kernel hacker 修炼之道——李万鹏

男儿立志出乡关， 学不成名死不还。 埋骨何须桑梓地， 人生无处不青山。 ——西乡隆盛诗

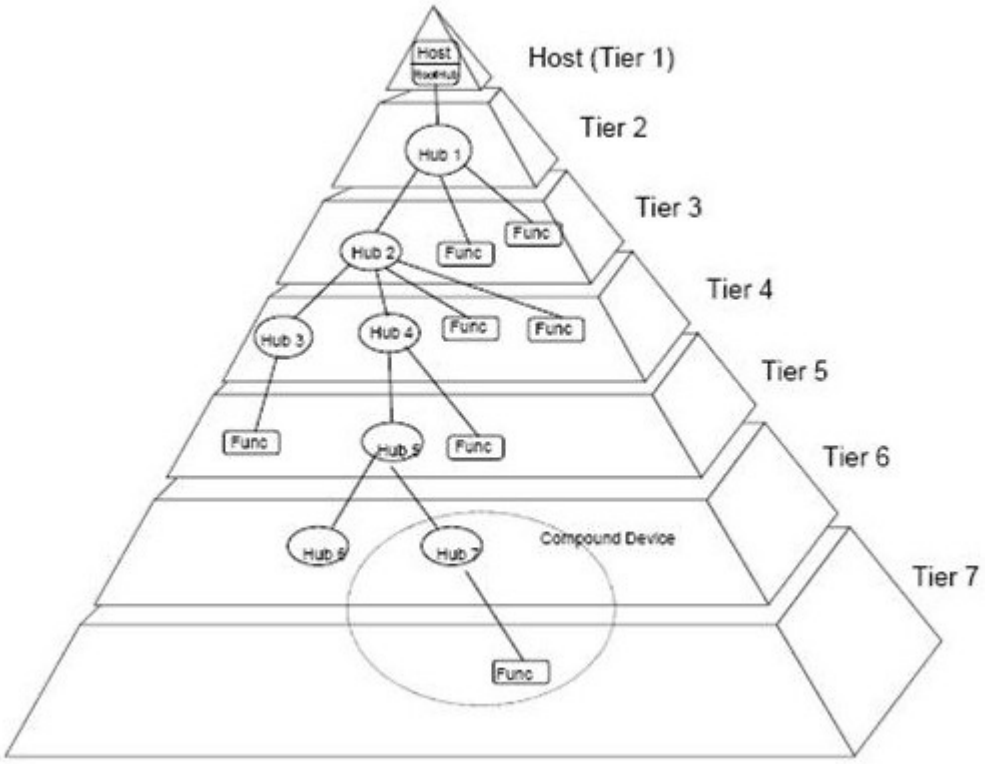
LINUX内核USB子系统学习笔记之初识USB

分类： linux驱动编程 2011-04-25 21:19 1375人阅读 评论(4) 收藏 举报

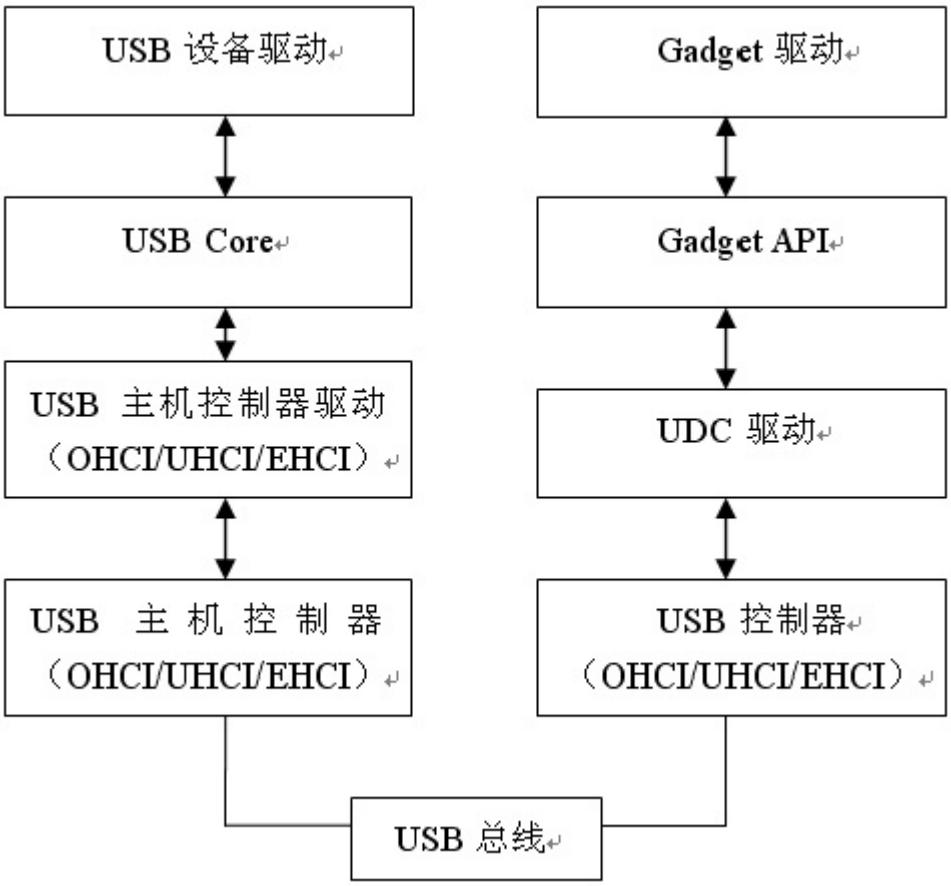
努力成为linux kernel hacker的人李万鹏原创作品，转载请标明出处

http://blog.csdn.net/woshixingaaa/archive/2011/04/25/6362603.aspx

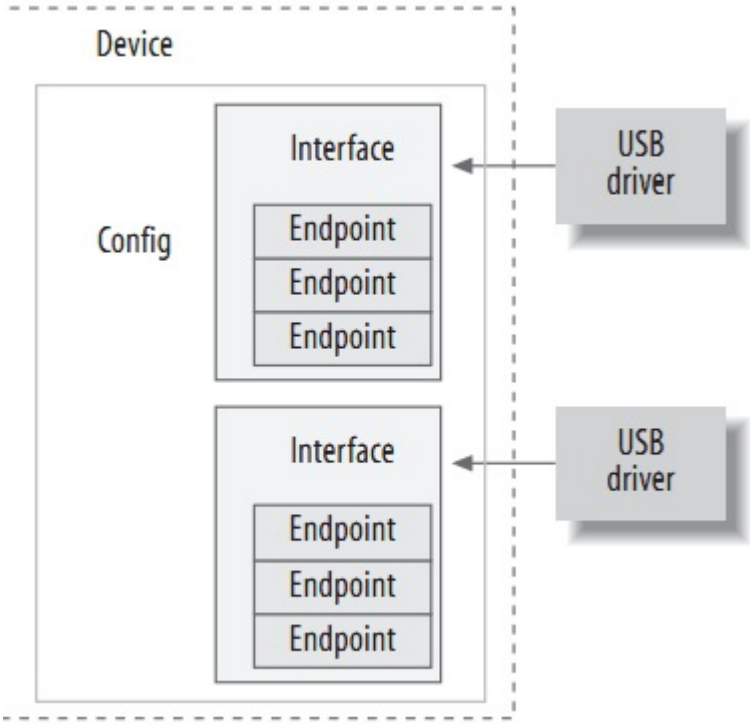
这个是USB系统的拓扑图，4个部分构成：USB主机控制器，根集线器，集线器，设备。其中Root Hub与USB主机控制器是绑定在一起的。



再了解一下USB驱动框架：

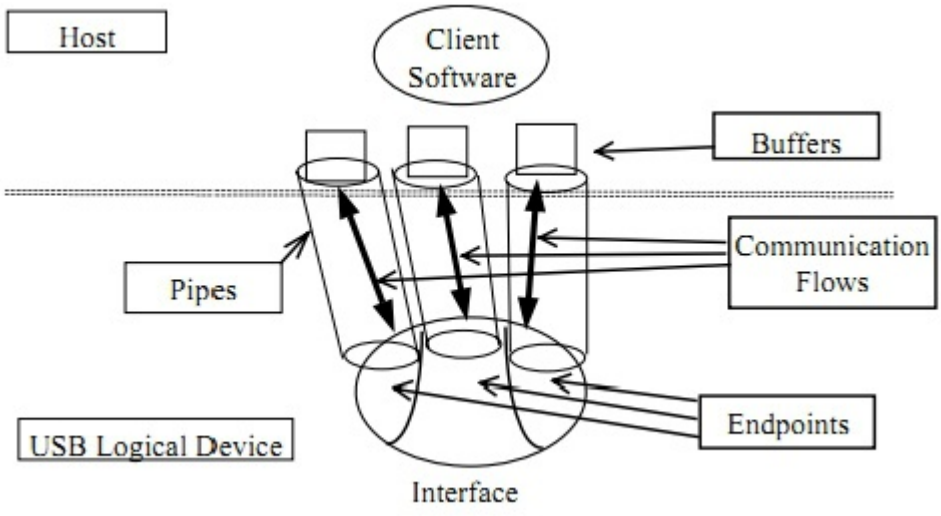


USB通信都是由host端发起的。USB设备驱动程序分配并初始化一个URB发给USB Core,USB Core改一改，发给USB主机控制器驱动，USB主机控制器驱动把它解析成包，在总线上进行传送。USB Core是由内核实现的，其实也就是把host control driver里的功能更集中的向上抽象了一层，它是用来对最上层的USB设备驱动屏蔽掉host control的不同。
USB设备的构成包括了配置，接口和端点。



1. 设备通常具有一个或者更多个配置
2. 配置经常具有一个或者更多个接口
3. 接口通常具有一个或者更多个设置
4. 接口没有或者具有一个以上的端点

需要注意的是，驱动是绑定到USB接口上，而不是整个设备。
USB通信最基本的形式是通过一个名为端点（endpoint）的东西。它是真实存在的。端点只能往一个方向传送数据（端点0除外，端点0使用message管道，它既可以IN又可以OUT），或者IN，或者OUT。除了端点0，低速设备只能有2个端点，高速设备也只能有15个IN端点和15个OUT端点。主机和端点之间的数据传输是通过管道。端点只有在device上才有，协议说端点代表在主机和设备端点之间移动数据的能力。



端点有4中不同的类型：控制，批量，等时，中断。
对应USB的4种不同的传输类型：

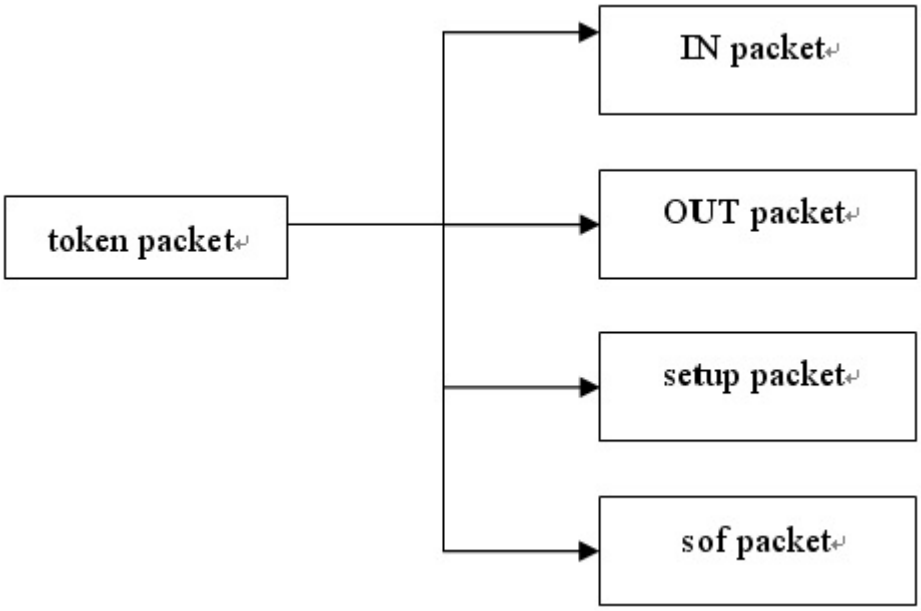
1. 控制传输：适用于小量的，对传输时间和速率没有要求的设备。如USB设备配置信息。

2. 批量传输：适用于类似打印机，扫描仪等传输量大，但对传输时间和速度无要求的设备。

3. 等时传输：适用于大量的，速率恒定，具有周期性的数据，对实时性有要求的，比如音视频。

4. 中断传输：适用于非大量，但具有周期性的数据，比如鼠标键盘。

这4大类由4个transaction组成：
1. IN transaction
IN事务为host输入服务，当host需要从设备获得数据的时候，就需要IN事务。
2. OUT transaction
OUT事务为host输出服务，当host需要输出数据到设备的时候，就需要OUT事务。
3. SETUP transaction
SETUP事务为host控制服务，当host希望传输一些USB规范的默认操作的时候就需要使用setup事务。
4. SOF transaction
这个用于帧同步。
然后这4种transaction又由3类包组成，每类又分几种：



1. in包

in包用于指明当前的事务为in类型的。

2. out包

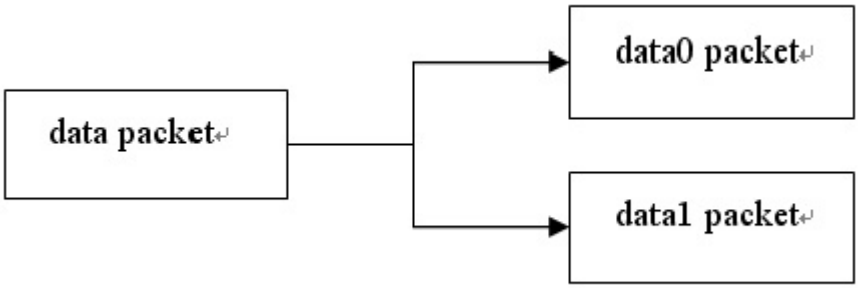
out包用于指明当前事务为out类型的。

3. setup包

setup包指明当前事务为setup类型的。

4. sof包

sof包指明当前事务为setup类型的。

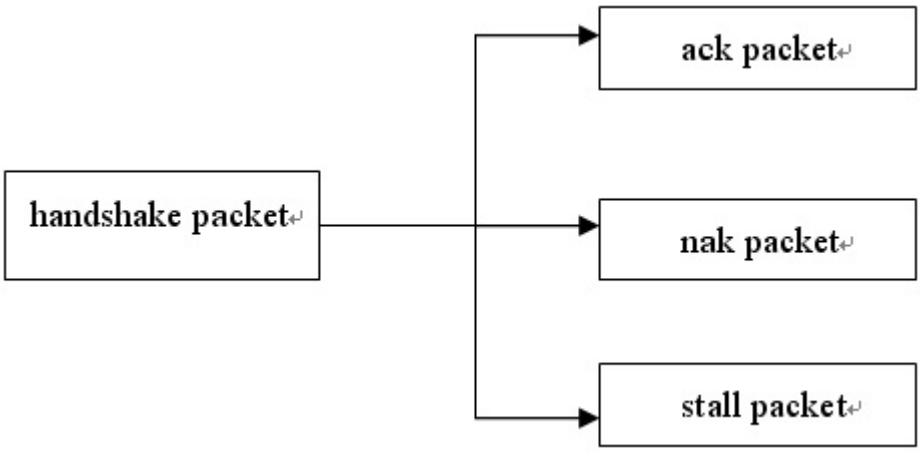


1. data0包

该数据包的类型为0。

2. data1包

该数据包的类型为1。



1. ack包

ack握手包指明当前的事务的数据包传输是成功的。

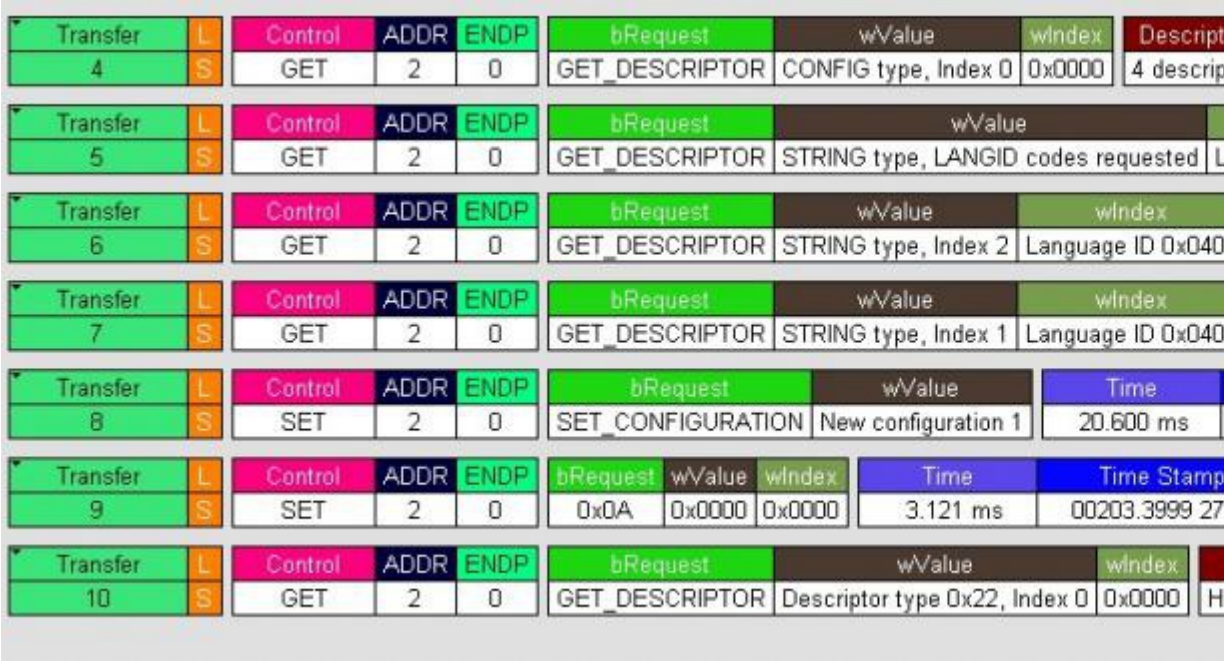
2. nak包

nak握手包指明当前设备忙，不能处理数据包，请主机稍后再次发送。

3. stall包

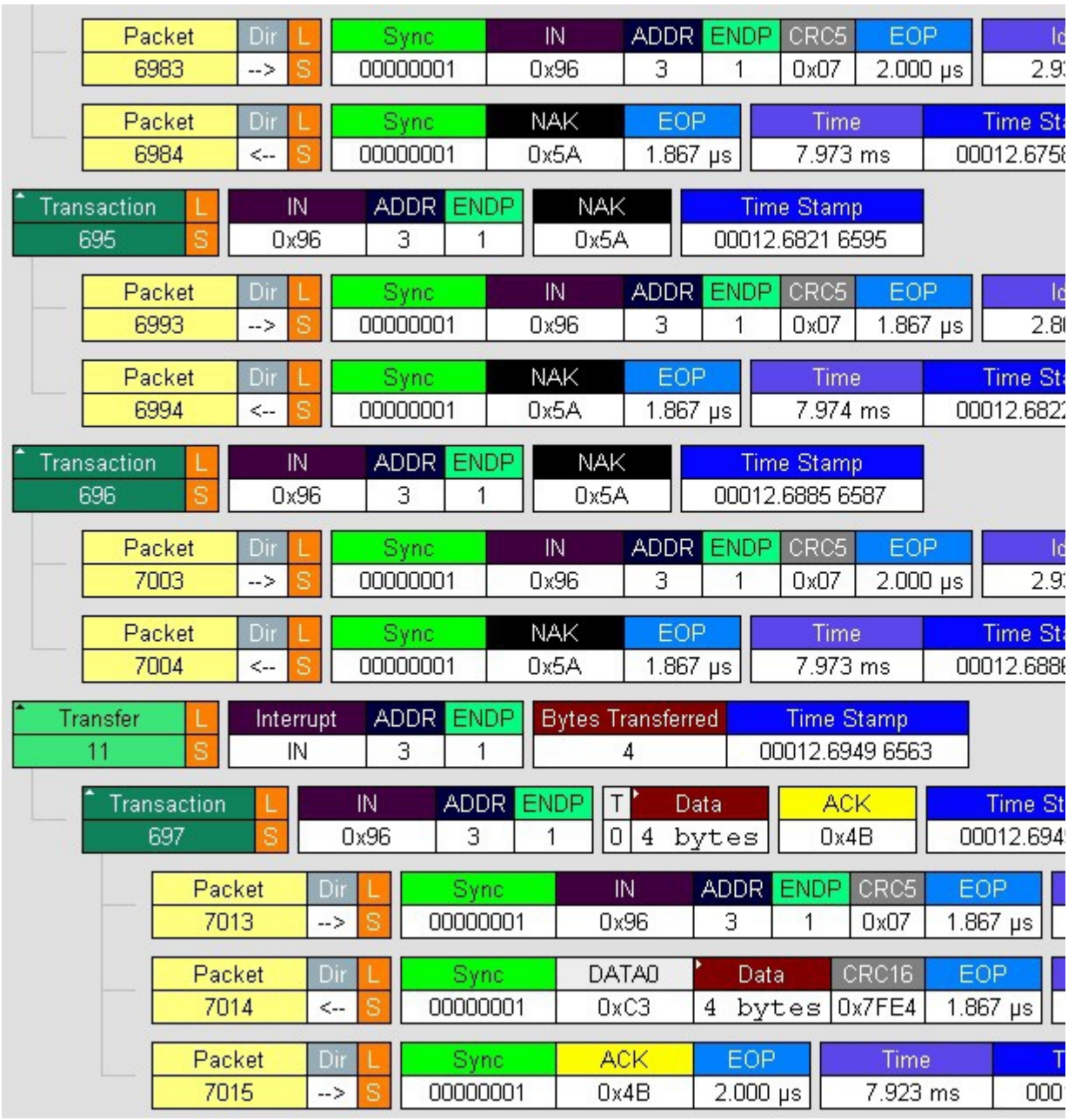
stall握手包指明当前设备不能接受或者传输数据，表示一个严重的错误。
下图是一个USB鼠标插入Linux系统时完整的枚举过程，一共发生了11次传输，每次传输包括几个事务，每个事务又包括几个包，每个包包括几个域。


Transfer L
0 S
Control ADDR ENDP
GET 0 0
bRequest wValue wIndex
GET_DESCRIPTOR DEVICE type 0x0000
Transaction L
0 S
SETUP ADDR ENDP
0xB4 0 0
T D Tp R bRequest wValue
0 D->H S D GET_DESCRIPTOR
Packet Dir L
134 --> S
Sync SETUP ADDR ENDP CRC5 EOP
00000001 0xB4 0 0 0x08 1.867 µs
Packet Dir L
135 --> S
Sync DATA0 Data CRC16 EOP
00000001 0xC3 8 bytes 0xBB29 2.000 µs
Packet Dir L
136 <-- S
Sync ACK EOP Time
00000001 0x4B 1.867 µs 24.667 µs
Transaction L
1 S
IN ADDR ENDP
0x96 0 0
T Data ACK Time
Transfer L
5 S
Control ADDR ENDP
GET 2 0
bRequest
GET_DESCRIPTOR
Transaction L
3 S
IN ADDR ENDP
0x96 0 0
Transaction L
4 S
OUT ADDR ENDP
0x87 0 0
Packet Dir
150 -->
Reset
10.017 ms
Time
133.718 ms
Transfer L
1 S
Control ADDR ENDP
SET 0 0
Transfer L
2 S
Control ADDR ENDP
GET 2 0
Transfer L
3 S
Control ADDR ENDP
GET 2 0
Time Stamp
0203.3638 0583
Time Time Stamp
1.997 ms 00203.3655 3819



Transfer	L	Control	ADDR	ENDP	bRequest	wValue	wIndex	Descript
4	S	GET	2	0	GET_DESCRIPTOR	CONFIG type, Index 0	0x0000	4 descrip
Transfer	L	Control	ADDR	ENDP	bRequest	wValue		
5	S	GET	2	0	GET_DESCRIPTOR	STRING type, LANGID codes requested		L
Transfer	L	Control	ADDR	ENDP	bRequest	wValue	wIndex	
6	S	GET	2	0	GET_DESCRIPTOR	STRING type, Index 2	Language ID 0x040	
Transfer	L	Control	ADDR	ENDP	bRequest	wValue	wIndex	
7	S	GET	2	0	GET_DESCRIPTOR	STRING type, Index 1	Language ID 0x040	
Transfer	L	Control	ADDR	ENDP	bRequest	wValue	Time	
8	S	SET	2	0	SET_CONFIGURATION	New configuration 1	20.600 ms	
Transfer	L	Control	ADDR	ENDP	bRequest	wValue	wIndex	Time	Time Stamp
9	S	SET	2	0	0x0A	0x0000	0x0000	3.121 ms	00203.3999 27
Transfer	L	Control	ADDR	ENDP	bRequest	wValue	wIndex	
10	S	GET	2	0	GET_DESCRIPTOR	Descriptor type 0x22, Index 0	0x0000	H

这里有一个概念需要注意，这里的中断传输与硬件中断那个中断是不一样的，这个中断传输实际是靠USB host control轮询usb device来实现的，而USB host control对于CPU则是基于中断的机制。拿USB鼠标为例，USB host control对USB鼠标不断请求，这个请求的间隔是很短的，在USB spec Table 9-13端点描述符中的bInterval域中指定的，当鼠标发生过了事件之后，鼠标会发送数据回host，这时USB host control中断通知CPU，于是usb_mouse_irq被调用，在usb_mouse_irq里，就可以读取鼠标发回来的数据，当读完之后，驱动再次调用usb_submit_urb发出请求，就这么一直重复下去，一个usb鼠标的驱动也就完成了。下面是USB鼠标中断传输图，可以看到USB host control向usb device发送了IN包，没有数据的时候device回复的是NAK,有数据的时候才向host control发送DATA包。


Packet Dir L Sync IN ADDR ENDP CRC5 EOP
6983 --> S 00000001 0x96 3 1 0x07 2.000 µs
Packet Dir L Sync NAK EOP Time
6984 <-- S 00000001 0x5A 1.867 µs 7.973 ms
Transaction L IN ADDR ENDP NAK Time Stamp
695 S 0x96 3 1 0x5A 00012.6821 6595
Packet Dir L Sync IN ADDR ENDP CRC5 EOP
6993 --> S 00000001 0x96 3 1 0x07 1.867 µs
Packet Dir L Sync NAK EOP Time
6994 <-- S 00000001 0x5A 1.867 µs 7.974 ms
Transaction L IN ADDR ENDP NAK Time Stamp
696 S 0x96 3 1 0x5A 00012.6885 6587
Packet Dir L Sync IN ADDR ENDP CRC5 EOP
7003 --> S 00000001 0x96 3 1 0x07 2.000 µs
Packet Dir L Sync NAK EOP Time
7004 <-- S 00000001 0x5A 1.867 µs 7.973 ms
Transfer L Interrupt ADDR ENDP Bytes Transferred Time Stamp
11 S IN 3 1 4 00012.6949 6563
Transaction L IN ADDR ENDP T Data ACK
697 S 0x96 3 1 0 4 bytes 0x4B
Packet Dir L Sync IN ADDR ENDP CRC5 EOP
7013 --> S 00000001 0x96 3 1 0x07 1.867 µs
Packet Dir L Sync DATA0 Data CRC16 EOP
7014 <-- S 00000001 0xC3 4 bytes 0x7FE4 1.867 µs
Packet Dir L Sync ACK EOP Time
7015 --> S 00000001 0x4B 2.000 µs 7.923 ms