Windows 内核实现的 个关键问题

王晓松　王明丽　著

东北大学出版社
Northeastern University Press

Windows 内核实现的 34 个关键问题

王晓松　王明丽　著

东北大学出版社
·沈　阳·

图书在版编目（CIP）数据

Windows 内核实现的 34 个关键问题 / 王晓松，王明丽著. — 沈阳 : 东北大学出版社，2017.2
ISBN 978-7-5517-1532-4

Ⅰ. ①W… Ⅱ. ①王… ②王… Ⅲ. ①Windows 操作系统－程序设计 Ⅳ. ①TP316.7

中国版本图书馆 CIP 数据核字(2017)第 030379 号

出 版 者：东北大学出版社
地址：沈阳市和平区文化路三号巷 11 号
邮编：110819
电话：024－83683655(总编室)　83687331(营销部)
传真：024－83687332(总编室)　83680180(营销部)
网址：http://www.neupress.com
E-mail: neuph@neupress.com
印 刷 者：沈阳航空发动机研究所印刷厂
发 行 者：东北大学出版社
幅面尺寸：185mm×260mm
印　　张：13
字　　数：316 千字
出版时间：2017 年 2 月第 1 版
印刷时间：2017 年 2 月第 1 次印刷
责任编辑：刘　莹　曲　直
责任校对：周　朦
封面设计：潘正一
责任出版：唐敏志

ISBN 978-7-5517-1532-4　　定　价：42.00 元

献给我的妻子小宋同志
和儿子王梓明小朋友！

——王晓松

前 言

20 世纪初，著名数学家希尔伯特提出了 23 个数学难题，希望在 20 世纪能够被解决，虽然到目前为止仍有一些问题没有得到解决，但正是对这些问题的不懈研究，不仅推动了相关数学分支的蓬勃发展，而且开拓出不少新的研究领域。“设立问题—寻求解答”的发展模式也同样适用于对 Windows 内核的学习。

笔者一直很想弄明白 Windows 内核的奥秘，从 2002 年开始，从没有放弃过对 Windows 内核资料的阅读。十几年来，倒是有了不少对 Windows 内核的理解。对于很多内容，刚开始有很多不明白之处，甚至是错误的地方，反反复复地翻阅资料，在脑海中还原系统实现的细节，不断地提问、追索，有时候仿佛进入了迷宫中的死胡同，毫无头绪可言，它是这么做的，可为什么要这么做呢？终于有一天，有那么一刻，豁然开朗，仿佛它的奥秘被完整地呈现在眼前，纤毫毕见，心中有种欣欣然的感觉。

现在回头看，在最开始学习 Windows 内核时，心中存在着 N 多困惑：

① 如何在脑海中构造一个清晰简洁的 Windows 内核框架？

② 对象是如何扮演好它基础性机制的角色？编程人员熟悉的句柄与对象又有怎样的联系呢？

③ 生活中意外难免，Windows 的异常情况也经常出现，Windows 的异常机制是如何从容不迫地处理好各种特殊情况的呢？

④ Windows 服务是幕后英雄，英雄是如何炼成的呢？

⑤ 消息是怎样成为窗口显示机制的灵媒？

⑥ 消息钩子既可以做些好事，也可以做些坏事，怎么做的呢？

⑦ Windows 独有的注册表机制是咋回事？

⑧ 权利之争也存在于 Windows 内核中，系统调用是如何实现特权升级的？

……

经过学习总结出 34 个问题，就像 34 个脚印，引领笔者迈进了深入理解 Windows 内核的殿堂，如何能帮助那些对 Windows 内核充满好奇，却踯躅不前的学友迈出前进的第

一步，并逐步地登堂入室、得见大观，既是写作本书的初衷，也是目标。

考虑到 Windows 内核的内容庞杂，初学者往往在扑面而来的概念、多如牛毛的术语、面目狰狞的代码面前望而却步，本书尽量以生动的语言、丰富的图表和贴近实际的例子，拉近 Windows 内核与读者之间的距离。当然，本书并没有能将 Windows 内核所有内容讲清楚的野心，而只是尝试着从整体的角度向读者介绍 Windows 内核实现的一些基本思想，试图让读者对 Windows 内核的实现有一个整体的框架。本书主要集中讲解 Windows xp 系统，希望读者能够以此为基石，通过自己的学习，了解其后各种 Windows 系统的实现。

除了本书说明的参考文献外，作者还参考了微软发布的 WRK1.2，以及网络上的 ReactOS（一个类似 linux 的草根模拟 Windows 操作系统），在此致谢！

有一本很有名的科普读物，叫作《从一到无穷大》，是 20 世纪 70 年代的作品，宏观到宇宙，微观到原子，将当时很多最新的科学进展、看起来深奥的问题，娓娓道来，用生动的语言、漫画般的图解讲解得鞭辟入里。我上初中时就拜读过这本书，十几岁的孩子读起来，对问题的理解不费力，还特别有兴趣去思考解决问题之道，前段时间上网特意买了一本，拿到手中的那一刻，仿佛和一个老朋友相见。书的译者序言中有一个例子，一个教授家有这本书，小保姆只有小学文化，却特爱翻看这本书，结果后来小保姆回了家乡，书也不翼而飞……书写到这种境界，也只能是“虽不能至，心向往之”。

作　者

2016 年 11 月

目 录

第 1 章　Windows 内核系统综述

1.1　一个清晰简洁的 Windows 内核框架

问题 1：*如何在脑海中构造一个清晰简洁的 Windows 内核框架？*

从创立到现在，微软公司一直坚持“以问题为导向，以解决实际问题为抓手”的原则，取得了几十年的持续发展，并一直保持着操作系统界的垄断地位。对于操作系统，大家并不陌生，但操作系统是如何实现的，却不是每个人都清楚，尤其是 Windows 这种非开源的操作系统，虽然被如此广泛地应用，也是经过几十年来无数的大牛对其进行逆向分析，以及伴随着微软公司逐步地开放老一些版本的源代码，人们才得以对 Windows 内核的实现有了较为全面的了解。

既然 Windows 是一种操作系统，那么其解决的问题自然也脱离不开操作系统的那些经典内容，如进线程管理、内存管理、硬件管理、同步等经典问题，同时 Windows 在具体实现上，也有与其他操作系统不同之处，如使用了对象管理器作为系统各组件的基础机制，独有的注册表机制储存了各种系统和用户数据，使用了多种手段来保证用户使用的便捷性、舒适性等。

Windows 系统虽然博大精深，对很多人来说是一个神秘的领域，但是其很多内容实现的基本思想并不深奥，和我们生活中的一些处理方法非常相似，下面先看看什么是操作系统。

抛开严肃刻板的定义，可以认为操作系统是一个承上启下的中间层，其下层是稀奇古怪的各种各样的硬件，其上提供美观便捷的用户界面，供对计算机几乎一无所知的用户使用。硬件、操作系统和用户的关系如图 1 - 1 所示。

“万丈高楼平地起”，既然操作系统是建立在硬件之上，那么首先看看电脑的硬件配置。通常，每个电脑都有一块主板，主板上有 CPU、内存、硬盘、网卡和显卡等典型器件，与之对应，操作系统会有相应的模块负责维护这些硬件，如图 1 - 2 所示。图 1 - 2 列出了一个电脑上的典型硬件配置，以及操作系统中与之对应的模块。

打开过电脑机箱的读者对电脑的硬件并不会陌生。CPU（主处理单元）主要负责运算功能，经常听说的几核处理器指的便是有几个 CPU；对于内存，游戏玩家的感触最深，内存的大小是决定游戏感受最为关键的因素，内存主要负责数据的临时存储，电脑掉电，内存中的数据会丢失，CPU 处理的数据基本都是直接针对内存的；硬盘可以一直保留数据，无论系统是否掉电，而且它们的容量很大，要比内存大好几个数量级；除了

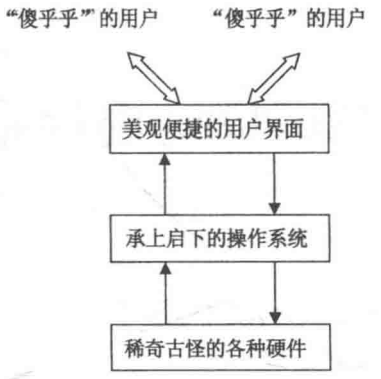

图 1-1　操作系统的角色

CPU、内存、硬盘三大件，主板上还有各种各样的外部硬件，如网卡、声卡、USB 接口等。电源是维护整个系统正常运行的基础，家里停电会很别扭，而主机没有电源便会无法运行。操作系统面对这些硬件大佬和林林总总的马仔，其功能就是将它们管理起来，实际上与硬件相对应，操作系统也可以大致分为以下模块：进线程管理（CPU 对应）、内存管理（内存对应）、I/O 管理（硬件对应），以及无处不在的安全机制，这里将 I/O 管理分为即插即用管理器、I/O 管理器、电源管理器三部分。针对这六部分内容，下面分别列出一些有意思的模块实现，希望有助于读者对 Windows 内核的初步理解。

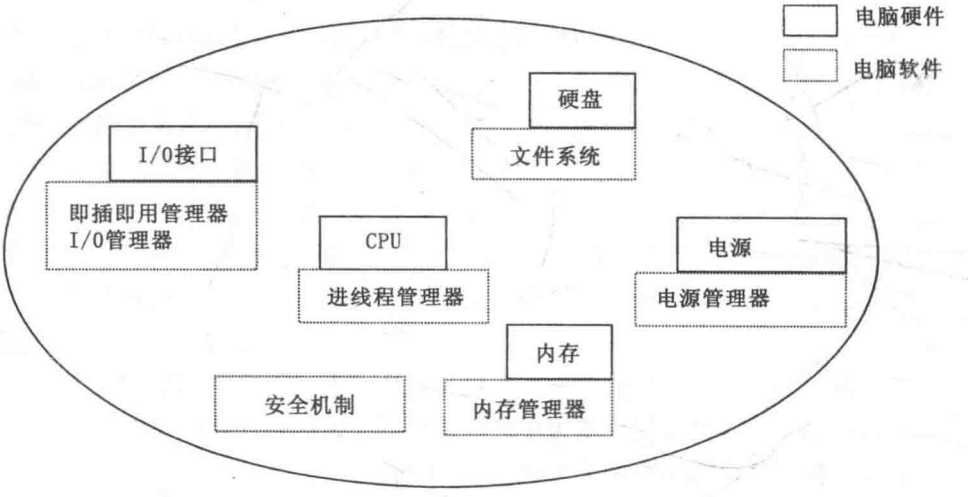

图 1-2　电脑硬件与操作系统构件的关系

1.1.1　CPU 相关

计算机只有一个 CPU，但是我们却可以一边写文档、一边听歌曲，同时干许多事情，其实现的奥秘在于虽然每个时点上只有一个任务在运行，但是经过不同的任务不停地以人无法意识到的速度切换，从而在整体上体现出多个任务同时运行的表象。操作系统一个重要的任务就是不停地决定让哪个任务运行，在何时运行，以及怎样运行的问题。

任务是一个笼统的概念，在 Windows 系统中，任务可以细化为进程和线程。进程这

个概念我们并不陌生，打开“任务管理器”，我们会看到当前系统中运行的进程，可以说，不同的任务使用了不同的进程，编辑文档的 Word 程序是一个进程，电影播放器是一个进程，上网使用的浏览器是一个进程，不同的需求使用不同的进程。

通常，一个进程的需求是一致的，但是也会有不同的分工，比如一个包含图形界面与复杂运算的程序，如果使用一个执行实体，那么当后台处理复杂运算时，有用户进行界面的操作，这个操作就会有明显的迟滞，甚至没有反应。如果将整个模块分为显示是一部分，后台程序处理又是一部分，分别使用不同的执行模块进行处理，线程就登场了。显示线程优先级高些，首先满足用户的需求；后台线程优先级低些，可以处理一些较为复杂的、慢的计算。

那么，进程和线程有什么区别呢？有的书提到，线程是轻量级的进程，这种说法在 Windows 中并不准确，因为它们之间完全不能类比。进程是一个整体、是一种环境，它包含运行的实体——线程，以及进程的内存空间、全局变量等；而线程存在于进程中，并且是专门作为运行实体存在的。

Windows 系统对运行实体的调度是以整个系统的所有线程为基本单位的，而不是以进程为单位，那么，它采取的调度策略是什么呢？每个线程并不平等，也分为三六九等，每个线程都有优先级。有些线程（如界面显示）为了照顾用户的感受，优先级高；而有些线程（如整理硬盘碎片）并不紧急，那么优先级就可以低一些。优先级高的先执行，优先级低的后执行。同一个优先级的线程连接成一个链表，由系统轮流调度，低优先级的线程只有在没有高优先级线程等待调度的条件下，才能够得到被执行的机会。当然，根据实际需要，每个线程的优先级是可以进行动态微调的，如图 1－3 所示。

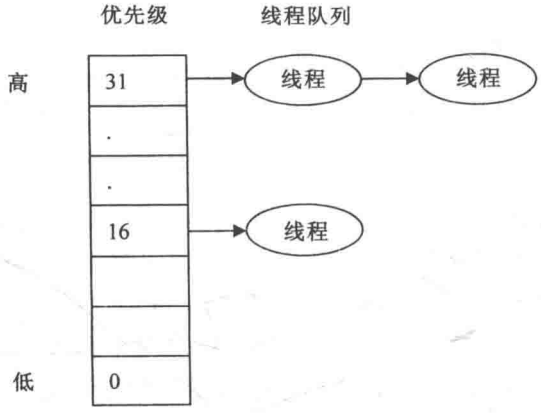

图 1－3　线程的调度策略说明

1.1.2　内存相关

对于内存的管理，按照 CPU 的处理方式，可以分为实模式和保护模式。在实模式情况下，所有的进程共用物理内存，一个进程可以访问其他进程的内存数据。保护模式是将每个进程的空间进行隔离，虽然最终使用的都是物理内存，但是每个进程的空间独立，不能互相访问。显然，保护模式的处理要较实模式复杂，那么，这么做的目的是什

么呢？下面举一个生活中的例子。

情景 1：银行保险库的寄存柜统一编号，小王在银行有五个寄存柜，分别是保险库的第 1，5，7，9，11 号柜，取东西时，他会告诉银行保管人员“麻烦取下第 11 号柜的物品”。

这种寄存柜的管理方式，对于用户来说，可能会存在以下两个问题：

① 用户记号码很费劲，枯燥的没有规律的数字 1，5，7，9，11 很容易记错。

② 有些用户的柜子不用，但银行也必须无条件地为该用户保留。

情景 2：精明的银行家想出了一个好办法，如图 1－4 所示，银行保险库的寄存柜仍然统一编号，但是对用户并不公开，给每个用户的卡号都是［1－10］号专用，用户只要有业务，不用记长长的、无规律的号码，由管理员负责找到其名下用户需要的柜子。当然，对于管理员来说，他很清楚从用户的柜号到实际上保险库柜号的映射关系。而且对于银行家来说，柜子可以得到尽量高效的使用，可能一个用户拥有的十个柜子，只有五个柜子被使用，那么银行家完全可以先分配五个柜子，其余五个等用时再分配，这样，柜子的使用效率也得到提高。

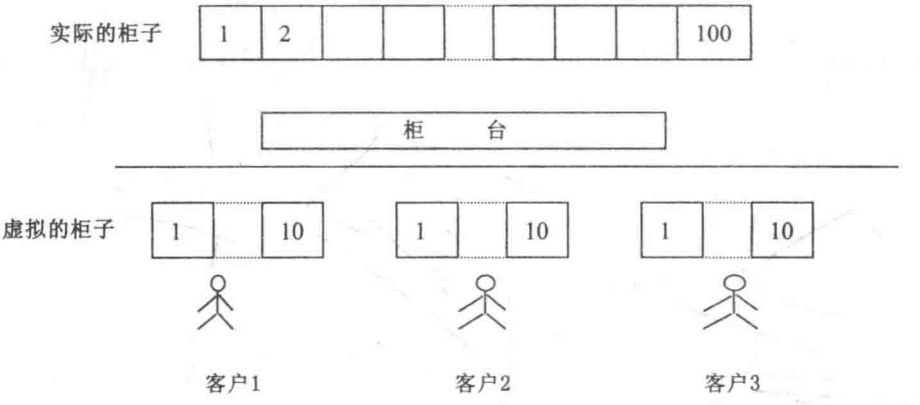

图 1－4　使用保护模式例子

跨过实模式，使用保护模式的原因与之类似。可以将每个进程想象成一个用户，柜子对应着物理内存，实模式就对应着情景 1，存在的问题也非常相似，内存的管理相当于由每个进程单独管理，每个进程除了实现自身的逻辑外，还要考虑使用内存地址时不要和其他进程产生冲突，不要误写入其他进程，这显然对于单个进程来说复杂，对于整个系统来说也是不稳定且低效的。若某个进程申请了较大的空间，那么对这段地址的使用是独占的，即使并不使用，也不能分配给别的进程，效率低。依照上面的思想，解决的办法也是显而易见的，给每个进程分配一个独立的地址空间，即每个进程都有一个虚拟的地址空间（对于 32 位的 CPU 来说，每个进程的地址空间为 $2^{32}=4G$），至于虚拟地址到物理地址的映射，用一个被称为 MMU（内存管理单元）的专门部件来实现，就会避免实模式使用中的问题，每个进程将精力放在处理自己的任务，虚拟地址到物理内存的映射及物理内存的管理由操作系统负责就可以了。对于每个进程来说，精简了任务。对于系统来说，整个内存由系统统一管理，将有效地提高内存的管理效率，均衡地分配负载，带动了整体性能的提高。由此，保护模式诞生了。

无论是对于虚拟地址空间，还是对于物理地址空间，均使用页面（大小为 4K）作为内存管理的基本单位，因此，虚拟地址到物理地址的映射可以归结为虚拟地址空间的页面到物理地址空间页面的映射，以图 1 -5 为例，进程 1 虚拟地址空间的 1 号页面被映射为物理地址空间的 1 号页面，而 6 号页面被映射为 7 号页面。不同进程之间的地址空间隔离，如同为 1 号页面，进程 1 中的映射为物理地址空间 1 号页面，而进程 2 中的映射为物理地址空间 3 号页面，除非为了完成内存共享，一般来说，不同进程中虚拟页面后台映射的物理页面并不相同。

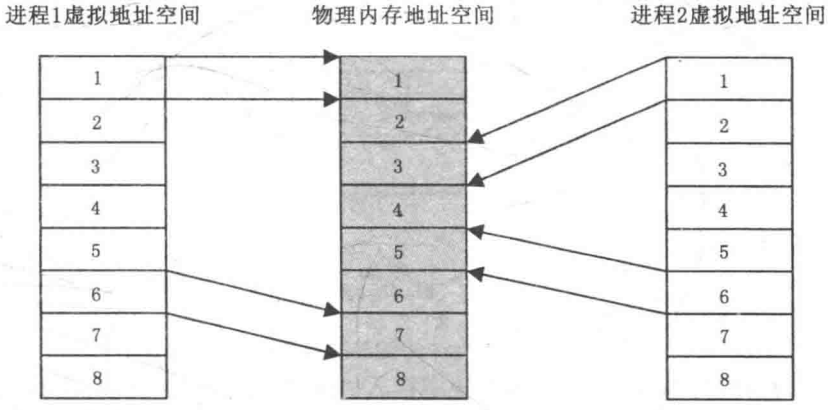

图 1 -5　保护模式下内存的映射

1.1.3　外设相关

U 盘我们都使用过，非常方便，将 U 盘插入 U 口，系统自动识别，然后文件管理器中出现一个新的盘符。U 盘只是 Windows 支持外设的一个典型例子，我们熟悉的网卡、显卡、硬盘等都属于电脑的外设。Windows 的外设管理有一套自身的体系，主要包括三剑客：即插即用管理器、I/O 管理器、电源管理器。

可以简单地将三者的关系理解为：即插即用管理器是硬件体系结构的人力资源部长，主要完成外部设备的加载、移除等操作，此过程并不需要用户手动地设置外设的端口、中断号、驱动程序等（当然，电脑中没有针对该设备的驱动程序，还是需要用户提供的）。I/O 管理器是行政部长，顾名思义，就是完成 I/O 管理功能的，当一台外部设备由即插即用管理器加载入系统后，应用程序及其他内核程序对该设备的操作（读、写、设置等）都需要经过 I/O 管理器。应用程序对外部设备的操作通常使用 ReadFile/WriteFile 函数，I/O 管理器将这个调用转化为 IRP 的形式传递给对应的驱动程序，同时 I/O 管理器将设备返回的信息传递回用户程序。电源管理器相当于后勤方面的能源部长，它不光管理主板上的接口电源，而是精细到每个带电源管理功能的芯片上，电源管理器的作用在移动设备上体现得特别明显，移动设备可以在不用时自动进入休眠模式，就会有效地减小电池电量的消耗。

（1）即插即用管理器

计算机即插即用的实现依托于硬件架构、即插即用管理器和硬件设备的协同配合，如果将计算机主板上的硬件抽象化，那么如图 1 -6 所示，整个硬件系统就是“总线 +

设备”级联组合而成的，总线包括我们熟悉的 PCI 总线、USB 总线、SCSI 总线、ISA 总线等，总线和总线之间通过桥接器级联，桥接器的功能就是完成总线协议之间的转换，在每条总线上，挂接着符合该总线标准的各种硬件设备。即插即用管理器的功能就是逐次扫描各条总线，枚举总线上的设备，从全局的角度分配中断号、I/O 空间、内存空间，并加载驱动程序。

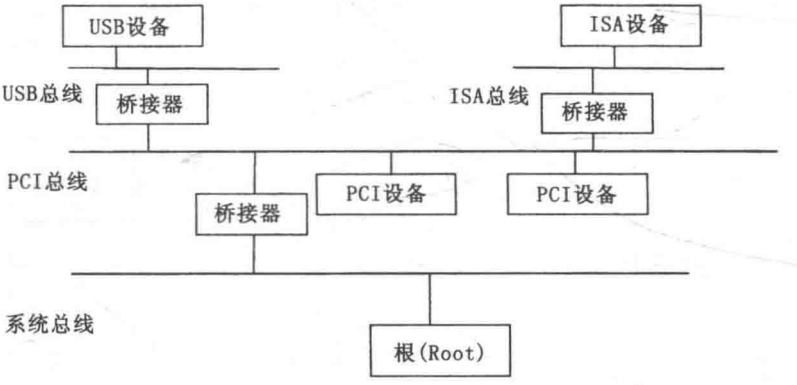

图 1-6　Windows 中的总线关系结构图

大多数人都会有这样的经历：在系统中接入一个新的设备，系统会自动地发现该设备，并尝试着加载驱动程序，老版本的 Windows 系统自带的驱动程序并不多，所以，通常会提示用户插入带有驱动程序的光盘。而现在的 Windows 系统附带了大多数常用设备的驱动，即使没有，也会连接到微软的服务器自动下载驱动。细心地读者可能会有这样的疑问：PCI 的设备有网卡、显卡、视频卡等诸多设备，那么 Windows 是如何知道应该加载哪个驱动的呢？

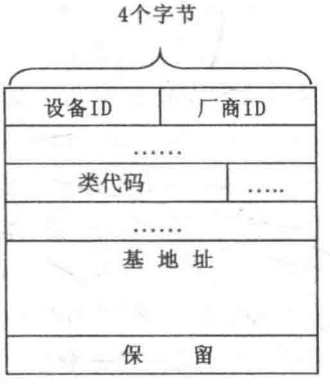

图 1-7　PCI 配置空间

道理在于每个 PCI 设备都会在固定的位置保留一个配置空间，如图 1-7 所示，在配置空间的最开始就是两个字段，厂商 ID（Vendor ID）和设备 ID（Device ID），每个字段 2 个字节，厂商 ID 指的是提供设备厂商的 ID，而设备 ID 指的是该厂商生产的这种设备的 ID，通常会说这是 BROADCOM 生产的 BCM5709 网卡，厂商 ID 就对应着 BROADCOM 公司，BROADCOM 公司生产的设备有很多种，而设备 ID 对应着 BCM5709

网卡这种设备，这两个标识的组合基本上可以帮助系统定位需要的驱动程序。

（2）I/O 管理器

顾名思义，I/O 管理器就是完成 I/O 管理功能的，当一台外部设备由即插即用管理器加载入系统后，应用程序及其他内核程序对该设备的操作（读、写、设置等）都需要经过 I/O 管理器，如图 1 - 8 所示。

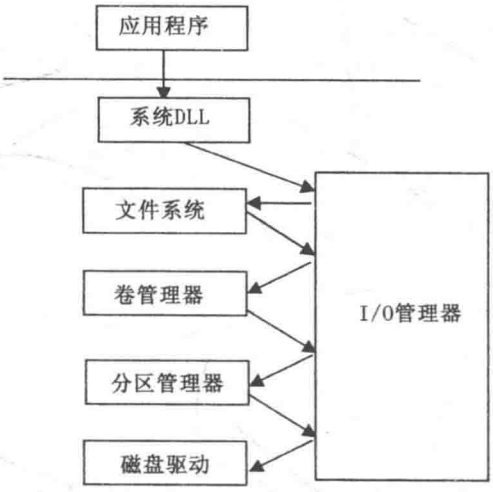

图 1 - 8　Windows 操作系统的存储结构

一块硬盘通常是由 512 字节大小的扇区组成，当我们拿到硬盘，通常会将其分区，比如将 C 盘作为系统区、D 盘作为工作区、E 盘作为备份区等。对于寻常用户的电脑来说，分区之上直接映射文件系统就可以了，卷的出现是为了有效地管理多个分区，比如电脑中有两块硬盘，一个卷可以包含不同硬盘中的两块分区，这样可以将存储的数据分配到不同的分区上，或者两个分区间互为备份，达到一定的灵活性。实际上，图 1 - 8 展示的是简洁的 Windows 存储结构，普通用户看到的是一个个结构清楚的目录和文件，一个文件到扇区的映射要经过文件系统、卷管理器、分区管理器、磁盘驱动器各个模块的协同配合方可完成，而各层之间的协调者就是 I/O 管理器。

应用程序对外部设备的操作通常使用 ReadFile/WriteFile 函数，I/O 管理器将这个调用转化为 IRP 的形式传递给对应的驱动程序，I/O 管理器会将 IRP 传递给文件系统，文件系统完成自身的功能后，调用 I/O 管理器中的 IoCallDriver 函数，变相地通过 I/O 管理器将 IRP 传递卷管理器，依次向下传递，直到磁盘驱动器。

通过图 1 - 8 也能看出 Windows 系统设计充分使用了“分层”的思想，将不同的功能模块分解到不同的层，每层向下看到的都是下一层向上呈现的数据结构，如文件系统对下是无法看到磁盘的扇区的，而只能看到卷管理器提供的 N 多的簇，而这些簇可能在不同的分区。

（3）电源管理器

由于智能手机耗电很快，因此大家对手机的电源管理会比较感同身受一些，但是在笔记本或者其他移动终端中，电源的管理同样非常重要。大家知道，电脑除了处于关闭状态和工作状态外，还会进入睡眠（sleep）、休眠（hibernate）、待机（standby）3 种状

态，这 3 种状态都属于中间态，在这种状态下，系统运行实时镜像被保存，并进入低耗电模式，因此，耗电相较工作状态要小，同时恢复到工作状态所使用的时间也较正常的由关闭进入工作状态的开机过程要快。

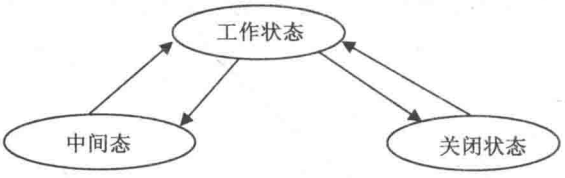

图 1-9　计算机工作的 3 种状态

这 3 种状态不是“眠”就是“待”，似乎很相似，那么，它们之间的区别是什么呢？请看表 1-1。

表 1-1

中间态	内存	硬盘	耗电情况	断电后果
睡眠	√	√	极低	从硬盘恢复
休眠		√	无	从硬盘恢复
待机	√		极低	系统重新启动

待机状态只在内存中保留镜像数据，系统启动，会免去加载映像的过程，启动速度会很快，但是如果断电，因为在硬盘中并没有镜像数据，那么只好重新启动系统。休眠状态是将内存镜像数据保留在硬盘上，那么可以说，进入这种状态后，系统完全不耗电；当退出这种状态时，直接从硬盘取镜像就可以直接运行了，但是时间会稍慢。为了弥补以上两者的缺陷，睡眠状态将内存镜像数据在内存和硬盘中各保留一份，仅仅维持内存的供电，正常恢复时，从内存取镜像即可，如果断电可以从硬盘中提取数据直接恢复。

当然，电脑中的电源管理并不局限于整个系统，而是精细到每个带电源管理功能的芯片上，比如一台设备的电源状态可以分为 D0 ~ D3 四个层次，根据不同的耗电需求，依次降低。D0 为正常使用状态；D3 为关闭状态；D1 和 D2 为中间使用状态，耗电情况介于 D0 与 D3 之间。

1.1.4　安全机制

安全机制现在越来越受到人们的重视，比如要进入一个严格保密的区域，通常要出示能证明身份的凭证。当然，每个人获得的权利并不相同。有的人是管理者，骑角旮旯哪里都可以去；有的人，比如像我这样普通的技术人员，管理者的办公室不是说去就能去的。Windows 系统使用的安全认证机制和我们生活中的情景非常类似。

上述情景可以简化为：需求和权限是否匹配。一个需求的内容可以包含两个方面：我是谁，我要做什么。在 Windows 系统中，最典型的应用就是进程对对象的访问，如图 1-10 所示，需求和权限仿佛被一堵认证之墙隔离。当一个用户创建了一个进程后，便将一个令牌赋予该进程，令牌中的内容说明了该进程属于的用户及用户组，当进程 1 需

要访问某一对象时，一般会说明执行什么动作，比如读、写、运行等，将令牌与动作结合，表明一个需求组合，即“我是谁，我要干什么”，如墙左侧所示。每个对象都会有一个 ACL（access control list，访问控制列表），这张列表说明的恰恰是“允许谁做什么动作”“禁止谁做什么动作”，如墙右侧所示。将墙两侧的内容进行对比，就可以判断出该动作是否被允许。

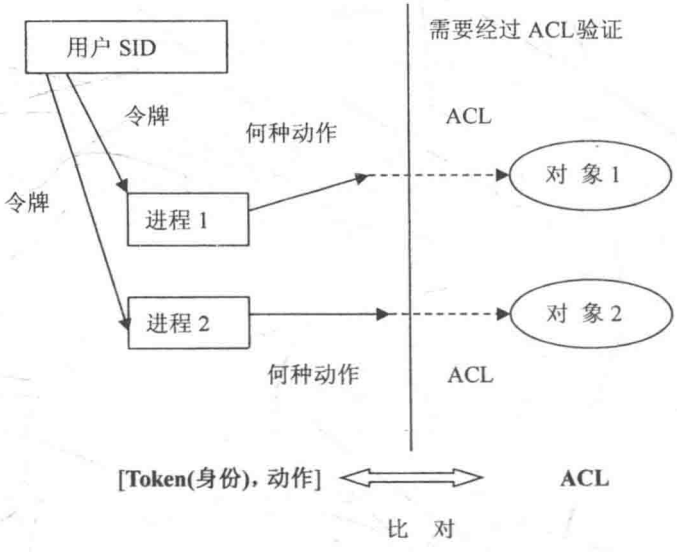

图 1－10　安全认证机制

1.1.5　一个清晰简洁的 Windows 内核框架

下面来看一个清晰简洁的 Windows 内核框架，如图 1－11 所示。应用程序对于各种 DLL 的调用基本都要归于以下三个 DLL 文件：Kernel32. dll、User32. dll、Gdi32. dll。其中，Kernel32. dll 控制着系统的内存管理、数据输入输出操作和中断处理；User32. dll 是 Windows 用户界面相关应用程序接口，如创建窗口和发送消息；Gdi32. dll 包含的是图形显示的函数。实际上，这 3 个动态链接库最终都要调用 Ntdll. dll 中对应的函数，由此进入内核模式。当进入内核模式后，会看到很多熟悉的身影，如进线程管理器、内存管理器、I/O 管理器等，通过前面的介绍，应该不会感到陌生。这些模块的下面是对象管理器，可以认为是上述各种组件的一个基础，因为各个组件的实现很多最终都可以归结为对对象的操作。那么，对象下面怎么又出现了一个内核？这个内核的重点在于实现更底层的进线程调度，以及异常处理、陷阱和各种中断，相对其上的各种组件，这个内核非常小，它的下面就是与硬件最亲密的硬件抽象层，硬件抽象层被操作系统用来屏蔽各种硬件的差别。

从图 1－11 可以看到，大部分内核模块最终都要调用对象管理器，可以说，对象管理器是上面 N 种机制的基础性机制，对 Windows 内核的介绍也从对象管理器说起。