Windows NT File System Internals

第一部分

概述

第一部分介绍 Windows NT 操作系统和文件系统驱动开发引出的相关知识。

第一章， NT 系统组成

第二章， 文件系统驱动开发

第三章， 结构化的驱动开发

第一章

NT 系统组成

本书的中心是 NT 文件系统和文件系统同其他核心操作系统组件的相互作用。如果你有兴趣为 NT 平台提供有附加值的软件，过滤驱动的设计和开发的主题将会给你提供设计那样的相关软件好的理解。文件系统和过滤驱动不存在真空中，而是和操作系统有许多很紧密地交互，这一章介绍 NT 系统的主要的组件。

基础

操作系统处理那些用户不关心的事情，包括初始化处理器状态，协调多个 CPU，维护 CPU 高速缓存一致，管理本地总线，物理内存，提供虚拟内存支持，管理外部设备，创建和调度用户线程和进程，负责把用户数据存储到外部设备上，提供易于管理和友好的基础系统。由此可见，操作系统必须是可靠和有效的，因为任何这种能力的缺乏几乎都能导致操作系统的很快死亡。

和你可能会听说相反，无论如何 NT 不是一个最新技术发展水平的操作系统。它借鉴了已经被实现了许多年的的许多商业操作系统的概念和观念，可以认为 NT 是广泛的商业系统和大学研究项目的概念和方法的汇集。

设计原理和方法逻辑来自伟大的 UNIX 和 OPEN VMS 操作系统，同样 CMU 开发的 MATH 系统在其中的影响也是很明显的。也受到一些 MS-DOS 和 OS/2 的影响，但是，不能认为 NT 就是其他系统设计原理和观点的糅合。事实是 NT 的设计者将会借鉴他们设计其他系统的经验，而这种经验会使他们设计出一个稳定和严肃的系统。

核心架构

在 NT 的设计中的一些从 MACH 系统继承过来的设计哲学是明显的。这包括努力最小化内核的大小和实现一个 C/S 模式的操作系统部分，在模块之间使用消息传递信息的方法。而且，设计者还试图实现一个层次化的系统，每一个组件和其他层的交互都是在一个定义良好的接口上进行。

NT 的设计是能运行在一个或者多个处理器环境的。

最后，一个主要的目标是能够简单地在不同的硬件架构上移植系统，为了达到这个目标，设计者使用基于对象的模式设计系统，抽象出那些硬件依赖的从而需要在不同平台重新实现的部分。而其他的移植组件就只要在不同的架构上做重新编译即可。

图 1-1 显示 NT 的结构，从图上看出 NT 可以划分成两个部分，用户模式和内核模式。

用户模式

操作系统提供保护子系统，每个保护子系统在自己的存储空间里处理自己的进程。内存保护是由 NT 的虚拟内存管理器提供支持的。

子系统提供定义良好的应用编程接口，从而用户模式的进程可以调用期望的功能。子系统再用定义良好的系统服务调用和内核模式的操作系统部分通信。

注意：微软没有真正文档化过操作系统提供系统服务调用。他们鼓励 NT 应用开发人员使用操作系统环境提供的子系统服务。

由于 NT 系统服务 API 未文档化，设计者试图保留一个操作系统的一个抽象试图。因此应用程序只是和自己的本地子系统交互，而子系统负责和操作系统的交互。微软这样做的意图（哲学观念）是使大多数应用和易于移植的 WIN32 子系统绑定，而且和大多数应用依赖着特定的 NT 系统服务比较也更容易改进。因此，有时 NT 应用程序和内核驱动程序能够直接访问系统服务会更有效。在附录 A 里有能找到由 I/O 管理器提供的处理文件 I/O 操作的系统服务。

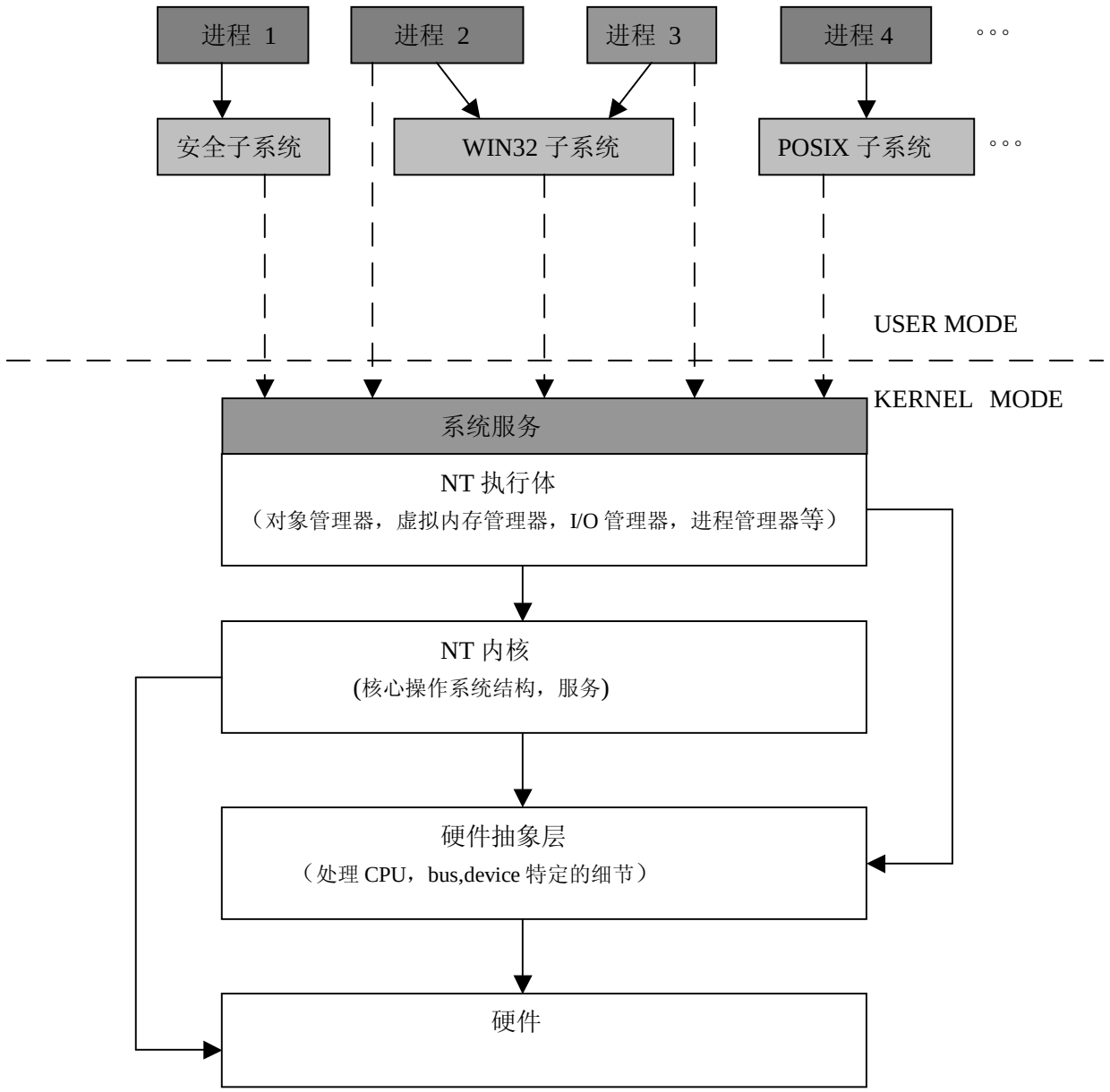

图 1-1 NT 操作系统环境

环境子系统向用户进程提供 API 和执行环境来仿效一些特定的操作系统如 OS/2 或 UNIX，可以认为子系统是用户进程看上去的操作系统，进程可以在操作系统提供的安全，自然的环境里舒适地执行而不用关心 NT 内核系统的能力，编程接口和要求。

NT 提供下面几种环境子系统：

WIN32

NT 本地执行环境，MICROSOFT 鼓励应用开发人员在软件中使用 WIN32 API 来获得操作系统服务。这个子系统也是更特别的系统，它是唯一的可靠的用来管理设备用于和用户交互的。监视器，键盘，和鼠标都是由 WIN32 子系统控制的。也是唯一的系统的窗口管理器，而且定义了策略来控制外观的图形用户接口。

POSIX

这个子系统是支持那些符合 POSIX 1003.1 代码标准的应用程序的。如果你的应用程序是使用这个标准里定义的 API 来开发的。理论上是能够早 NT 平台上编译，链接和执行。

POSIX 子系统有一些功能上的限制，应用程序不得不面对，比如，没有网络支持。

OS/2

在 INTEL X86 硬件平台上提供 16 位 OS/2 应用程序的支持。

WOW（WINDOWS ON WINDOWS）

对 16 位的 WINDOWS 3.X 应用程序的支持，注意，16 位的试图控制和直接访问硬件的程序不能在 NT 平台上执行。

VDM（VIRTUAL DOS MACHINE）

支持 16 位 DOS 应用程序，和 16 位的 WINDOWS 3.X 应用程序一样，试图控制和直接访问硬件的程序不能在 NT 平台上执行。

完整的子系统扩展到用户空间，提供重要的系统功能。包括用户空间的安全子系统（如本地服务权力）用户空间的 NT LAN MANAGER 网络软件组件，还有负责加载，卸载和管理内核模式驱动和系统服务的服务管理器。

内核模式

执行代码在内核模式和在用户模式之间的不同是 CPU 在不同的硬件特权等级执行代码。

大多数 CPU 架构有至少两个硬件特权等级，许多还有更多等级。CPU 的硬件特权等级决定代码能够执行的指令集。例如，执行在用户模式中的进程不能直接修改或者访问 CPU 寄存器或者用于虚拟内存管理的分页表。允许所有的用户进程访问这些数据将会很快陷入混乱，使重要的任务得不到 CPU 的处理。

NT 使用简化的两个硬件特权等级模式：内核模式允许代码在处理器上做任何必要的事情。用户模式进程则被严格的约束在一个被允许的操作的范围里。

如果你熟悉 INTEL X86 架构，内核模式处理器等于 RING0 特权层而用户模式等于 RING3。术语内核模式和用户模式实际上是表示和处理器相关的特权等级。虽然常用来说代码。术语内核模式就表示在执行某些代码时 CPU 将处于内核模式特权等级，而用户模式表示 CPU 将在用户特权等级执行代码。

通常，作为第三方开发人员，除非你设计和开发 NT 内核模式驱动程序，否则不能在 CPU 的内核模式特权等级上执行程序。

NT 的内核模式部分由以下组成：

硬件抽象层（HAL）

NT 系统是设计成支持多个硬件架构的。实际上，可以在 INTEL X86 平台，DEC ALPHA 平台，以及其他基于 MIPSD 的平台。而且，有多种外部总线可以使用，包括 ISA，EISA，VL-BUS 和 PCI 总线结构。NT 开发人员建立 HAL 来隔离硬件相关的代码。HAL 是直接和 CPU 和其他硬件系统交互而且负责向系统提供适当抽象的相对较薄的软件层。

NT 内核看见的是一个由 HAL 提供的理想化的硬件视图，所有的不同硬件结构之间的差异由 HAL 在内部处理。它导出的函数被操作系统组件和设备驱动程序调用。

HAL 导出的函数允许访问系统时钟，I/O 总线，DMA 和中断控制器。设备寄存器等。

NT 内核

NT 内核提供操作系统其余部分使用的基本功能，可以认为 NT 内核提供那些基本的块让 NT

执行体向操作系统提供有力的功能。内核负责处理进程和线程调度，用自旋锁处理多处理器同步，中断处理和分派和其他的功能。NT 内核将在下一节详细描述。

NT 执行体

这是 NT 的最大的组成部分。他使用内核和 HAL 的服务，因此可以在不同硬件平台和结构上容易的迁移。为各种子系统提供丰富发系统服务，允许他们访问操作系统功能。

NT 执行体的主要组成部分包括对象管理器，虚拟内存管理器，进程管理器，I/O 管理器，安全信任监视器，本地过程调用工具，配置管理器和高速缓存管理器。

I/O 子系统的文件系统，设备驱动程序和中间层驱动由 I/O 管理器管理也是 NT 执行体的一部分。

WINDOWS NT KERNEL

NT KERNEL 被作为操作系统的心脏进行描述，虽然比起 NT 执行体要小很多。内核负责提供的基本功能：

支持内核对象
线程分派
多处理器同步
硬件异常处理
中断处理和分派
陷入处理
其他硬件特定的功能

NT 内核在处理器的最高特权等级上执行，被设计为在并发的对称多处理器环境上执行。内核不能发生页面错误，因此内核所有的代码和数据总是驻留在系统内存里。而且不能被抢占，因此当处理器在执行属于内核的代码时不能发生上下文切换。但是，所有在任何处理器上执行的代码都是可以中断的，提供比正在执行的代码更高的中断等级就行。

IRQL LEVELS

NT 定义和使用中断请求等级（IRQLS）来决定内核组件的执行顺序。决定内核模式代码的 IRQL 是硬件优先权。底于或者等于当前执行的内核执行代码的 IRQL 将被 NT 内核屏蔽。因此当前在处理器上执行的代码能够被那些 IRQL 更高的软件或者硬件中断。IRQL 的等级定义如下（由第到高）：

PASSIVE_LEVEL

普通线程执行中断等级,大多数文件系统驱动被要求提供线程在 IRQL PASSIVE_LEVEL 等级执行的功能,虽然这不被关心.大多数 LOWER-LEVEL 驱动,例如设备驱动,在高于 PASSIVE_LEVEL 的 IRQL 等级被调用.这个 IRQL 也叫做 LOW_LEVEL.

APC_LEVEL

异步过程调用(APC)中断等级.APC 被软件软件中断调用,然后会影响目标线程的控制流.APC 的目标线程将被中断,然后创建 APC 是指定的过程将会在中断线程的上下文中在 APC_LEVEL 等级执行.

DISPATCH_LEVEL

线程分派(调度)和延迟过程调用(DPC)中断等级.DPC 在第三章 结构化驱动开发中 描述,一旦线程的 IRQL 升到 DPC 或更高,线程调度自动被暂停.

DEVICE INTERRUPT LEVELS(DIRQL)

设备 IRQL 等级的平台相关的数字和值.

PROFILE_LEVEL

定时器用来压型(PROFILING)

CLOCK1_LEVEL

定时器时钟一间隔时间
CLOCK2_LEVEL
定时器时钟二间隔时间
IPI_LEVEL
只用于多处理器系统的处理器间中断等级
POWER_LEVEL
电源故障中断
HIGHEST_LEVEL
通常用于机器自检和总线错误的中断
APC_LEVEL 和 DISPATCH_LEVEL 中断是软件中断,他们由内核模式代码请求,比其他所有的硬件中断等级的优先权底.在 CLOCK1_LEVEL 和 HIGHEST_LEVEL 的中断是 TIME_CRITICAL(时钟中断)中断,因此在线程执行中有最高优先级.
支持内核对象
NT 内核努力维持一个基于对象的环境.提供一个核心对象集让 NT 执行体使用,同时提供访问和操作这些对象的函数.注意 NT 内核不依赖对象管理器来管理内核定义的对象类型.NT 执行体使用内核输出的对象来构造更复杂的对象供用户使用.
内核对象有以下两种类型:
DISPATCHER OBJECT
这些对象控制同步和分派系统线程.包括 THREAD,EVENT,TIMER,MUTEX 和 SEMAPHORE 对象类型.在第三章有这些对象的描述.
CONTROL OBJECT
这些对象影响内核模式代码的运行但不影响分派和同步.包括 APC,DPC,中断,进程和设备队列对象.
NT 内核还维护下列的数据结构:
INTERRUPT DISPATCHER TABLE(中断分派表)
内核维护的和中断源相关的适当的中断服务例程的表格.
PROCESSOR CONTROL BLOCKS(PRCBS)
系统中的每个处理器有一个 PRCB,包含各种处理器特有的信息,包括当前线程的执行指针,下一个要被调度的线程和空闲线程.
注意:每个处理器有一个空闲线程,每当没有其他的线程可以调度的时候就执行,他比系统中所有线程的优先级底.不断的循环寻找有没有要处理的任务比如 DPC 队列,还有每当其他线程就绪了初始化上下文以便在处理器上执行.
处理器控制块(PROCESSOR CONTROL REGION)
这是硬件结构相关的内核数据结构,包含 PRCB 的指针,GLOBLE DESCRIPTOR TABLE(GDT),INTERRUPT DESCRIPTOR TABLE(IDT)和其他的信息.
DPC 队列
这个全局队列包含将要调用的过程列表,每当处理器的 IRQL 低于 DISPATCH_LEVEL.
定时器队列
内核维护的一个全局定时器队列,包含在将来某个时间预定的要到时的定时器.
分派器数据库
线程分派器模块维护一个包含所有处理器执行状态和系统中的线程的数据库,被分派器模块用于调度线程执行.
除了上面提到的对象类型,NT 内核还维护设备队列,电源通知队列,处理器请求队列和其他的内核为了自己的功能必须的数据结构.

进程和线程

进程是表示一个执行程序的一个实例的对象.在 NT 中,每个进程必须至少有一个执行线程.进程由私有的虚拟地址空间,虚拟空间中的私有的代码和数据,还有执行的过程中分配的系统资源组成.

注意进程对象不是调度实体,因此不能调度一个进程去执行,每个进程包含一个或者多个调度线程来执行.

线程对象上下文由线程的用户和核心栈指针,指示当前执行指令的程序计数器,系统寄存器(包括整形和浮点寄存器)包含的状态信息,其他线程执行是维护的处理器状态信息组成.

每个线程有一个相关的调度状态.可能的有 INITIALIZED,READY-TO-RUN,STANDBY,RUNNING,WAITING 和 TERMINATED.在任何给定的时刻只有一个线程处于 RUNNING 状态,但是在多处理器上可以有多个(每个处理器一个).

线程有相关的执行优先级,高优先级的总是被优先调度和抢占底优先级线程的执行,优先级分为 REAL-TIME 优先级和可变优先级.

注意:在 NT 系统中可能遇到优先级倒置的情况,当一个底优先级的线程拥有一个高优先级请求的资源的时候,那么那些比这个拥有资源的线程的优先级高的线程就得到机会执行,尽管他的优先级比等待资源的线程底.

上面描述的情况违反了高优先级线程总是抢占底优先级线程执行的设定,这可能导致不正确的行为,特别在线程优先级必须维持的情况下(例如适时进程).内核模式设计师必须预见和理解那些情况会发生,除非保证资源获得的顺序(层次)是正确定义和维护的.

NT 没有提供比如优先级继承的特性来自动避免优先级倒置的问题.

多数内核提供的例程用于程序处理(programmatically manipulating)或者访问线程或者进程结构不向第三方驱动开发人员暴露.

线程上下文和陷阱

陷阱是处理器提供的当某些事件发生时来捕获执行线程的一种机制。陷阱处理器是一段汇编代码，高度的处理器和架构特定的。也是 NT 内核提供的功能的一部分。导致陷阱的事件包括中断，异常情况，或者把处理器模式从用户模式转换到内核模式的系统服务调用。

当陷阱发生时，将调用操作系统的陷阱处理器。在调用适当的例程处理陷阱情况前陷阱处理器要保存执行线程的信息到一个叫做“调用帧（CALL FRAME）”的表里。调用帧有两个组成部分：

陷阱帧

包含寄存器（变化的）的状态

异常帧

当异常情况的发生导致陷阱处理器被调用的时候，也要保存寄存器（不便的）的状态。另外，陷阱处理器还要保存当前一刻的机器状态和其他的用于陷阱处理完后让线程继续执行的信息。

WINDOWS NT 执行体

NT 执行体由不同模块，或者子系统组成，他们中的每一个都要负责功能的一部分。通常说 NT 内核模式代码实际上是说的执行体中模块。

执行体为子系统提供了大量的系统服务调用来访问他的服务。另外，执行体还给那些期望扩展现有功能的开发者提供了广泛的支持。通常用于开发第三方驱动程序，可安装文件系统驱动，和其他中间层和过滤驱动等提供增值服务的软件。

组成 NT 执行体的不同的组件自己或多或少的严格维护自己的边界。重复，基于对象的操作系统大量使用抽象数据类型和方法来表明自己。模块并不直接访问其他模块的内部数据结

构，注意这点，虽然设计者要管理互相粘连的定义良好的内部接口，模块在相互调用时还是要做许多假定。这个假定通常在“期望的”调用的模块会执行什么样的处理和错误状态会怎样处理或者报告的情况之中。最后，如你将在本书中看见的，NT 执行体采用的同步层次组件在他们相互调用的时候非常的复杂。

WINDOWS NT 对象管理器

所有的 NT 执行体组件导出的供其他的内核模式模块使用的数据类型使用对象管理器来定义，创建，和管理数据类型，还有数据的实例。

NT 对象管理器管理对象。对象是特定的内核组件实现和操作的不透明的数据结构。每个对象可能有相应的操作集；这些操作包括创建对象的实例，删除对象的实例，等待对象被通知和通知对象。

对象管理器提供以下的能力：

动态的添加对象的类型到系统中（注意对象管理器并不关心这个对象的内部数据结构）。

允许模块指定对象实例的安全和保护特性。

提供创建和删除对象实例的方法。

允许模块定义对象类型，提供自己的方法（比如创建，关闭，和删除操作）来管理对象类型的实例。

提供兼容的方法逻辑来维护对对象类型实例的引用。

提供基于更通用的文件系统的层次化的“翻转树”格式的全局命名层次。

对象管理器维护一个全局名字空间。系统中所有的命名对象可以在这个空间访问到。对象的名字空间模仿通常的文件命名习惯。因此，有一个全局的叫做“\”的根目录在系统初始化的时候被对象管理器创建出来。执行组件可以在根目录下创建目录和子目录。然后可以在那些子目录中创建对象类型的实例。当一个对象被创建或者插入后（甚至文件系统定义的对象如文件和目录），从对象管理器维护的名字空间的根目录开始分析对象的名字。如果有为对象类型定义的分析方法（例如，文件对象具有的打开文件系统文件和目录），对象管理器为对象调用分析方法，第二章，文件系统驱动开发，提供更多的关于对象管理器怎样处理对盘上文件或者目录对象的打开或者创建请求。

NT 对象管理器维护的对象类型结构包含比如实例应该被分配的内存池，对象的有效的访问类型，和对象关联的处理过程的指针（这些是可选的，包括比如创建，打开，关闭，删除和其他的过程指针），和对象管理器为特定类型的所有实例维护的同步结构。

每个对象实例有一个标准的对象头和相关的对象类型特有的对象体。标准对象头包含如对象名字的指针（如果有），和对象相关安全描述符（如果有）对象的访问模式，引用记数，对象类型的指针（指向拥有的对象实例），和其他相关属性。

当线程成功打开一个特定对象类型的实例，对象管理器返回一个对象实例的不透明句柄给请求线程。注意，任何对象的实例同时可以有不只有一个句柄，例如，对象句柄可以隐含地继承。

对象管理器维护和每个对象句柄关联的信息，包含一个对象实例的指针，打开操作的访问信息和其他的属性。注意，在句柄和指向打开的对象类型的实例之间没有直接联系。句柄通常是一个对象数组的索引。

警告：句柄是特定于进程的，因此如果一个线程成功执行创建和打开操作得到一个返回句柄，这个进程中的所有线程都可以使用这个句柄。

但是，如果这个同样的句柄在其他的进程的线程上下文中使用，将会收到一个错误代码指出这个句柄是无效的。

其他 WINDOWS NT 执行组件

主要的其他的 NT 执行组件有这些：

进程管理器

负责创建和删除进程和线程，他使用 NT 内核提供的服务执行比如挂起一个正在执行的进程，继续执行一个进程，提供进程信息等任务。

本地过程调用（LPC）工具

一种可以在同一接点上的进程之间传递消息的机制。客户端进程常常传递一些参数到服务器进程来请求请求一些服务。作为回报，他可以收到一些处理后的数据。

客户端到服务器的调用被客户端进程中的一个存根（STUB）截取，然后打包要被送到服务器过程的参数。然后 LPC 给客户端进程提供机制来传输数据到服务器，然后等待服务器的回应。这是使用一种 LPC 定义和创建的叫“端口”对象的来完成的。

LPC 模仿 RPC 的机制，RPC 用于实现客户-服务器模式通过本地或者广域范围相连的机器。LPC 做了更好的优化，因为在同一接点中所有进程访问相同的物理内存。

安全引用监控

负责在本地节点上强制执行安全策略。还提供对象审记工具。

虚拟内存管理

NT 虚拟内存管理器（VMM）管理节点上所有可用的物理内存，还负责操作系统其他部分提供虚拟内存管理，还有所有节点上执行的所有应用程序。

几乎所有内核和用户模块必须和虚拟内存管理器组件交互，大部分模块是 VMM 的客户端因此依赖于 VMM 提供的内存管理服务。但是文件系统是特殊的，因为它通常必须向 VMM 提供服务（如读，写页文件）。文件系统设计者透彻地理解文件系统驱动和 VMM 模块之间的交互，在第五章更详细的讨论 VMM。

缓存管理器

NT 执行体包含一个专门的缓存模块来为存储在二级存储介质的文件数据提供缓存功能（在系统内存中）。他使用 NT 执行体的 VMM 的服务来提供缓存功能。所有的本地文件系统驱动程序都使用缓存管理器，将在第 6-8 章讨论缓存管理器的细节。

I/O 管理器

NT I/O 管理器定义和管理所有内核驱动（包括文件驱动，网络，磁盘，中间层，过滤驱动程序）的框架。将在第四章描述细节。