

Linux那些事儿

系列丛书

之

我是EHCI

1 原文为blog.csdn.net/fudan_abc 上的《linux 那些事儿之我是EHCI》，有闲情逸致的或者有批评建议的可以到上面做客，也可以email 到ilttv.cn@gmail.com

目录

目录	2
引子	3
接口体系.....	3
套路	10
pci match 和 probe.....	12
data structure of ehci driver and device	14
2008 年的这一场雪.....	18

引子

转眼之间，到了 2008 年，先祝大家新年快乐，希望新的一年里好运连连，工资猛涨。好久没有写了，一个原因在于，作为一个 PHD 学生，难免要做一些读 paper 写 paper 的琐事，另一个原因就是自己太懒了。大哥甲一如既往，坚持的写作，着实让人钦佩。此时此刻，我情不自禁，作诗一首，北飘奇男子，江南大丈夫。海上常常生明月，江湖就此一枝花。不服不行。

这里主要就 linux ehci host controller 这部分的代码，谈谈我自己理解。不当之处，请多指正。参考资料，ehci spec 和 linux-2.6.22.1 内核。我尽量少贴代码，把问题讨论清楚。

现在开说，要进行 usb 传输，得有一个 usb host controller, usb 主控制器。它与插入系统的 USB 设备进行相互操作，并负责处理 USB 设备与系统其它部分通信所必需的所有低层次细节。

一个 usb2.0 主控制器如图所示

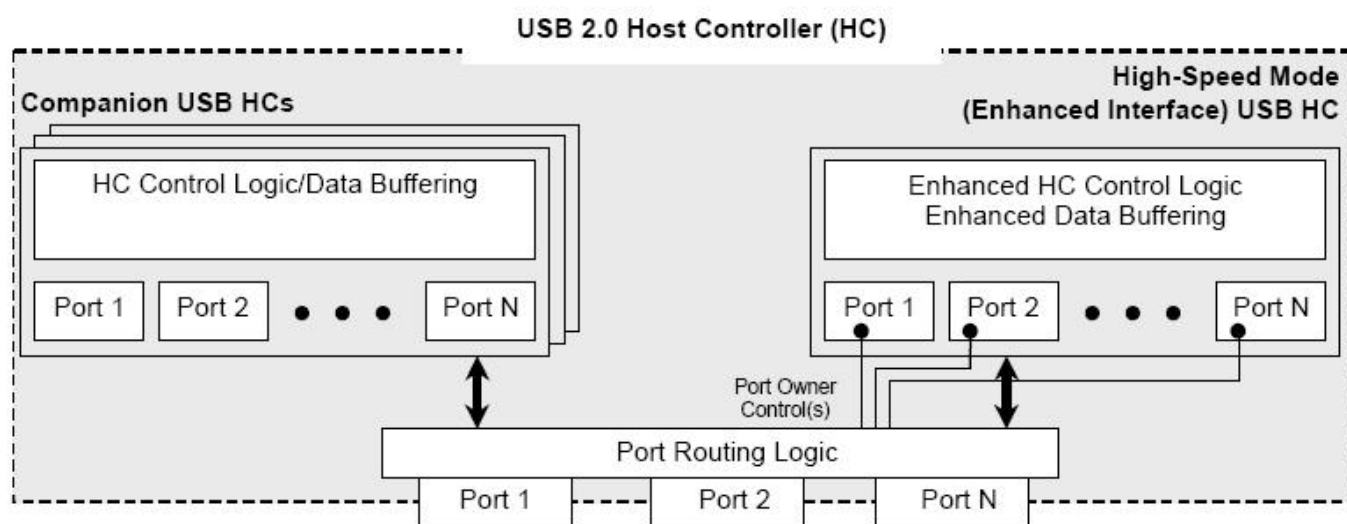


Figure 1-2. USB 2.0 Host Controller

usb 2.0 定义了低速(ls)，全速(fs)，高速(hs)传输。EHCI 仅仅支持高速传输，所以它必须还要有一个 companion HC，如(UHCI)来支持低速和全速设备，情况是这样的：

- 1), fs/ls 设备插入到 root hub port, 会由 companion HC(uhci/ohci)发现并管理设备；
- 2), fs/ls 设备插入到 usb 2.0 hub(not root hub), 那么由 ehci 通过 split transaction 和 transaction translation(tt)支持 fs/ls 设备。

比如，当一个 usb 设备插入 root hub port 时，先要做一件 routing 的事情。所有的 root hub port 默认是被 EHCI 占有的，所以，EHCI 和插入的 usb 设备通信，看是不是 hs 设备，如果是好说。如果不是，EHCI 就放弃这个 port 的占有权，让给 companion HC(uhci/ohci)去管理。

接口体系

EHCI 首先是一个 PCI 设备，我们可以 lspci 一下看看。

00:1a:7 USB Controller: Intel Corporation USB2 EHCI Controller #1 (rev 03)

我们与外围硬件打交道，可以把数据用 in(out)指令传递给外围硬件，还可以把数据传输到 cpu 和外围硬件共享的内存里面去。这些都是计算机与硬件的接口。（参见 ldd3 第 9 章）

那么我们的程序如何与 EHCI 联系，交流呢？EHCI 定义了三个接口空间。如图

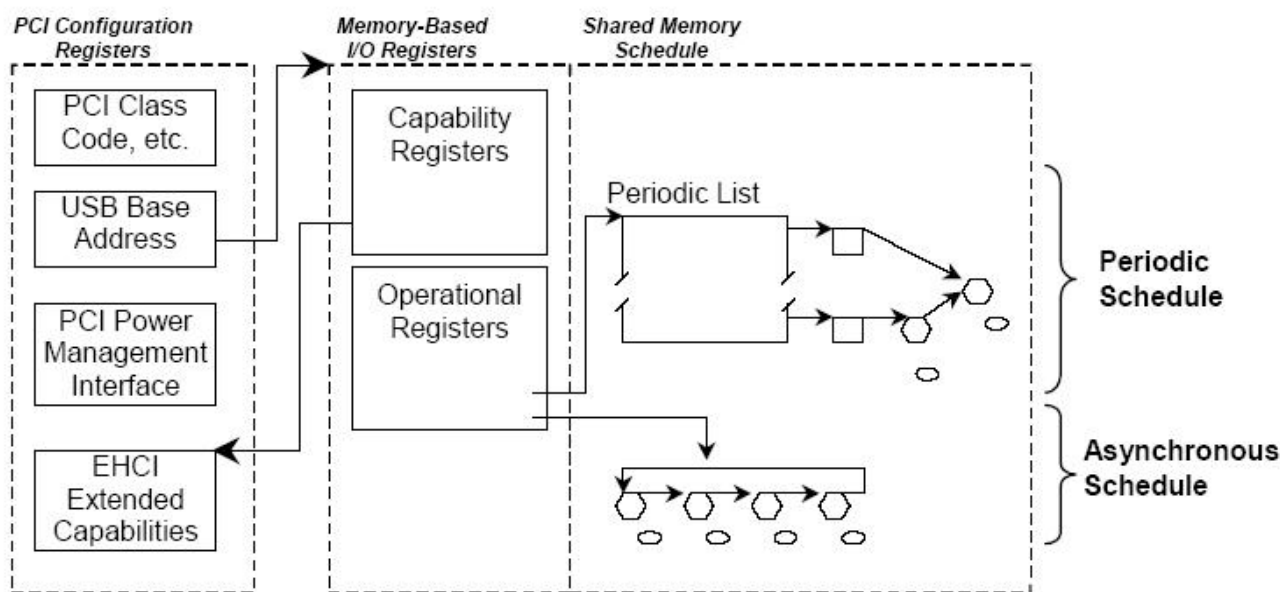


Figure 1-3. General Architecture of Enhanced Host Controller Interface

作为一个程序员，我们关心的是如何在代码中读/写这些地方的内容。概念性的东西肯定是 LDD3 写的最好，我就不赘述了。

1) pci configuration space. (ldd3 第 12 章)

由于 EHCI 是一个 PCI 设备，这里用于系统组件枚举和 PCI 的电源管理。

以 x86 为例，读取 PCI 总线套路是这样的。我们要读取 PCI 总线上地址为 add，长度为 4 个字节的内容。

```
outl(add, 0xcf8); // 先把add的out到地址为 0xcf8 的地方
value = inl(0xcfc); // 然后再读取 0xcfc的内容
```

网上找到了一段程序，大家可以试验一下。

```
/* name: pci.c */
#include <stdio.h>
#include <assert.h>
#include <sys/io.h>

#define IO_PORTS1 1 /* ioport <= 0x3ff */
#define IO_PORTS2 2 /* ioport >0x3ff && ioport < 0xffff */
#define IO_PERMOFF 0
#define IO_PERMON 1
#define IO_PERMON2 3

#define RW_DELAY 10000 /*delay 100000 microseconds for reading and writing I/O ports. */
#ifdef BOOL
```

```

typedef unsigned char BOOL;
#endif

#ifndef BYTE
typedef unsigned char  BYTE;
#endif



#ifndef DWORD
typedef unsigned long  DWORD;
#endif

#ifndef INT
typedef unsigned int INT;
#endif

#ifndef ULONG
typedef unsigned long  ULONG;
#endif

#ifndef WORD
typedef unsigned short WORD;
#endif

```

  /*

** Function : Write the value of the specified I/O port by giving the length and the

e

| ** starting address.

| ** Parameter: PortAddr: the port address



| ** PortVal : the value to set

| ** size : size = 1 for reading 1 byte, 2 for word, 4 for double words

| ** Return : 1 returned if success, or 0 returned

|_* /

BOOL SetPortVal(WORD PortAddr, DWORD PortVal, BYTE size)

  {

| BOOL Ret = 0;



| INT tmpRet = 0;

| ULONG numperm = 1;

| INT privilege = 0;

| assert(PortAddr>0);

| if(PortAddr <=0x3ff)

  {

| tmpRet = ioperm((ULONG)PortAddr, numperm, IO_PERMON);

| privilege = IO_PORTS1;

| }


```

| ** Parameter: PortAddr : the port address
| **          PortVal  : value from port
| **          size    : size = 1 for reading 1 byte, 2 for word, 4 for double words
| ** Return   : 1 returned if success, or 0 returned.
| */
|
| BOOL GetPortVal(WORD PortAddr, DWORD * PortVal, BYTE size)
| {
|     BOOL Ret    = 0;
|     int  tmpRet = 0;
|     unsigned long numperm = 1;
|     int  privilege = 0;
|     assert(PortAddr>0);
|     assert(PortVal!=NULL);
|     if(PortAddr <=0x3ff)
|     {
|         tmpRet = ioperm((unsigned long)PortAddr, numperm, IO_PERMON);
|         privilege = IO_PORTS1;
|     }
|     else if( PortAddr > 0x3ff)
|     {
|         tmpRet = iopl(IO_PERMON2);
|         privilege = IO_PORTS2;
|     }
|     else
|         return Ret;
|     if(tmpRet<0)
|     {
|         fprintf(stderr, "can't set the io port permission for reading ! ");
|         return Ret;
|     }
|     else
|     {
|         switch(size)
|         {
|             case 1: /*read one byte from the port */
|                 *PortVal = inb(PortAddr);
|                 break;
|             case 2: /*read one word from the port */
|                 *PortVal = inw(PortAddr);
|                 break;
|             case 4: /*read double words from the port */
|                 *PortVal = inl(PortAddr);
|                 break;
|             default:

```

```

        Ret = 0;
        break;
    }
    usleep(RW_DELAY);
    Ret = 1;
}

if( privilege == IO_PORTS1 )
    ioperm( (unsigned long)PortAddr, numperm, IO_PERMOFF );
else if( privilege == IO_PORTS2 )
    iopl(IO_PERMOFF);
return Ret;
}

int main (int argc, char * argv[])
{
    WORD add_port = 0xcf8;
    WORD data_port = 0xcfc;
    DWORD addr = 0x80000000;
    DWORD port_value;
    BYTE size = 4;
    int input;
    printf("Please select the option number as follow: ");
    printf("1--bus 0:dev:0 fun:0 as address 0x80000000 ");
    printf("2--bus 0:dev:1 fun:0 as address 0x80000800 ");
    printf("3--input your own defined address value: ");
    scanf("%d",&input);
    switch(input)
    {
        case 1:
            addr=0x80000000;
            break;
        case 2:
            addr=0x80000800;
            break;
        case 3:
            printf("please input the 32 bits address in Hex format(such as 8000
7800): ");
            scanf ("%x", &addr);
            break;
        default:
            printf("input invalid option num, exit program. ");
            return -1;
    }
}

```



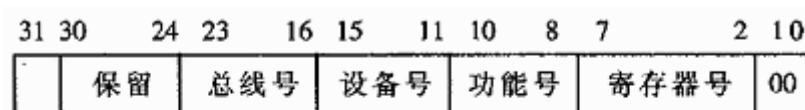
```

|   printf ("The addr is :%X ", addr);
|   printf ("The add_port is : %X ", add_port);
|   printf ("The data_port is : %X ", data_port);
|   if (SetPortVal(add_port, addr, size))
|   {
|       if (GetPortVal(data_port, &port_value, size))
|       {
|           printf("port value is :%08X ", port_value);
|           return 0;
|       }
|   }
|   return -1;
| }

```

打印出来的内容与(1)用 `lspci -xxx` 命令输出；(2)EHCI spec 2.1 章的内容对照一下。

好了，现在问题是我们怎么知道 PCI 总线上 EHCI 的地址 `add`。`lspci` 可以看到所有 PCI 设备的地址。首先，EHCI 不管有没有驱动，它这个 PCI 设备在 PCI 总线枚举时就被探测到了，这时候它就被分配了地址。每个 PCI 外设有一个总线号，一个设备号，一个功能号标识号。比如 00:1a:7，00 总线号，1a 设备号，7 功能号。这些个号组成了独一无二的 ID。ID 和地址的转换关系是这样的：



使能位 (1=允许, 0=禁止)

图 1 配置空间地址寄存器的格式

我们只要 ID，就知道了外设的地址，然后就可以读写 PCI 寄存器的内容。另外可以看看 `pci_read()`，`pci_write()` [\\arch\\i386\\pci\\common.c](#) 的内容，这样会有更深的理解。

2) register space.

这是基于内的 i/o 寄存器，就是 i/o 内存。这里包含了 Capability Registers 和 Operational Registers。我们可以读取 `/proc/iomem` 看看 io 内存的分配情况。我们可以看到 ehci 的地址是 `fe226400-fe2267ff`。这段内存不可以直接读写，先要调用 `ioremap`（或是 `ioremap_nocache`）影射成虚拟地址后再使用。

我写了一段程序。大家可以试验一下。

```

#include <asm/io.h>
static int hello_init(void)
{
|   unsigned long port_value, mem_value;
|   void __iomem *add;
|   int i;
|   printk(KERN_ALERT "Hello, world ");
|   add = ioremap(0xfe226400, 0x400);
|   for(i = 0; i < 100; i++){

```

```

|         mem_value = ioread32(add+i*4);
|         printk("%08X mem value is :%08X ", add+i*4, mem_value);
|     }
|     iounmap(add);
|     return 0;
| }

```

以上是基于 ldd3 中那个最简单的模块 hello.ko 改的。主要是为了可以在内核空间运行。大家可以把打印出来的内容与 ehci spec 2.2 对照一下。

3) Schedule Interface Space.

这里就是普通的内存。我们直接就可以访问它。

套路

子曰：按套路出牌。的确，什么东西都有套路，泡妞有泡妞的套路，花前月下不如花钱日下。打麻将会有打麻将的套路，少吃少碰少放炮，多摸多杠多发财。星际有星际的套路，linux 也有 linux 的套路。刘涛姐姐的故事再一次告诉我们，年龄不是问题，身高不是距离，有 cai 就行。

我们不妨看看 modprobe ehci-hcd 之后发生了什么事情。ehci-hcd 是一个驱动程序，不知您记不记得我在 sysfs 中谈论过设备模型。有两个重要的链表挂在 bus 上，一个是设备 device 链表，一个是驱动 driver 链表。

每当我们向一根 bus 注册一个驱动 driver 时，套路是这样的：

```

driver_register(struct device_driver * drv) -> bus_add_driver() -> driver_attach() ->
bus_for_each_dev(drv->bus, NULL, drv, __driver_attach);
bus_for_each_dev 遍历该总线上所有的 device，执行一次__driver_attach()，看能不能将驱动
关联(attach)到某个设备上去。
__driver_attach()
->driver_probe_device()
->drv->bus->match(dev, drv), // 调用 bus 的 match 函数，看 device 和 driver 匹不匹
配。如果匹配上，
                                继续执行 really_probe()。
->really_probe()
->driver->probe()。(如果 bus->probe 非空，则调用 bus->probe)

```

而每当我们向一根 bus 添加一个硬件时时，套路是这样的：

```

device_add()
    // device_add 中有很多操作kobject，注册sysfs，形成硬件hierarchy结构的代码。
    如果您忘记了，先回头去参考参考"我是 sysfs"
    ->bus_attach_device() -> device_attach() ->bus_for_each_drv()
bus_for_each_drv 与 bus_for_each_dev 类似，遍历该总线上所有的 driver，执行一次
__device_attach()，看能不能将设备关联(attach)到某个已登记的驱动上去。
__device_attach()
->driver_probe_device() //后面与上面一样

```

总结一些，一句话，注册一个某个 bus 的驱动就是先把驱动自己链入到 bus 驱动链表中去，再从 bus 的设备链表中一一寻找，看有没有自己可以关联上的设备。找到就 probe，再把二者 bind 起来。反之，添加设备道理也是一样的。

好吧，我们还是看看 modprobe ehci-hcd 后的事情。一切从此开始，

module_init(ehci_hcd_init);

我们把不必要的预编译代码去掉后，ehci_hcd_init 如下：

```
static int __init ehci_hcd_init(void)
{
    int retval = 0;

    pr_debug("%s: block sizes: qh %Zd qtd %Zd itd %Zd sitd %Zd ",
            hcd_name,
            sizeof(struct ehci_qh), sizeof(struct ehci_qtd),
            sizeof(struct ehci_itd), sizeof(struct ehci_sitd));

    retval = pci_register_driver(&PCI_DRIVER);
    if (retval < 0) {
        return retval;
    }
    if (retval < 0) {
        platform_driver_unregister(&PLATFORM_DRIVER);
        pci_unregister_driver(&PCI_DRIVER);
        return retval;
    }
}
```

PCI_DRIVER 是一个宏，#define PCI_DRIVER ehci_pci_driver。

```
static struct pci_driver ehci_pci_driver = {
    .name = (char *) hcd_name,
    .id_table = pci_ids,

    .probe = usb_hcd_pci_probe,
    .remove = usb_hcd_pci_remove,

#ifdef CONFIG_PM
    .suspend = usb_hcd_pci_suspend,
    .resume = usb_hcd_pci_resume,
#endif
    .shutdown = usb_hcd_pci_shutdown,
};
```

ehci_hcd_init 很简单就是调用了 pci_register_driver()，就是__pci_register_driver()。

```
int __pci_register_driver(struct pci_driver *drv, struct module *owner,
        const char *mod_name)
```

```

{
    int error;

    /* initialize common driver fields */
    drv->driver.name = drv->name;
    drv->driver.bus = &pci_bus_type;
    drv->driver.owner = owner;
    drv->driver.mod_name = mod_name;
    drv->driver.kobj.ktype = &pci_driver_kobj_type;

    spin_lock_init(&drv->dynids.lock);
    INIT_LIST_HEAD(&drv->dynids.list);

    /* register with core */
    error = driver_register(&drv->driver);
    if (error)
        return error;

    error = pci_create_newid_file(drv);
    if (error)
        driver_unregister(&drv->driver);

    return error;
}

```

`driver_register(struct device_driver * drv)`就是前面讲过了，就是 linux 的套路。那我们看看 pci 总线的 `match`, `probe` 函数是什么样的吧。

pci match 和 probe

`pci_bus_type` 定义如下，

```

struct bus_type pci_bus_type = {
    .name      = "pci",
    .match     = pci_bus_match,
    .uevent    = pci_uevent,
    .probe     = pci_device_probe,
    .remove    = pci_device_remove,
    .suspend   = pci_device_suspend,
    .suspend_late = pci_device_suspend_late,
    .resume_early = pci_device_resume_early,
    .resume    = pci_device_resume,
    .shutdown  = pci_device_shutdown,
    .dev_attrs = pci_dev_attrs,
};

```

```

static int pci_bus_match(struct device *dev, struct device_driver *drv)
{
    struct pci_dev *pci_dev = to_pci_dev(dev);
    struct pci_driver *pci_drv = to_pci_driver(drv);
    const struct pci_device_id *found_id;

    found_id = pci_match_device(pci_drv, pci_dev);
    if (found_id)
        return 1;

    return 0;
}

```

总的来说，判断一个设备和驱动是否匹配，是看设备的描述符是否和驱动所支持的一样。`pci_match_device()` 分别在 `driver->dynid`，`driver->id_table` 这两个列表（由一系列的 `pci_device_id` 构成）里面查找。找到则返回这个设备的 `pci_device_id`。（不妨比较一下 `pci_bus_type->match` 和 `usb_bus_type->match`）

```

struct pci_device_id {
    __u32 vendor, device;    /* Vendor and device ID or PCI_ANY_ID */
    __u32 subvendor, subdevice; /* Subsystem ID's or PCI_ANY_ID */
    __u32 class, class_mask; /* (class, subclass, prog-if) triplet */
    kernel_ulong_t driver_data; /* Data private to the driver */
};

```

注意，`pci_device_id->driver_data` 指向了每个 `pci` 设备驱动所特有的数据结构，比如 `ehci` 来说：`.driver_data = (unsigned long) &ehci_pci_hc_driver`。

另外就是，

```

static int pci_device_probe(struct device *dev)
{
    int error = 0;
    struct pci_driver *drv;
    struct pci_dev *pci_dev;

    drv = to_pci_driver(dev->driver);
    pci_dev = to_pci_dev(dev);
    pci_dev_get(pci_dev);
    error = __pci_device_probe(drv, pci_dev);
    if (error)
        pci_dev_put(pci_dev);

    return error;
}

```

`pci_device_probe()` ---> `__pci_device_probe()` ---> `pci_call_probe()` ---> (`pci_driver->probe()`)
而 `ehci_pci_driver->probe = usb_hcd_pci_probe()`。像 `pci_device_probe` 的外包函数，就是一种面向对象的设计。不管怎样，经历了千辛万苦，咱终于绕到 `usb` 了。

data structure of ehci driver and device

阿扁"辞职"了，kde4 发布了，更让我激动的是，英雄志过两天又有更新了，这部连载长达 8 年的小说，终于要进入精彩的大结局。卢云的命运究竟如何？观海云远，四个性格理念完全不同的人，谁是好，谁是坏，谁是对，谁是错？何谓正道？

接着上回说，usb_hcd_pci_probe 这个函数在"我是 UHCI"中也有讨论，不过我想按照我的思路写下去。

```

46 /**
47  * usb_hcd_pci_probe - initialize PCI-based HCDs
48  * @dev: USB Host Controller being probed
49  * @id: pci hotplug id connecting controller to HCD framework
50  * Context: !in_interrupt()
51  *
52  * Allocates basic PCI resources for this USB host controller, and
53  * then invokes the start() method for the HCD associated with it
54  * through the hotplug entry's driver_data.
55  *
56  * Store this function in the HCD's struct pci_driver as probe().
57  */
58 int usb_hcd_pci_probe (struct pci_dev *dev, const struct pci_device_id *id)
59 {
60     struct hc_driver    *driver;
61     struct usb_hcd      *hcd;
62     int                  retval;
63
64     if (usb_disabled())
65         return -ENODEV;
66
67     if (!id || !(driver = (struct hc_driver *) id->driver_data))
68         return -EINVAL;
69
70     if (pci_enable_device (dev) < 0)
71         return -ENODEV;
72     dev->current_state = PCI_D0;
73     dev->dev.power.power_state = PMSG_ON;
74
75     if (!dev->irq) {
76         dev_err (&dev->dev,
77                 "Found HC with no IRQ. Check BIOS/PCI %s setup! ",
78                 pci_name(dev));
79         retval = -ENODEV;

```

```

80         goto err1;
81     }
82
83     hcd = usb_create_hcd (driver, &dev->dev, pci_name(dev));
84     if (!hcd) {
85         retval = -ENOMEM;
86         goto err1;
87     }
88
89     if (driver->flags & HCD_MEMORY) {    // EHCI, OHCI
90         hcd->rsrc_start = pci_resource_start (dev, 0);
91         hcd->rsrc_len = pci_resource_len (dev, 0);
92         if (!request_mem_region (hcd->rsrc_start, hcd->rsrc_len,
93             driver->description)) {
94             dev_dbg (&dev->dev, "controller already in use ");
95             retval = -EBUSY;
96             goto err2;
97         }
98         hcd->regs = ioremap_nocache (hcd->rsrc_start, hcd->rsrc_len);
99         if (hcd->regs == NULL) {
100             dev_dbg (&dev->dev, "error mapping memory ");
101             retval = -EFAULT;
102             goto err3;
103         }
104
105     } else {    // UHCI
106         int    region;
107
108         for (region = 0; region < PCI_ROM_RESOURCE; region++) {
109             if (!(pci_resource_flags (dev, region) &
110                 IORESOURCE_IO))
111                 continue;
112
113             hcd->rsrc_start = pci_resource_start (dev, region);
114             hcd->rsrc_len = pci_resource_len (dev, region);
115             if (request_region (hcd->rsrc_start, hcd->rsrc_len,
116                 driver->description))
117                 break;
118         }
119         if (region == PCI_ROM_RESOURCE) {
120             dev_dbg (&dev->dev, "no i/o regions available ");
121             retval = -EBUSY;
122             goto err1;
123         }

```

```

124     }
125
126     pci_set_master (dev);
127
128     retval = usb_add_hcd (hcd, dev->irq, IRQF_SHARED);
129     if (retval != 0)
130         goto err4;
131     return retval;
132
133 err4:
134     if (driver->flags & HCD_MEMORY) {
135         iounmap (hcd->regs);
136 err3:
137         release_mem_region (hcd->rsrc_start, hcd->rsrc_len);
138     } else
139         release_region (hcd->rsrc_start, hcd->rsrc_len);
140 err2:
141     usb_put_hcd (hcd);
142 err1:
143     pci_disable_device (dev);
144     dev_err (&dev->dev, "init %s fail, %d ", pci_name(dev), retval);
145     return retval;
146 }

```

64 行, `usb_disabled()` 判断内核有没有开启支持 `usb`, 要是这都不支持, 一切都免谈。

70 行, `pci_enable_device()` 这是对 `ehci` 三类接口中的 `pci configuration space` 进行操作, 设置其中某个寄存器的值, 使设备处于工作状态。调用 `pci_read_config_word()` 和 `pci_write_config_word` 来读写 `pci` 配置空间的寄存器。`pci_enable_device -> pci_enable_device_bar -> do_pci_enable_device ...` 原理不难, 只要对照 `spec` 可以看懂。

72, 73 是电源管理的内容。75 判断设备的中断号是否为空。

83, `usb_create_hcd()` 创建一个 `usb_hcd` 机构体。

```

1480 /**
1481  * usb_create_hcd - create and initialize an HCD structure
1482  * @driver: HC driver that will use this hcd
1483  * @dev: device for this HC, stored in hcd->self.controller
1484  * @bus_name: value to store in hcd->self.bus_name
1485  * Context: !in_interrupt()
1486  *
1487  * Allocate a struct usb_hcd, with extra space at the end for the
1488  * HC driver's private data. Initialize the generic members of the
1489  * hcd structure.
1490  *
1491  * If memory is unavailable, returns NULL.

```



```

1492 */
1493 struct usb_hcd *usb_create_hcd (const struct hc_driver *driver,
1494     struct device *dev, char *bus_name)
1495 {
1496     struct usb_hcd *hcd;
1497
1498     hcd = kzalloc(sizeof(*hcd) + driver->hcd_priv_size, GFP_KERNEL);
1499     if (!hcd) {
1500         dev_dbg (dev, "hcd alloc failed ");
1501         return NULL;
1502     }
1503     dev_set_drvdata(dev, hcd);
1504     kref_init(&hcd->kref);
1505
1506     usb_bus_init(&hcd->self);
1507     hcd->self.controller = dev;
1508     hcd->self.bus_name = bus_name;
1509     hcd->self.uses_dma = (dev->dma_mask != NULL);
1510
1511     init_timer(&hcd->rh_timer);
1512     hcd->rh_timer.function = rh_timer_func;
1513     hcd->rh_timer.data = (unsigned long) hcd;
1514 #ifdef CONFIG_PM
1515     INIT_WORK(&hcd->wakeup_work, hcd_resume_work);
1516 #endif
1517
1518     hcd->driver = driver;
1519     hcd->product_desc = (driver->product_desc) ? driver->product_desc :
1520         "USB Host Controller";
1521
1522     return hcd;
1523 }

```

一下子出现了很多数据结构，现在有必要捋一捋它们之间的关系。

(1)描述驱动的数据结构

ehci 主控器的驱动程序需要包括些什么东西？站在 pci 总线的角度来说，ehci 是一个 pci 的设备，驱动程序里面必须提供操作 ehci 设备，比如开启(start)，关闭(stop)，重启(reset)，中断的函数。另外，站在 usb 控制器的角度来说，驱动程序里面还要与底层 usb 系统交互的函数，如插入\删除 urb(rb_enqueue, urb_dequeue)。

```
pci_driver ehci_pci_driver{
```

```
.....
```

```

        .id_table = pci_ids,      ----->driver_data = (unsigned long) &ehci_pci_hc_driver,
        .probe = usb_hcd_pci_probe
    }
}
hc_driver ehci_pci_hc_driver{
    .....
    .hcd_priv_size = sizeof(struct ehci_hcd),

    .irq = ehci_irq,
    .reset = ehci_pci_setup,
    .start = ehci_run,
    .stop = ehci_stop,
    .shutdown = ehci_shutdown,

    .urb_enqueue = ehci_urb_enqueue,
    .urb_dequeue = ehci_urb_dequeue,
}

```

(2)描述设备的数据结构

```

usb_hcd{
    .....
    usb_bus .self
    .hcd_priv = ehci_hcd
}

```

usb_hcd 交大人甲有过论述:

linux 那些事儿 之 戏说 USB(28)设备的生命线 (七)

http://blog.csdn.net/fudan_abc/archive/2007/10/18/1831459.aspx

usb_hcd 注释上说 USB Host Controller Driver,但我更愿意认为它是一个描述一个 usb 主控制器设备的数据结构。usb_hcd 描述了 usb 主控制器共有的属性,usb_hcd.hcd_priv 指向了特定的主控制器数据结构,它描述各自特有的属性。对 ehci,是 ehci_hcd,对 uhci,是 uhci_hcd。usb_hcd 的成员 usb_bus self 是 HCD bus-glue layer,usb 主控制器与总线之间粘合剂。这样,1498 就好理解了。1506 行 usb_bus_init 初始化 usb_hcd.self。1511-1513 初始化 root harbor 计时器。然后就有返回到 usb_hcd_pci_probe()

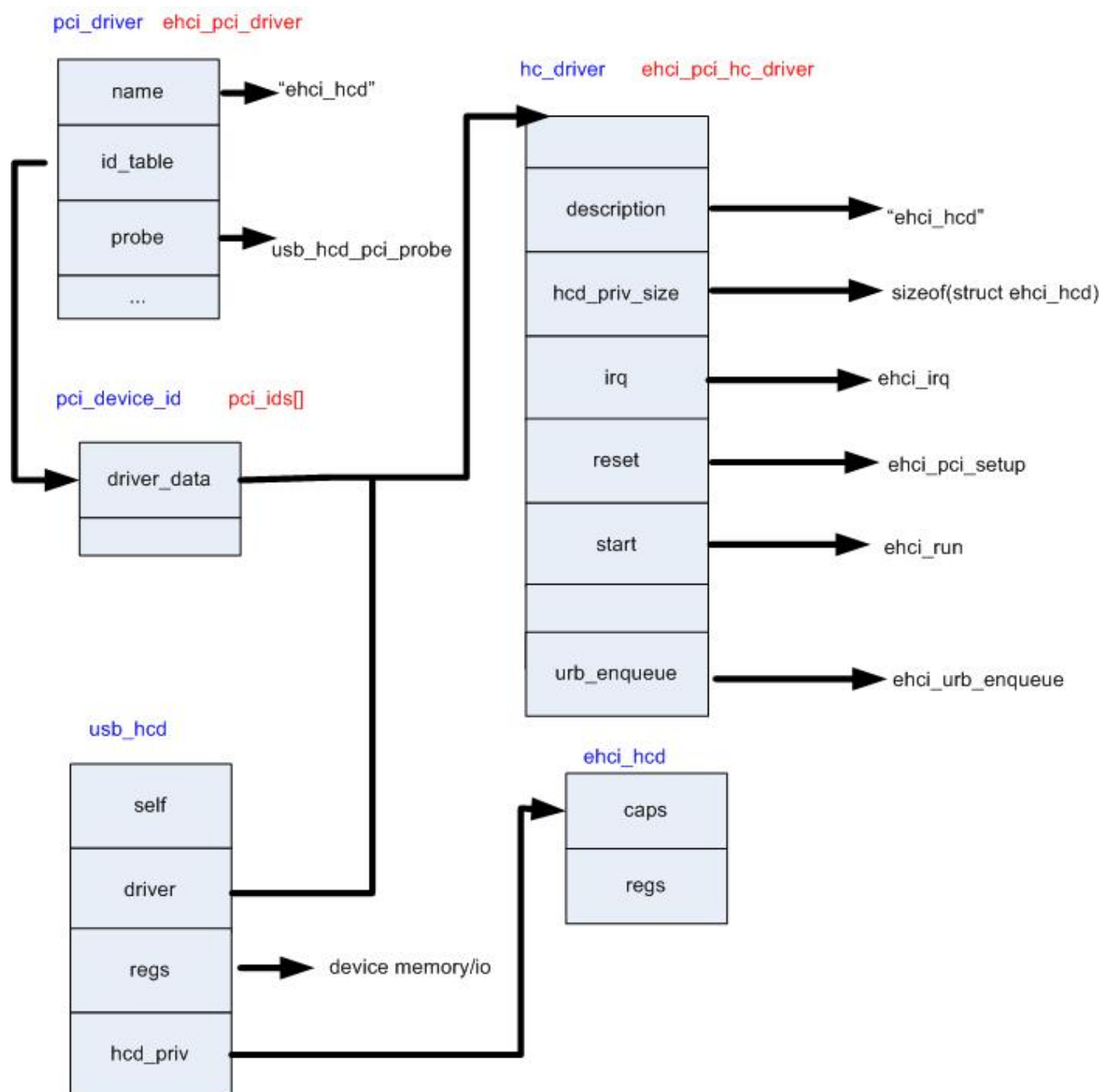
2008 年的这一场雪

2008 年的这一场雪,比以往时候来的要大一些。以前是买不到票,这次有票也走不了了。此时此刻,多少兄弟姐妹被困在路上,饥寒交迫,多少城市停电停水,物价飞涨。我真是忍不住破口大骂,好一个和谐盛世!虽说是天灾,但岂无人祸?直接原因是那该死的冻雨。但我认为罪魁乃是三峡工程,以及所谓南水北调。

祝愿那些困在铁路，公路上的人们早日回家团圆，大家过个好年。

还是先看看股市，有消息称，"导致大盘暴跌的原因是因为大面积的降雪，导致大量的股民滞留车站,机场.无法进入股市抄底，待在家中人员又因大面积的停电而无法上网操作，导致了今天的大资金无法介入，请大家不要恐慌。据不完全统计有 60% 的基金操盘手被困在机场吃方便面。"

闲话少讲，先补上一张图，



Device and Driver data structures

我们接着 `usb_hcd_pci_probe()` 往下走，89 - 108 行，是为 EHCI 申请 io 内存，92 行 `request_mem_region()` 申请，98 行做一次 mapping，在 EHCI(1) 中曾经提到过，EHCI 的接口有 3 种，第一种是 PCI 配置寄存器，第二种是 io 内存，第三种就是普通的内存。这里先把

EHCI 的 io 内存登记，再做一次 ioremap 映射成虚拟地址。

接下来 126 pci_set_master() 见 UHCI 4

http://blog.csdn.net/fudan_abc/archive/2007/10/04/1811451.aspx

128 行进入了 usb_add_hcd()

```
1548 /**
1549  * usb_add_hcd - finish generic HCD structure initialization and register
1550  * @hcd: the usb_hcd structure to initialize
1551  * @irqnum: Interrupt line to allocate
1552  * @irqflags: Interrupt type flags
1553  *
1554  * Finish the remaining parts of generic HCD initialization: allocate the
1555  * buffers of consistent memory, register the bus, request the IRQ line,
1556  * and call the driver's reset() and start() routines.
1557  */
1558 int usb_add_hcd(struct usb_hcd *hcd,
1559                 unsigned int irqnum, unsigned long irqflags)
1560 {
1561     int retval;
1562     struct usb_device *rhdev;
1563
1564     dev_info(hcd->self.controller, "%s\n", hcd->product_desc);
1565
1566     set_bit(HCD_FLAG_HW_ACCESSIBLE, &hcd->flags);
1567
1568     /* HC is in reset state, but accessible. Now do the one-time init,
1569      * bottom up so that hcds can customize the root hubs before khubd
1570      * starts talking to them. (Note, bus id is assigned early too.)
1571      */
1572     if ((retval = hcd_buffer_create(hcd)) != 0) {
1573         dev_dbg(hcd->self.controller, "pool alloc failed\n");
1574         return retval;
1575     }
1576
1577     if ((retval = usb_register_bus(&hcd->self)) < 0)
1578         goto err_register_bus;
1579
1580     if ((rhdev = usb_alloc_dev(NULL, &hcd->self, 0)) == NULL) {
1581         dev_err(hcd->self.controller, "unable to allocate root hub\n");
1582         retval = -ENOMEM;
1583         goto err_allocate_root_hub;
1584     }
1585     rhdev->speed = (hcd->driver->flags & HCD_USB2) ? USB_SPEED_HIGH :
1586                 USB_SPEED_FULL;
1587     hcd->self.root_hub = rhdev;
```

```

1588
1589     /* wakeup flag init defaults to "everything works" for root hubs,
1590     * but drivers can override it in reset() if needed, along with
1591     * recording the overall controller's system wakeup capability.
1592     */
1593     device_init_wakeup(&rhdev->dev, 1);
1594
1595     /* "reset" is misnamed; its role is now one-time init. the controller
1596     * should already have been reset (and boot firmware kicked off etc).
1597     */
1598     if (hcd->driver->reset && (retval = hcd->driver->reset(hcd)) < 0) {
1599         dev_err(hcd->self.controller, "can't setup\n");
1600         goto err_hcd_driver_setup;
1601     }
1602
1603     /* NOTE: root hub and controller capabilities may not be the same */
1604     if (device_can_wakeup(hcd->self.controller)
1605         && device_can_wakeup(&hcd->self.root_hub->dev))
1606         dev_dbg(hcd->self.controller, "supports USB remote wakeup\n");
1607
1608     /* enable irqs just before we start the controller */
1609     if (hcd->driver->irq) {
1610         snprintf(hcd->irq_descr, sizeof(hcd->irq_descr), "%s:usb%d",
1611                 hcd->driver->description, hcd->self.busnum);
1612         if ((retval = request_irq(irqnum, &usb_hcd_irq, irqflags,
1613                 hcd->irq_descr, hcd)) != 0) {
1614             dev_err(hcd->self.controller,
1615                     "request interrupt %d failed\n", irqnum);
1616             goto err_request_irq;
1617         }
1618         hcd->irq = irqnum;
1619         dev_info(hcd->self.controller, "irq %d, %s 0x%08llx\n", irqnum,
1620                 (hcd->driver->flags & HCD_MEMORY) ?
1621                     "io mem" : "io base",
1622                 (unsigned long long)hcd->rsrc_start);
1623     } else {
1624         hcd->irq = -1;
1625         if (hcd->rsrc_start)
1626             dev_info(hcd->self.controller, "%s
0x%08llx\n", 1627                 (hcd->driver->flags & HCD_MEMORY) ?
1628                     "io mem" : "io base",
1629                     (unsigned long long)hcd->rsrc_start);
1630     }

```

```
1631
1632     if ((retval = hcd->driver->start(hcd)) < 0) {
1633         dev_err(hcd->self.controller, "startup error %d\n", retval);
1634         goto err_hcd_driver_start;
1635     }
1636
1637     /* starting here, usbcore will pay attention to this root hub */
1638     rhdev->bus_mA = min(500u, hcd->power_budget);
1639     if ((retval = register_root_hub(hcd)) != 0)
1640         goto err_register_root_hub;
1641
1642     if (hcd->uses_new_polling && hcd->poll_rh)
1643         usb_hcd_poll_rh_status(hcd);
1644     return retval;
1645
1646 err_register_root_hub:
1647     hcd->driver->stop(hcd);
1648 err_hcd_driver_start:
1649     if (hcd->irq >= 0)
1650         free_irq(irqnum, hcd);
1651 err_request_irq:
1652 err_hcd_driver_setup:
1653     hcd->self.root_hub = NULL;
1654     usb_put_dev(rhdev);
1655 err_allocate_root_hub:
1656     usb_deregister_bus(&hcd->self);
1657 err_register_bus:
1658     hcd_buffer_destroy(hcd);
1659     return retval;
1660 }
```

1572 行 `hcd_buffer_create()` 见 UHCI 5

http://blog.csdn.net/fudan_abc/archive/2007/10/10/1818462.aspx。

简单总结一下，创建，摧毁 `buffur`: `hcd_buffer_create()` / `hcd_buffer_destroy()`，创建之后，从池子里索取或者把索取的释放回去，`usb_buffer_alloc()/usb_buffer_free()`。

1577 行 `usb_register_bus` 见 UHCI 6

http://blog.csdn.net/fudan_abc/archive/2007/10/13/1823287.aspx。

简单来说，把自己连入到全局变量 `usb_bus_list` 中，并调用 `class_device_create` 往 `sysfs` 新加入一个 `class`。

1580 行 `usb_alloc_dev()`，UHCI 7

http://blog.csdn.net/fudan_abc/archive/2007/10/16/1827449.aspx。

1598 行之前，都和 UHCI 没有什么分别，`hcd->driver->reset` 指向的是，`ehci_pci_hc_driver.ehci_pci_setup()`