般来说服务器不能使用客户端的实际令牌，因为客户端和服务器可运行于不同的系统。）

I/O处理也经常需要关注线程。当执行同步I/O时会阻塞线程，并且异步I/O相关的未完成的I/O请求也关联到线程。当一个线程完成执行，它可以退出，此时任何等待该线程的I/O请求将被取消。当进程中最后一个活跃线程退出时，这一进程将终止。

需要注意的是线程是一个调度的概念，而不是一个资源所有权的概念。任何线程可以访问其所属进程的所有对象，只需要使用句柄值，并进行合适的Win32调用。一个线程并不会因为一个不同的线程创建或打开了一个对象而无法访问它。系统甚至没有记录是哪一个线程创建了哪一个对象。一旦一个对象句柄已经在进程句柄表中，任何在这一进程中的线程均可使用它，即使它是在模拟另一个不同的用户。

正如前面所述，除了用户态运行的正常线程，Windows有许多只能运行在内核态的系统线程，而其与任何用户态进程都没有联系。所有这一类型的系统线程运行在一个特殊的称为*系统进程*的进程中。该进程没有用户态地址空间，其提供了线程在不代表某一特定用户态进程执行时的环境。当学到内存管理的时候，我们将讨论这样的一些线程。这些线程有的执行管理任务，例如写脏页面到磁盘上，而其他形成了工作线程池，来分配并执行部件或驱动程序需要系统进程执行的工作。

11.4.2 作业、进程、线程和纤程管理API调用

新的进程是由Win32 API函数CreatProcess创建的。这个函数有许多参数和大量的选项，包括被执行文件的名称，命令行字符串（未解析）和一个指向环境字符串的指针。其中也包括了控制诸多细节的令牌和数值，这些细节包括了如何配置进程和第一个线程的安全性，调试配置和调度优先级等。其中一个令牌指定创建者打开的句柄是否被传递到新的进程中。该函数还接受当前新进程的工作目录和可选的带有关于此进程使用GUI窗口的相关信息的数据结构。Win32对新进程和其原始线程都返回ID和句柄，而非只为新进程返回一个ID号。

大量的参数揭示了Windows和UNIX在进程创建的开发设计上的诸多的不同之处。

1) 寻找执行程序的实际搜索路径隐藏在Win32的库代码里，但UNIX中则显式地管理该信息。

2) 当前工作目录在UNIX操作系统里是一个内核态的概念，但是在Windows里是用户态字符串。Windows为每个进程都打开当前目录的一个句柄，这导致了和UNIX一样的麻烦：除了碰巧工作目录是跨网络的情况下可以删除它，其他工作目录都是不能删除的。

3) UNIX解析命令行，并传递参数数组；而Win32需要每个程序自己解析参数。其结果是，不同的程序可能采用不一致的方式处理通配符（如*.txt）和其他特殊字符。

4) 在UNIX中，文件描述符是否可以被继承是句柄的一个属性。不过在Windows中，其同时是句柄和进程创建参数的属性。

5) Win32是面向图形用户界面的，因此新进程能直接获得其窗口信息，而在UNIX中，这些信息是通过参数传递给图形用户界面程序的。

6) Windows中的可执行代码没有SETUID位属性，不过一个进程也可以为另一个用户创建进程，只要其能获得该用户的信用标识。

7) Windows返回的进程、线程句柄可以用在很多独立的方法中修改新进程/线程，例如复制句柄、在新进程中设置环境变量等。UNIX则只在fork和exec调用的时候修改新进程。

这些不同有些是来自历史原因和哲学原因。UNIX的设计是面向命令行的，而不是像Windows那样面向图形用户界面的。UNIX的用户相比来说更高级，同时也懂得像PATH环境变量的概念。Windows Vista继承了很多MS-DOS中的东西。

这种比较也有点偏颇，因为Win32是一个用户态下的对NT本地进程执行的包装器，就像UNIX下的系统库函数fork/exec的封装。实际的NT中创建进程和线程的系统调用NtCreateProcess和NtCreateThread比Win32版本简单得多。NT进程创建的主要参数包括代表所要运行的程序文件句柄、一个指定新进程是否默认继承创建者句柄的标志，以及有关安全模型的相关参数。由于用户态下的代码能够使用新建进程的句柄能对新进程的虚拟地址空间进行直接的操作，所有关于建立环境变量、创建初始线程的细节就留给用户态代码来解决。

为了支持POSIX子系统，本地进程创建有一个选项可以指定，通过拷贝另一个进程的虚拟地址空间