PCI 设备 Windows 通用驱动程序设计

北京理工大学电子工程系(100081) 李海

电子邮件：haili@public.bta.net.cn

本文首次发表于《电子技术应用》2000 年第 1 期

http://articles.126.com

PCI 设备 Windows 通用驱动程序设计

北京理工大学电子工程系(100081) 李海

电子邮件：haili@public.bta.net.cn

摘要：Windows 操作系统为了保证系统的安全性、稳定性和可移植性，对应用程序访问硬件资源加以限制，这就要求设计设备驱动程序以实现 PC 机的软件对 PCI 设备的访问。本文结合“通用高速 PCI 总线目标模块”[1]的驱动程序设计，全面地讨论了 Windows 设备（特别是 PCI 设备）驱动程序编写时所面临的主要问题及解决方案，并提出了封装设备驱动的方法。

关键词：PCI 设备驱动程序 端口 内存 中断 封装

在设计和使用 PCI 设备时，经常要在 PC 机的软件中访问和控制硬件设备，但 Windows 操作系统（包括 Windows 95/98、Windows NT、Windows 2000）为了保证系统的安全性、稳定性和可移植性，对应用程序访问硬件资源加以限制，这就要求设计设备驱动程序以实现 PC 机的软件对 PCI 设备的访问。

Windows 下的驱动程序不仅仅包括物理设备的驱动程序，也包括为文件系统等非物理设备编写的虚拟设备驱动程序。为了简化问题，下面只讨论硬件物理设备的驱动程序。本文将以“通用高速 PCI 总线目标模块”[1]的驱动设计为例，探讨 PCI 设备的驱动程序设计方案。我们开发了一套通用的 PCI 设备驱动程序，它可以完成一般 PCI 设备驱动所需的功能，可以作为其它 PCI 设备驱动开发的框架。

一、驱动程序的模式和开发工具的选择

设备驱动程序是指管理某个外围设备的一段代码。驱动程序不会独立地存在，而是操作系统的一部分。通过设备驱动程序，多个进程可以同时使用这些资源，从而可以实现多进程并行运行。在下文中，将调用设备驱动程序的 PC 机程序称为**用户程序**。

Windows 95 和 Windows NT 采用的驱动程序体系不同，所以大多数情况下驱动程序也不能通用。如果设备需要在 Windows 9x/NT 下使用，一般至少要设计 Windows 9x 和 Windows NT 两个驱动程序版本。Windows 98 可以兼容 Windows 95 的驱动程序，同时它又推出一个新的 Win32 Drivers Mode(WDM)驱动类型。Windows 98 中有些设备(如 USB 设备)的驱动程序必须为 WDM 模式的。这个新的类型实际是在 Windows NT 的驱动模型的基础上增加了即插即用等内容。WDM 驱动也可以用在 Windows 2000（先前叫 Windows NT 5.0）中。从长远的角度看，今后开发人员只要开发 WDM 驱动就可以了，但从目前的市场情况来看，Windows 95 是无法放弃的，所以 WDM 在近一两年还无法替代其它类型的设备驱动。

Intel 80386 以上的微处理器有 4 个优先级别：0 级、1 级、2 级和 3 级，一般操作系统运行于优先级 0 级上，而用户程序运行在 3 级上，在对硬件操作上有一些限制（具体的限制在不同的操作系统中是不同的）。Windows 95 支持的驱动类型很多，但针对一般硬件设备而言，主要是 VxD 和打印机驱动两类。VxD 指的是 Virtual Device Drivers。VxD 运行在 Intel 系统的 0 级上的，可以执行特权级指令，对任何 I/O 设备有全部访问权，所以大多数硬件驱动程序都是 VxD。

VxD 驱动通常以.vxd 为扩展名，放在 Windows\System 目录下，可以在 Windows 95 启动时装入，也可以在程序运行时根据需要动态地载入。动态加载有助于节约系统内存和资源。但打印机驱动程序不是 VxD，它是运行在 3 级上的。同 Windows 95 类似，Windows NT 的驱动也有可以运行在 0 级的内核模式(Kernel Mode)和运行在 3 级的用户模式(User Mode)之分。由于 Windows NT 禁止用户模式的程序访问 I/O 端口（Windows 95/98 则允许用户程序直接访问 I/O 端口），直接控制物理设备的驱动程序都是内核模式的。我们设计的 PCI 通用驱动程序要求对各种硬件资源的访问，所以应该选择工作在 0 级的驱动程序模式。

开发设备驱动采用的主要开发工具是微软为设备开发者提供的软件包 Device Driver Kit(DDK)。这个软件包中包括有关设备开发的文档、编译需要的头文件和库文件、调试工具和程序范例。在 DDK 中还定义了一些设备驱动可以调用的系统底层服务，象 DMA 服务、中断服务、内存管理服务、可安装文件系统服务等等。这些都是编写设备驱动所必须的。但 Windows 95 的 DDK 由于主要使用汇编语言描述，开发起来比较困难。因此，我们在 Windows 95 操作系统中同时采用了 Numega 公司的产品 VtoolsD 。VtoolsD 是基于 C/C++的，支持 Borland C++和 Visual C++，使用和维护都较 Windows 95 DDK 容易。

二、PCI 驱动程序的特点

在设计驱动程序之前，首先要对欲控制的硬件设备进行细致地分析，更需要详细了解硬件设备的特性。硬件设备的特性会对驱动程序设计产生重大的影响。需要了解的最主要的硬件特性包括：

1、设备的总线结构

设备采用什么总线结构非常关键，因为不同的总线类型（如 ISA 和 PCI）在许多硬件工作机制上是不同的，所以驱动程序设计也不同。

2、寄存器

要了解设备的控制寄存器、数据寄存器和状态寄存器，以及这些寄存器工作的特性。

3、设备错误和状态

要了解如何判断设备的状态和错误信号，这些要通过驱动程序返回给用户。

4、中断行为

要了解设备产生中断的条件和使用中断的数量。

5、数据传输机制

最常见的数据传输机制是通过 I/O 端口(port)，也就是通过 CPU 的 IN/OUT 指令进行数据读写。PC 的另一种重要的传输机制是 DMA，但 PCI 规范不包括从属 DMA 的说明。

6、设备内存

许多设备自身带有内存，PCI 设备大多是采用映射的方式映射到 PC 系统的物理内存。有的设备还要通过驱动程序设置设备的接口寄存器。

有关驱动程序的加载和响应用户请求的内容，在 DDK 文档中有规定，所以设计设备驱动程序主要面临的问题是如何进行硬件操作，这是根据设备的不同而不同的。而硬件驱动程序的功能虽然千差万别，但基本功能就是完成设备的初始化、对端口的读写操作、中断的设置、响应和调用以及对内存的直接读写。如前面说的，Windows 9x 和 Windows NT 的操作系统模型不同，但驱动程序所要完成的工作却是相同的，所以下面以 Windows 9x 为主进行介绍，仅在需

要的地方指出两个操作系统的不同。

下面从这几方面讨论解决这些问题的途径：

1、设备初始化

PCI 设备驱动程序要实现识别 PCI 器件、寻址 PCI 器件的资源和对 PCI 器件中断的服务。PCI 系统 BIOS 功能提供了 BIOS 的访问与控制的具体特性，所有软件（设备驱动程序、扩展 ROM 码）将通过标准中断号 1AH 调用 BIOS 功能访问特殊部件。PCI BIOS 规范有完整的有关 PCI BIOS 功能的描述[3]。

PCI 设备驱动程序的初始化过程中，利用指定器件识别号（device_id）、厂商识别号（vendor_id）、检索号（index）搜索 PCI 器件，通过调用 PCI BIOS 确认其存在，并确定其物理位置：总线号、器件号和功能号，这是该器件/功能在系统中的唯一寻址标志。利用总线号、器件号和功能号可以寻址该器件/功能的 PCI 配置空间（configuration space）。

接下来，设备驱动就需要从配置空间获得硬件的参数。PCI 设备的许多参数，包括所用的中断号，端口地址的范围（I/O）方式、存储器的地址（存储器映射方式）等，都可以从 PCI 配置空间的各基址所对应的寻址空间中得到。读写配置空间可以调用 BIOS 中断 1AH，也可以先向配置空间地址寄存器（0CF8H）写入总线和设备号（在前面搜索 PCI 器件时得到的）和寄存器号，再对配置空间数据寄存器（0CFCH）进行读写。对设备驱动来说，最重要的是获得基址寄存器（BADR），不能认为 PCI 器件资源总是设计设备时设置的初值，系统可能会根据硬件情况为 PCI 设备分配新的资源。我们所设计的 PCI 设备使用的基址 1-3 都是按 I/O 空间映射的，而基址 4 是按内存方式映射的。确定一个端口是按什么方式映射的，可以读对应端口的配置寄存器(Configuration Register)。读出后，判断其 0 位，如果 0 位为数值 0，表示其是按内存方式设置的，否则为 I/O 方式设置的。内存方式和 I/O 方式的配置寄存器的含义参见文献[3]的 3.11。如果要获得基址的大小，可以向基址寄存器写入 FFFFH，然后读基址寄存器，如果是内存方式，从第 4 位开始的 0 的数目表示基址的大小，如果是 I/O 方式，则从第 2 位开始的 0 的数目表示基址的大小。

在 Windows NT 下，查找 PCI 设备的工作是由 HalGetBusData 完成的，也可以使用前述的办法读取配置寄存器，但 DDK 推荐使用 HalGetBusDataOffset 函数。

2、端口操作

在 PC 机上，I/O 端口寻址空间和内存寻址空间是不同的，所以处理方法也不同。I/O 空间是一个 64K 字节的寻址空间，它不象内存有实模式和保护模式之分，在各种模式下寻址方式相同。在 Windows 9x 下，用户程序可以直接使用 I/O 指令，而不一定非通过专门的驱动程序来完成，所以如果软件对硬件的操作完全是通过 I/O 端口操作来完成的，甚至可以不用专门设计驱动程序，直接由应用程序来完成对硬件的控制。由于 PCI 总线是 32 位的总线标准，在进行 I/O 操作时通常要进行双字（DWORD）操作，而目前大多数 C/C++编译软件都没有提供双字的函数，所以需要构造双字操作读写函数 inpd/outpd。

在 Windows NT 下，系统不允许处于优先级 3 级的用户程序和用户模式驱动程序直接使用 I/O 指令，如果使用了 I/O 指令将会导致特权指令意外（privileged instruction exception）。所以任何对 I/O 的操作对需要借助内核模式驱动来完成。具体的做法有两种：一是在驱动程序中使用 IoReportResourceUsage 报告资源占用，然后使用 READ_PORT_XXX、WRITE_PORT_XXX 函数读写，最后使用 IoReportResourceUsage 取消资源占用；另一种是驱动程序修改 NT 的 I/O Permissions Map (IOPM)，以使系统允许用户程序对指定的 I/O 端口进行操作，这时用户程序就是采用通常的 I/O 指令进行操作。后者的优点的速度快、用户程序设计简单，但牺牲了移植性，

程序不能移植到非 Intel 的系统中，而且如果多个程序同时读写同一端口容易导致冲突。

3、内存的读写

Windows 工作在 32 位保护模式下，保护模式与实模式的的根本区别在于 CPU 寻址方式上的不同，这也是 Windows 驱动程序设计中需要着重解决的问题。Windows 采用了分段、分页机制（图 1），这样使应用程序产生一种错觉，好象程序中可以使用非常大的物理存储空间。这样做最大的好处就是一个程序可以很容易地在物理内存容量不一样的、配置范围差别很大的计算机上运行，编程人员使用虚拟存储器可以写出比任何实际配置的物理存储器都大得多的程序。每个虚拟地址由 16 位的段选择子和 32 位段偏移量组成。通过分段机制，系统由虚拟地址产生线性地址。再通过分页机制，由线性地址产生物理地址。线性地址被分割成页目录 (Page Directory)、页表(Page Table)和页偏移(Offset)三个部分。当建立一个新的 Win32 进程时，操作系统会为它分配一块内存，并建立它自己的页目录、页表，页目录的地址也同时放入进程的现场信息中。当计算一个地址时，系统首先从 CPU 控制器 CR3 中读出页目录所在的地址，然后根据页目录得到页表所在的地址，再根据页表得到实际代码/数据页的页帧，最后再根据页偏移访问特定的单元。硬件设备读写的是物理内存，但应用程序读写的是虚拟地址，所以存在着将物理内存地址映射到用户程序线性地址的问题。

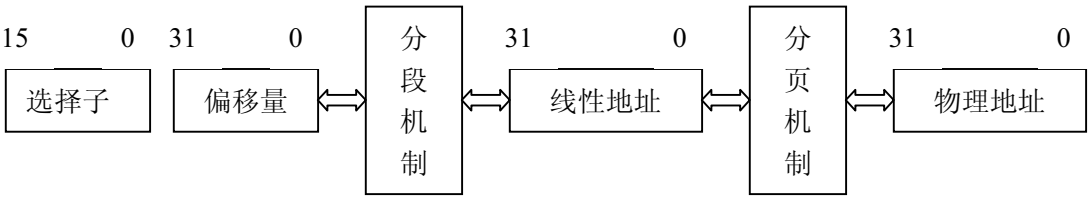

图 1 虚拟地址和物理地址的对应关系

从物理地址到线性地址的转换工作也是由驱动程序来完成的。在 Windows 95 下，使用 DDK 的 VMMCall_MapPhysToLinear 进行地址映射。驱动程序的内存映射部分主要是调用 VxD 的系统服务 MapPhysToLinear。在 VtoolsD 中这个函数的定义如下：

PVOID MapPhysToLinear(CONST VOID * PhysAddr, DWORD nBytes, DWORD Flags) ;

其中第一个参数 PhysAddr 就是要映射的内存的物理地址的起始位置，而 nBytes 是内存区域的长度，Flags 必须设置为 0。这个函数返回的就是这段物理地址映射的线性内存地址。如果指定的内存不能存取，函数将返回 FFFFFFFFH。

比如要映射物理内存 ED000000H 开始的 4096 个字节，可以这样做：

```
PCHAR *PointerToPage = (PCHAR)MapPhysToLinear
        ((PVOID) 0xED000000, 4096, 0);
```

而将 PointerToPage 传递给调用驱动的用户程序，在用户程序中使用

```
DWORD *pFIFOBodyBase = (DWORD*) PointerToPage;
```

而这个 pFIFOBodyBase 指针就可以象普通的指针一样进行读写操作，而通过对这个指针的操作就可以实现对物理内存 ED000000H 进行读写。

在 Windows NT 下，首先调用 IoReportResourceUsage 请求使用设备的内存。然后调用 HalTranslateBusAddress 转换与总线相关的内存为系统的物理内存地址。再使用 MmMapIoSpace 把设备的内存映射到虚拟空间。在设备驱动卸出时，调用 MmUnmapIoSpace 断开设备的内存和虚拟空间的连接。

4、中断的设置、响应与调用

对中断的设置、响应与调用应该在驱动程序中完成。

对中断的调用（象前面调用 BIOS 的 1A 中断读取配置寄存空间）可以由 DDK 的 Exec_Int 完成。

PCI 设备驱动程序应当从 PCI 配置寄存器的中断寄存器（INTLN）和中断引脚寄存器（INTPIN）中获得有关中断的信息。DDK 还提供了响应中断事件的服务。如在 Windows 95 中，VPICD 服务用来管理所有硬件中断事件。PC 机的硬件中断需要确定硬件中断的 IRQ，对一个特定的 IRQ 中断源，VPICD 或者提供缺省的中断处理函数，或者允许其它 VxD 重载中断处理函数。在 VtoolsD 中，要处理硬件中断应该从 VHardwareInt 继承一个类。在这个类中，VtoolsD 提供了编写中断响应程序所需的功能。

在 Windows NT 中，同 VPICD 对应的中断服务为中断请求层（IRQL）。设备驱动首先使用 HalGetInterruptVector 将与总线有关的中断向量转换为系统的中断向量，然后利用 IoConnectInterrupt 指定中断服务。

三、设备驱动的调用

编写设备驱动并不是最终的目的，总是需要由用户程序来调用驱动并实现一定的功能。一般调用设备驱动是使用 CreateFile 函数打开设备文件，得到一个文件句柄。具体到我们的设备驱动程序，使用如下的语句就可以打开文件。

```
hVxD = CreateFile("\\\\.\\PCIBIOS.VXD", 0, 0, 0,
        CREATE_NEW, FILE_FLAG_DELETE_ON_CLOSE, 0);
```

打开设备文件后，调用 DeviceIoControl 函数就可以同设备驱动程序交换数据了。

完成硬件操作之后，可以调用 CloseHandle(hVxD);关闭设备驱动。

这种调用方式也是 Windows NT 调用设备驱动的标准方法。对于 VxD 来说还有其它的调用方式，如 DPMI 方式，但采用 DeviceIoControl 的方法可以保证程序在 Windows NT 和 Windows 9x 下的兼容性，在两个操作系统下，仅有 CreateFile 语句是不同的。

四、设备驱动的进一步封装

至此，完成了对驱动程序的初步设计。但考虑到在上面调用设备驱动时使用的 DeviceIoControl 函数仍是比较复杂的，程序也不太容易具有通用性。而且，在有些开发工具中，如 Visual Basic，不包括直接读写 I/O 端口的语句，所以可以考虑根据不同软件的需要对驱动程序进行不同的封装。目前，我们实现了以 DLL、ActiveX、VCL 和 C++类库中进行封装。DLL 可以在大多数软件环境中进行调用。ActiveX 可以在 Visual Basic 等可视编程环境中使用。VCL 可以在 Delphi 和 C++ Builder 中使用。考虑到许多用户使用 Visual C++，所以也提供了 C++类库方式。

参考文献：

[1] 马卫国、何佩琨，通用高速 PCI 总线目标模块的设计，电子技术应用，1999 年 1 期

[2] Art Baker，Windows NT 设备驱动程序设计指南，机械工业出版社，1997 年 12 月

[3] AMCC S5933 PCI Controller Data Book，Applied Micro Circuits Corporation，1996