(1 条消息) 浅谈 Linux PCI 设备驱动（下）_zjy900507 的博客 - CSDN 博客_linux pcie 设备驱动

我们在 浅谈 Linux PCI 设备驱动（上） 中 (以下简称 浅谈 (一)) 介绍了 PCI 的配置寄存器组，而 Linux PCI 初始化就是使用了这些寄存器来进行的。后面我们会举个例子来说明 Linux PCI 设备驱动的主要工作内容 (不是全部内容)，这里只做文字性的介绍而不会涉及具体代码的分析，因为要分析代码的话，基本就是对 Linux 内核源代码情景分析(下册) 第八章的解读，读者若想分析代码，可以参考该书的内容 , 这里就不去深入分析这些代码了。

Linux PCI 设备驱动代码必须扫描系统中所有的 PCI 总线，寻找系统中所有的 PCI 设备 (包括 PCI-PCI 桥设备)。系统中的每条 PCI 总线都有个编号 number，根 PCI 总线的编号为 0。系统当前存在的所有根总线 (因为可能存在不止一个 Host/PCI 桥，那么就可能存在多条根总线) 都通过其 pci_bus 结构体中的 node 成员链接成一个全局的根总线链表，其表头由 struct list_head 类型的全局变量 pci_root_buses 描述，我们在 / linux-2.4.18/linux/drivers/pci/pci.c 的 38 行可以看到如下定义：

```
112:  /* Header type 1 (PCI-to-PCI bridges) */

113:  #define PCI_PRIMARY_BUS           0x18       /* Primary

114:  #define PCI_SECONDARY_BUS     0x19       /* Secondary bu

115:  #define PCI_SUBORDINATE_BUS  0x1a       /* Highest bus n
```

- 1

而根总线下面的所有下级总线则都通过其 pci_bus 结构体中的 node 成员链接到其父总线的 children 链表中。这样，通过这两种 PCI 总线链表，Linux 内核就将所有的 pci_bus 结构体以一种倒置树的方式组织起来。

另外，每个 PCI 设备都由一个 pci_dev 结构体表示，每个 pci_dev 结构体都同时连入两个队列，一方面通过其成员 global_list 挂入一个总的 pci_dev 结构队列 (队列头是 pci_devices)；同时又通过成员 bus_list 挂入其所在总线的 pci_dev 结构队列 devices(队列头是 pci_bus.devices, 即该 pci 设备所在的 pci 总线的 devices 队列)，并且使指针 bus(指 pci_dev 结构体里的 bus 成员) 指向代表着其所在总线的 pci_bus 结构。

如果具体的设备是 PCI-PCI 桥，则还要使其指针 subordinate 指向代表着另一条 PCI 总线的 pci_bus 结构。同样我们在 / linux-2.4.18/linux/drivers/pci/pci.c 的 39 行可以看到如下定义：

```
LIST_HEAD(pci_devices);
```

- 1

对于 PCI 设备链表，我们可以通过图 1 来理解。

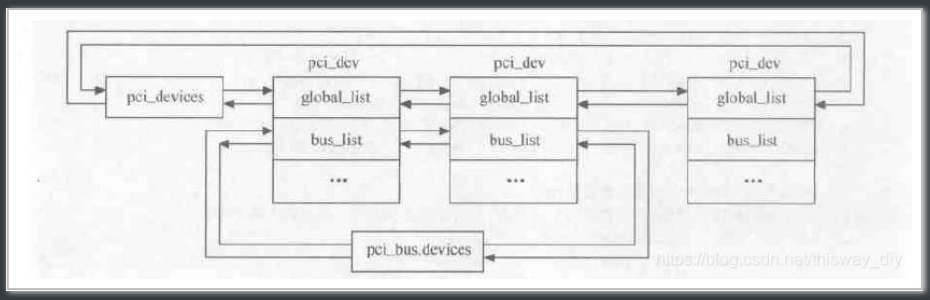

图 1 Linux PCI 设备链表

注：该图摘自 Linux 设备驱动开发详解 第 21 章 PCI 设备驱动。

而对于我们在浅谈 (一) 中贴出的图 1 的 PCI 系统结构示意图，Linux 内核中对应的数据结构如图 2 所示。

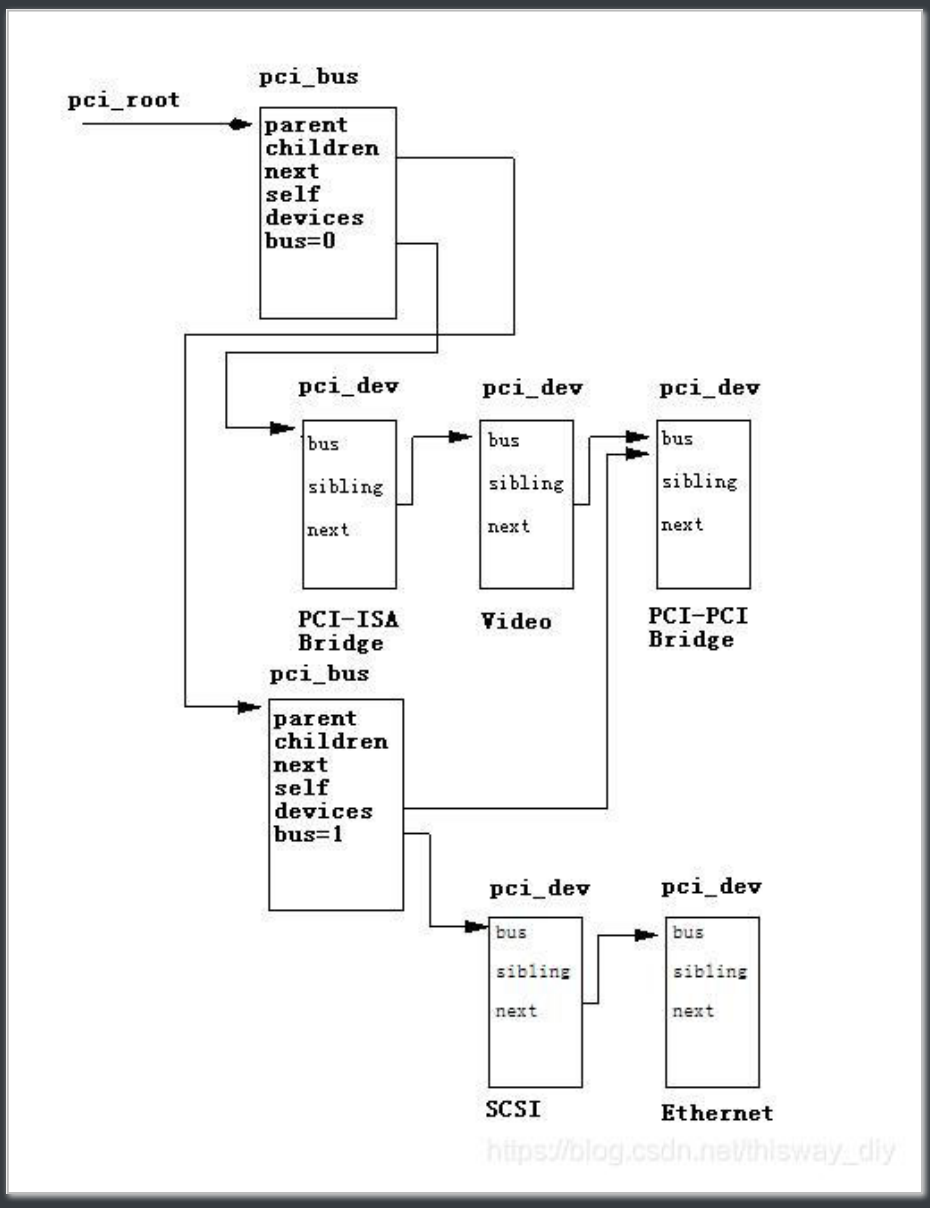

图 2 Linux 内核 PCI 数据结构

Linux PCI 初始化代码从 PCI 总线 0 开始扫描，它通过读取 "Vendor ID" 和 "Device ID" 来试图发现每一个插槽上的设备。如果发现了一个 PCI-PCI 桥，则创建一个 pci_bus 数据结构并且连入到由 pci_root_buses 指向的 pci_bus 和 pci_dev 数据结构组成的树中。PCI 初始化代码通过设备类代码 0x060400 来判断一个 PCI 设备是否是 PCI-PCI 桥。

然后，Linux 核心开始构造这个桥设备另一端的 PCI 总线和其上的设备。如果还发现了桥设备，就以同样的步骤来进行构建。这个处理过程称之为深度优先算法。PCI-PCI 桥横跨在两条总线之间，寄存器 PCI_PRIMARY_BUS 和 PCI_SECONDARY_BUS 的内容就说明了其上下两端的总线号，其中 PCI_SECONDARY_BUS 就是该 PCI-PCI 桥所连接和控制的总线，而 PCI_SUBORDINATE_BUS 则说明自此以下、在以此为根的子树中最大的总线号是什么。

我们可以在 / linux-2.4.18/linux/include/linux/pci.h 看到如下定义：

```
112:  /* Header type 1 (PCI-to-PCI bridges) */

113:  #define PCI_PRIMARY_BUS          0x18       /* Primary

114:  #define PCI_SECONDARY_BUS     0x19       /* Secondary bu

115:  #define PCI_SUBORDINATE_BUS  0x1a       /* Highest bus n
```

- 1
- 2
- 3
- 4

- 5
- 6
- 7
- 8

由于在枚举阶段做的是深度优先扫描，所以子树中的总线号总是连续递增的。当 CPU 往 I/O 寄存器 0xCF8 中写入一个综合地址以后，从 0 号总线开始，每个 PCI-PCI 桥会把综合地址中的总线号与自身的总线号相比，如果相符就用逻辑设备号在本总线上寻访目标设备；

否则就进一步把这个总线号与 PCI_SUBORDINATE_BUS 中的内容相比，如果目标总线号落在当前子树范围中，就把综合地址传递给其下的各个次层 PCI-PCI 桥，要不然就不予理睬。这样，最终就会找到目标设备。当然，这个过程只是在 PCI 设备的配置阶段需要这样做，一旦配置完成，CPU 就直接通过有关的总线地址访问目标设备了。

PCI-PCI 桥要想正确传递对 PCI I/O，PCI Memory 或 PCI Configuration 地址空间的读和写请求，必须知道下列信息：

(1)Primary Bus Number(主总线号)

该 PCI-PCI 桥所处的 PCI 总线称为主总线。

(2)Secondary Bus Number(子总线号)

该 PCI-PCI 桥所连接的 PCI 总线称为子总线 / 次总线号。

(3)Subordinate Bus Number

PCI 总线的下属 PCI 总线的总线编号最大值。有点绕，看后面的分析就明白了。

PCI I/O 和 PCI Memory 窗口

PCI 桥的配置寄存器与一般的 PCI 设备不同。一般 PCI 设备可以有 6 个地址区间，外加一个 ROM 区间，代表着设备上实际存在的存储器或寄存器区间。而 PCI 桥，则本身并不一定有存储器或寄存器区间，但是却有三个用于地址过滤的区间。每个地址过滤区间决定了一个地址窗口，从 CPU 一侧发出的地址，如果落在 PCI 桥的某个窗口内，就可以穿过 PCI 桥而到达其所连接的总线上。

此外，PCI 桥的命令寄存器中还有”memory access enable”和”I/O access enable ”的两个控制位，当这两个控制位为 0 时，这些窗口就全都关上了。在未完成对 PCI 总线的初始化之前，还没有为 PCI 设备上的各个区间分配合适的总线地址时，正是因为这两个控制位为 0，才不会对 CPU 一侧造成干扰。例如， 对于浅谈 (一) 的 PCI 系统示意图 ，仅当读和写请求中的 PCI I/O 或 PCI memory 地址属于 SCSI 或 Ethernet 设备时，PCI-PCI 桥才将这些总线上的请求从 PCI 总线 0 传递到 PCI 总线 1。这种过滤机制可以避免地址在系统中没必要的繁衍。

为了做到这点，每个 PCI-PCI 桥必须正确地被设置好它所负责的 PCI I/O 或 PCI memory 的起始地址和大小。当一个读或写请求地址落在其负责的范围之内，这个请求将被映射到次级的 PCI 总线上。系统中的 PCI-PCI 桥一旦设置完毕，如果 Linux 中的设备驱动程序存取的 PCI I/O 和 PCI memory 地址落在在这些窗口之内，那么这些 PCI-PCI 桥就

是透明的。这是个很重要的特性，使得 Linux PCI 设备驱动程序开发者的工作容易些。

问题是配置一个 PCI-PCI 桥的时候，并不知道这个 PCI-PCI 桥的 subordinate bus number。那么就不知道该 PCI 桥下面是否还有其他的 PCI-PCI 桥。即使你知道，也不清楚如何对它们赋值。解决方法是利用上述的深度扫描算法来扫描每个总线。每当发现 PCI-PCI 桥就对它进行赋值。

当发现一个 PCI-PCI 桥时，可以确定它的 secondary bus number。然后我们暂时先将其 subordinate bus number 赋值为 0xFF。紧接着，开始扫描该 PCI-PCI 桥的 downstream 桥。这个过程看起来有点复杂, 下面的例子将给出清晰的解释：

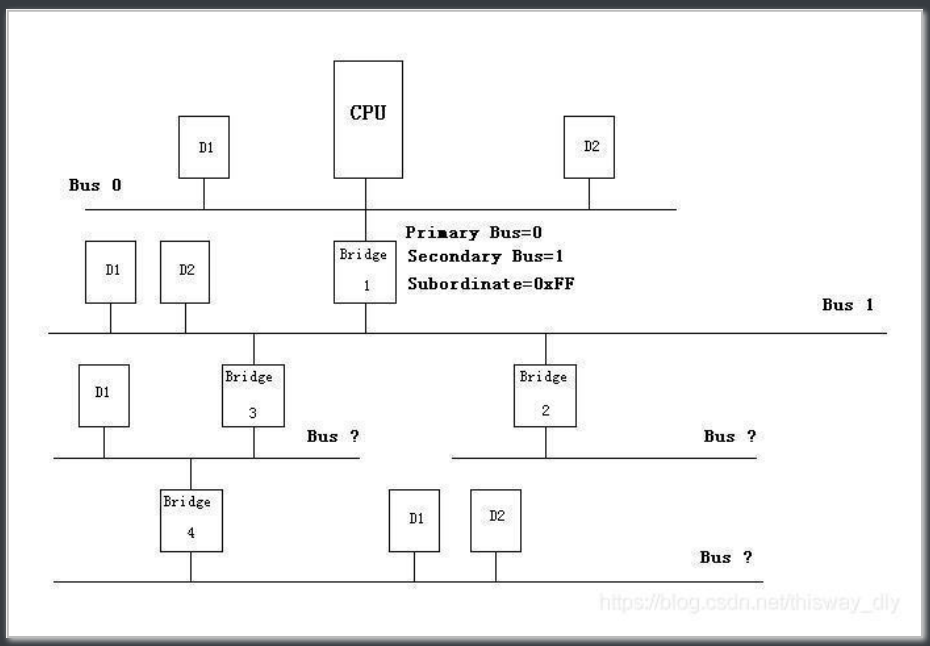

图 3，配置 PCI 系统 第一步

PCI-PCI 桥编号 – 第一步

以图 3 的拓扑结构为例，扫描时首先发现的桥是 Bridge1。Bridge 1 的 downstream PCI 总线号码被赋值 1。自然该桥的 secondary bus number 也是 1。其 subordinate bus number 暂时赋值为 0xFF。上述赋值的含义是所有类型 1 的含有 PCI 总线 1 或更高 (<255) 的号码的 PCI 配置地址将被 Bridge 1 传递到 PCI 总线 1 上。

如果 PCI 总线号是 1，Bridge 1 还负责将配置地址的类型转换成类型 0(对于这里说的类型 0 和类型 1，请参考浅谈 (一))。否则，就不做转换。上述动作就是开始扫描总线 1 时 Linux PCI 初始化代码所完成的对总线 0 的配置工作。

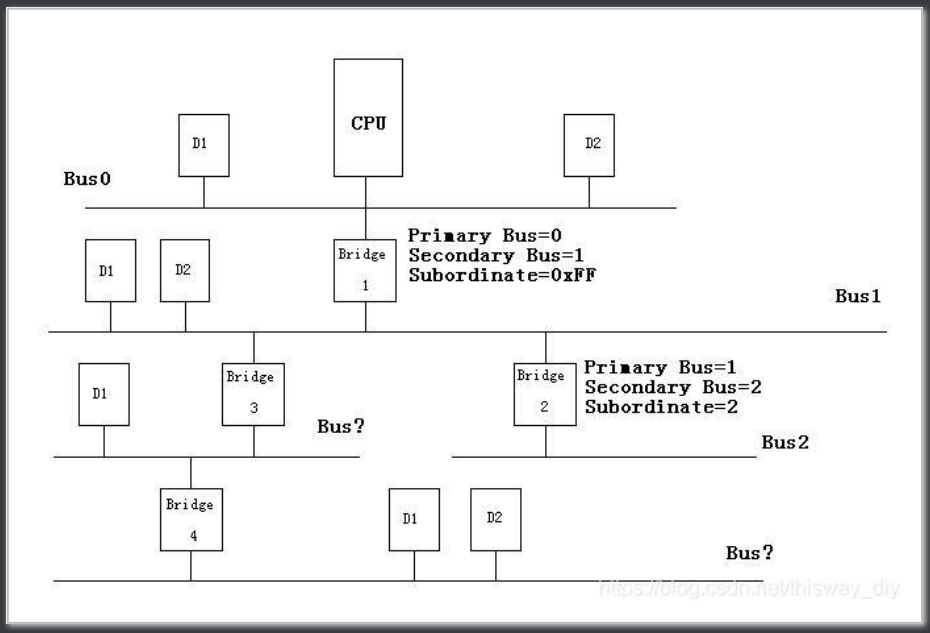

图 4，配置 PCI 系统 第二步

PCI-PCI 桥编号 – 第二步

由于 Linux PCI 设备驱动使用深度优先算法进行扫描，所以初始化代码开始扫描总线 1。从而 Bridge 2 被发现。因为在 Bridge 2 下面发现不再有 PCI-PCI 桥，所以 Bridge 2 的 subordinate bus number 是 2，等于它的 secondary bus

number。图 4 显示了在这个时刻总线和 PCI-PCI 桥的赋值情况。

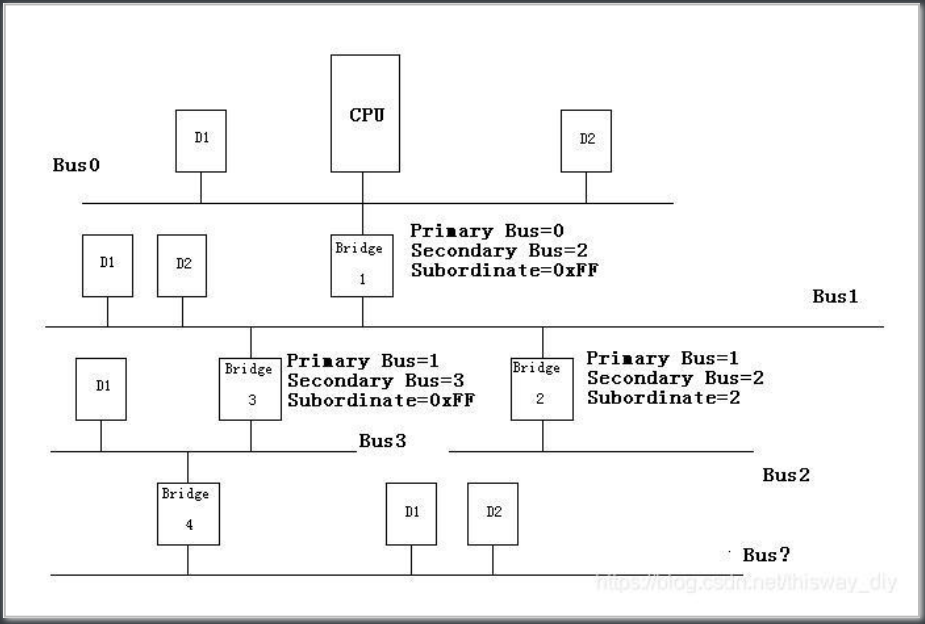

图 5，配置 PCI 系统 第三步

PCI-PCI 桥编号 – 第三步

Linux PCI 设备驱动代码从总线 2 的扫描中回来接着进行扫描总线 1，发现 Bridge 3。它的 primary bus number 被赋值为 1，secondary bus number 为 3。因为总线 3 上还发现了 PCI-PCI 桥，所以 Bridge 3 的 subordinate bus number 暂时赋值 0xFF。图 5 显示了这个时刻系统配置的

状态。到目前为止，含有总线号 1，2，3 的类型 1 的 PCI 配置都可以正确地传送到相应的总线上。

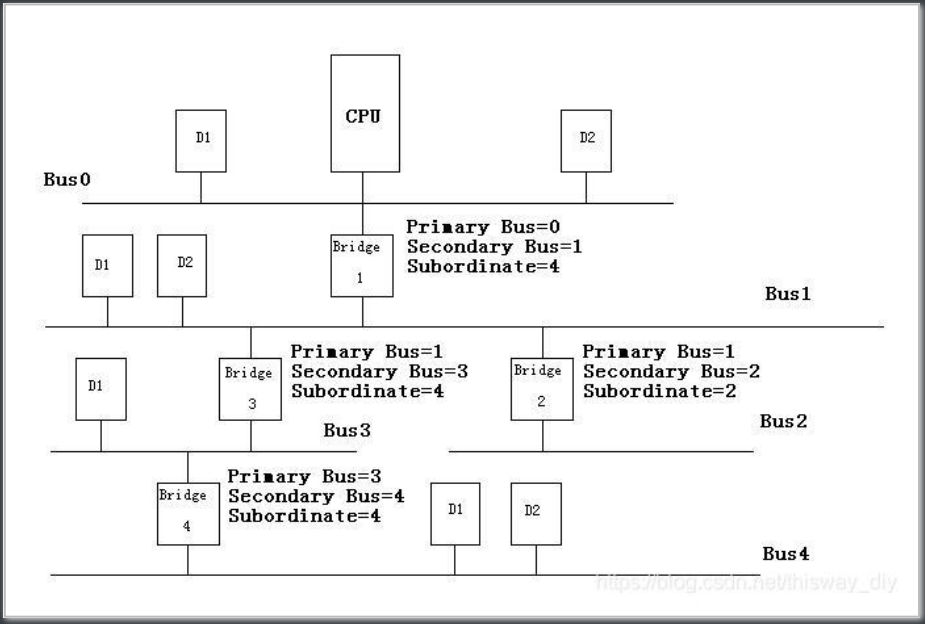

图 6 , 配置 PCI 系统 第四步

PCI-PCI 桥编号 – 第四步

现在 Linux 开始扫描 PCI 总线 3，Bridge 3 的 downstream。PCI 总线 3 上有另外一个 PCI-PCI 桥，Bridge 4。因此 Bridge 4 的 primary bus number 的值为 3，secondary bus number 为 4。由于 Bridge 4 下面没有

别的桥设备，所以 Bridge 4 的 subordinate bus number 为 4。

然后回到 PCI-PCI Bridge 3。这时就将 Bridge 3 的 subordinate bus number 从 0xFF 改为 4，表示总线 4 是从 Bridge 3 往下走的最远的 PCI-PCI 桥。最后，Linux PCI 设备驱动代码将 4 以同样的道理赋值给 Bridge 1 的 subordinate bus number。图 6 反映了系统最后的状态。

注：浅谈 Linux PCI 设备驱动（下）暂时的整体结构就是这样了。在此强烈推荐想学 Linux PCI 设备驱动的朋友结合《Linux 内核源代码情景分析下册》第八章和《Linux 设备驱动开发详解》第 21 章 来学习。

全文完