ 2409.043255] count is 21 time!
[ 2409.051243] count is 22 time!
[ 2409.059254] count is 23 time!
[ 2409.067254] count is 24 time!
[ 2409.075240] count is 25 time!
[ 2409.083242] count is 26 time!
[ 2409.091255] count is 27 time!
```

那么传输过来的数据在哪里呢？其实就在 data 这个数据指针中，在下一篇文档中，我们增加输入子系统部分，在应用层打印数据。

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 2 月