# 第2章 Windows 2000/xp的体系结构

- 1 操作系统的设计
- 2 Windows2000/xp操作系统模型
- 3 Windows2000/xp体系结构
- 4 Windows2000/xp系统机制
- 5 Windows2000/xp注册表
- 6 WMI的体系结构

# 操作系统的设计

设计操作系统的复杂性。

例：IBM公司的OS／360系统

- 由4000个模块组成
- 共约100万条指令
- 花费5000人年
- 经费达数亿美元
- 每个版本都仍然隐藏着无数的错误

# 操作系统的设计问题

- 操作系统设计有着不同于一般应用系统设计的特征：
  - 复杂程度高
  - 研制周期长
  - 正确性难以保证
- 解决途径：
  - 良好的操作系统结构
  - 先进的开发方法和工程化的管理方法
  - 高效的开发工具

# 操作系统的设计目标

- 可靠性：正确性和健壮性
- 高效性：提高系统的运行效率
- 易维护性：易读、易扩充、易剪裁、易修改性
- 易移植性：作系统程序中与硬件相关的部分相对独立
- 安全性：计算机软件系统安全性的基础
- 可适应性
- 简明性

# 操作系统的设计考虑

- 功能设计：操作系统应具备哪些功能
- 算法设计：选择和设计满足系统功能的算法和策略，并分析和估算其效能
- 结构设计：选择合适的操作系统结构

# 操作系统结构

- 程序结构
  - 程序结构的两层含义
    整体结构
    局部结构
- 软件结构：大型程序是小规模程序组成
- 操作系统结构

# Windows 的术语“服务”

- 在不同的场合有不同的意义
- 可以指操作系统中可调用的例程、设备驱动程序或服务器进程

- 1. Win32 API函数：Win32 API中文档化的、可调用的子程序。例如CreateProcess、 CreateFile、GetMessage。
- 2 Windows 系统服务(执行体系统服务) 。例如，NtCreateProcess是由CreateProcess函数调用，用来创建新进程的内部系统服务。

- **3 Windows内部例程：**

位于Windows执行体、内核或硬件抽象层(HAL)内的子例程，只能从核心态调用。例如，ExAllocatePool是由设备驱动程序调用的

- **4 Windows服务：**

由Windows服务控制管理器启动的进程。

- **5 DLL(动态链接库)：**

作为二进制映像连接的、可调用的子例程集。

# 两种机器状态

- 用户态或者说目态

处于目态时为用户服务

- 系统态或者说核心态、管态

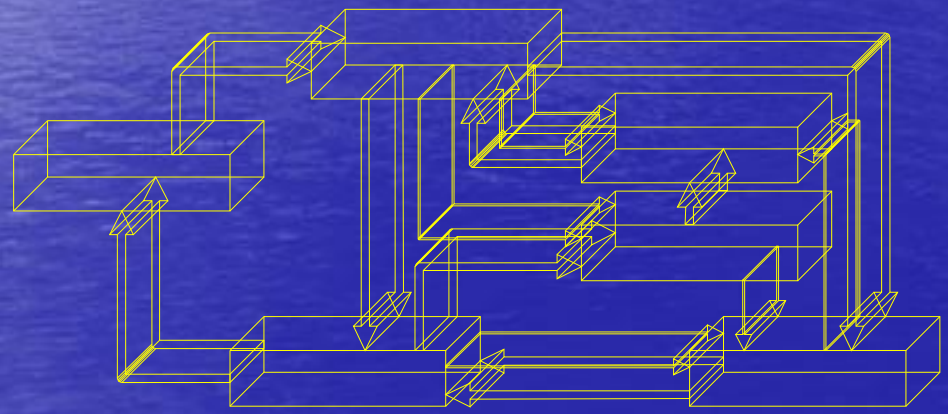
当其通过系统调用或访管指令进入到OS内核运行时，处于管态时可能为用户服务，也可能做系统维护工作。

# 操作系统的结构设计

- 1. 模块组合结构
- 2. 层次结构
- 3. 虚拟机结构
- 4. 客户/服务器体系结构

# 模块组合结构

整个系统按功能进行设计和模块划分。系统是一个单一的、庞大的的软件系统。这种结构思想来源于服务功能观点，而不是资源管理的观点。



# 模块组合结构

- 模块结构的特点：模块由众多服务过程（模块接口）组成，可以随意调用其他模块中的服务过程
  - 优点：具有一定灵活性，在运行中的高效率
  - 缺点：功能划分和模块接口难保正确和合理；模块之间的依赖关系，降低了模块之间的相对独立性——不利于修改

# 2. 层次结构

从资源管理观点出发，划分层次。在某一层次上代码只能调用低层次上的代码，使模块间的调用变为有序性。系统每加一层，就构成一个比原来功能更强的虚拟机。有利于系统的维护性和可靠性。

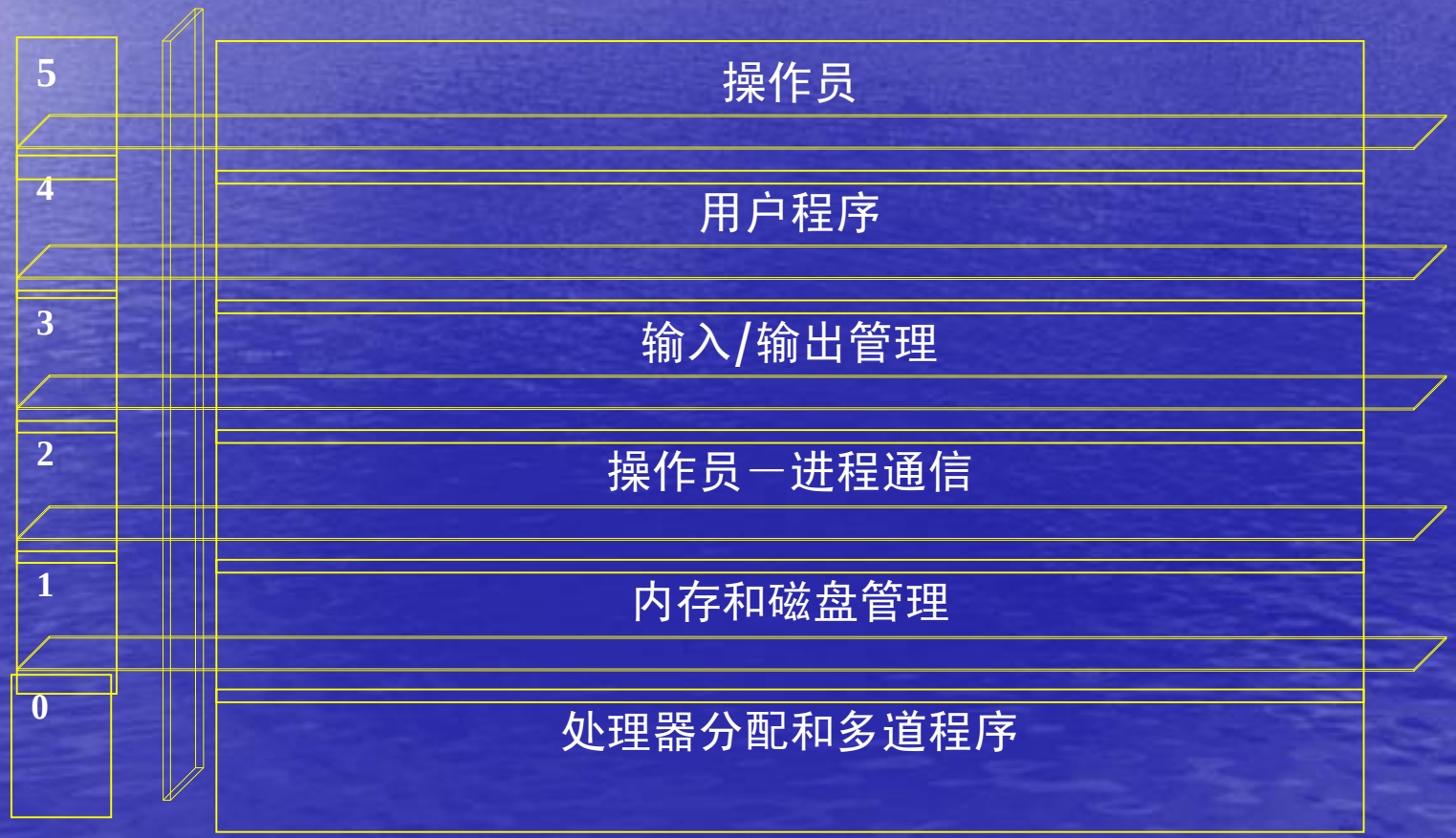
目的：要清除模块接口法的缺点就必须减少各模块之间毫无规则地相互调用、相互依赖的关系，特别是清除循环现象

方法：操作系统的所有功能模块按功能的调用次序分别排列成若干层 （单向依赖或单向调用 ）

如只允许上层或外层模块调用下层或内层模块)

# 层次结构

- THE系统：



# 分层结构的特点

- 优点：
  - 功能明确，调用关系清晰（高层对低层单向依赖），有利于保证设计和实现的正确性
  - 低层和高层可分别实现（便于扩充）；高层错误不会影响到低层；避免递归调用

# 分层原则

- 被调用功能在低层：如文件系统管理——设备管理——设备驱动程序
- 资源管理的公用模块放在最低层：如缓冲区队列、堆栈操作
- 存储器管理放在次低层：便于利用虚拟存储功能
- 最低层的硬件抽象层：与机器特点紧密相关的软件放在最低层。如Windows NT中的HAL
- 资源分配策略放在最外层，便于修改或适应不同环境

# 分层原则

便于将操作系统移植到其他机器上：

- 机器特点紧密相关的软件(如中断处理、输入输出管理等)放在紧靠硬件的最低层
- 与硬件有关的BIOS(管理输入输出设备)放在最内层。所以当硬件环境改变时只需要修改这一层模块就可以了

# 分层原则

- 前台处理分时作业，又可在后台以批处理方式运行作业
- 共同使用的基本部分放在内层随着这些操作方式而改变的部分放在外层(例如，调度程序、键盘命令解释程序和作业控制语言解释程序等)

# 分层原则

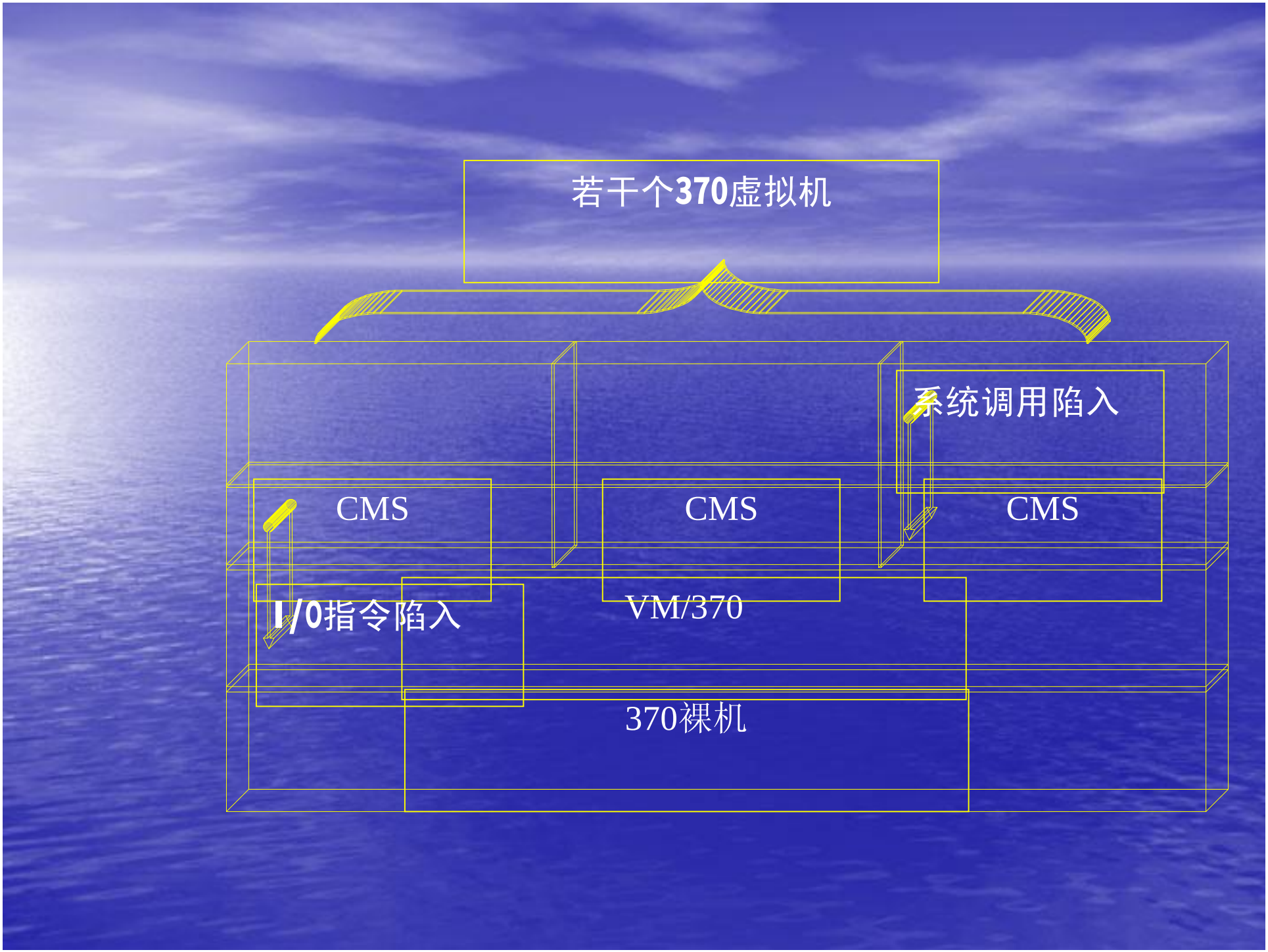
- 系统调用：为进程提供服务，这些功能模块(各系统调用功能)构成操作系统内核，放在系统的内层。

# 3．虚拟机结构

- 如IBM大型机上的系列操作系统
- 基本思想：系统应该提供

  1）多道程序能力

  2）一个比裸机有更方便扩展界面的计算机 。但是二者的实现应该相互独立
- 优缺点
  - 虚拟机概念可以实现完全保护
  - 用软件从硬件逐层扩展
  - 虚拟机方法把多道程序和扩充机器的功能完全分开


若干个370虚拟机
系统调用陷入
CMS
CMS
CMS
I/O指令陷入
VM/370
370裸机

# 微内核（客户/服务器结构）

- 非常适宜于应用在网络环境下，应用于分布式处理的计算环境中
- 由下面两大部分组成：
  - “微”内核
  - 若干服务

# 客户/服务器模型

把操作系统分成若干分别完成一组特定功能的服务进程（如内存管理服务、进程创建服务和处理器调度服务），等待客户提出请求；而系统内核只实现操作系统的基本功能(如：虚拟存储、消息传递)。

- 微内核(micro-kernel)：将更多操作系统功能放在核心之外，作为独立的服务进程运行；
  - 服务进程
  - 客户进程
- 内核消息：是一定格式的数据结构。①发起调用，送出请求消息②请求消息到达并进行处理③送出回答消息④整理回答消息，返回结果；

# Windows 2000/XP系统模型

- 融合了分层操作系统和微内核（客户/服务器）操作系统的设计思想，使用面向对象的分析与设计，采用整体式的实现
- Windows 2000/XP通过硬件机制实现了核心态以及用户态两个特权级别 。对性能影响很大的操作系统组件运行在核心态。
- 内存管理器、高速缓存管理器、对象及安全管理器、网络协议、文件系统和所有线程和进程管理，都运行在核心态。

- Windows 2000/XP的核心态组件使用了面向对象设计原则这些组件只能使用外部的接口传送参数并访问或修改这些数据
- 出于可移植性以及效率因素的考虑，大部分代码使用了基于C语言的对象实现。
- Windows 2000/XP的很多系统服务运行在核心态，这使得Windows 2000/XP更加高效，而且也是相当稳定的。

# Windows 2000 / XP的构成

- 计算机的处理器支持两种模式：用户态和核心态。用户应用程序代码在用户态下运行，操作系统代码(如系统服务和设备驱动程序)在核心态下运行
- 目的：保证了应用程序的不当行为在总体上不会破坏系统的稳定性 。装入第三方的设备驱动程序时一定要谨慎，因为一旦进入核心态，软件就能完全访问所有的操作系统数据。

# 系统调用

- 系统调用是操作系统提供给软件开发人员的唯一接口，开发人员可利用它使用系统功能。OS核心中都有一组实现系统功能的过程（子程序）
- 中断处理程序（系统调用子例程）

# 系统调用描述

- 用户所需要的功能有些功能可由硬件完成，并设有相应的指令，如启动外设工作，就有用于输入/输出的硬指令。
- 系统资源的分配、控制不能由用户干预，而必须由操作系统统一管理。
- Msdos 通过int 21 实现
- Linux int 0x80

# 系统调用目的与使用

- 1 用户程序和内核程序相分离
- 2 内核程序为用户提供相关功能，使用不必了解系统程序内部结构和相关硬件细节，用户提供系统调用名、参数。
- 通过高级程序语言内部库函数使用。

# 系统调用的过程

- 1 当系统调用发生时，处理器通过一种特殊的机制，通常是中断或者异常处理，把控制流程转移到监控程序内。同时，处理器模式转变为特权模式。
- 2 由监控程序执行被请求的功能代码。
- 3 处理结束后，监控程序恢复系统调用之前的现场；把运行模式从特权模式恢复成为用户方式；最后将控制权转移回原来的用户程序。

# 系统调用功能分类

- 1. 设备管理：这类系统调用被用来请求和释放设备，以及启动设备操作等。
- 2. 文件管理：这类系统调用包括创建、删除文件，读、写文件操作以及移动文件指针等。
- 3. 进程控制：进程是一个在功能上独立的程序的一次执行过程。
- 4. 进程通信：进程间传递消息或信号的系统调用。
- 5. 存储管理：内存块的申请、释放，获取作业占用内存块的首址、大小等。
- 6．线程管理：包括线程的创建、调度、执行、撤销等。

# 系统调用的功能

- 每个操作系统都提供几百种系统调用
- 1) 设备管理：

设备的读写和控制；

Ioctl 设备配置
Open 设备打开
Close 设备关闭
Read 读设备
Write 写设备

# 系统调用的功能

2) 文件管理：文件读写和文件控制；

Open 文件打开
Close 文件关闭
Read 读文件
Write 写文件
seek 读写指针定位
Creat 文件创建
Stat 读文件状态
Mount 安装文件系统
chmod修改文件属性

# 系统调用的功能

3) 进程控制：创建、中止、暂停等控制；
Fork　　创建进程
Exit　　进程自我终止
Wait　　阻塞当前进程
Sleep　　进程睡眠
Getpid　读父进程标识

4) 进程通信：消息队列、共享存储区、socket等通信渠道的建立、使用和删除；

5) 存储管理：内存的申请和释放；

6) 系统管理：设置和读取时间、读取用户和主机标识等；
gtime　　读取时间
Stime　　设置时间
getuid　读取用户标识

# 陷阱指令（访管指令）：

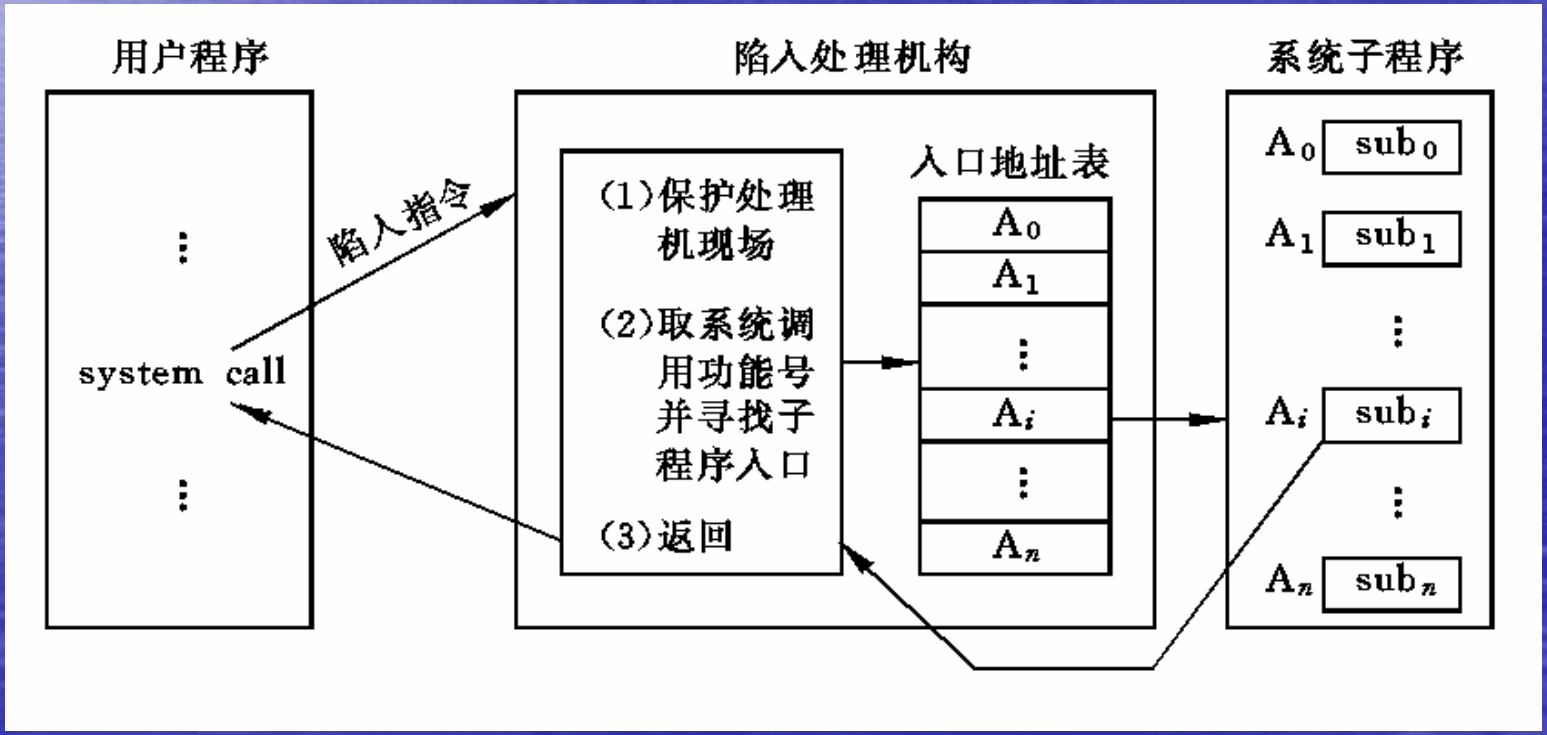
控制系统调用服务的机构称为陷阱（trap)处理机构

为了实现对这些事先编制好的、具有特定功能的例行子程序的调用，现代计算机系统一般提供访管指令。

当处理机执行到这一条指令时就发生中断，该中断称为访管中断，借助中断可使机器状态由目态转为管态。

# 系统调用的实现过程

- 陷阱指令中功能号--入口地址表
- 入口地址表—系统子程序



- 设置系统调用号和参数。
  - 调用号作为指令的一部分（如早期UNIX），或装入到特定寄存器里（如：DOS int 21h，AH=调用号。）
  - 参数装入到特定寄存器里，或以寄存器指针指向参数表（内存区域）。

- 执行trap(int)指令：入口的一般性处理，查入口跳转表，跳转到相应功能的过程。
  - 保护CPU现场(将PC与PSW入栈)，改变CPU执行状态（PSW切换）
- 执行操作系统内部代码；
- 执行iret指令：将执行结果装入适当位置（类似于参数带入），恢复CPU现场（以栈顶内容置PSW和PC）。

# 系统调用举例

凡是与硬件相关、与应用无关的工作，都通过操作系统程序来完成。

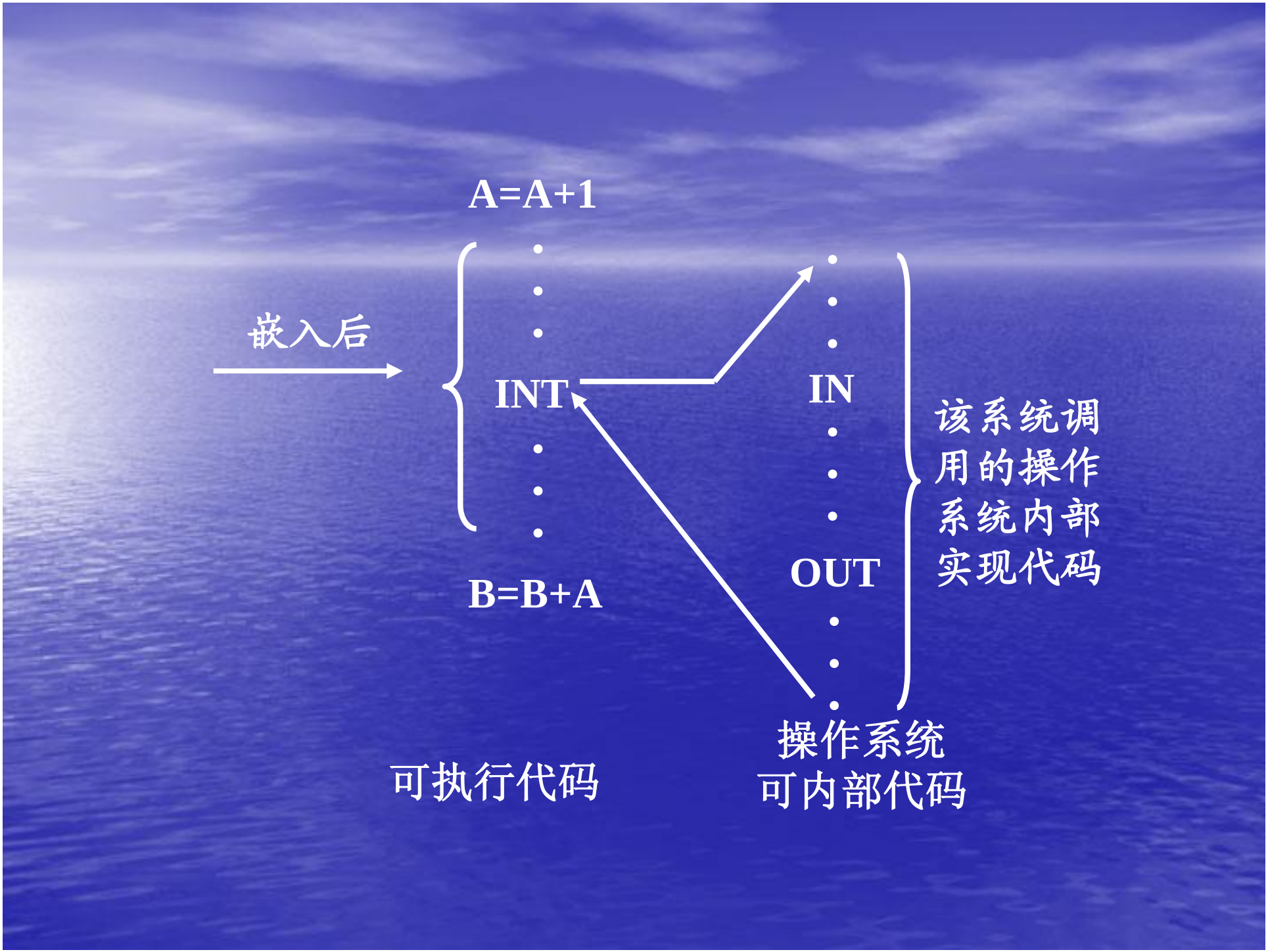
向打印机输出字符

方法1：调用DOS功能向打印机输出

```
MOV AH,05H
MOV DL,AL
INT 21H
```

方法2：用OUT指令直接打印

```
L1：MOV A，I
IN ADDR1，B
OR B，BS
JNC L1
OUT ADDR2，A
```


A=A+1
嵌入后
INT
B=B+A
IN
OUT
该系统调用的操作系统内部实现代码
可执行代码
操作系统可内部代码

系统调用是动态调用，程序中不包含被调用代码，好处：

（1）用户程序长度缩短

（2）当OS升级时，调用方不必改变

- 操作系统环境是由运行于用户模式的组件和运行于核心模式的组件组成的

# Windows 2000/XP的构成

- 用户态组件
  - 系统支持进程（system support process），不是Windows 2000/XP服务，不由服务控制器启动。
  - 服务进程（service process），Windows 2000/XP的服务。
  - 环境子系统（enviroment subsystems），它们向应用程序提供操作系统功能调用接口包括：Win32、POSIX和OS/2 1.2。
  - 应用程序（user applications），五种类型：Win32、Windows 3.1、MS-DOS、POSIX 或OS/2 1.2。
  - 子系统动态链接库：(服务进程和应用程序不能直接调用操作系统服务，通过文档化函数转换window内部系统调用

# 系统支持进程

- Idle进程
- 系统进程
- 会话管理器SMSS
- Win32子系统CSRSS
- 登录进程WINLOGIN
- 本地安全身份验证服务器LSASS
- 服务控制器SERVICES及其相关的服务进程

# Idle进程

- 在Windows 2000 / XP中Idle进程的ID总是0，而不管进程的名称是什么
- 对于每一个CPU,统计空闲的CPU时间
- **Idle**进程的名称是**System**

# 系统进程和系统线程

- 系统进程的ID总是2，它是一种特殊类型的、只运行在核心态的“系统线程”的宿主。
- 系统线程只能从核心态调用
- Windows 2000 / XP以及不同的设备驱动程序在系统初始化时创建系统线程以执行那些需要线程描述表的操作。如：发布和等待I / O
- 从系统的内存堆中分配动态存储区

# 会话管理器SMSS

会话管理器是第一个在系统中创建的用户进程

- 1)创建LPC端口对象，等待客户的请求
- 2)创建系统环境变量。
- 3)定义用于MS-DOS设备名称的符号链接(例如COMI或LPTI)。
- 4)创建附加的页面调度文件。
- 5)打开已知的动态链接库。
- 6)加载Win32子系统的核心态部分(WIN32K．EXE)
- 7)启动子系统进程。
- 8)启动登录进程。
- 9)创建用于调试事件消息的LPC端口并创建一些线程

启动了子系统进程**csrss.exe**和登录进程**WINLOGON**,如果这两个进程意外终止，**smss.exe**将使系统崩溃

# Win32子系统CSRSS

- csrss.exe(子系统进程)将负责管理windows2000的主子系统---win32子系统。

# 登录进程WINLOGIN

- 负责处理用户登录和注销的内部活动。
- 负责启动**SERVICES.exe**---系统服务器进程和**LSASS.exe**---本地安全身份验证服务进程
- 系统的启动过程，当登录界面出现后，用户输入用户名和密码，**WINLOGON.exe**平时是挂起状态的，但当从键盘截取到**ctrl+alt+del**时，它将被激活。

# 本地安全身份验证服务器LSASS

- 本地安全身份验证服务器进程接收来自WINLOGON的身份验证请求，并调用适当的身份验证包来执行实际的验证，例如检查一个密码是否与存储在SAM文件中的密码匹配

# 服务控制器SERVICES及其相关的服务进程

- 一些Windows 2000组件是作为服务来实现的，例如假脱机、事件日志、用于RPC的支持和其他各种各样的网络组件。服务由服务控制器启动和停止。
- 服务控制器是一个运行映像SERVICES．EXE的特殊系统进程，它负责启动、停止和与服务控制器交互。

# 服务进程

- spoolsv.exe 缓冲（spooler）服务是管理缓冲池中的打印和传真作业。
- service.exe 用于管理启动和停止服务
- winmgmt.exe 是win2000客户端管理的核心组件。透过Windows Management Instrumentation data (WMI)技术处理来自应用客户端的请求
- Svchost 进程提供很多系统服务,系统服务是以动态链接库（dll）形式实现的，由svchost调用相应服务的动态链接库来启动服务

# 环境子系统

- 环境子系统是用户模式进程，建立在windows执行体服务程序之上。Windows得以运行其他操作系统开发的程序。
- 将基本的执行体系统服务的某些子集以特定的形态展示给应用程序，函数调用不能在不同子系统之间混用，因此每一个可执行的映像都受限于唯一的子系统

- 三种环境子系统：POSIX、OS/2和Win32（OS/2 只能用于x86系统） 它们提供api
- Win32子系统必须始终处于运行状态，其他两个子系统只是在需要时才被启动，Win32子系统是Windows 2000/XP运行的基本条件之一。

# Win32子系统

- Win32环境子系统进程CSRSS，包括对下列功能的支持：控制台（文本）窗口、创建及删除进程与线程、支持16位DOS虚拟机（VDM）进程的部分。
- 核心态设备驱动程序（WIN32K.SYS）。
- 图形设备接口（GDI，Graphics Device Interfaces）
- 子系统动态链接库，它调用NTOSKRNL.EXE和WIN32.SYS将文档化的Win32 API函数转化为适当的非文档化的核心系统服务。
- 图形设备驱动程序，包括依赖于硬件的图形显示驱动程序、打印机驱动程序和视频小型端口驱动程序。
- 其他混杂的函数，如几种自然语言支持函数。

- POSIX子系统
  - 设计的强迫性目标
  - 实现了POSIX.1，功能局限，用处不大
  - Windows XP实际上并不包含POSIX子系统
  - 今后产品的POSIX/UNIX子系统将大大加强

# NTDLL

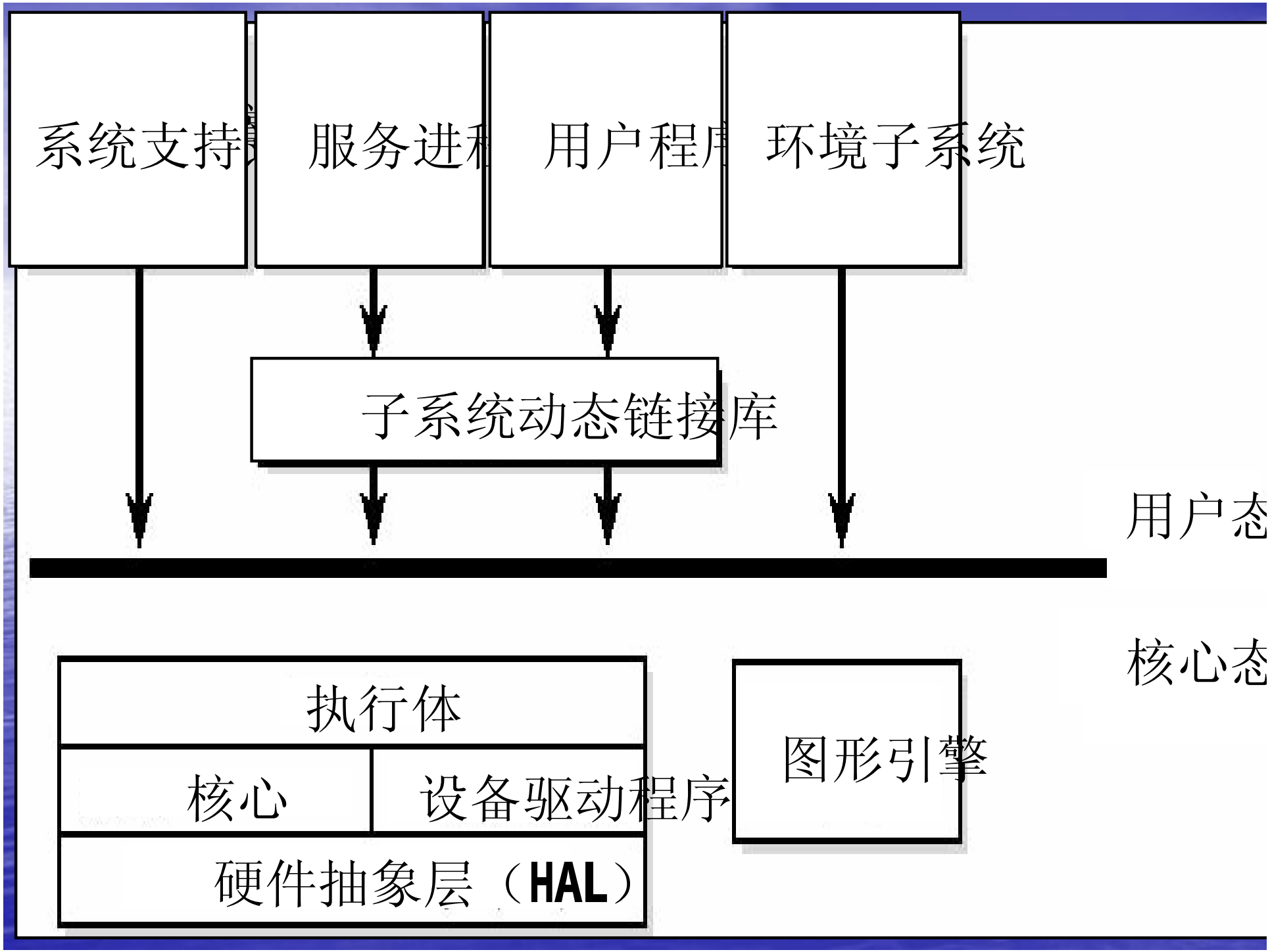
- 主要用于子系统动态链接库的特殊系统支持库
- 功能
  - 提供系统调用入口
  - 为子系统、子系统动态链接库、及其他本机映像提供内部支持函数
  - 提供从用户态调用执行体系统服务接口

# 核心态组件

- 硬件抽象层（HAL, Hardware Abstraction Layer）将内核、设备驱动程序以及执行体同硬件分隔开来，实现硬件映射。
- 设备驱动程序（Device Drivers）包括文件系统和硬件设备驱动程序等，其中硬件设备驱动程序将用户的I/O函数调用转换为对特定硬件设备的I/O请求。
- 窗口和图形系统包含了实现图形用户界面（GUI，Graphical User Interface）的基本函数。

# 核心态组件

- 核心（kernel）包含了最低级的操作系统功能，例如线程调度、中断和异常调度、多处理器同步等。同时它也提供了执行体（Executive）用来实现高级结构的一组例程和基本对象。
- 执行体包含基本的操作系统服务，例如内存管理器、进程和线程管理、安全控制、I/O以及进程间的通信。

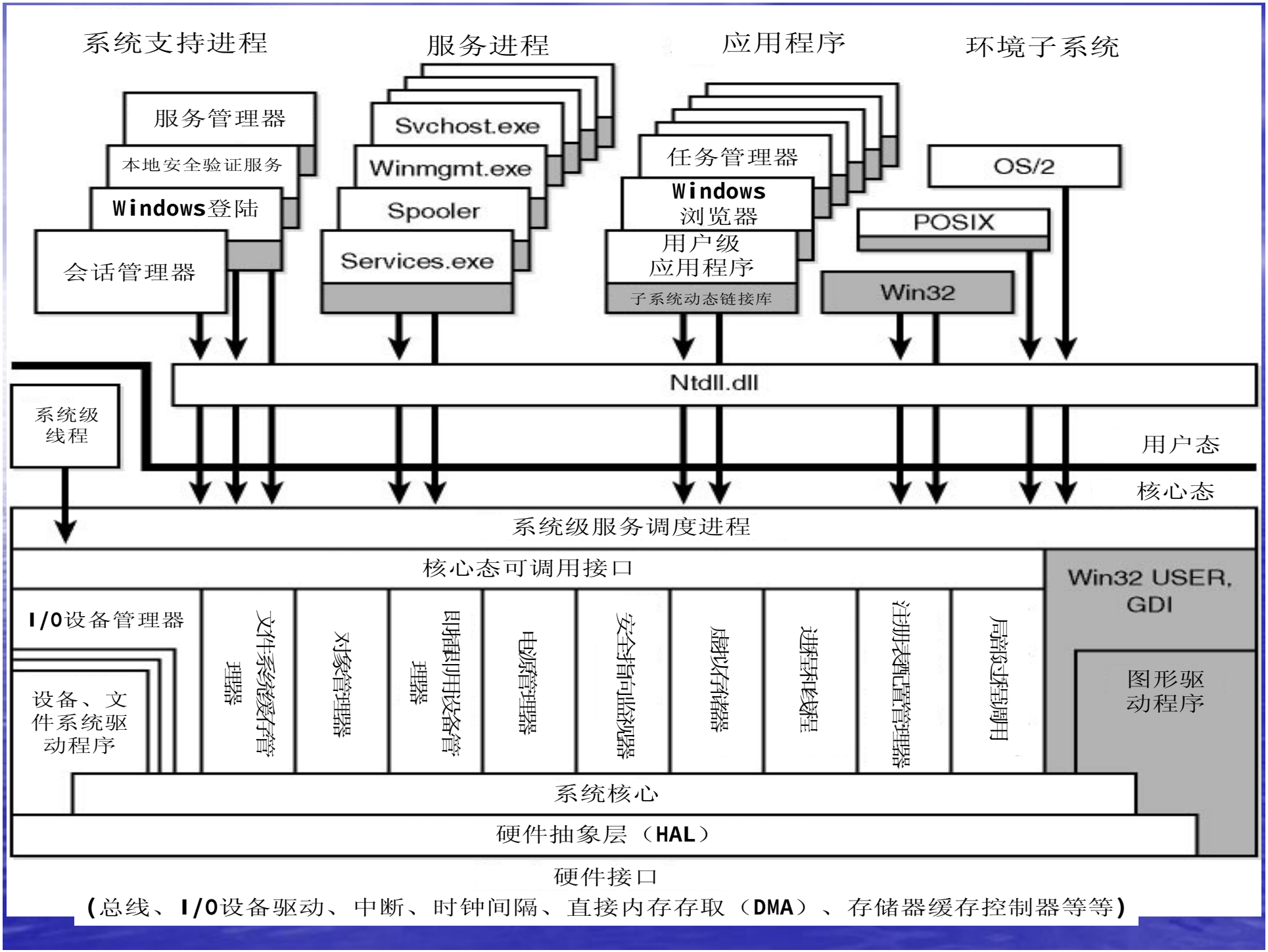

系统支持
服务进程
用户程序
环境子系统
子系统动态链接库
用户态
核心态
执行体
核心
设备驱动程序
硬件抽象层（HAL）
图形引擎

# windows2000可移植性的获得

- 两种手段
  - 分层的设计。依赖于处理器体系结构或平台的系统底层部分被隔离在单独的模块之中，系统的高层可以被屏蔽在千差万别的硬件平台之外。提供操作系统可移植性的两个关键组件是HAL和内核。
  - Windows 2000/XP大量使用高级语言——执行体、设备驱动程序等用C语言编写，图形用户界面用C++编写。只有那些必须和系统硬件直接通信的操作系统部分，或性能极度敏感的部分是用汇编语言编写的。汇编语言代码分布集中且少。

# 对称多处理器支持

- 非对称式多处理(Asymmetric Multiprocessing, ASMP)：又称主从模式(Master-slave mode)。
  - 主处理器：只有一个，运行OS。管理整个系统的资源，为从处理器分配任务；
  - 从处理器：可有多个，执行应用程序或I/O处理。
  - 特点：不同性质任务的负载不均，可靠性不够高
- 对称式多处理(Symmetric Multiprocessing, SMP)：OS交替在各个处理器上执行。任务负载较为平均，性能调节容易


系统支持进程
服务进程
应用程序
环境子系统
服务管理器
本地安全验证服务
Windows登陆
会话管理器
Svchost.exe
Winmgmt.exe
Spooler
Services.exe
任务管理器
Windows
浏览器
用户级
应用程序
子系统动态链接库
OS/2
POSIX
Win32
Ntdll.dll
系统级
线程
用户态
核心态
系统级服务调度进程
核心态可调用接口
I/O设备管理器
设备、文件系统驱动程序
文件系统缓存管理器
对象管理器
即插即用设备管理器
电源管理器
安全引用监视器
虚拟存储器
进程和线程
注册表配置管理器
局部过程调用
Win32 USER, GDI
图形驱动程序
系统核心
硬件抽象层（HAL）
硬件接口
(总线、I/O设备驱动、中断、时钟间隔、直接内存存取（DMA）、存储器缓存控制器等等)

# 对象管理器

- 对象是一个重要的概念，windows 的系统管理和程序设计都基于对象。
- 系统对象包括文件对象、设备对象和I/O 管理器所使用的驱动程序对象、执行进程对象和由进程管理的所使用的线程对象等。

# I/O管理器

- 对驱动程序的编写至关重要
- 驱动程序所进行的一切操作是以数据包为基础的，而驱动数据包的正是I/O 管理器。
- I/O 管理器通过支持创建、读、写、设定信息、获取信息和作为对于文件对象其他操作的完全主体来行使它的功能。

# 安全监视器

- 安全监视器其主要功能是负责实现安全访问策略。
- 举例来说，在NTFS 的文件系统中文件访问生效是通过安全监视器提供的服务实现的。

# 内核

- 功能
  - 线程安排和调度
  - 陷阱处理和异常调度
  - 中断处理和调度
  - 多处理器同步
  - 供执行体使用的基本内核对象
- 始终运行在核心态，代码精简，可移植性好。除了中断服务例程（ISR，interrupt service routine），正在运行的线程不能抢先内核。

# 内核对象

帮助控制、处理并支持执行体对象的操作

- 控制对象，这个对象集和包括内核进程对象、异步过程调用（APC，asynchronous procedure call）对象、延迟过程调用（DPC，deferred procedure call）对象和几个由I/O系统使用的对象，例如中断对象。
- 调度程序对象集合负责同步操作。调度程序对象包括内核线程、互斥体（Mutex）、事件（Event）、内核事件对、信号量（Semaphore）等

# 硬件抽象层（HAL）

- 使得执行体和设备驱动程序同硬件无关
- 实际硬件与Windows 2000/XP抽象计算机描述的接口层和功能映射层
- 隐藏各种与硬件有关的细节，例如I/O接口、中断控制器以及多处理器通信机制等
- 实现多种硬件平台上的可移植性
- 各种系统组件使用HAL函数与CPU外的硬件打交道

# 硬件抽象层（HAL）

- HAL通过动态链接库 HAL.DLL

设备驱动程序->硬件 使用HAL函数(IN指令访问设备端口)

HAL隐藏各种与硬件有关的细节，例如I/O接口、中断控制器及多处理器通信机制等任何体系结构专用的和依赖于计算机的函数。也即，当需要依赖于平台的信息时，Windows 2000/xp内部组件和用户书写的设备驱动程序通过调用HAL例程来保持可移植性。

# 硬件抽象层（HAL）

- 1 在操作系统引导过程中，完成总线控制器、中断控制器等硬件初始化工作
- 2 提供一种标准接口用于管理各种不同I/O总线映射（总线变化后要修改该函数）
- 3 提供包括硬中断和软中断在内的标准IRQL

# 执行体

- 提供的功能性调用
  - 从用户态导出并且可以调用的函数。这些函数的接口在NTDLL.DLL中。通过Win32API或一些其他的环境子系统可以对它们进行访问。
  - 从用户态导出并且可以调用的函数，但当前通过任何文档化的子系统函数都不能使用。
  - 在Windows 2000 DDK（设备开发包）中已经导出并且文档化的核心态调用的函数。
  - 在核心态组件中调用但没有文档化的函数。例如在执行体内部使用的内部支持例程。
  - 组件内部的函数。

- 包含的功能实体
  - 进程和线程管理器创建及中止进程和线程。对进程和线程的基本支持在Windows 2000内核中实现。
  - 虚拟内存管理器实现“虚拟内存”。内存管理器也为高速缓存管理器提供基本的支持。
  - 安全引用监视器在本地计算机上执行安全策略。它保护了操作系统资源，执行运行时对象的保护和监视。
  - I/O管理器（文件系统、设备驱动程序）
  - 高速缓存管理器通过将最近引用的磁盘数据驻留在主内存中来提高文件I/O的性能，

- 对象管理：创建、管理以及删除Windows 2000/XP的执行体对象和用于代表操作系统资源的抽象数据类型，例如进程、线程和各种同步对象。
- 本地过程调用（LPC，Local Procedure Call）机制，在同一台计算机上的客户进程和服务进程之间传递信息。LPC是一个灵活的、经过优化的“远程过程调用”（RPC，Remote Procedure Call）版本。
- 一组广泛的公用运行时函数，例如字符串处理、算术运算、数据类型转换和完全结构处理。
- 执行体支持例程，例如系统内存分配（页交换区和非页交换区）等。

# 对象管理器

- 执行体组件之一
- 用于创建、删除、保护和跟踪对象
- 提供使用系统范围内资源使用的公共、一致的机制
- 实现对象的集中保护
- 实现了资源的访问控制
- 对象管理器有一套对象命名方案和统一的保留规则，能够容易地操纵现有对象

- 执行体对象
  - 由执行体的各种组件实现
  - 进程管理器、内存管理器、I/O子系统等
- 内核对象
  - 由内核实现的原始的对象集合
  - 这些对象对用户态代码是不可见的，它们仅在执行体内创建和使用
- 内核对象提供了一些基本性能，许多执行体对象内包含着一个或多个内核对象。

# 设备驱动程序

- 可加载的核心态模块 (通常以．SYS为扩展名)
- I/O系统和相关硬件之间的接口
- Windows 2000 / XP上的设备驱动程序不直接操作硬件，而是调用HAL功能作为与硬件的接口。
- 分类
  - 硬件设备驱动程序操作硬件。
  - 文件系统驱动程序接受面向文件的I/O请求，并把它们转化为对特殊设备的I/O请求。
  - 过滤器驱动程序截取I/O并在传递I/O到下一层之前执行某些特定处理。

# Windows 2000/XP的系统机制

- 陷阱调度，包括中断、延迟过程调用（DPC，Deferred Procedure Call）、异步过程调用（APC，Asynchronous Procedure Call）、异常调度（Exception Dispatching）和系统服务调度（System Service Dispatching）
- 执行体对象管理器（Executive Object Manager）
- 同步（Synchronization），包括自旋锁（Spin lock）、内核调度程序对象（Kernel Dispatcher Objects）
- 本地过程调用（LPC，Local Procedure Call）

# 陷阱调度

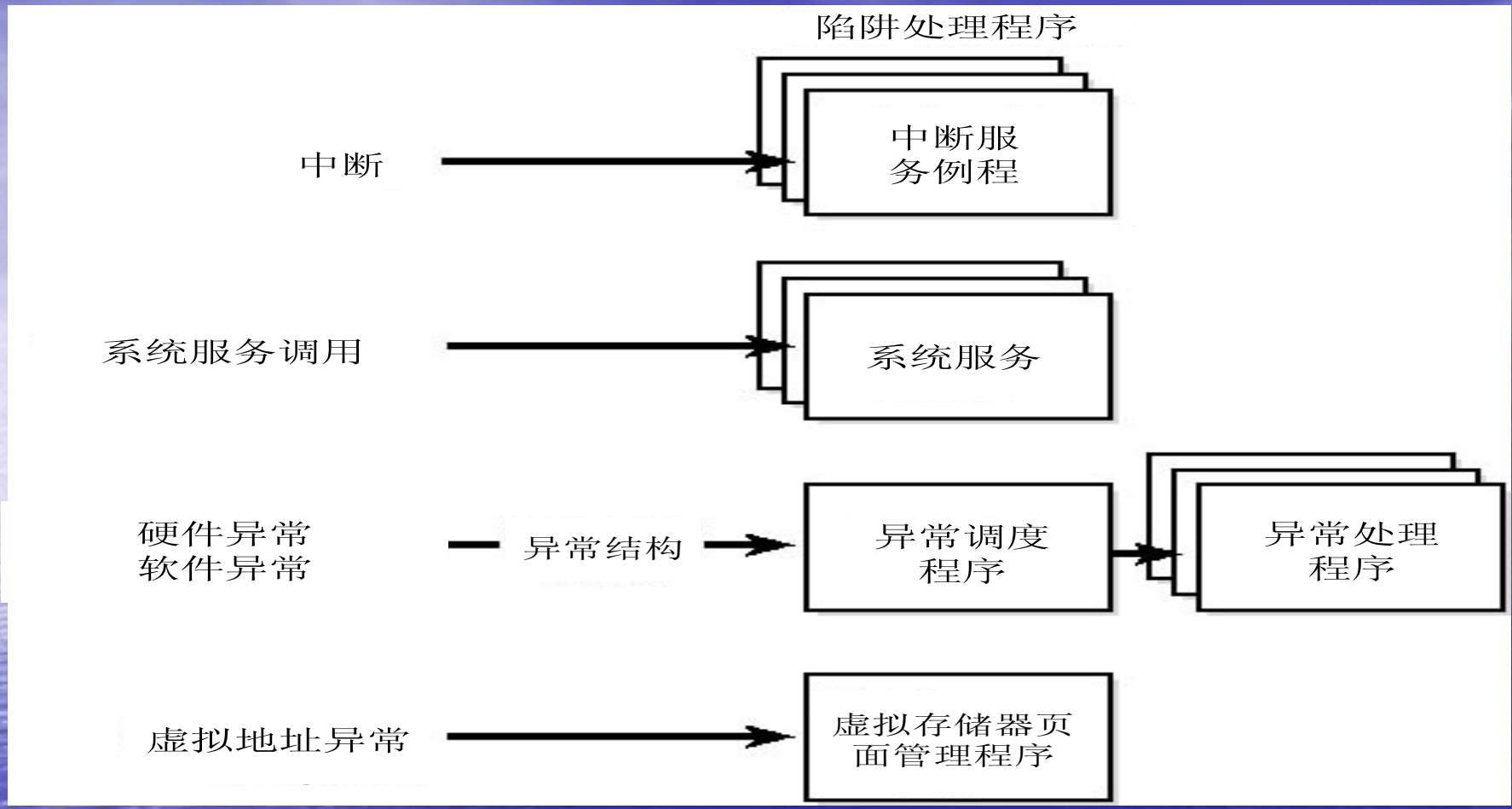
- 提供给操作系统处理意外事件的硬件机制
- 当异常或中断发生时，硬件或软件可以检测到，处理器会从用户态切换到核心态，并将控制转交给内核的陷阱处理程序，该模块检测异常和中断的类型，并将控制交给处理相应情况的代码

Ø中断是异步事件，可能随时发生，与处理器正在执行的内容无关。中断主要由I/O设备、处理器时钟或定时器产生，可以被启用或禁用

Ø异常是同步事件，它是某一特定指令执行的结果。例如内存访问错误、调试指令以及被零除。

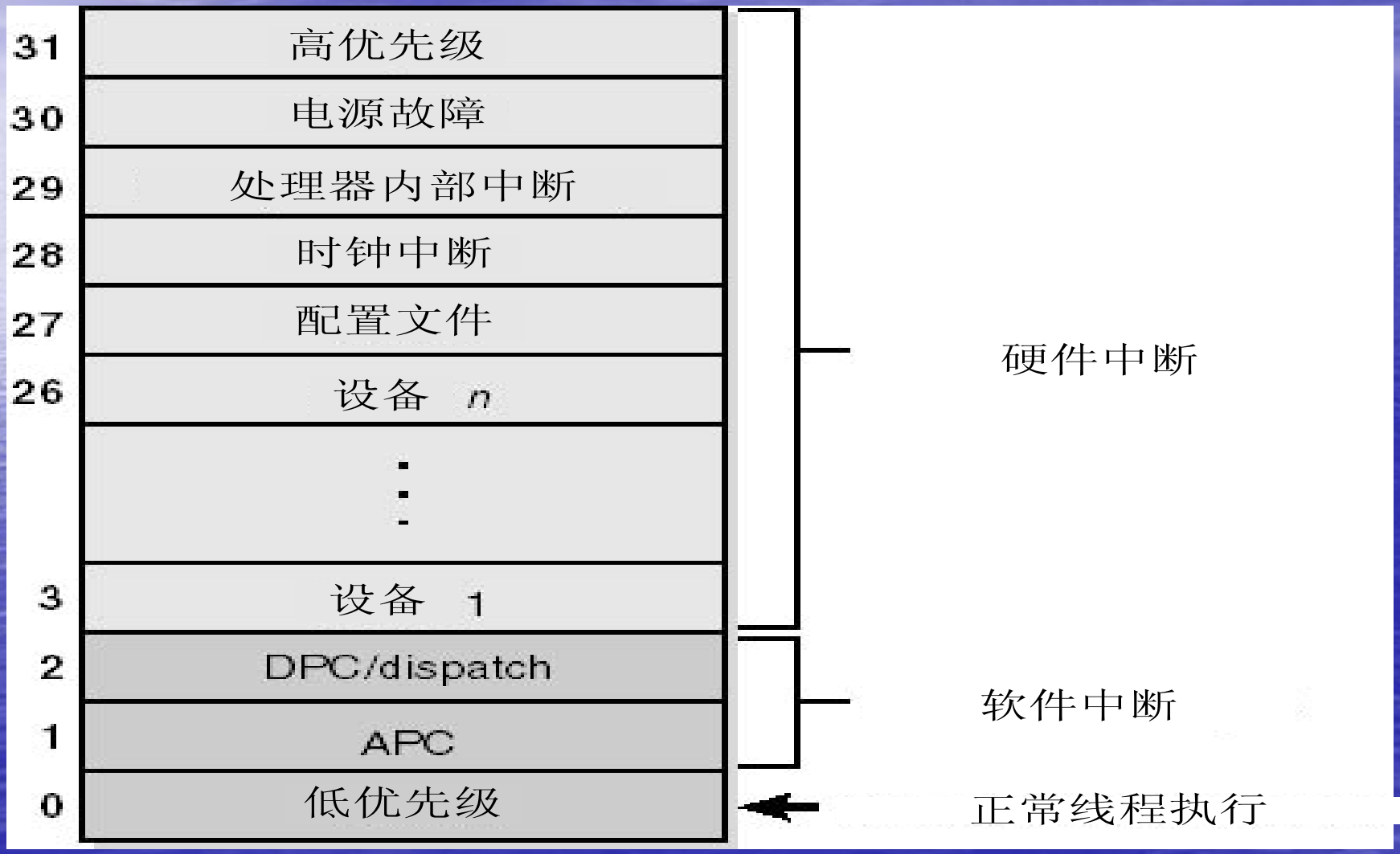
系统服务调用也视作异常。

软件和硬件都可以产生异常和中断。




- 陷阱帧：完整的线程描述表的子集，用于现场保护
- 陷阱处理程序处理少量事件，多数转交给其他的内核或执行体模块处理

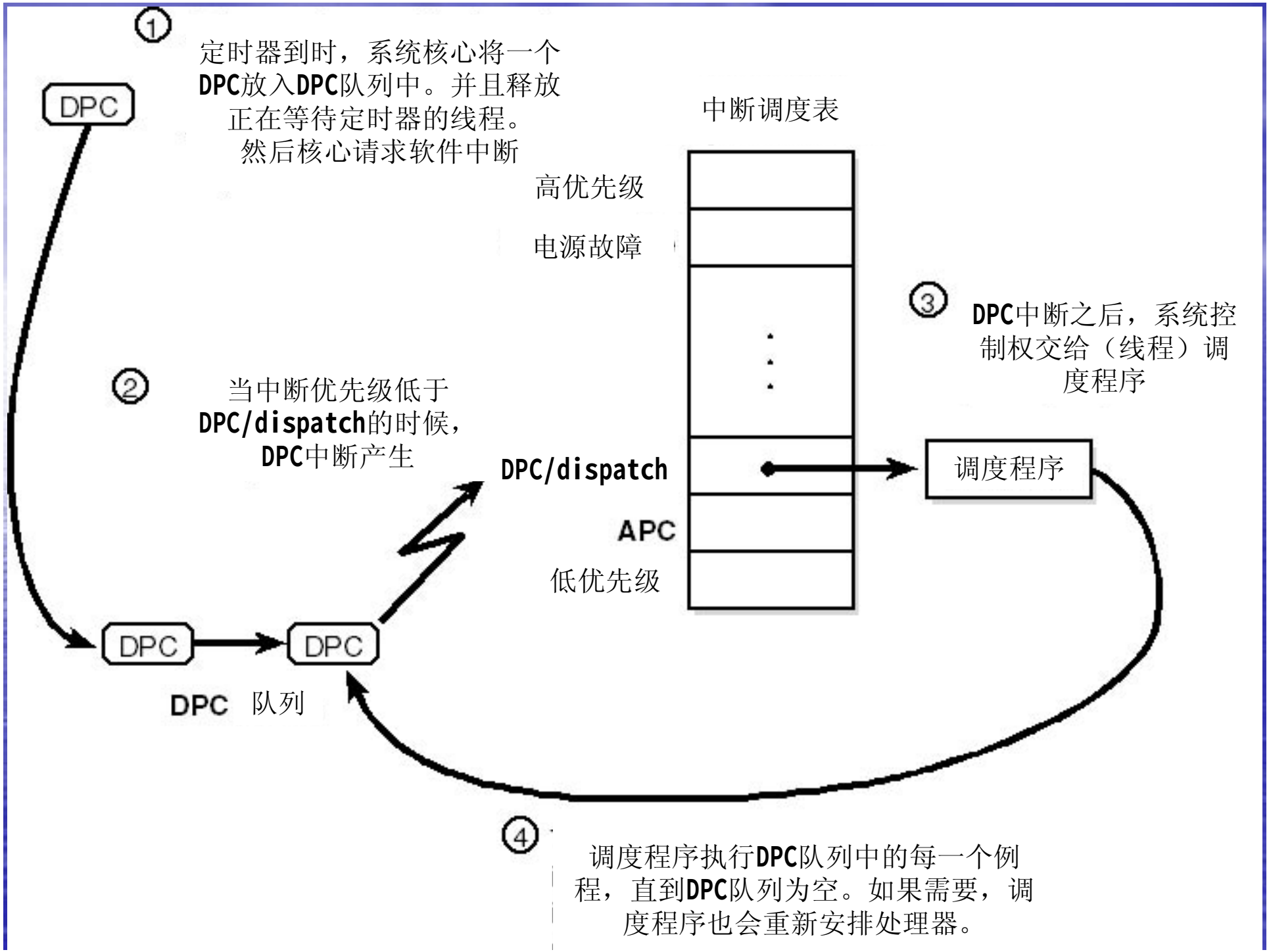
# 中断分类和优先级


31
高优先级
30
电源故障
29
处理器内部中断
28
时钟中断
27
配置文件
26
设备 n
3
设备 1
2
DPC/dispatch
1
APC
0
低优先级
硬件中断
软件中断
正常线程执行

- 用户线程运行在中断优先级0
- 内核态的异步过程调用(APC) 运行在中断优先级1
- 线程调度代码运行在DPC/ dispatch 优先级
- 所有的硬件中断与DPC/ dispatch 中断都可以中断用户线程的运行

Windows 2000 的中断处理分两步:

- 第一步,中断服务例程( ISR) 执行,为了使设备准备好再次中断, ISR 执行尽可能少的代码,然后将一个DPC 例程加入DPC 队列,并准备再次中断
- 第二步DPC 例程做中断处理的大部分工作


①
定时器到时，系统核心将一个DPC放入DPC队列中。并且释放正在等待定时器的线程。然后核心请求软件中断
DPC
中断调度表
高优先级
电源故障
③
DPC中断之后，系统控制权交给（线程）调度程序
②
当中断优先级低于DPC/dispatch的时候，DPC中断产生
DPC/dispatch
调度程序
APC
低优先级
DPC
DPC
DPC 队列
④
调度程序执行DPC队列中的每一个例程，直到DPC队列为空。如果需要，调度程序也会重新安排处理器。

# 延迟过程调用(DPC)

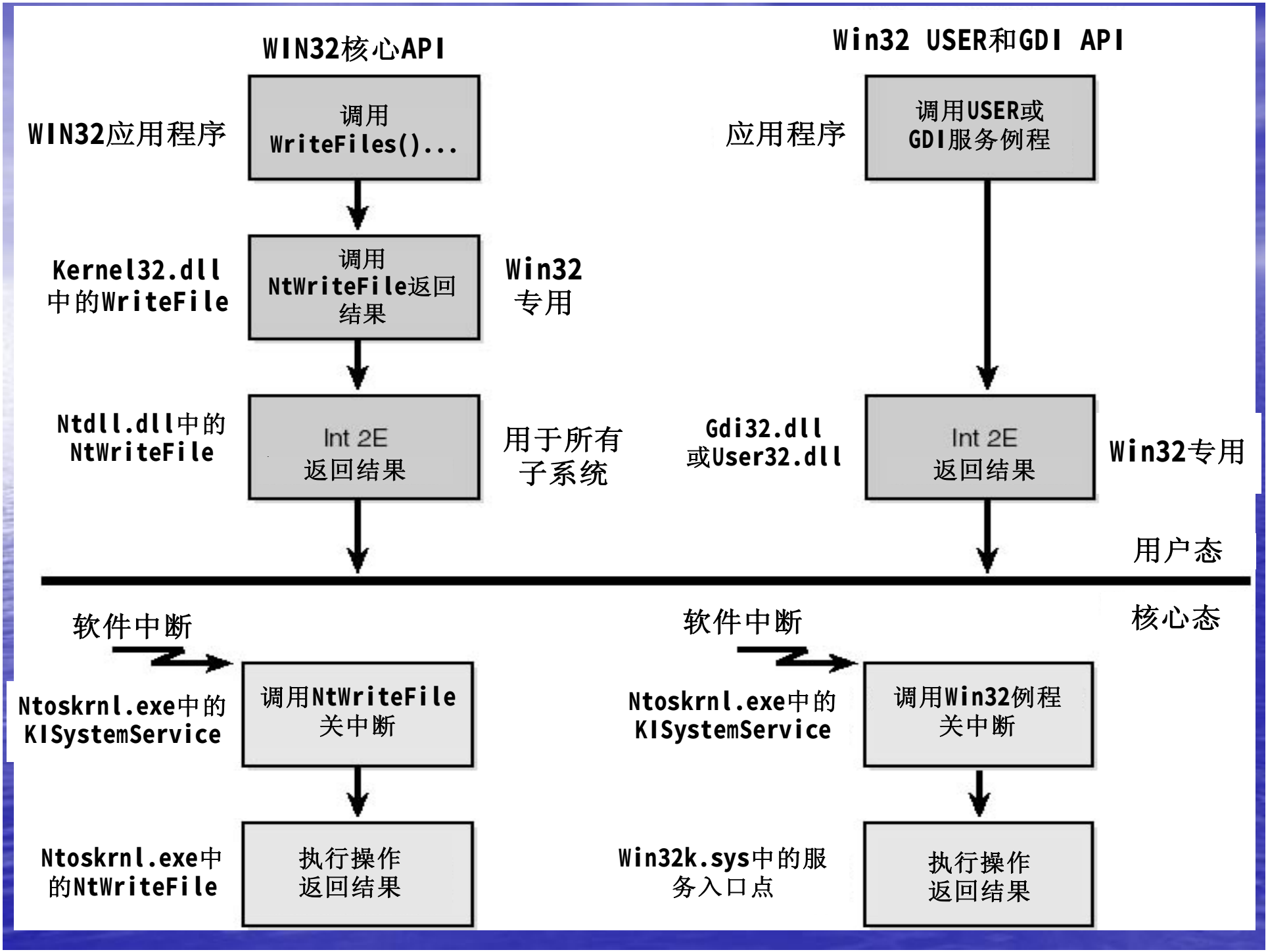
- 大幅度缩短中断服务程序的执行时间
- 当一个中断服务程序完成了它应该执行的工作时,应请求运行它的DPC。
- DPC 运行在DISPATCH L EVEL 中断级,属于软件中断,它的中断请求级低于所有的硬件中断,但同时其优先级又高于所有其他线程,因而DPC 队列中的DPC 将在所有硬件中断服务完毕后立即得到执行。
- 这种机制的实质是把中断服务程序代码根据它所进行操作的紧急程度再次分成两个优先级别,这样既保证了中断服务程序能及时完成它所需的全部操作,同时所有的硬件中断都能尽可能快地得到服务。
- 除了在中断服务程序中使用DPC 外,在驱动程序中使用自定义定时器时,也要进行DPC 的自定义。

# 异步过程调用（APC）

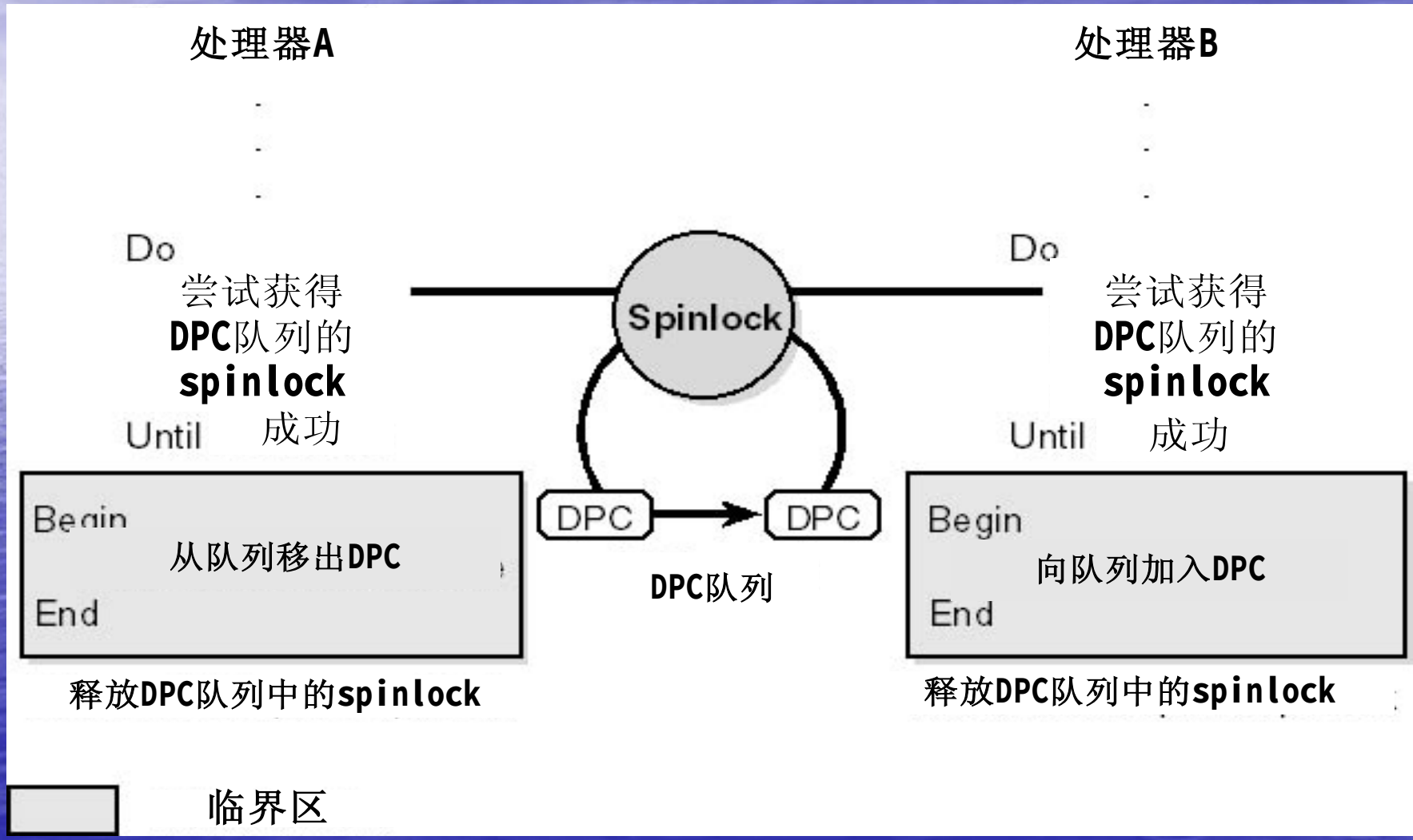
- 异步过程调用为用户程序和系统代码提供了一种在特殊用户线程的描述表（一个特殊的进程地址空间）中执行代码的方法
- 有用户态APC和核心态APC
- 设备驱动程序使用

# 异常调度

- 异常直接由运行程序的执行所产生。
- 除了那些可由陷阱处理程序解决的简单异常外，所有异常都是由异常调度程序接管


WIN32核心API
Win32 USER和GDI API
WIN32应用程序
调用
WriteFiles()...
应用程序
调用USER或
GDI服务例程
Kernel32.dll
中的WriteFile
调用
NtWriteFile返回
结果
Win32
专用
Ntdll.dll中的
NtWriteFile
Int 2E
返回结果
用于所有
子系统
Gdi32.dll
或User32.dll
Int 2E
返回结果
Win32专用
用户态
核心态
软件中断
软件中断
Ntoskrnl.exe中的
KISystemService
调用NtWriteFile
关中断
Ntoskrnl.exe中的
KISystemService
调用Win32例程
关中断
Ntoskrnl.exe中
的NtWriteFile
执行操作
返回结果
Win32k.sys中的服
务入口点
执行操作
返回结果

# 同步机制


处理器A
处理器B
Do
尝试获得
DPC队列的
spinlock
Until 成功
Do
尝试获得
DPC队列的
spinlock
Until 成功
Spinlock
Begin
从队列移出DPC
End
DPC
DPC
DPC队列
Begin
向队列加入DPC
End
释放DPC队列中的spinlock
释放DPC队列中的spinlock
临界区

# 本地过程调用（LPC）

- 一个用于高速信息传输的进程间通信机制（使用共享内存）
- LPC常常被使用在一个服务器进程与该服务器的一个或多个客户进程之间，以及在两个用户态进程之间或一个核心态组件和一个用户态进程之间

- 远程过程调用使用LPC 在同一系统中的进程之间通信。
- 少数Win32 API导致向Win32子系统进程发送消息。
- Winlogon利用LPC与本机安全服务器进程(Lsass)通信。
- 安全引用监测器利用LPC 与Lsass 进程通信。

- LPC的三种交换信息的方法：
  - 使用包含信息的缓冲区调用LPC可以发送少于256字节的信息。然后，这个信息又从发送进程的地址空间复制到系统地址空间，再从那里拷贝到接收进程的地址空间。
  - 如果用户和服务器想交换大于256字节的数据，那么他们可以选择使用双方都映射了的共享区。
  - 直接从客户地址空间读出或向客户地址空间写入。

# LPC 的通信过程

- 客户进程传送请求到服务器连接端口
- 当客户进程发出一个请求时,请求消息就被插入到服务器进程的消息队列中,客户进程转入等待状态;
- 当服务器进程收到请求时,它的一个工作线程被唤醒,并从消息队列中取出请求,执行完毕后将结果送至客户端的未命名端口
- 客户端进程也在结果到达后继续执行

# 注册表

- 保存所有有关系统和每个用户的设置信息
- Windows 2000系统管理机制的关键数据库
- 组成：
  - 主键
  - 键值
  - 主键可以包含若干主键（或称为这个主键的子键）和键值，而键值则存储数据，顶级主键称为根键
  - 六个根键

- HKEY_CURRENT_USER 存储与当前登录用户有关的信息。
- HKEY_USER 存储了所有用户的信息。
- HKEY_CLASSES_ROOT 存储与文件类型和COM对象相关的信息
- HKEY_LOCAL_MACHINE 存储与系统设置相关的信息
- HKEY_PERFORMANCE_DATA 存储与系统性能相关的信息
- HKEY_CURRENT_CONFIG 存储了当前硬件配置文件的信息

# WMI的体系结构

- 主要的组件：

1 WMI数据消费者：

处理和显示WMI被管理对象的应用及服务，例如性能监测，查询和控制被管理对象的状态和行为

2 WMI被管理对象

被管理对象是指需要管理的对象的模型，是基于公共信息模型（CIM）的描述被管理对象的数据结构，被管理的对象可以是小到一根电缆、一个阀门，大到一个应用程序、一个数据库系统等都可以是被管理的对象。

3 WMI数据生产者

数据生产者的可能表现形式有：WMI管理进程加载的动态连接库、独立的 应用程序以及独立的 服务进程。微软提供了一套内建的数据生产者，它们提供了常用的数据源，如注册表、事件管理器、活动目录设备驱动程序

4 WMI内部基础设施

内部基础设施是联系数据消费者和数据生产者的桥梁

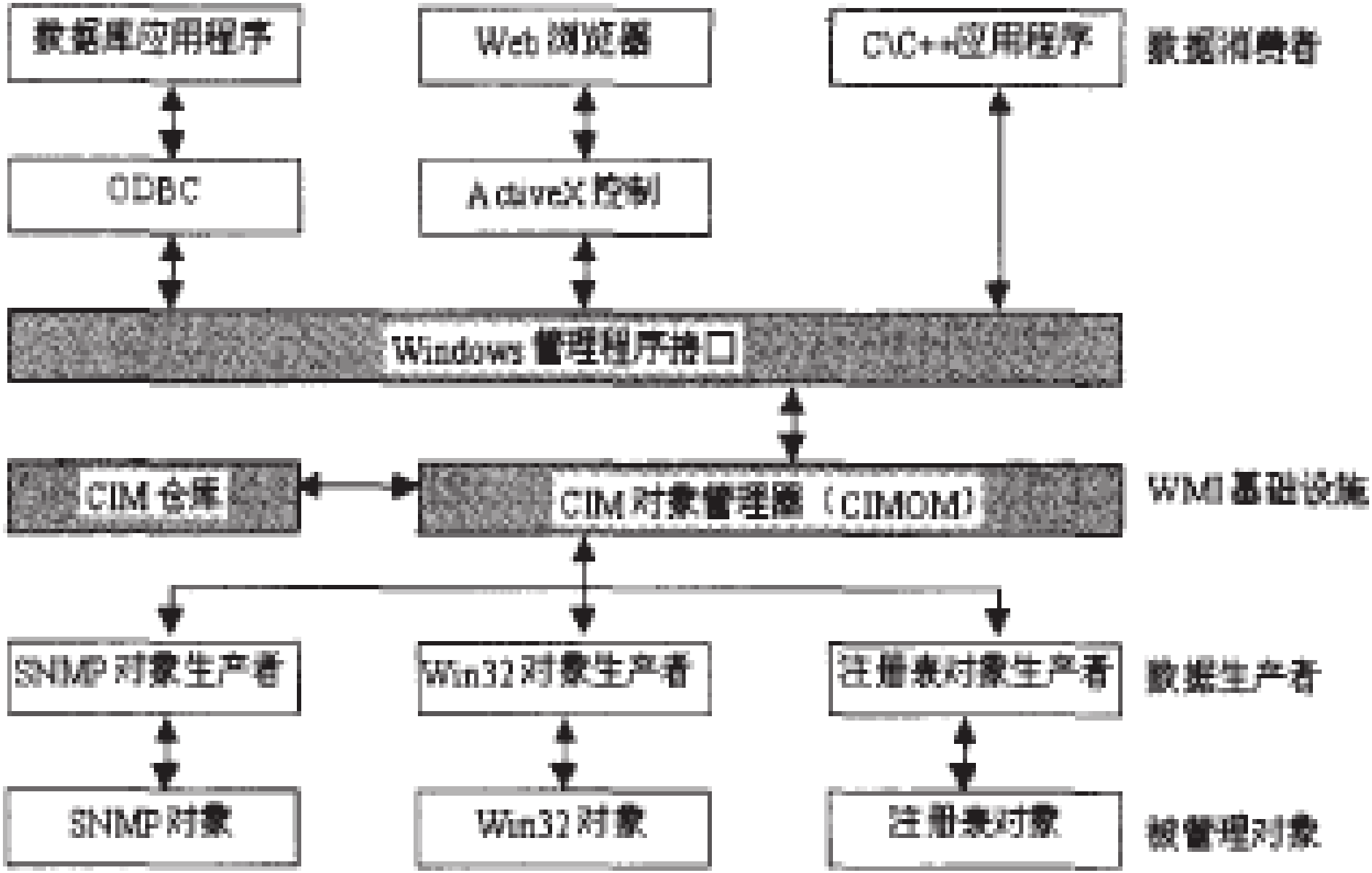



图 1　WMI 技术中各组件之间的关系

# WMI给设备驱动程序带来很多好处：

- 1 任何驱动程序定义的数据都可以由用户应用程序广泛使用。
- 2 驱动程序可以主动通知应用程序驱动程序自定义的事件的发生。而不必应用程序轮询或者发送IRP请求包
- 3 驱动程序只需收集应用程序要求的数据和事件，从而可以降低耗费、提高驱动程序性能。
- 4 客户应用程序需要时可以将驱动程序定义的数据和事件的描述显示给用户