计算机专业人员书库

Windows 2000/XP

WDM设备驱动程序开发

武安河 郜铭 于洪涛 编著

電子工業出版社
Publishing House of Electronics Industry
http://www.phei.com.cn

Windows 2000/XP

WDM设备驱动程序开发

计算机专业人员书库

Windows 2000/XP WDM 设备驱动程序开发

武安河　邰　铭　于洪涛　编著

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

内 容 简 介

本书主要介绍用 DriverStudio 开发工具开发 Windows 2000/XP 下的 WDM 设备驱动程序的原理及编程方法。本书详细介绍了 WDM 基本程序框架和编程，IRP 的基本概念及编程，WDM 和应用程序之间的通信、即插即用、电源管理、WMI 的编程技术，IRP 串行处理、过滤器驱动程序，WDM 访问硬件设备、处理硬件中断、实现 DMA 操作的编程技术，以及大量的基本编程技术，还有 USB 接口和 PCI 接口设备驱动程序 WDM 的开发。

本书是一本技术性较强的工具书，附有 20 个典型的编程实例，适合具有一定计算机硬件及 C++ 语言基础的计算机应用开发人员阅读，也是计算机应用爱好者和高等院校学生的实用参考书。

图书在版编目（CIP）数据

Windows 2000/XP WDM 设备驱动程序开发/武安河，邰铭，于洪涛编著. —北京：电子工业出版社，2003.4
（计算机专业人员书库）
ISBN 7-5053-8647-6

Ⅰ. W… Ⅱ. ①武… ②邰… ③于 Ⅲ. 窗口软件，Windows 2000/XP—驱动程序—程序设计 Ⅳ. TP316.7

中国版本图书馆 CIP 数据核字（2003）第 026105 号

责任编辑：毛兆余 王 斌
印 刷：北京大中印刷厂
出版发行：电子工业出版社 http://www.phei.com.cn
北京市海淀区万寿路 173 信箱 邮编 100036
经 销：各地新华书店
开 本：787×1092 1/16 印张：27.25 字数：672 千字 附光盘 1 张
版 次：2003 年 4 月第 1 版 2003 年 4 月第 1 次印刷
印 数：6000 册 定价：45.00 元 （含光盘）

凡购买电子工业出版社的图书，如有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系。联系电话：（010）68279077

前　言

2001 年 9 月，电子工业出版社出版了由我主编的《Windows 设备驱动程序（VxD 与 WDM）开发实务》一书，这本书主要介绍了用 VtoolsD 开发 Windows 9*x* 设备驱动程序 VxD 的原理及编程方法，虽然也介绍了用 DriverWorks 开发 Windows 设备驱动程序 WDM 的原理及编程方法，但仅是入门性的。

WDM（Windows Driver Model）是微软公司全新的驱动程序模式，支持即插即用、电源管理和 WMI 技术，它的运行平台是 Windows 98/2000/XP 操作系统。随着 Windows 操作系统的不断发展，致力于 Windows 操作系统下微机接口设备开发的广大科技开发人员，必须学习和掌握 WDM 的原理及编程方法。

我对 WDM 进行了深入的研究，在学习的过程中，不仅在原理上有许多一时难以理解的问题，而且在编写实例时，也遇到了许多未曾想到的技术困难。虽然最终解决了所有问题，但却花费了大量的时间。可以自信地说，我的经验是宝贵的，读者将会少走很多弯路和节省大量的时间。

我对参考文献中提到的《Windows WDM 设备驱动程序开发指南》、《Programming the Microsoft Windows Driver Model》和《Windows 2000 驱动程序开发大全》这三本书做了深入的研究，基本掌握了 WDM 的编程思想。我之所以说编程思想，是因为这三本书所用的是 DDK 工具，而我所用的是 DriverWorks 工具。我不大喜欢用 DDK 编程，它太复杂了，有点非高手莫入的味道。有人把 DDK 比做汇编语言，把 DriverWorks 比做 C 语言，这比喻比较恰当。也许有人认为还是 DDK 正宗，话虽如此，但对于大多数的计算机应用开发人员来说，时间及效率可能更重要。这就是我为什么向读者积极推荐使用 DriverStudio 工具来开发 Windows 设备驱动程序的原因。

开发工具

下面谈谈 WDM 设备驱动程序的开发工具。

开发 WDM 离不开 DDK，微软公司提供了 Windows 98 DDK，Windows 2000 DDK 和 Windows XP DDK 三个版本，分别对应相应的三种操作系统。WDM 程序仅在源代码级兼容这三个操作系统，但所生成的可执行代码是不同的，必须用相应的 DDK 来生成某一操作系统下的 WDM 驱动程序。

本书没有将 Windows 98 考虑在内，因为 Windows 98 操作系统对 WDM 的支持不是特别好，有一些语句它并不支持。关于这一点，请参考附录 1 和 2。

DriverStudio 是一个开发工具包，包含 VtoolsD，DriverWorks 和 SoftICE 等开发工具，现在其较高的版本为 2.5，2.6 和 2.7，其中，2.6 版本以后可用于 Windows XP WDM 设备驱动程序的开发。用 DriverWorks 开发 WDM 必须有 DDK 的支持。

WDM 程序框架有一定的要求，若用 DDK 开发 WDM，读者可参考附录 1 和 2 的范例。用 DriverWorks 所生成的 WDM 程序框架对于开发人员来说非常简单。但实质上，仍满足

DDK 对 WDM 的要求，调用的仍是 DDK 所提供的基本函数。

本书还用到 Visual C++ 6.0 和 Platform SDK（February 2001 Edition）工具。Platform SDK 只在编译 PnPEvent 和 WMISample 实例的应用程序时需要。

本书配套光盘

本书的配套光盘含有书中所有实例的驱动程序和应用程序的全部源代码，以及生成的驱动程序和可执行的应用程序。除 USBCounter 和 PCI9054 实例因需要硬件设备的支持而无法运行外，其他 18 个实例均可运行。

光盘中的实例分为 Windows 2000 和 Windows XP 操作系统两部分，其中 Windows XP 下的实例不包含 USBCounter 和 PCI9054 实例。

另外，还有 USB 相关资料和 PCI9054 的文档资料。

Chinesedot 目录下的 HZK16 点阵字库是 FileThread 实例需要使用的。

设备类注册表下的注册表文件用于创建 Windows 2000 和 Windows XP 操作系统下的“WDM 范例”设备类信息。

有许多读者购买了《Windows 设备驱动程序（VxD 与 WDM）开发实务》一书，这里向大家表示谢意。如果我的经验能对你们学习和掌握 Windows 设备驱动程序有一些帮助，那将是我最高兴的事情。感谢各位读者和电子工业出版社的支持，使我得以续写《Windows 2000/XP WDM 设备驱动程序开发》一书。我以自己的实战经验写书，我的目标是实用第一、简单第一，本书的特点是入门和实例。希望这本书能对你现在或将来的工作有所帮助。既然是经验，难免有高低之分，请大家见谅，我是尽心尽力写这两本书的。

感谢六系王大会主任和钟庆山政委的支持和鼓励。

感谢江华、周慧琴夫妇的支持和合作。

感谢邵高平、程保炜、胡君杰和刘立柱教员的支持。

感谢张瑾和李晶晶教员的支持。

感谢张勇和孙勇研究生的支持。

感谢教研室领导和同事的支持和鼓励。

特别感谢 http://e.pku.edu.cn 和其他一些优秀网站。

由于我们的理论水平有限，书中难免出现差错和遗漏，敬请广大计算机应用开发人员批评指正。

武安河

目　录

第 1 章　Windows 2000 和 WDM 驱动程序

1.1　Windows 2000 组件概述

图 1-1 显示了 Windows 2000 操作系统环境的主要组件。在 Windows 2000 操作系统环境中，一部分组件运行在用户模式下，其他的则运行在内核模式下。

Windows 2000 操作系统包括了许多内核模式组件，它们被精心地定义为功能相互独立的组件。对内核模式驱动程序设计者来说，最感兴趣的就是内核、I/O 管理器、即插即用（PnP）管理器、电源管理器、硬件抽象层、配置管理器、内存管理器、运行支持和进程结构组件。对另一些设计者来说，感兴趣的其他组件可能还包括对象管理器和安全引用监视器。

即插即用管理器和电源管理器是 Windows 2000 中的新组件。它们仅仅支持 Windows 2000 驱动程序和 WDM 驱动程序。

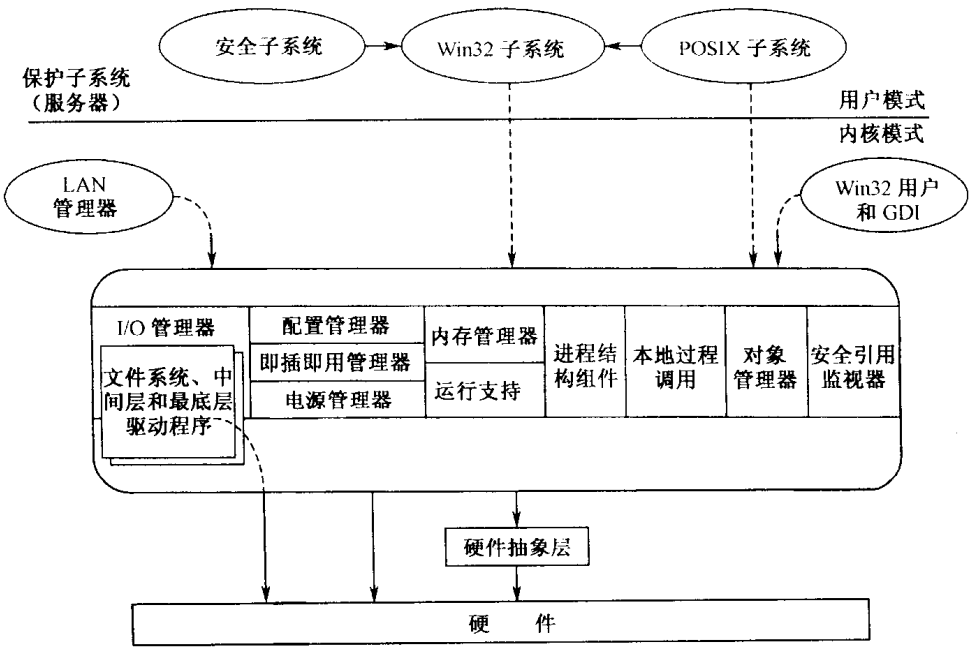

图 1-1　Windows 2000 组件总览

当用户模式程序需要读取设备数据时，它就调用 Win32 API 函数，如 ReadFile。Win32 子系统模块（如 KERNEL32.DLL）通过调用平台相关的系统服务接口实现该 API，而平台相关的系统服务将调用内核模式支持例程。在 ReadFile 调用中，调用首先到达系统 DLL（NTDLL.DLL）中的一个入口点 NtReadFile 函数，然后这个用户模式的 NtReadFile 函数调用系统服务接口，最后由系统服务接口调用内核模式中的服务例程，该例程同样命名为 NtReadFile。

系统中还有许多与 NtReadFile 相似的服务例程，它们同样运行在内核模式中，为应用

程序请求提供服务，并以某种方式与设备交互。它们首先检查传递给它们的参数以保护系统安全或防止用户模式程序非法存取数据，然后创建一个称为“I/O 请求包（IRP）”的数据结构，并把这个数据结构送到某个驱动程序的入口点。在刚才的 ReadFile 调用中，NtReadFile 将创建一个主功能代码为 IRP_MJ_READ（DDK 头文件中的一个常量）的 IRP。实际的处理细节可能会有不同，但对于 NtReadFile 例程，可能的结果是，用户模式调用者得到一个返回值，表明该 IRP 代表的操作还没有完成。用户模式程序也许会继续其他工作，然后等待操作完成，或者立即进入等待状态。不论哪种方式，设备驱动程序对该 IRP 的处理都与应用程序无关。

执行 IRP 的设备驱动程序最后可能会访问硬件。对于 PIO 方式的设备，一个 IRP_MJ_READ 操作将导致直接读取设备的端口（或者是设备实现的内存寄存器）。尽管运行在内核模式中的驱动程序可以直接与其硬件会话，但它们通常都使用硬件抽象层（HAL）访问硬件。读操作最后会调用 READ_PORT_UCHAR 从某个 I/O 口读取单字节数据。HAL 例程执行的操作是平台相关的。在 Intel x86 计算机上，HAL 使用 IN 指令访问设备端口；在 Alpha 计算机上，HAL 使用内存读取指令访问设备实现的内存寄存器。

驱动程序完成一个 I/O 操作后，通过调用一个特殊的内核模式服务例程来完成该 IRP。完成操作是处理 IRP 的最后动作，它使等待的应用程序能够继续运行。

1.2 Windows 2000 中的驱动程序种类

如图 1-2 所示，Windows 2000 系统可以使用多种驱动程序。

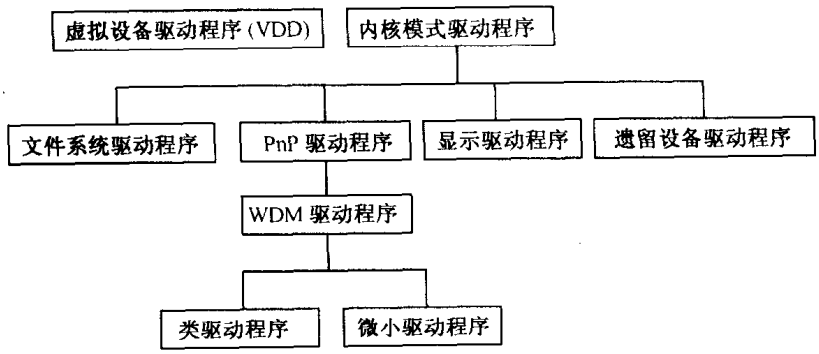

图 1-2 Windows 2000 中的设备驱动程序种类

- 虚拟设备驱动程序（VDD）是一个用户模式部件，可以使 DOS 应用程序访问 x86 平台上的硬件。VDD 通过屏蔽 I/O 权限掩码来捕获端口存取操作，它基本上是模拟硬件操作，这对于那些直接对微机硬件编程的应用程序特别有用。

内核模式驱动程序的分类如下：

- PnP 驱动程序就是一种遵循 Windows 2000 即插即用协议的内核模式驱动程序。
- WDM 驱动程序是一种 PnP 驱动程序，它同时还遵循电源管理协议，并能在 Windows 98，Windows 2000 和 Windows XP 间实现源代码级兼容。WDM 驱动程序还细分为类驱动程序（class driver）和微小驱动程序（minidriver），类驱动程序管理属于已定义类的设备，微小驱动程序给类驱动程序提供厂商专有的支持。
- 文件系统驱动程序在本地硬盘或网络上实现标准 PC 文件系统模型（包括分级目录

结构和命名文件概念）。

- 显示驱动程序是用于显示和打印设备的内核模式驱动程序。
- 遗留设备驱动程序也是一种内核模式驱动程序，它直接控制一个硬件设备而不用其他驱动程序帮助。这种驱动程序主要包括 Windows NT 早期版本的驱动程序，它们可以不加修改地运行在 Windows 2000 中。

1.3 WDM 驱动程序特点

1.3.1 内核模式驱动程序的设计目标

内核模式驱动程序的设计目标与 Windows 2000 的许多设计目标相符合，特别是系统 I/O 管理器部分。这些设计目标包括：

- 平台之间的可移植性
- 硬件和软件的可配置性
- 永远抢占优先和永远中断
- 多处理器平台上的多处理器安全
- 基于对象
- 带可重用 I/O 请求包（IRP）的包驱动 I/O
- 支持异步 I/O

1．平台之间的可移植性

内核模式驱动程序的源代码应该可以移植到所有 Windows NT 平台。WDM 驱动程序在其定义中就规定了其源代码可以在 Windows 98，Windows 2000 和 Windows XP 之间相互移植。为了实现这种可移植性，驱动程序应该全部用 C 语言编写，并且只使用 ANSI C 标准规定的语言元素。应避免使用编译器厂商专有的语言特征，并避免使用没有被操作系统内核输出的运行时库函数。如果不能避免驱动程序中的平台依赖，至少应该用条件编译指令隔离这些代码。如果严格遵循这些设计方针，仅需要重新编译连接源代码，生成的驱动程序就可以运行在任何新的 Windows NT 平台上。

如果仅使用 WDM.H 中声明的内核模式支持函数，那么可以很容易地实现驱动程序的源代码级兼容。但是，操作系统之间的差异会在某些地方影响驱动程序的移植性。

2．硬件和软件的可配置性

内核模式驱动程序应避免对设备特征或某些系统设置做绝对假设，这些系统设置会随着平台的改变而变化。例如，在 x86 平台上，标准串行口使用一个专用的 IRQ 和 8 个 I/O 端口，这些数值持续 20 年从未改变。把这些值直接写到驱动程序中将使驱动程序失去可配置性。

为了实现可配置性，首先应该在代码中避免直接引用硬件，即使是在平台相关的条件编译块中也是这样。应该使用 HAL 工具或调用低级总线驱动程序，或者实现一个标准或定制的控制接口，并通过控制面板程序与用户交互。另外，还应该支持 Windows 管理诊断（WMI）控件，这种控件允许用户和管理员在分布式企业环境中配置硬件特征。最后，应该使用注册表作为配置信息的数据库，这可以使配置信息在系统重新启动后仍然存在。

3. 永远抢占优先和永远中断

Windows 2000 是多任务操作系统，可以为任意多个线程分配 CPU 时间。在大部分时间中，驱动程序例程执行在可以被其他线程（在同一个 CPU 上）抢先的环境中。线程抢先取决于线程的优先级，系统使用系统时钟为线程分配 CPU 时间片。

Windows 2000 还使用了一个中断优先级的概念，即 IRQL。本章后面将详细讨论 IRQL，在这里先简单介绍一下。你可以认为 CPU 中有一个 IRQL 寄存器，它记录着 CPU 当前的执行级别。有三个 IRQL 值对设备驱动程序有重要意义：PASSIVE_LEVEL，DISPATCH_LEVEL，以及设备中断请求级 DIRQL。在大部分时间里，CPU 执行在 PASSIVE_LEVEL 级上，所有的用户模式代码也运行在 PASSIVE_LEVEL 级上，并且驱动程序的许多活动也都发生在 PASSIVE_LEVEL 级上。当 CPU 运行在 PASSIVE_LEVEL 级时，当前运行的线程可以被任何优先级大于它的线程抢先。然而，一旦 CPU 的 IRQL 大于 PASSIVE_LEVEL 级，线程抢先将不再发生，此时 CPU 执行在使 CPU 越过 PASSIVE_LEVEL 级的任意线程上下文中。

你可以把高于 PASSIVE_LEVEL 级的 IRQL 看成是针对中断的优先级。这是一个与支配线程抢先机制不同的优先级方案，正如前面说的，在 PASSIVE_LEVEL 级上没有线程抢先发生。但是，运行在任何 IRQL 级上的活动都可以被更高 IRQL 级上的活动中断。所以，驱动程序必须假定在任何时刻都可能失去控制权，而此时系统可能需要执行更基本的任务。

4. 多处理器平台上的多处理器安全

Windows 2000 可以运行在多处理器计算机上。Windows 2000 使用对称多处理器模型，即所有的处理器都是相同的，系统任务和用户模式程序可以执行在任何一个处理器上，并且所有处理器都平等地访问内存。多处理器的存在给设备驱动程序带来了一个困难的同步问题，因为执行在多个 CPU 上的代码可能同时访问共享数据或共享硬件资源。Windows 2000 提供了一个同步对象——自旋锁（Spin Lock），驱动程序可以使用它来解决多处理器的同步问题。

5. 基于对象

Windows 2000 内核是基于对象的，即驱动程序和内核例程使用的许多数据结构都有公共的特征，这些特征由对象管理器集中管理。这些特征包括名称、参考计数、安全属性等。在内部，内核中包含了许多执行公共对象管理的方法例程，例如打开和关闭对象。

驱动程序使用内核部件输出的服务例程来维护对象或对象中的域。某些内核对象，比如内核中断对象是不透明的，DDK 头文件中没有其数据成员的声明。其他内核对象，如设备对象或驱动程序对象则是部分不透明的，DDK 头文件中声明了其结构的全部成员，但 DDK 文档中仅描述了可访问的成员并警告驱动程序开发者不要直接访问或修改其他成员。对于驱动程序必须间接访问的不透明域，可以用支持例程访问。部分不透明的对象类似于 C++的类，有任何人都能访问的公共成员，还有必须通过方法才能访问的私有成员和保护成员。

6. 带可重用 I/O 请求包（IRP）的包驱动 I/O

I/O 管理器和设备驱动程序使用 IRP 来管理 I/O 操作的具体细节。首先，某个内核模式部件创建一个 IRP，该 IRP 可以是一个让设备执行一个操作、向驱动程序发送一个命令或者向驱动程序询问某些信息的请求。然后，I/O 管理器把这个 IRP 发送到驱动程序的输出例程上。一般说来，每个驱动程序例程仅执行 IRP 指定的一部分工作，然后返回 I/O 管理器。最后，

某个驱动程序例程完成该 IRP，之后 I/O 管理器删除该 IRP 并向原始请求者报告结束状态。

7．支持异步 I/O

Windows 2000 允许应用程序和驱动程序在启动一个 I/O 操作后继续执行，而此时 I/O 操作仍在进行。所以，需要长时间运行的 I/O 操作应该以异步方式执行。即当驱动程序接收到一个 IRP 后，它首先初始化用于管理该 I/O 操作的任何状态信息，然后安排 IRP 的执行，而该 IRP 将在以后的某个时刻完成，最后返回到调用者。由调用者决定是否等待该 IRP 的完成。作为一个多任务操作系统，Windows 2000 根据线程的适合性和优先级来调度它们在有效处理器上的执行。通常，驱动程序在某些不可预测线程的上下文中应该使用异步方式处理 I/O 请求。我们使用术语任意线程上下文（Arbitrary Thread Context）来描述驱动程序并不知道或并不关心处理器当前正执行在哪一个线程上的上下文中。驱动程序应避免阻塞任意线程，这使得驱动程序有了这样的架构：通过执行离散的操作并返回来响应硬件事件。

1.3.2　WDM 驱动程序模型

在 WDM 驱动程序模型中，每个硬件设备至少有两个驱动程序。其中一个驱动程序称为功能（Function）驱动程序，通常它就是你认为的那个硬件设备驱动程序。它了解硬件工作的所有细节，负责初始化 I/O 操作，有责任处理 I/O 操作完成时所带来的中断事件，有责任为用户提供一种设备适合的控制方式。

另一个驱动程序称为总线（Bus）驱动程序。它负责管理硬件与计算机的连接。例如，PCI 总线驱动程序检测插入到 PCI 槽上的设备并确定设备的资源使用情况，它还能控制设备所在 PCI 槽的电流开关。

WDM 功能驱动程序通常由两个分离的执行文件组成。一个文件是类驱动程序，它了解如何处理操作系统使用的 WDM 协议（有些协议相当复杂），以及如何管理整个设备类的基本特征。USB 照相机类驱动程序就是一个例子。另一个文件称为微小驱动程序（minidriver），它包含类驱动程序用于管理设备实例的厂商专有特征例程。类驱动程序和微小驱动程序合在一起才成为一个完整的功能驱动程序。

一个完整的驱动程序包含许多例程，当操作系统遇到一个 IRP 时，它就调用驱动程序中的例程来执行该 IRP 的各种操作。有些例程是必需的，比如 DriverEntry 和 AddDevice，还有 DispatchPnP，DispatchPower 和 DispatchWMI 派遣函数。需要对 IRP 排队的驱动程序一般都有一个 StartIo 例程；大部分能生成硬件中断的设备，其驱动程序都有一个中断服务例程和一个延迟过程调用例程；执行 DMA 传输的驱动程序应有一个 AdapterControl 例程。图 1-3 描述了 WDM 驱动程序的一些例程。所以，WDM 驱动程序开发者的一项任务就是选择所需要的例程。

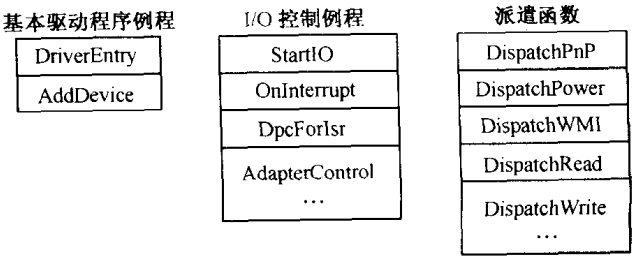

图 1-3　WDM 驱动程序例程

1.3.3 设备和驱动程序的层次结构

WDM 驱动程序采用分层的结构模型，如图 1-4 所示。图中左边是一个设备对象堆栈，设备对象是操作系统为帮助软件管理硬件而创建的数据结构。处于堆栈最底层的设备对象称为物理设备对象（Physical Device Object），或简称为 PDO。在设备对象堆栈的中间有一个对象称为功能设备对象（Functional Device Object），或简称为 FDO。在 FDO 的上面和下面可能还会有一些过滤器设备对象（Filter Device Object），位于 FDO 上面的过滤器设备对象称为上层过滤器，位于 FDO 下面的过滤器设备对象称为下层过滤器。

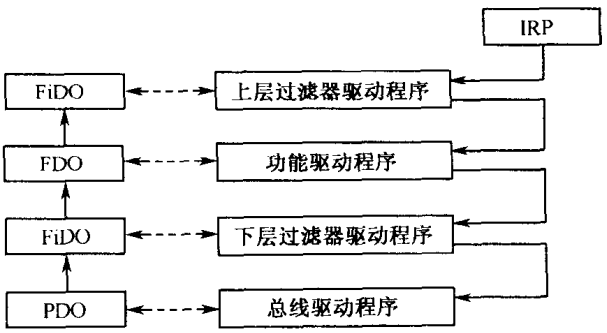

图 1-4 WDM 设备对象和驱动程序的层次结构

操作系统的 PnP 管理器按照设备驱动程序的要求构造设备对象堆栈。总线驱动程序的一项任务就是枚举总线上的设备，并为每个设备创建一个 PDO。一旦总线驱动程序检查到新硬件存在，PnP 管理器就创建一个 PDO，之后便开始描绘如图 1-4 所示的结构。

创建完 PDO 后，PnP 管理器参照注册表中的信息查找与这个 PDO 相关的过滤器和功能驱动程序，它们出现在图的中部。系统安装程序负责添加这些注册表项，而驱动程序包中控制硬件安装的 INF 文件负责添加其他表项。这些表项定义了过滤器和功能驱动程序在堆栈中的次序。PnP 管理器先装入最下层的过滤器驱动程序并调用其 AddDevice 函数。该函数创建一个 FiDO，这样就在过滤器驱动程序和 FiDO 之间建立了水平连接。然后，AddDevice 把 PDO 连接到 FiDO 上，这就是设备对象之间连线的由来。PnP 管理器继续向上执行，装入并调用每个下层过滤器、功能驱动程序、上层过滤器，直到完成整个堆栈。

层次结构可以使 I/O 请求过程更加明了，见图 1-4 的右侧。影响到设备的每个操作都使用 I/O 请求包。通常，IRP 先被送到设备堆栈的最上层驱动程序，然后逐渐过滤到下面的驱动程序。每一层驱动程序都可以决定如何处理 IRP。有时，驱动程序不做任何事，仅仅是向下层传递该 IRP；有时，驱动程序直接处理完该 IRP，不再向下传递；还有时，驱动程序既处理了 IRP，又把 IRP 传递下去。这取决于设备以及 IRP 所携带的内容。

在单个硬件的驱动程序堆栈中，不同位置的驱动程序扮演了不同的角色。总线驱动程序管理计算机与 PDO 所代表的设备的连接。功能驱动程序管理 FDO 所代表的设备。过滤器驱动程序用于监视和修改 IRP 流。

1.3.4 中断级别 IRQL

在 Windows 内核中，中断按其优先等级分为 32 个级别。表 1-1 列出了这些中断级别的排列顺序。