“十五”国家重点电子出版物规划项目·计算机知识普及和软件开发系列

2002 开发人员宝典系列（3）

# VC++.NET 开发驱动程序详解

Windows 2000/XP

北京希望电子出版社 总策划
郭益昆 编写

中国科学出版集团
北京希望电子出版社

“十五”国家重点电子出版物规划项目·计算机知识普及和软件开发系列

2002 开发人员宝典系列（3）





# VC++.NET 开发驱动程序详解

## Windows 2000/XP

北京希望电子出版社 总策划
郭益昆 编 写

中国科学出版集团
北京希望电子出版社

## 内 容 简 介

本书是一本介绍 Windows 2000 Professional 和 Windows XP 的核心驱动程序的专著。提供了多种核心程序模型，极大地方便了读者的学习和应用。

本书内容由 6 部分组成，第 1 部分为核心模式基础，主要内容有系统与驱动，对 I/O、I/O 对象和 IRPs 进行分层、调度方法和优先级、系统定义的对象与驱动的关系、基本驱动结构。第 2 部分为核心流驱动，主要讲述了核心流驱动的概念、流小驱动、音频驱动、音频小端口驱动、视频捕获驱动以及 DVD 驱动模型设计。第 3 部分为即插即用，主要内容有 PnP 需要的驱动支持，增加新 PnP 设备的步骤。第 4 部分为电源管理，包括电源管理的含义及其实现步骤。第 5 部分为 Windows 2000 的管理机制。第 6 部分为调试与开发驱动程序，介绍了 1 0 个调试工具以及如何使用 VC++.NET 开发驱动程序。

本书结构清晰，逻辑严密，内容环环相扣，不但是从事用 VC++.NET 进行开发与应用的广大编程人员的技术指导书，同时也可作为大专院校计算机专业、非专业师生重要的参考读物。

本版 CD 为驱动程序源代码。


系 列 盘 书：“十五”国家重点电子出版物规划项目　计算机知识普及和软件开发系列
2002 开发人员宝典系列（3）
盘 书 名：VC++ .NET 开发驱动程序详解—Windows 2000/XP
总 策 划：北京希望电子出版社
文本著作者：郭益昆 编写
C D 制作者：希望多媒体开发中心
C D 测试者：希望多媒体测试部
责任编辑：杨如林　周 艳　周凤明
出版、发行者：北京希望电子出版社
地 址：北京中关村大街 26 号，100080
网址: www.bhp.com.cn　E-mail: lwm@hope.com.cn
电话: 010-62562329, 62541992, 62637101, 62637102, 62633308, 62633309
（图书发行和技术支持）
010-62613322-215（门市）　010-62547735（编辑部）
经 销：各地新华书店、软件连锁店

排 版：希望图书输出中心 马君 李毅
C D 生产者：北京中新联光盘有限责任公司
文本印刷者：北京媛明印刷厂
开本 / 规格：787 毫米×1092 毫米　16 开本　23 印张　534 千字
版次 / 印次：2002 年 4 月第 1 版　2002 年 4 月第 1 次印刷
印 数：1-4000 册
本 版 号：ISBN 7-900088-76-8
定 价：42.00 元（本版 CD）

# 前　言

核心开发是近年来开发人员关注的重点，因为通过核心基础知识的学习，对应用程序的开发是有很大好处的。它使开发人员可以进行更深入的编程，并提高程序的效率。本书内容是关于 Windows2000 Professional 和 Windows XP 的核心程序，内容由 6 部分组成：核心模式基础知识、核心流、即插即用、电源管理、系统管理机制和驱动程序调试。其中，既有在 VC++.NET 环境下关于描述、处理、改变数据格式的方法，也有针对硬件的驱动开发的描述、处理、拓扑分布的实现方法。

本书对于编程过程中所需要注意的重点进行了清晰的描述，对于常见问题给予了实际的解决方法，并阐述了关于解决方法的原理。为了便于读者的阅读理解，在每节中都对专业词汇进行了详尽的解释，并通过对于本书中模型的学习，使读者在短期内达到实际编写系统核心模式程序的水平。

由于本书中涉及的概念较多，并且在实际编程中所需要的原理比较广泛，建议读者在阅读的时候可以先略过暂时不懂的概念或词汇，这些概念在后续的章节中都给出了详细的解释与说明。

通过本书对各种概念的描述，读者既可以搞清教科书中常常提到的 Windows 系统与中间件，中间件与硬件层的关系，也能够熟悉用 VC++.NET 实施对各个层面的控制。

如果要读懂本书，读者应具备基本的英语知识及 VC++基本命令。

本书由郭益昆主编，此外，参加编写工作的还有王宝华、李冬伟、张宝仪、胡广达、周贵勤、李宗武、张达、王华生等。本书中的所有解释性描述都是在实际工作中总结的很实用的经验，尽管在编写本书时尽了最大努力，但由于水平有限，难免会出现一些不足之处，还望广大读者给予批评指正。

编者

# 目　录

## 第 1 部分　核心模式驱动基础



## 第 2 部分　核心流驱动

## 第 3 部分 即插即用

# 第 1 部分　核心模式驱动基础

## 第 1 章　系统与驱动

### 1.1　Win 2000 中的驱动结构

学习核心模式驱动首先要知道什么是核心模式。核心（Kernel）也叫核心、核心程序。在操作系统中，实现诸如分配硬件资源、进程调度等基本功能，是与硬件机器直接打交道的的部分，始终驻留内存。

核心模式（Kernel Mode）又称为核心模式。在计算机系统中的最高特许处理器存取模式，即计算机正在执行核心中的指令时的模式。操作系统的最高特定服务功能（如 I/O 设备驱动）均可在这个模式下运行。

核心对象（Kernel Object）是在 Windows 系统中，一个核心定义的抽象数据模型的运行实例，核心为核心对象定义的特别的句义，并执行系统执行对象为操作核心对象所能调用的核心程序。核心对象可分为控制对象和调度程序对象两类，他们是 Windows 系统执行对象的基础。

下面我们先全面的了解一下系统结构,这样，有助于我们学习驱动，了解驱动的作用。图 1.1 显示了 Windows2000 运行上下文的主要部件和工作结构。

从登陆进程、Win32 客户、和 POSIX 客户到各个子系统之间是消息处理机制。而从子系统到系统服务，用的是系统捕获机制。也就是说 Win2000 中包含了一些在用户模式中运行的部件。用户模式是计算机运行的一种模式，在这个模式上，处理器只能存取分配给该用户进行作业的存储区和其他资源，不能执行特权指令，此时处理器处于最小优先级运行状态。而其他的系统部件是在核心模式中，它们是相互独立的部件，这些部件，都是逻辑上的部件，它们实际上是一些指定的程序或操作原则。其中，对核心模式驱动开发者来说，必须明确知道：核心（Kernel），I/O 管理器（I/O manager），即插即用管理器（Plug And Play Manager），电源管理（Power Manager），硬件应用层（HAL，Hardware Abstraction Layer），配置管理器（Configuration Manager），内存管理器（Memory Manager），执行支持（Executive Support），和进程结构部件（Process Structure Components）。这里也会提到对象管理器（Object Manager）和安全引用原则监视（Security Reference Monitor）。

这里的 XX 器，实际上就是 XX 接口或 XX 程序。

总的来说有如下 11 个方面：

（1）核心（Kernel）也叫核心，核心程序。在操作系统中，实现诸如分配硬件资源、进程调度等基本功能，是与硬件机器直接打交道的的部分，始终驻留内存。

（2）I/O 管理器（I/O manager）就是输入/输出的管理程序，用于管理关于 I/O 的各种资源的分配与回收的程序。

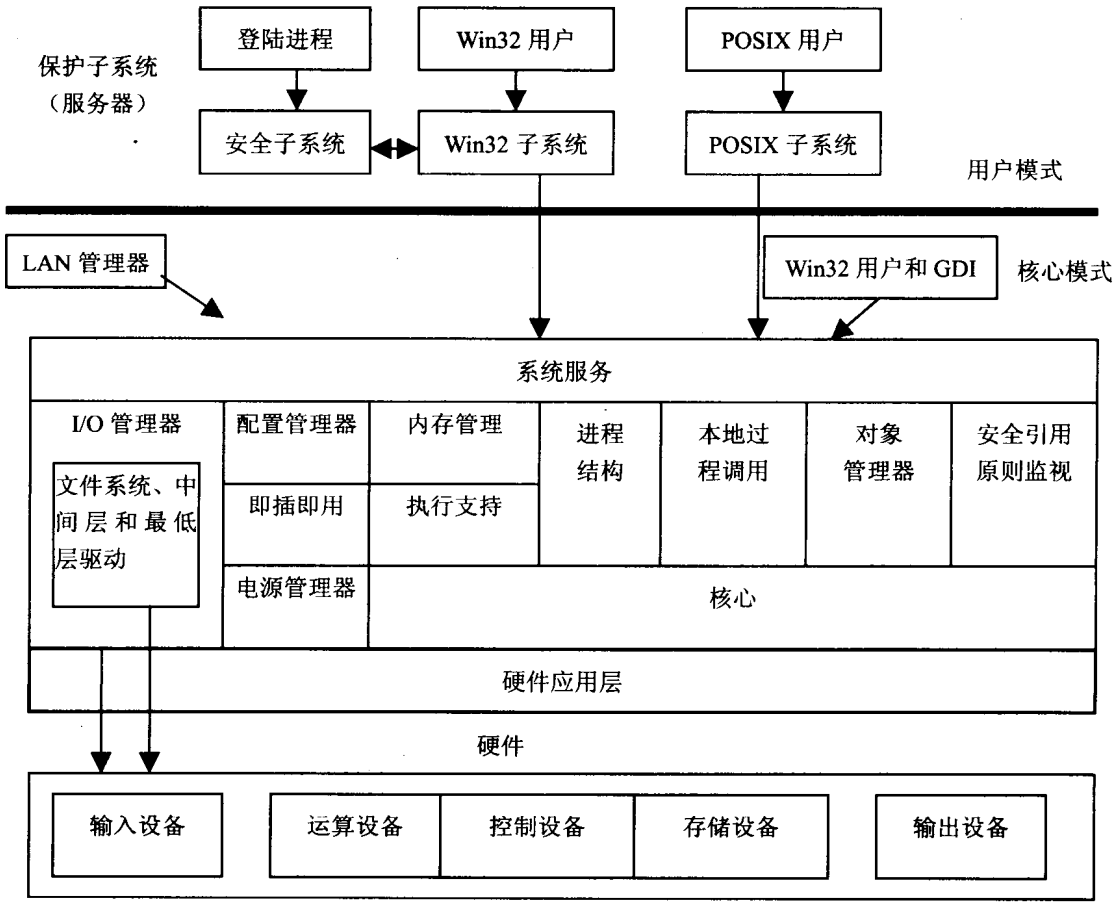



图 1.1　Windows 2000 运行上下文主要部件和工作结构

（3）即插即用管理器，也叫 PnP，它是 IBM 在推出 PS/2 计算机的时候，通过微通道技术而体现的一种技术。Intel 公司在推出 PCI（Peripheral Compenent Interconnect）局域总线的时候也采用了这个技术。PCI 部件内部设有系统配置寄存器，其中保存着系统中现有外围设备与 CPU 沟通的地址与中断向量。系统启动时，配置寄存器的内容会自动完成系统配置操作。主机板上支持即插即用的芯片组允许隔离每一个适配器，当用户将新的适配卡插入总线扩展槽时，配置软件会自动选择空闲的中断向量，确保各扩充卡的中断不会发生冲突，因而不必再用手工调整跨接线和 DIP 开关，也不必考虑中断向量的设置问题。即插即用管理器就是即插即用的的管理程序，用于管理关于即插即用的各种资源的分配与回收的程序。

（4）电源管理器，是对计算机电源的管理程序，首先，电源管理是指绿色计算机的一个基本要求。当计算机较长时间未执行任何操作时，可自动进入省电状态，其管理程序，是对其中各个细微过程的管理。

（5）硬件应用层，是系统与硬件之间的一个层次。用来上下传送数据的一个层次，对

用户来讲，是封闭的，不透明的（这个层，对我们来说，也是不透明的，但我们可以操作它，控制它，管理它）。

（6）配置管理器，配置的意思是计算机系统或网络按照其功能部件的性质、数量和主要特性而确定的软件和硬件的安排，即建立一个自选的系统工作上下文。配置管理器是根据要求而进行全面控制、安排的程序。

（7）内存管理器，是对内存的低层控制程序。基本上是在分页层次的管理、控制程序，也讲了些虚拟与实际硬件之间的相互联系与操作。分页是：在采用虚拟存储器的计算机系统中，将内存与外部存储器划分为具有固定长度的块，用以存储数据和指令块，这被称为页面。页面是分配给作业的基本存储单位，一个作业可以存放在物理上不连续的多个页面中。作业执行时，仅将当前操作所必须的部分页面装入内存中，称为实页。如果作业操作过程中需要其他页的内容，则必须将当前内存中已有的某个或某些页按规定的调度法淘汰掉，才能腾出空间装入所需的。利用这个方法，可以在有限的存储空间中执行较大的程序。有时，它也指在内存或辅助存储器中的指令块和数据块。

（8）执行支持，是一些对执行程序进行控制和管理的程序。执行程序是：编码指令的一个主集，是控制其他程序的装入和重定位的一种程序。在有些情况下，执行程序可利用一般程序员都不知道的指令。它可提供书写程序、编译、组合、装入、及执行等功能的程序。用此程序，计算机可以综合各程序的要求，分别按调度程序执行的顺序，若加上其他一些程序的配合执行程序就可扩大为操作系统。执行程序的主要任务是作业调度，内存分配、监督管理控制输入 / 输出装置等。

（9）进程结构部件，是通过进程提供支持的系统结构部件。进程的意思是：具有一定功能的程序与某个有关的数据集合相结合而实现的一次运行活动。在本书中，是对各个核心对象进行支持的固定的系统实例结构。

（10）对象管理器，是指系统对对象的管理手段，分为目录对象和符号链接对象。

（11）安全引用原则监视：安全引用原则是在计算机保密中，抽象计算机对主体存取客体的安全控制机制进行实施和调整的一种保密控制原则，原则上，基准原则是完整的独立于系统变量的和可证明的。它是一些概念。根据这些概念而对系统进行监查管理的程序集，就是安全引用原则监视机制。

## 1.2　Win2000 驱动分类

在 Win2000 中，驱动的作用是非常大的，可以说，所有的操作都需要驱动的运行才能完成。驱动可分为两个基本类型：

### 1. 用户模式中的被保护的子系统

和用于逻辑的、虚拟的或物理设备的核心模式驱动。在 Windows2000 中有三个子系统：POSIX，OS/2 和 Win32 子系统。其中 Win32 子系统最重要，如果没有它，Windows2000 就不能运行。

子系统的作用是将基本的执行支持系统服务的某些子集提供给应用程序。每个子集都可以访问 Windows2000 中本地服务的不同子集，函数调用不能在子系统之间混用。用户应用程序不能直接调用 Window2000 系统服务，这种调用必须通过一个或多个子系统动态链接库作为中介才能够完成。例如 Win32 动态链接库（如 KERNEL32.DLL、USER32.DLL 和 GDI32.DLL）实现 Win32API 函数，POSIX 子系统动态链接库则实现 POSIX 1003.1API。

（1）Win32 子系统。Win32 子系统由下列重要部件组成：

- Win32 子系统支持控制台（文本）窗口、创建和删除进程与线程、16 位 DOS 虚拟机进程的部分。
- 核心模式设备驱动程序（WIN32K.SYS）中的：窗口管理器控制窗口显示；管理屏幕输出；收集来自键盘、鼠标和其他设备的输入信息；以及将用户信息传送给应用程序。
- 图形设备接口（GDI），一个用于图形输出设备的函数库，它包括了线条，文本、绘图和图形操作函数。
- 子系统动态链接库（例如 WSER32.DLL、ADVAPI32.DLL、GDI32.DLL 和 DERNEL32.DLL），它调用 NTOSKRNL.EXE 和 WIN32.SYS 将文档化的 Win32API 函数转化为适当的非文档化的核心系统服务。

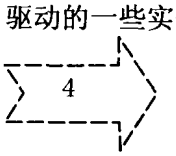
（2）POSIX 子系统。POSIX 代表了 UNIX 类型的操作系统接口的国际标准集，它鼓励制造商实现兼容的 UNIX 风格接口，这样，编程者就会很容易的将他们的应用程序从一个操作系统移到另一个操作系统。Windows2000 只实现了 POSIX.1 标准（ISO/IEC9945-1 1990 或 IEEE POSIX 1003.1-1990）。

（3）OS/2 子系统。单用户多任务系统，它在 Windows2000 中，仅支持 X86 系统以及基于 16 位字符的 OS/2 1.2 或视频 I/O 应用程序。

（4）这里还要提一提 NTDLL.DLL。它实际上包含了用户模式中的和核心模式中的一部分。可以说它是一个很特殊的系统支持库，主要用于系统动态链接库。NTDLL.DLL 包含两种类型的函数：执行支持所提供的系统服务调度定位程序；子系统、子系统动态链接库以及其他本地映像使用的内部支持函数。其中，第一组函数提供了可以从用户模式调用的作为 Windows2000 执行支持系统服务的接口。这些函数的大部分功能可以通过 Win32API 访问。对于这些函数中的每个函数，NTDLL 都包含了一个有相同名称的入口点。在函数内的代码含有体系结构专用指令，它能够产生一个进入核心模式的转换来调用系统服务调度程序。在进行一些验证后，系统服务调度程序将调用包含在 NTOSKRNL.EXE 内的实例代码的实际的核心模式系统服务。NTDLL 也包含许多支持函数，例如映像加载程序、堆栈管理器和 Win32 子系统进程通信函数以及通用运行时库例程。此外，它还包含了用户模式异步过程调用调度例程和异步调度例程。

### 2. 用于逻辑的、虚拟的、或物理设备的核心模式驱动

这些驱动是执行程序的一部分。在 Windows2000 中，一些驱动也是 WDM 驱动。WDM 的英文是 Windows Driver Model（Windows 驱动模型）。它是一些在 Windows 系统中，构建驱动的一些实例化的概念和思路，其中有很多的结构，半结构的东西。通过对这些结构进



行适当的调整，可以很快的开发出很好的驱动程序来。所有的 WDM 驱动都是 PnP 驱动，而且都支持电源管理。WDM 驱动跨越了 Windows98 与 Window2000 操作系统，从系统的角度看，它是资源兼容的。在后面的几章中，还会出现一个叫遗留驱动的，它的意思是为以前版本 WindowsNT 而写的驱动。而且不支持 PnP。

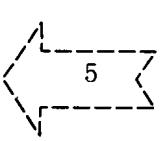
核心模式驱动分为 3 个基本类型，每个类型在结构和功能上都不一样。

最高层驱动（Highest Level Drivers），如：系统提供的 FAT,NTFS，和 CDFS 文件系统驱动（简称 FSDs）。FAT 是文件分配表。由操作系统建立，用来在记录媒体上分配文件存储空间，并提供查找手段的一种表。在 DOS 中，一个文件往往存放于多个不连续的区域（称为碎片）中，这些碎片之间采用链表结构组织在一起，链表的指针保存在文件分配表中。磁盘空间以簇为单位进行分配，每个簇在文件分配表中为一个项。对于 5.25 英寸软磁盘，每个簇为 1KB，在文件分配表中占用一个 12 位（或 16 位，根据盘上包含的簇数来定）的项。为了可靠，一张磁盘上建立两份相同的文件分配表。如果此表被破坏了，在操作系统中将无法读出原有文件的内容。NTFS 是设计用于 WindowsNT 操作系统的文件系统.与以前的 Windows3.x 版本下的文件系统相比，它具有以下特点，①支持长文件名，最长 255 个字节。②大范围的共享文件许可权。③可建立事物处理日志。④支持 OS/2（单用户多任务操作系统）使用的文件分配表和 HPFS 系统。CDFS 的意思是 CD-ROM 文件系统。

换个角度说，文件系统驱动（File System Driver，简称 FSD）可分为本地 FSD 和远程 FSD。本地 FSD 允许用户访问本地计算机上的数据；远程 FSD 允许用户通过网络访问远程计算机上的数据。本地 FSD 包括 Ntfs.sys，Fastfat.sys，Udfs.sys，Cdfs.sys 和 Raw FSD 等等。本地 FSD 负责向 I/O 管理器注册自己。当开始访问某个卷时，I/O 管理器将调用 FSD 来进行卷（磁盘分区）识别。远程 FSD 则由两部分组成，客户端 FSD 和服务器端 FSD。客户端 FSD 允许应用程序访问远程的文件和目录。对于 Windows2000 而言，客户端 FSD 是 LANMan 重定向器，而服务器端是 LANMan 服务器。重定向器是通过端口 / 小端口驱动协作体来实现的，其中端口驱动实现为驱动程序函数库，而小端口驱动则是特定函数或例程来实现的服务功能子集。Windows2000 是通过互联网文件系统（CIFS）协议来实现重定向与服务器之间的通信的。CIFS 是微软服务器消息段（SMB）协议的改进版。

最高层驱动始终需要下面的更低层驱动的支持。无论一个特定文件系统驱动是否会获得一个中间层驱动支持，任何一个文件系统驱动最终都依靠一个或多个低层外围设备驱动的支持。

中间层驱动如虚拟磁盘，镜像，或设备类型指定的类驱动。虚拟磁盘的是指在某些系统中，由操作系统管理物理（实际）磁盘或磁盘组的部分逻辑子区。这些逻辑子区都有自己的虚拟设备，它好像就是一个独立的磁盘。镜像是指对一个数据库内容的精确拷贝。在网络文件服务器上，为了提高数据的可靠性，有时需要建立它的镜像，这样，当原数据因故障而被损坏时，就可从镜像中得到恢复。设备类型指定的类驱动是某类设备的驱动主体上下文，就是该设备类型所指定的类驱动。这里的“类”与 C++中的“类”不同，这里，它是对一个或多个具有相同属性和服务集的对象以及如何在这个对象上创建新对象的一种



描述。而 C++中的“类”是在面向对象程序设计中，用来描述对象的一个术语，一个类就是一个样板，用来定义一组属性或操作，来表示这个类的特性。

中间层驱动也依靠来自下面的更低层驱动的支持。PnP 功能驱动用来控制那些由 PnP 硬件总线驱动控制的 I/O 总线上的，指定外围设备。如 PnP 过滤器驱动，它们在（为任何外围设备而指定的）驱动栈中，把自己插入到 PnP 功能驱动的上层或下层。一个 PnP 功能和过滤器驱动的子集，也是 WDM 功能和过滤器驱动。任何系统提供的，输出一个系统定义的 WDM 类 / 小类接口的类驱动，都是与一个或多个 WDM 小类驱动（也叫 WDM 小驱动）相连的一个中间层驱动。每个与其相连的 WDM 类 / 小类驱动协作体都提供了与 WDM 功能驱动或 WDM 软件总线驱动相同的功能。PnP 软件总线驱动代表了一组子设备。那些能与这些子设备相连接的，静态的更高级别的类，功能，或过滤器驱动也是中间层驱动。所谓协作体是指，具有相同、相似、有时甚至可能是相反特性的两个元件/组件等，合起来组成的一个相互协作的单位。

最低层驱动是直接与硬件打交道的驱动。如用来控制一些与外围设备相连接的 I/O 总线的 PnP 硬件总线驱动。最低层驱动，不依靠较低层的驱动，它是用来控制一个物理外围设备的，如一个总线。

Windows2000 网络驱动可以被划分为以上基本类型之一。比方说，一个 NT Server 或 Redirector 可以被指定为一个文件系统服务器；任何在传送栈中的驱动都是一个中间层驱动；一个物理网卡（也叫网络接口控制器或 NIC）驱动就是一个底层驱动。

实际上，如果从实践出发来看待驱动链，大体结构如图 1.2 所示。

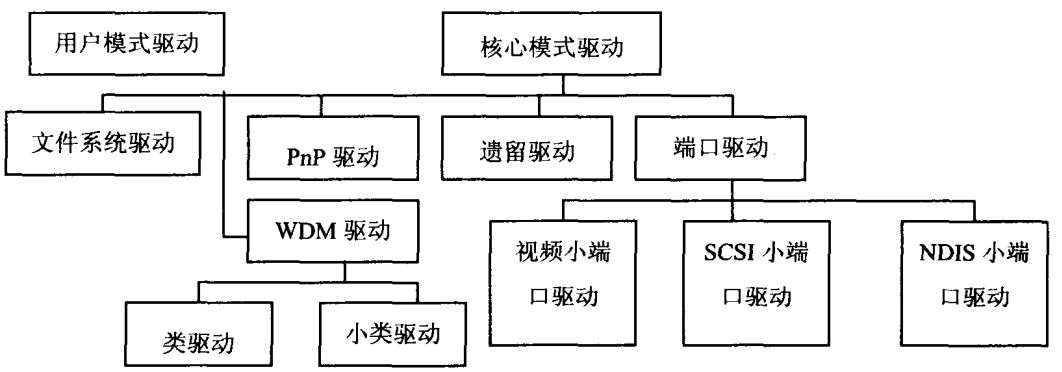



图 1.2　驱动链示意图

图中的 NDIS 驱动是为网络设备提供的驱动。

在 WDM 驱动模型中，每个硬件设备至少有两个驱动，功能驱动和总线驱动功能驱动，即人们常说的硬件设备驱动程序。它了解硬件工作的所有细节，负责初始化 I/O 操作，处理 I/O 操作完成时所产生的中断事件，并为用户提供一种适当的设备控制方式。总线驱动，它负责管理硬件与计算机的连接。实际上总线这个词是一个泛称。它可以是 PCI 总线，SCSI 卡，并行口，串行口，USB 集线器等等。它可以是任何能插入多个设备的硬件设备。

# 第 2 章　对 I/O、I/O 对象和 IRPs 进行分层

既然完成所有的运行都需要驱动，那么驱动对数据的传递将是整体运行的关键之一。我们前面提过了，核心模式的数据传递是基于“包”的，这个“包”就是输入 / 输出请求包，简称 IRP。

首先我们要以 IRP 为线索来看一看驱动链的工作过程，如图 2.1 所示。

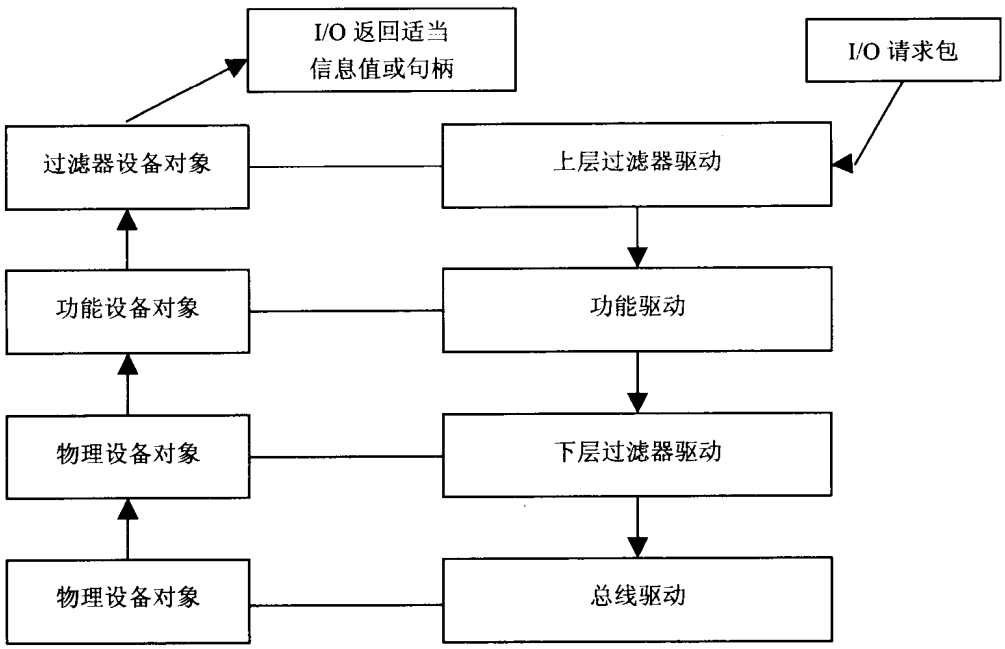



图 2.1　驱动链示意图

对于 I/O、IRPs 和 I/O 对象的分层是理解驱动链处理过程、请求包处理过程的关键。可以说对它们的理解是所有驱动编写者必须知道的重中之重。

I/O 的意思是输入 / 输出。IRP 的意思是输入 / 输出请求包。

每个运行系统都有一个隐含的或外在的 I/O 模型来处理数据在外围设备的进出。总的来说，在 Win2000 中的 I/O 模型有以下几个特点：

（1）I/O 管理器代表了一个对所有核心模式驱动都可用的一个一致的接口，包括最低层的，中间层的，和文件系统的驱动。所有到驱动的 I/O 请求都作为一个 I/O 请求包（IRPs）来传递。

（2）I/O 运行是分层的。I/O 管理器输出 I/O 系统服务，而用户模式保护子系统代表了它们的应用程序和 / 或最终用户，调用它们完成 I/O 操作。I/O 管理器解释这些调用，创建一个或多个 IRPs，并以分层的驱动为路径将这些 IRPs 传递到物理设备。

（3）I/O 管理器定义了一组标准例程。一些是任何驱动都要使用的，一些是可选的。所有驱动都可以按照一个相对稳定的应用模型来给出在不同外围设备的不同特征和由总线、功能、过滤器、和文件系统驱动所要求的不同功能。例程也叫例行程序，是一种可在程序中多

次调用，完成相同操作功能的指令序列。将这样的指令模块序列组成具有独立性的模块，可赋予一个符号名称，并规定此指令序列执行时所需的形式参数。按名称调用例程时，以实际参数替代形式参数，即可得到运算结果。合理利用例程，可减少程序设计的重复工作量。

（4）Win2000 驱动也是基于对象的。在系统中的驱动、设备、和系统硬件都被看作是对象。I/O 管理器和其他的操作系统部件都通过调用核心模式例程来完成特定对象的操作。

再加上通过 IRPs 来传送 I/O 请求，I/O 管理器与 PnP 管理器和电源管理器一起配合工作来发送包含 PnP 和电源请求的 IRPs。

通用的 IRP 结构示意图如图 2.2 所示。

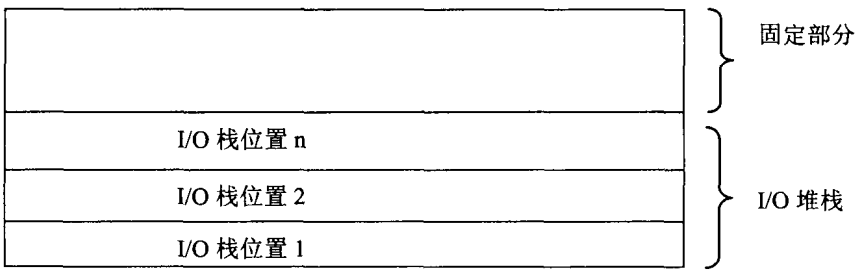



图 2.2　IRP 结构示意图

可以说，Windows 2000 的 I/O 体系是一个包驱动的系统，在这个系统中，每次的 I/O 操作，都是一个 IRP 的描述，传递，管理操作。从编程的角度看，IRP 是 I/O 管理器在相应一个 I/O 请求时，从非分页系统内存中分配的一块大小可变的数据结构内存，I/O 管理器每受到一个来自用户的请求就创建一个该结构，并将其作为参数传递驱动程序的适当例程中。

每个 IRP 都可以被看作由两个部分组成，固定部分和一个 I/O 堆栈。IRP 的固定部分包含了关于请求的信息。I/O 堆栈则包含了一系列 I/O 栈位置。可以说，该 IRP 经过了多少个驱动，就要有多少个栈位置。在栈位置中，一般存放的是处理 I/O 请求的驱动、被处理的对象等的函数指针和适当参数。而 IRP 的固定部分，是一个数据结构。总体来说它是相对稳定的。

下面我们简单介绍一下固定部分的主要结构。

（1）MdlAddress，指向一个内存描述符表（Memory Descriptor List，简称 MDL）。当驱动使用直接 I/O 操作时，MDL 用来描述一个与该请求相关联的用户模式缓冲区。

（2）Flags，包含一些对驱动程序只读的标志。该域通常仅仅是文件系统所需要的标志，这个标志与 WDM 驱动程序无关。

（3）AssociatedIrp，该域是一个三指针的联合。其中，与 WDM 驱动相关的指针是 AssociatedIrp.SystemBuffer。如果设备执行缓冲 I/O，则 SystemBuffer 指针指向系统空间缓冲区，否则为 NULL。

（4）IoStatus，是一个仅仅包含两个域的结构，驱动在最终完成请求时设置这个结构。当 IoStatus.Status 域收到一个 NTSTATUS 值，并且 IoStatus.Information 的类型为 ULONG_PTR 时，它将收到一个信息值，该值的确切含义取决于具体的 IRP 类型和请求完成的状态。

（5）RequestorMode，取值为一个枚举常量 UserMode 或 KernelMode，指定请求初始

化的模式为用户模式或核心模式。驱动程序有时需要查看这个值来决定是否信任某个参数。

（6）PendingReturned，该域是一个 Boolean 值，如果为 TRUE，则表明处理该 IRP 的最低级调度例程返回了 STATUS_PENDING。完成例程通过参考该值来避免自己与其它调度例程间竞争资源。

（7）Cancel，该值为 Boolean 值，如果为 TRUE，则表明 IoCancelIrp 已经被调用，该值用于取消请求。

（8）CancelIrql，是一个 IRQL 值，表明那个专用的取消螺旋锁是在这个 IRQL 上获取的。当驱动在 Cancel 例程中释放螺旋锁时会参考这个值。

（9）CancelRoutine，指向驱动程序取消例程的地址。必须使用 IoSetCancelRoutine 来设置 CancelRoutine 域，而不能直接修改。

总体上看，系统中处理每个请求都是按照以下关系来运作的。

图 2.3 阐释了最终用户、一个子系统和 I/O 管理器之间运作关系。

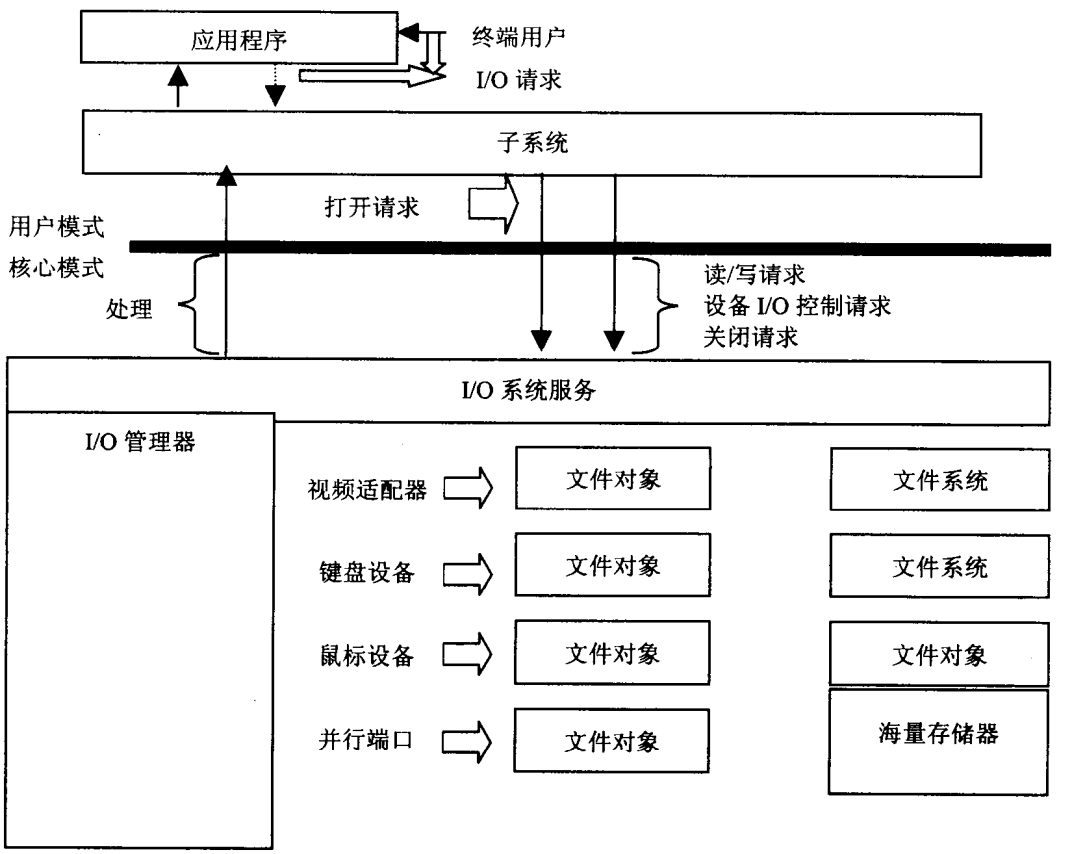



图 2.3　用户与子系统和 I/O 管理器示意图

I/O 管理器提供了一个 I/O 模型，一组核心模式支持的例程（驱动可以用这个例程进行 I/O 操作），和一个稳定的接口（在 I/O 请求的始发者和要回应这个 I/O 请求的驱动之间）。如上图所示，一个子系统和在子系统上运行的本地应用程序，只要运用由 I/O 管理器所提

供的文件对象句柄，就可以进入一个驱动设备或在海量存储设备上的一个文件。如果，要为打开这样一个文件对象，或为 I/O 到一个设备或数据文件而获得一个句柄，则一个子系统通过请求来调用 I/O 系统服务来打开指名文件。这个指名文件可以有一个指定子系统别名（(符号链接）在核心模式中的指定的名称）。在 I/O 管理器输出这些系统服务之后，要回应定位或产生一个文件对象（代表设备或数据文件），或定位一个适当的驱动。

图 2.4 显示了当一个子系统打开一个文件对象（以数据文件来代表一个应用程序）时所发生的全过程：

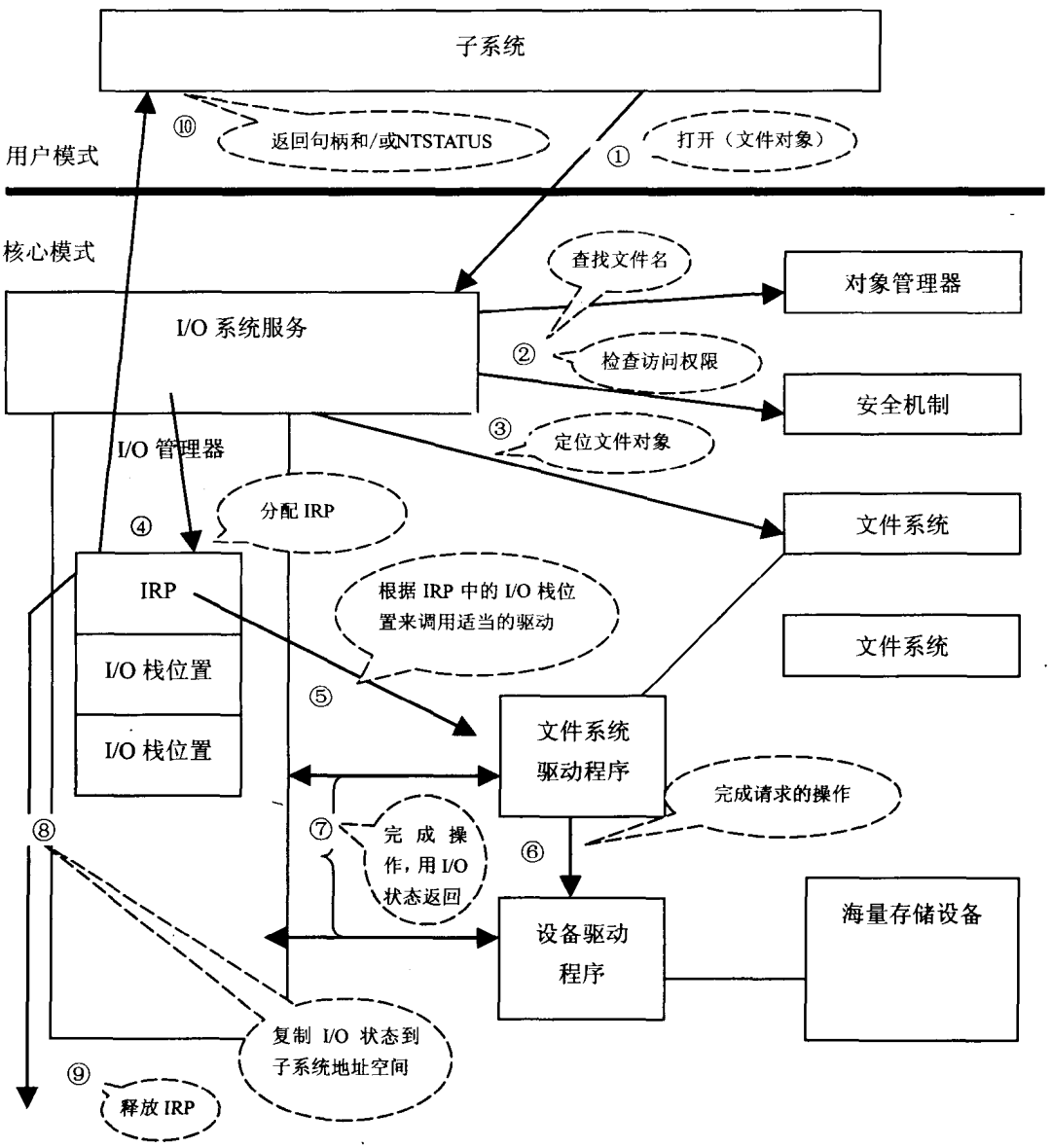



图 2.4　子系统打开文件过程示意图