Windows内核由一系列在Ntoskrnl.exe中提供基本机制的函数组成。这些函数包括了线程调度和同步服务，用于执行体组件以及低级硬件架构支持，比如在每种处理器架构都不同的中断和异常处理。内核代码大部分是用C语言编写的，只有需要专用处理器方法和用C语言不易编译的部分使用汇编语言编写。

像各种执行支持函数，大部分在WDK的内核部分是已经文档化的(并且可以通过以Ke开头的函数找到)因为编写设备驱动时需要它们。

内核对象

内核提供了一组定义明确的、可预知的操作系统低层原语和机制，从而使得执行体中的高层组件可以做它们需要做的事情。内核实现了操作系统的基本机制，并且避免各种策略决定，从而将自己与执行体的其余部分分离开。它几乎将所有的策略决定留给了执行体，唯一的例外是线程调度和分发，这是由内核自己来实现的。

从内核外部来看，执行体将线程和其他可共享的资源都表示为对象。这些对象需要一些策略开销，比如用以维护它们的对象句柄(object handle)，以及用以保护它们的各种安全检查，还有相应的资源配额（当它们被创建时资源配额就会被扣除）。这些开销在内核中时不存在的，内核实现了一组更为简单的对象，称为**内核对象**(kernel object)，它们帮助内核控制好中心处理器过程，并且支持执行体对象的创建工作。绝大多数执行体层的对象都封装了一个或者多个内核对象，把它们的内核属性合并在一起。

一组称为**控制对象**(control object)的内核对象建立了有关控制各种操作系统功能的语义。这包括APC对象、DPC(Deferred Procedure Call，**延迟过程调用**)对象，以及I/O管理器使用的一些对象，比如中断对象等。

另一组称为**分发器对象**(dispatcher object)的内核对象融合了同步的能力，可以改变或者影响线程的调度。分发器对象包括内核线程、互斥体(mutant)、事件、内核事件对(event pair)、信号量(semaphore)、定时器，以及可等待的定时器(waitable timer)

内核处理器控制区和控制块(KPCR和KPRCB)

内核使用一个称为**处理器控制区**(kernel processor control region, KPCR)的数据结构来存放与处理器有关的数据。KPCR包含了基本的信息，处理器中断分发表(IDT)、任务状态段(TSS)和全局描述符表(GDT)都在其中。还有中断控制器的状态，这是内核与其他模块(ACPI和HAL)共享的数据。为了便于访问KPCR，在32位Windows上，内核在fs寄存器中保存一个指向KPCR的指针；在x64 Windows系统上，指向KPCR的指针存放在gs寄存器中。在IA64系统上，KPCR总是位于0xe0000000ffff0000

KPCR也包含一个称为**内核处理器控制块**(kernel process control block，KPRCB)的内嵌数据结构。KPCR是已经文档化的数据结构，因而第三方的驱动程序和其他的Windows内核组件可以使用；但是KPRCB是一个私有的数据结构，仅在Ntoskrnl.exe中的内核代码使用该结构。KPRCB包含了调度信息(处理器正在调度的当前线程、下一个执行线程以及空闲线程)、该处理器的分发器数据库(其中包含了每个优先级的就绪队列)、DPC队列、CPU厂商和标识符信息(型号(model)、步进(stepping)、速度、特征位)、CPU和NUMA拓扑(节点信息、每个芯片的核、每个核的逻辑处理器，等等)、缓存大小、时间计数信息(比如DPC和中断时间)，等等。KPRCB还包含了所有关于该处理器的统计信息，比如I/O统计、缓存管理器的统计、DPC统计，以及内存管理器的统计。最后，KPRCB有时候也被用来存储一些缓存对齐的、针对每个处理器的数据结构，以便优化内存访问，系统中非换页的和换页的内存池快查表也存储在KPRCB中。

**Windows驱动程序模型**

Windows2000增加了对即插即用、电源选项的支持，同时也扩展了WindowsNT的驱动程序模型，新的模型称为WDM(Windows Driver Model，Windows驱动程序模型)。Windows2000及以后的版本都可以运行老的WindowsNT 4驱动程序，但是因为这些驱动程序不支持即插即用和电源选项，所以运行这些驱动程序的系统在这两方面的能力都会有所退化。

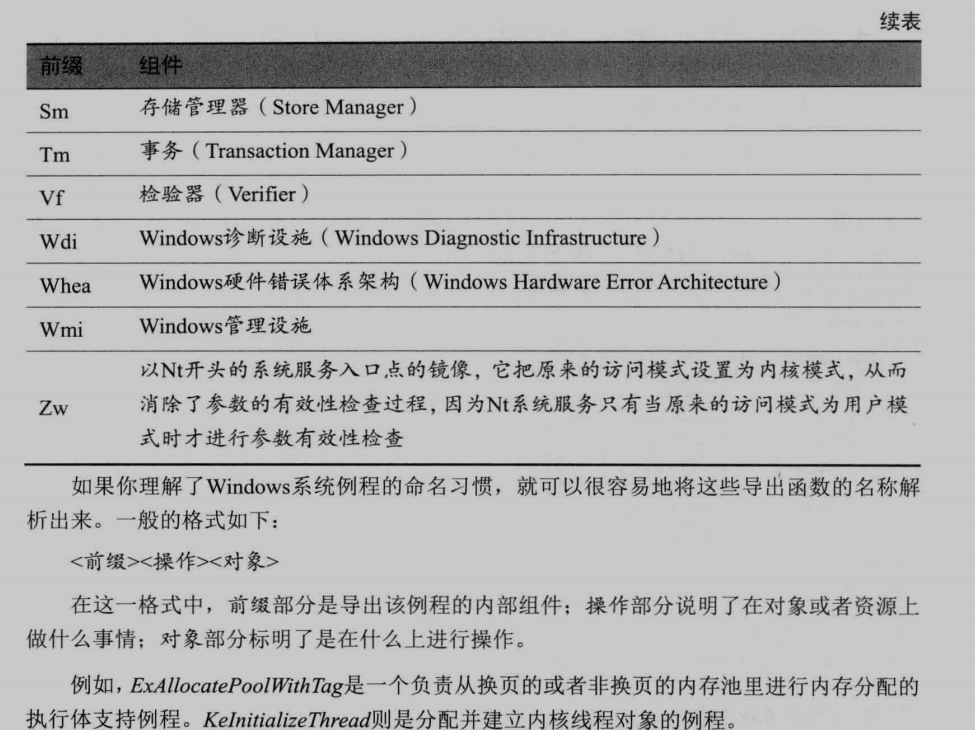
从WDM的角度来看，有以下三种驱动程序。

**总线驱动程序**(bus driver)为总线控制器、适配器、桥或任何带有子设备的设备提供服务。总线驱动程序是必需的驱动程序，通常Microsoft会提供此类驱动程序；系统中的每一种总线类型(比如PCI、PCMCIA和USB)都有一个总线驱动程序。第三方也可以编写总线驱动程序来为新的总线提供支持，比如VMEbus、Multibus和Futurebus。

**功能驱动程序**(function driver)是主要的设备驱动程序，它为相应的设备提供可操作的接口。功能驱动程序也是必需的，除非相应的设备可以按照原始的方式直接使用（指一种特殊的实现，其I/O可通过总线驱动程序和总线过滤驱动程序来完成，比如SCSI PassThru）。根据定义，功能驱动程序最了解具体的设备，它往往是唯一能访问与该设备相关的寄存器的驱动程序。

**过滤驱动程序**(filter driver)用来为某一设备（或已有的驱动程序）增加新的功能，或者修改来自其他驱动程序的I/O请求或应答（也常常用来修补那些未能提供正确的硬件资源需求信息的硬件设备）。过滤驱动程序是可选的，可以有任意数目，可以放在功能驱动程序之上或之下，也可以放在总线驱动程序之上。通常，系统原始设备制造商(OEM)或者独立硬件供应商(IHV)会提供过滤驱动程序





System进程和系统线程

System进程(Pid=4)是某种特殊线程的母体，这种特殊线程只能在内核模式下运行，称为**内核模式系统线程**(kernel-mode system thread)。系统线程具备普通用户模式线程的所有属性和环境（比如硬件环境、优先级等），但是不同的地方在于，它们只有在内核模式下运行系统空间加载的代码，无论这些代码是否在Ntoskrnl.exe中，还是在任何其他加载进来的设备驱动程序中。而且，系统线程没有用户进程地址空间，因此，任何动态存储需求，都必须从操作系统的内存堆中分配，比如从一个换页的或者非换页的内存池中分配

系统线程是通过PsCreateSystemThread函数来创建的，该函数只能从内核模式中调用。Windows以及各种设备驱动程序在系统初始化阶段创建系统线程，以便执行各种要求线程环境操作的，内核会创建一个称为**平衡集管理器**(balance set manager)的系统线程，它每秒钟被唤醒一次，从而有可能会发出各种调度和内存管理相关的事件。缓存管理器也使用系统线程来实现“预读(read-ahead)”和“滞后写(write-behind)”的I/O。文件服务器设备驱动程序(Srv2.sys)利用系统线程来相应那些“针对已共享胖次分区的文件数据”的网络I/O请求。

默认情况下，系统线程是属于System的，但是，设备驱动程序可以在任何一个进程中创建系统线程。例如，Windows子系统设备驱动程序(Win32k.sys)在规范的显示驱动程序(Canonical Display Driver, Cdd.dll)中创建一个系统线程，Cdd.dll是Windows子系统进程(Csrss.exe)的一部分，因而新线程可以很容易地访问该进程用户地址空间中的数据。

在诊断或者进行系统分析时，若能够将一个系统线程的执行过程映射回驱动程序上，甚至映射到该代码的子例程，一定非常有用。例如，在一个负载很重的文件服务器上，System进程很有可能会消耗掉相当可观的CPU时间。但是，仅仅知道“System进程运行时‘某个系统线程’正在运行”，是不足以确定哪个设备驱动程序或者操作系统组件正在运行的。所以如果System进程中的线程正在运行，则首先要确定哪些线程正在运行（例如，通过性能监视器）。找到正在运行的线程后就可以查看一下该系统线程是在那个驱动程序中开始执行的（至少可以知道可能是哪个驱动创建了这个线程），或者检查一下所涉及的线程调用栈（或至少当前的地址），通过调用栈可以知道线程当前在哪里执行。

