PEB（进程环境块）。PEB包括已加载的模块（如EXE 和 DLL）列表，包含环境字符串的内存、当前的工作目录和管理进程堆的数据——以及很多随着时间的推移已添加的Win32 cruft。

线程是在Windows中调度CPU的内核抽象。优先级是基于进程中包含的优先级值来为每个线程分配的。线程也可以通过亲和处理只在某些处理器上运行。这有助于显式分发多处理器上运行的并发程序的工作。每个线程都有两个单独调用堆栈，一个在用户态执行，另一个内核态执行。也有TEB（线程环境块）使用户态数据指定到线程，包括每个线程存储区（线程本地存储区）和Win32字段、语言和文化本地化以及其他专门的字段，这些字段都被各种不同的功能添加上了。

除了PEB与TEB外，还有另一个数据结构，内核态与每个进程共享的，即用户共享数据。这个是可以由内核写的页，但是每个用户态进程只能读。它包含了一系列的由内核维护的值，如各种时间、版本信息、物理内存和大量的被用户态组件共享的标志，如COM、终端服务和调试程序。有关使用此只读的共享页，纯粹是出于性能优化的目的，因为值也能获得通过系统调用到内核态获得。但系统调用是比一个内存访问代价大很多，所以对于大量由系统维护的字段，例如时间，这样的处理就很有意义。其他字段，如当前时区更改很少，但依赖于这些字段的代码必须查询它们往往只是看它们是否已更改。

1. 进程

进程创建是从段对象创建的，每个段对象描述了磁盘上某个文件的一个内存对象。在创建一个过程时创建的进程将接收一个句柄，这个句柄允许它通过映射段、分配虚拟内存、写参数和环境变量数据、复制文件描述符到它的句柄表、创建线程来修改新的进程。这非常不同于在UNIX中创建进程的，反映了Windows与UNIX 初始设计目标系统的不同。

正如11.1节所描述，UNIX 是为16 位单处理器系统设计的，而这样的单处理器系统是用于在进程之间交换共享内存的。这样的系统中，进程作为并发的单元，并且使用像fork这样的操作来创建进程是一个天才般的设计主意。如果要在很小的内存中运行一个新的进程，并且没有硬件支持的虚拟内存，那么在内存中的进程就不得不换出到磁盘以创建空间。UNIX操作系统（一种多用户的计算机操作系统）最初仅仅通过简单的父进程交换技术和传递其物理内存给它的子进程来实现fork。这种操作和运行几乎是没有代价的。

相比之下，在Cutler小组开发NT的时代，当时的硬件环境是32位多处理器系统与虚拟内存硬件共享1～16兆字节的物理内存。多处理器为部分程序并行运行提供了可能，因此NT使用进程作为共享内存和数据资源的容器，并使用线程作为并发调度单元。

当然，随后几年里的系统就完全不同于这些环境了。例如拥有64位地址空间并且一个芯片上集成十几个（乃至数百个）CPU内核，存储体系结构中若干GB大小的物理内存以及闪存设备和其他非易失存储存设备的加入，更广泛虚拟化、普适网络的支持，以及例如事件型内存（transactional memory）这类同步技术的创新。Windows和UNIX操作系统无疑将继续适应现实中新的硬件，但我们更感兴趣的是，会有哪些新的操作系统会基于新硬件而被特别设计出来。

2. 作业和纤程

Windows可以将进程分组为作业，但作业抽象并不足够通用。原因是其专为限制分组进程所包含的线程而设计，如通过限制共享资源配额、强制执行受限令牌（restricted token）来阻止线程访问许多系统对象。作业最重要的特性是一旦一个进程在作业中，该进程创建的进程、线程也在该作业中，没有特例。就像它的名字所示，作业是为类似批处理环境而非交互式计算环境而设计的。

一个进程最多属于一个作业。这是有道理的，因为很难去定义一个进程必须服从多个共享配额或限制令牌的情况。但这也意味着，如果有多个系统服务尝试使用作业来管理同一部分进程，则会产生冲突。例如，如果进程首先将自己加入到了一个作业中，或者一个安全的工具已经将其加入了带有一定受限令牌的作业中，则当一个管理工具试图将进程加入其他作业以限制其资源时将会失败。因此在Windows中很少使用作业。

图11-24显示了作业、进程、线程和纤程之间的关系。作业包含进程，进程包含线程，但是线程不包含纤程。线程与纤程通常是多对多的关系。