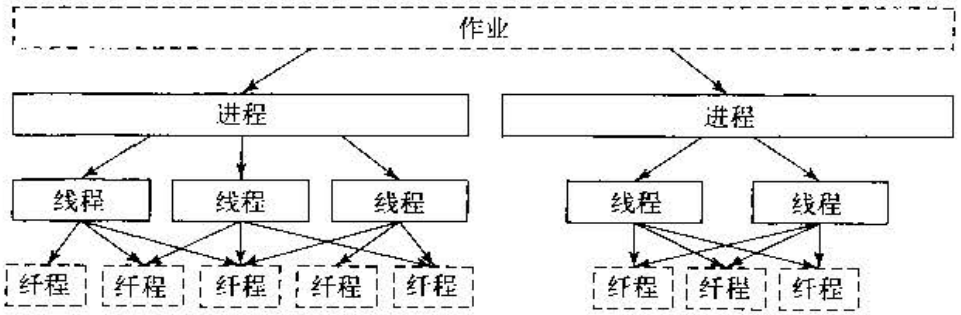

图11-24 作业、进程、线程、纤程之间的关系。作业和纤程是可选的；并不是所有的进程都在作业中或者包含纤程

纤程通过分配栈与用来存储纤程相关寄存器和数据的用户态纤程数据结构来创建。线程被转换为纤程，但纤程也可以独立于线程创建。这些新创建的纤程直到一个已经运行的纤程显式地调用SwitchToFiber函数才开始执行。由于线程可以尝试切换到一个已经在运行的纤程，因此，程序员必须使用同步机制以防止这种情况发生。

纤程的主要优点在于纤程之间的切换开销要远远小于线程之间的切换。线程切换需要进出内核而纤程切换仅需要保存和恢复几个寄存器。

尽管纤程是协同调度的，如果有多个线程调度纤程，则需要非常小心地通过同步机制以确保纤程之间不会互相干扰。为了简化线程和纤程之间的交互，通常创建和能运行它们的内核数目一样多的线程，并且让每个线程只能运行在一套可用的处理器甚至只是一个单一的处理器上。

每个线程可以运行一个独立的纤程子集，从而建立起线程和纤程之间一对多的关系来简化同步。即便如此，使用纤程仍然有许多困难。大多数的Win32库是完全不识别纤程的，并且尝试像使用线程一样使用纤程的应用会遇到各种错误。由于内核不识别纤程，当一个纤程进入内核时，其所属线程可能阻塞。此时处理器会调度任意其他线程，导致该线程的其他纤程均无法运行。因此纤程很少使用，除非从其他系统移植那些明显需要纤程提供功能的代码。图11-25总结了上面提到的这些抽象。

名称	描述	注释
作业	一组共享时间配额和限制的进程	很少使用
进程	持有资源的容器	
线程	内核调度的实体	
纤程	在用户空间管理的轻量级线程	很少使用

图11-25 CPU和资源管理所使用的基本概念

3. 线程

通常每一个进程是由一个线程开始的，但一个新的进程也可以动态创建。线程是CPU调度的基本单位，因为操作系统总是选择一个线程而不是进程来运行。因此，每一个线程有一个调度状态（就绪态、运行态、阻塞态等），而进程没有调度状态。线程可以通过调用指定了在其所属进程地址空间中的开始运行地址的Win32库函数动态创建。

每一个线程均有一个线程ID，其和进程ID取自同一空间，因此单一的ID不可能同时被一个线程和一个进程使用。进程和线程的ID是4的倍数，因为它们实际上是通过用于分配ID的特殊句柄表来执行分配的。该系统复用了如图11-18和图11-19所示的可扩展句柄管理功能。句柄表没有对象的引用，但使用指针指向进程或线程，使通过ID查找一个进程或线程非常有效。最新版本的Windows采用先进先出顺序管理空闲句柄列表，使ID无法马上重复使用。ID马上被重复使用的问题将在本章的最后问题部分再讨论。

线程通常在用户态运行，但是当它进行一个系统调用时，就切换到内核态，并以其在用户态下相同的属性以及限制继续运行。每个线程有两个堆栈，一个在用户态使用，而另一个在内核态使用。任何时候当一个线程进入内核态，其切换到内核态堆栈。用户态寄存器的值以上下文（context）数据结构的形式保存在该内核态堆栈底部。因为只有进入内核态的用户态线程才会停止运行，当它没有运行时该上下文数据结构中总是包括了其寄存器状态。任何拥有线程句柄的进程可以查看并修改这个上下文数据结构。

线程通常使用其所属进程的访问令牌运行，但在某些涉及客户机/服务器计算的情况下，一个服务器线程可能需要模拟其客户端，此时需要使用基于客户端令牌的临时令牌标识来执行客户的操作。（一