Broadview
www.broadview.com.cn

做程序员的最高境界
就要像和尚研究佛法一样研究算法
这次函数调用就像流星，短暂地划过却能照亮整个天空
终于，四大传输也就这样结束了，而我们的故事也即将ALT+F4了。我只是说也
忽如一夜春风来，内核处处是模块
到这里，我该恭喜你，因为你已经能够编写Linux内核模块了
这种感觉很美妙，不是吗
它的心只有编译器才懂

Linux那些事儿之我是USB

◎华清远见嵌入式培训中心 肖林甫 肖季东 任桥伟 著

每天万余人争相阅读
CSDN Blog专家 fudan_abc 最新力作

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

Linux那些事儿之我是USB

◎华清远见嵌入式培训中心 肖林甫 肖季东 任桥伟 著

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书基于 2.6.22 内核，对 USB 子系统的大部分源代码逐行进行分析，系统地阐释了 Linux 内核中 USB 子系统是如何运转的，子系统内部的各个模块之间是如何互相协作互相配合的。

本书使用幽默诙谐的笔调对 Linux 内核中的 USB 子系统源代码进行了分析，形象且详尽地介绍了 USB 在 Linux 中的实现。本书从 U 盘、Hub、USB Core 直到主机控制器覆盖了 USB 实现的方方面面，被一些网友誉为 USB 开发的“圣经”。

对于 Linux 初学者，可以通过本书掌握学习内核、浏览内核代码的方法；对于 Linux 驱动开发者，可以通过本书对设备模型有形象深刻的理解；对于 USB 开发者，可以通过本书全面的理解 USB 在一个操作系统中的实现；对于 Linux 内核开发者，也可以通过本书学习到很多 Linux 高手开发维护一个完整子系统时的编程思想。

图书在版编目（CIP）数据

Linux 那些事儿之我是 USB / 肖林甫，肖季东，任桥伟著. -- 北京:电子工业出版社, 2010.7
ISBN 978-7-121-11178-5

Ⅰ. ①L… Ⅱ. ①肖… ②肖… ③任… Ⅲ. ①Linux 操作系统－程序设计②电子计算机－接口－程序设计
Ⅳ. ①TP316.89②TP334

中国版本图书馆 CIP 数据核字（2010）第 117276 号

责任编辑：孙学瑛
文字编辑：王　静
印　　刷：北京天宇星印刷厂
装　　订：三河市皇庄路通装订厂
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编 100036
开　　本：860×1092　1/16　　印张：36　　字数：843 千字
印　　次：2010 年 7 月第 1 次印刷
印　　数：4000 册　　定价：79.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

导　读

Linux 那些事儿之我是 USB Core

2 - 6，对 USB 协议规范的简单描述。

7，USB 设备在 sysfs 文件系统中的表示。

8 - 9，通过对 README、Kconfig、Makefile 文件的分析，定位要分析的目标代码范围。

10 - 11，USB 子系统的初始化函数 usb_init()。

12 - 13，2.6 内核的设备模型，以及设备模型在 USB 子系统中的映射。

14 - 19，USB 子系统实现中的几个重要数据结构。

- 14，struct usb_interface，接口。
- 15，struct usb_host_interface，设置。
- 16，struct usb_host_endpoint，端点。
- 17，struct usb_device，设备。
- 18，struct usb_host_config，配置。
- 19，struct usb_driver，USB 接口驱动，struct usb_device_driver，USB 设备驱动，以及 USB 设备与 USB 驱动的匹配函数 usb_device_match()。

20 - 30，USB 设备连接到 Hub 上之后，内核中 USB 子系统的处理过程。

- 20，设备生命线（一），usb_alloc_dev()，USB 设备的构造函数。
- 21，设备生命线（二），usb_control_ msg()，创建一个控制 urb，并把它发送给 USB 设备，然后等待它完成。
- 22-23，设备生命线（三）、设备生命线（四），struct urb，描述 USB 数据传输的结构。
- 24，设备生命线（五），usb_alloc_urb()，创建一个 urb，usb_fill_control_urb()，初始化控制 urb，usb_start_wait_urb()，将 urb 提交给 USB Core，以便分配给特定的主机控制器驱动进行处理，然后默默地等待处理结果。
- 25，设备生命线（六），usb_submit_urb，启动和停止 USB 数据传输，对 urb 做些前期处理后扔给 HCD。
- 26-27，设备生命线（七）、设备生命线（八），struct usb_hcd，USB 主机控制器驱动，struct usb_bus，USB 总线。

- 28，设备生命线（九），usb_hcd_submit_urb()，将提交过来的 urb 指派给合适的主机控制器驱动程序。
- 29，设备生命线（十），usb_get_device_descriptor()，获得设备描述符，usb_get_configuration()，获得配置描述符。
- 30，设备生命线（十一），usb_parse_configuration()，解析配置描述符。

31-34，USB 驱动从注册到卸载的处理过程。

- 31，驱动生命线（一），usb_register_device_driver()，注册 USB 世界里唯一的那个 USB 设备驱动（不是 USB 接口驱动）struct device_driver 结构体对象 usb_generic_driver。
- 32-34，驱动生命线（二）、驱动生命线（三）、驱动生命线（四），usb_set_configuration()，配置设备。

35，字符串描述符。

36，usb_register()，注册接口的驱动。

37，usb_device_match()，匹配 USB 设备与 USB 驱动。

Linux 那些事儿之我是 HUB

3，Root Hub，与 USB 主机控制器集成在一起的根 Hub。usb_hub_init()，Hub 的初始化程序。

4，struct usb_driver hub_driver，Hub 的接口驱动程序。

5，hub_thread()，Hub 驱动中最精华的代码。struct usb_hub，Hub 数据结构。hub_event_list，Hub 事件链表。

6，hub_events()，处理线程 khubd 的工作链表 hub_event_list。

7，hub_probe()，Hub 驱动的 probe 函数。

8，工作队列。

9-13，配置 Hub 的过程。

- 9，hub_configure()，配置 Hub 设备的函数。
- 10，get_hub_descriptor()，获得 Hub 描述符。
- 11，struct usb_tt，transaction translator 的数据结构，负责 Hub 在高速和低速/全速的数据转换。hub_hub_status()，返回 Hub 状态的函数。
- 12-13，kick_khubd()，唤醒等待队 khubd_wait 上的等待线程 hub_thread()。

14，locktree()，锁住 USB 设备树。

15 - 16，hub_port_status()，返回 Hub 各个端口的状态。hub_port_connect_change()，当 Hub 端口上有连接变化时调用这个函数。

18-22，Hub 端口上有连接变化时的处理过程。

- 18，usb_set_device_state()，设置设备状态的函数。
- 19，choose_address()，为设备选择地址的函数。
- 20，hub_port_init()，Hub 所含端口的初始化函数。
- 22，usb_new_device()， 寻找驱动程序，调用驱动程序的 probe，跟踪这个函数就能一直跟踪到设备驱动程序的 probe()函数的调用。

23，hub_power_on()，Hub 上电。

24，hub_irq()，Hub 的中断处理函数，负责调用 kick_khubd()。

Linux 那些事儿之我是 UHCI

1 - 2，UHCI 驱动的初始化和退出。

- uhci_hcd_init()，UHCI 驱动的初始化函数。
- uhci_hcd_cleanup()，UHCI 驱动的退出函数。
- struct pci_driver uhci_pci_driver，从 PCI 层面上描述，UHCI 的驱动（注意 UHCI 是一个 PCI 设备）。

3，usb_hcd_pci_probe()，UHCI 驱动的 probe 函数。usb_create_hcd()，创建 struct usb_hcd。struct hc_driver uhci_driver，UHCI 驱动结构。

4，I/O 内存与 I/O 端口。

5，usb_add_hcd()，初始化并注册 struct usb_hcd。

6，usb_register_bus()，注册 USB 总线函数。

7，uhci_init()，UHCI 初始化。

8，UHCI 的中断资源。

9-13，uhci_start()，初始化 UHCI 的帧列表。uhci_alloc_td()，创建 Transfer Descriptors。uhci_alloc_qh()，创建 Queue Heads。

14，usb_hcd_poll_rh_status()，轮询 Root Hub 的状态变化情况。

15 - 16，Root Hub 的控制传输过程。

- usb_submit_urb，启动和停止 USB 数据传输，对 urb 做些前期处理后扔给 HCD。
- usb_hcd_submit_urb()，主机控制器向设备提交 urb。
- rh_urb_enqueue()，对 Root Hub 的 urb 进行排队。
- rh_call_control()，向 Root Hub 发送控制命令。

17，uhci_submit_bulk()，非 Root Hub 的批量传输。

18，uhci_irq()，UHCI 的中断服务程序。

19，rh_queue_status()，Root Hub 的中断传输。

20，uhci_submit_interrupt()，非 Root Hub 的中断传输。

21，uhci_submit_isochronous()，非 Root Hub 的等时传输。

Linux 那些事儿之我是 U 盘

1，通过对 Kconfig、Makefile 文件的分析，定位要分析的目标代码范围。

2 - 4，usb_stor_init()，USB storage 模块的初始化函数。usb_stor_exit()，USB storage 模块的退出函数。

5 - 9，2.6 内核的设备模型，以及设备模型在 USB 子系统中的映射。struct usb_driver usb_storage_driver，U 盘驱动结构。

10，struct usb_device_id，USB 设备的 ID。

14，storage_probe()，USB storage 模块的 probe 函数。struct us_data，会为每一个 USB storage 设备申请一个 us_data。

16，associate_dev()，为 us_data 的各个成员赋值。

17 – 19，get_device_info()，获得设备信息。

20，get_transport()，决定用的是哪种传输模式。

21，get_protocol()，决定用的是哪种通信协议。

22，usb_stor_acquire_resources()，初始化所有需要的动态资源。

23-40，struct urb，传说中的 urb。

- 24– 25，usb_stor_control_thread ()，USB storage 模块的内核守护进程，负责监听命令。
- 26，usb_stor_scan_thread()，Usb storage 模块的内核守护进程，负责扫描设备。
- 27，queuecommand()，排队 SCSI 命令。
- 28，struct scsi_device，SCSI 设备。
- 33，usb_stor_show_command()，打印 SCSI 命令信息。
- 34 – 40，usb_stor_Bulk_transport()，USB storage 模块中批量传输的函数。

45，storage_disconnect()，USB 设备离开主机时的处理。

前 言

1991 年，Linux 诞生了。又因为开放源代码的缘故，十几年来 Linux 是越来越火，熊熊火焰也烧到了华夏大地。诸多高校开始开设 Linux 相关的课程，诸多企业开始招聘 Linux 相关的人才。市面上关于 Linux 的书籍也层出不穷，而这其中大致分为两类，一类是应用方面的书籍，比如介绍如何组建各种服务器；另一类是内核方面的书籍，主要都是对内核源代码进行分析，这方面的书籍则以《Linux 设备驱动程序》和《深入理解 Linux 内核》为经典代表。而从眼下国内的人才市场来看，懂 Linux 内核的人找工作肯定不用发愁。事实上，毫不夸张地说，当代大学生，如果能够看完以上这两本书，并且能够看懂，那么在北京、上海、深圳这些一线城市，很容易就能找到一份体面的技术类工作。

那么为什么我们还要另起炉灶再写一本 Linux 内核方面的书籍呢？因为 Linux 内核包含大量的代码，以上面两本书为代表的很多 Linux 内核的书籍涵盖的内容太广，大多数书籍都是把 Linux 内核中的各个部分逐一地进行介绍和分析，然而实际情况是没有任何一个人能够对 Linux 内核的各个部分都很精通，包括 Linus Torvalds 本人。一个对 Linux 开发感兴趣的人也不一定需要并且有足够的时间对 Linux 的每个部分都去深入理解。而另一方面，很多对 Linux 内核感兴趣的朋友常常被一个问题所困扰，那就是 Linux 内核那么庞大的代码量，对于初学者来说，应该从哪里学起呢？关于这一点，其实《Linux 设备驱动程序》给出了很好的答案，学习驱动程序代码是最有效的入门方法。第一是因为在庞大的 Linux 内核源代码中，大约 87.53%是各种驱动程序代码，其重要性可想而知，第二是因为相对来说，驱动程序的代码其难度是比较低的，很少涉及复杂高深的算法，所以适合初学者研读。

然而网友“永不堕落”曾经问过我们：“既然已经有了《Linux 设备驱动程序》，为什么你们还要写一本 Linux 设备驱动程序相关的书呢？你们这样做是不是行为艺术呢？”这里我们想说的是，虽然《Linux 设备驱动程序》这本书很强大了，把各种类型的设备驱动程序都给介绍了一番，可是当一些读者读完这本书之后，他们敢说自己会写 Linux 设备驱动程序了吗？他们敢说自己完全了解一个真实的 Linux 设备驱动程序是怎么写的吗？至少本书作者当年在看完这本书之后，虽然觉得获益匪浅，可是仍然不太清楚真实的 Linux 设备驱动程序是怎么写的，仍然不敢认为自己就会写 Linux 设备驱动程序了。这才有了后来决定亲自选择一个子系统进行研

究，并在研究好了之后把研究心得写出来，与大家进行分享。最终我们选择的是 USB 子系统，原因很简单，USB 总线及连接在 USB 总线上的各种 USB 设备已经广泛地出现在了当代计算机上，广大计算机用户，尤其是高校学生，接触得最多的设备也正是 USB 设备，所以研究和分析 USB 总线，以及它上面的各种 USB 设备应该能让大家感到很亲切很真实并且具有相当的实用价值和怀旧意义。有人曾经说过：“USB 总线就像一条河，左岸是我无法忘却的回忆，右岸是我值得紧握的璀璨年华，中间流淌的，是我年年岁岁淡淡的感伤！”

本书的编写得到了许多人的帮助，在此向他们致以诚挚的谢意。首先感谢孙学瑛编辑，没有她的努力，这本书的内容将会一直偏居网络一隅，将不可能被出版从而去帮助更多需要的人。然后要感谢很多在技术上给予我们指导与帮助的老师和朋友，特别是 USB 这边的 maintaner 之一，Alan Stem 大侠对我们的问题的耐心解答与回复。

目 录

第 1 篇 Linux 那些事儿之我是 USB Core

第 2 篇 Linux 那些事儿之我是 HUB

第 3 篇　Linux 那些事儿之我是 UHCI

第 4 篇　Linux 那些事儿之我是 U 盘

附录　Linux 那些事儿之我是 sysfs

第 1 篇

Linux 那些事儿之我是 USB Core

1. 引子

老夫子们痛心疾首地总结说，现代青年的写照——自负太高，反对太多，商议太久，行动太迟，后悔太早。上天戏弄，我不幸地混进了“80后”的革命队伍里，成了一名现代青年，前有老夫子的忧心忡忡，后有“90后”的轻蔑嘲弄，终日在“迷失”与“老土”这样的两极词汇里徘徊。

这里我就讲一讲USB，让他们看一看“80后”还知道什么叫USB。

还是要说在前面，在这里耗费青春写USB，并不是因为喜欢它，相反，对它毫无感觉可言，虽然每天都必须和它相依为伴，不离不弃，不过那可是丝毫没有办法的事情，非我所愿。是不是特说到心坎儿里去了？不过您别多想，咱这里只谈USB。

一句话总结：哥写的不是USB，是寂寞。

2. 它从哪里来

“你从哪里来，我的朋友，好像一只蝴蝶，飞进我的窗口。”

在嘹亮的歌声中，USB好像一只蝴蝶飞进了千家万户。它从哪里来，它从Intel来。Intel不养蝴蝶，而是做CPU，它只是在蝴蝶的翅膀上烙上“Intel inside”，蝴蝶让咱们的同胞去养了，然后带着Intel飞进了千万家。

不过，与PCI、AGP属于Intel单独提出的硬件标准不同，Compaq、IBM、Microsoft等也一起参与了USB这个游戏。他们一起于1994年11月提出了USB，并于1995年11月制定了0.9版本，1996年制定了1.0版本。不过USB并没有因为有这些大佬的支持立即迎来它的春天，只怪它诞生在了冬季，生不逢时啊！

因为缺乏操作平台的良好支持和大量支持它的产品，这些标准都成了空谈。1998年USB1.1的出现，忽如一夜春风来，它就像春天里的一朵油菜花，终于涂上了浓重的一抹黄色。

为什么要开发USB？

在USB出现以前，电脑的接口处于“春秋战国时代”，串口并口等多方割据，键盘、鼠标、MODEM、打印机、扫描仪等都要连接在这些不同种类的接口上，一个接口只能连接一个设备。

不过咱们的电脑不可能有那么多接口，所以扩展能力不足，而且速度也确实很有限。还有关键的一点是，热插拔对它们来说也是比较危险的操作。

USB 正是为了解决速度、扩展能力、易用性等问题应景而生的。

3. PK

在 2006 年，最火的是“超级女生”，最流行的词是“PK”。

USB 的一生也充满了“PK”，不过 USB 还不够老，说一生还太早了，发哥说得好：“我才刚上路呢！”

USB 最初的设计目标就是替代串行、并行等各种低速总线，以一种单一类型的总线连接各种不同的设备。它现在几乎可以支持所有连接到 PC 上的设备，1999 年提出的 USB 2.0 理论上可以达到 480 MB/s 的速度，2008 年公布的 USB 3.0 标准更是提供了十倍于 USB 2.0 的传输速度。

因此，USB 与串口、并口等的这场“PK”从一开始就是不平等的，这样的开始也注定了以什么样的结果结束，只能说命运选择了 USB。我们很多人都说命运掌握在自己手里，但是从 USB 充满“PK”的一生中可以知道，只有变得比别人更强，命运才能掌握在自己手里。

有了 USB 在这场 PK 中的大获全胜，才有了 USB 键盘、USB 鼠标、USB 打印机、USB 摄像头、USB 扫描仪、USB 音箱等。至于将来，“PK 自己的，让别人去说吧！”USB 如是说。

4. 漫漫辛酸路

USB 的一生充满了“PK”，并在“PK”中发展，从 USB 1.0、USB 1.1、USB 2.0 到 USB 3.0，漫漫辛酸路，一把辛酸泪。

USB 2.0 的高速模式（High-Speed）最高已经达到了 480 MB/s，也就是说，以这个速度，你将自己从网上下载的短片备份到自己的移动硬盘上的时间长约为一秒钟。而 USB 3.0 的 Super-Speed 模式比这个速度还要提高了几乎 10 倍，达到了 4.8GB/s。

USB 走过的这段辛酸路，对咱们来说最直观的结果也就是传输速度提高了，过程很艰辛，结果很简单。

USB 的各个版本都是兼容的。每个 USB 2.0 控制器带有 3 个芯片，根据设备的识别方式将信号发送到正确的控制芯片。我们可以将 USB 1.1 设备连接到 USB 2.0 的控制器上使用，不过它只能达到 USB 1.1 的速度。同时也可以将 USB 2.0 的设备连接到 USB 1.1 的控制器上，不过不能指望它能以 USB 2.0 的速度运行。

显然，Linux 对 USB1.1 和 USB 2.0 都是支持的，并抢在 Windows 前，在 2.6.31 内核中率先对 USB 3.0 进行了支持。

5. 我型我秀

USB 既然能一路“PK”走过来，也算是一个挺能“秀”的角色了，不然也不会有那么多的拥护者。

USB 为所有 USB 外设都提供了单一的标准的连接类型，这就简化了外设的设计，也让我们不用再去想哪个设备对应哪个插槽的问题，就像种萝卜，一个萝卜一个坑，但是哪个萝卜种到哪个坑里是不用我们关心的。

USB 支持热插拔，而其他的比如 SCSI 设备等只有在关掉主机的前提下才能增加或移走外围设备。所以说，USB 的一生不仅仅是“PK”的一生，也是丰富多彩的一生，可以不用关机就能更换不同种类的外设。

USB 在设备供电方面提供了灵活性。USB 设备可以通过 USB 电缆供电，不然移动硬盘、IPod 等常备外设也用不了了。相对应，有的 USB 设备也可以使用普通的电源供电。

USB 能够支持从每秒几十 KB 到几十 MB 的传输速率，来适应不同种类的外设。它可以支持多个设备同时操作，也支持多功能的设备。多功能的设备当然指的就是一个设备同时有多个功能， 比如 USB 扬声器。这通过在一个设备中包含多个接口来支持，一个接口支持一个功能。

USB 可以支持多达 127 个设备。

USB 可以保证固定的带宽，这个对视频音频设备是利好。

6. 我是一棵树

“我是一棵树，静静地站在田野里，风儿吹过，我不知它的去向，人儿走过，我不知谁会为我停留。”

如图 1.6.1 所示，USB 子系统的拓扑也是一棵树，它并不以总线的方式来部署。

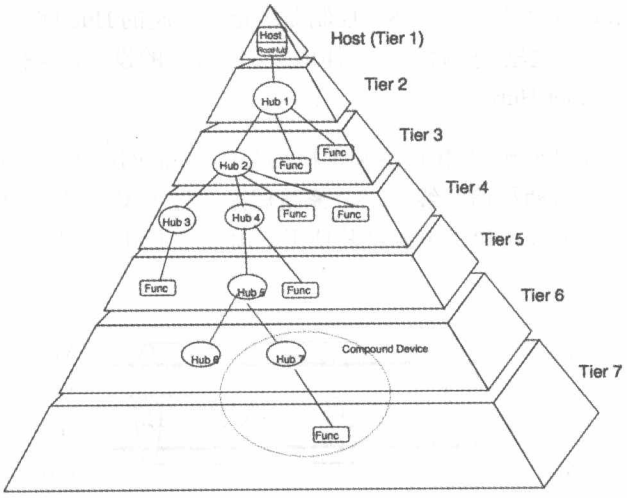

图 1.6.1 USB 子系统的树形结构

我曾经指着路边一棵奇形怪状的树问朋友：“这是什么树？”朋友的回答让我很晕：“大树。”那图 1.6.1 指的是什么树？自然也是大树了，不过却是 USB 的大树。这棵大树主要包括了 USB 连接、USB Host Controller（USB 主机控制器）和 USB 设备三个部分。而 USB 设备还包括了 Hub 和功能设备（也就是上图里的 Func)。

什么是 USB 主机控制器？控制器，顾名思义，用于控制。控制什么？控制所有的 USB 设备的通信。通常计算机的 CPU 并不是直接和 USB 设备打交道，而是和控制器打交道。它要对设备做什么，它会告诉控制器，而不是直接把指令发给设备。然后控制器再去负责处理这件事情，它会去指挥设备执行命令，而 CPU 就不用管剩下的事情。控制器替他去完成剩下的事情，事情办完了再通知 CPU。否则让 CPU 去盯着每一个设备做每一件事情，那是不现实的。

这就好像让一个学院的院长去盯着我们每一个学生上课，管理学生的出勤，这是不现实的。所以学生就被分成了几个系，通常院长有什么指示直接跟各系领导说就可以了，如果他要和三个系主任说事情，他即使不把三个人都召集起来开会，也可以给三个人各打一个电话，打完电话他就忙他自己的事情去了。而三个系主任就会去安排下面的人去执行具体的任务，然后他们就会向院长汇报。