同时也可以指定最大值。信号量是一个内核态对象，因此拥有安全描述符和句柄。信号量的句柄可以通过使用DuplicateHandler来进行复制，然后传递给其他进程使得多个进程可以通过相同的信号量来进行同步。在Win32的名字空间中一个信号量也可以被命名，可以拥有一个ACL集合来保护它。有些时候通过名字来共享信号量比通过拷贝句柄更合适。

对up和down的调用也是有的，只不过它们的函数名看起来比较奇怪：ReleaseSemaphore（up）和WaitForSingleObject（down）。可以给WaitForSingleObject一个超时时间，使得尽管此时信号量仍然是0，调用它的线程仍然可以被释放（尽管定时器重新引入了竞态）。WaitForSingleObject和WaitForMultipleObject是将在11.3节中讨论的分发者对象的常见接口。尽管有可能将单个对象的API封装成看起来更加像信号量的名字，但是许多线程使用多个对象的版本，这些对象可能是各种各样的同步对象，也可能是其他类似进程或线程结束、I/O结束、消息到达套接字和端口等事件。

互斥量也是用于同步的内核态对象，但是比信号量简单，因为互斥量不需要计数器。它们其实是锁，上锁的函数是WaitForSingleObject，解锁的函数是ReleaseMutex。就像信号量句柄一样，互斥量的句柄也可以复制，并且在进程间传递，从而不同进程间的线程可以访问同一个互斥量。

第三种同步机制是临界区，实现的是临界区的概念。临界区在Windows中与互斥量类似，但是临界区相对于主创建线程的地址空间来说是本地的。因为临界区不是内核态的对象，所以它们没有显式的句柄或安全描述符，而且也不能在进程间传递。上锁和解锁的函数分别是EnterCriticalSection和LeaveCriticalSection。因为这些API函数在开始的时候只是在用户空间中，只有当需要阻塞的时候才调用内核函数，它们比互斥量快得多。在需要的时候，可以通过合并自旋锁（在多处理器上）和内核同步机制来优化临界区。在许多应用中，大多数的临界区几乎不会被竞争或者只被锁住很短的时间，以至于没必要分配一个内核同步对象，这样会极大地节省内核内存。

我们讨论的最后一种同步机制叫事件，它使用内核态对象。就像我们前面描述的，有两类的事件——通知事件和同步事件。一个事件的状态有两种：收到信号和没收到信号。一个线程通过调用WaitForSingleObject来等待一个事件被信号通知。如果另一个线程通过SetEvent给事件发信号，会发生什么取决于这个事件的类型。对于通知事件来说，所有等待线程都会被释放，并且事件保持在set状态，直到手工调用ResetEvent进行清除；对于同步事件来说，如果有一个或多个线程在等待，那么有且仅有一个线程会被唤醒并且事件被清除。另一个替换的操作是PulseEvent，像SetEvent一样，除了在没有人等待的时候脉冲会丢失，而事件也被清除。相反，如果调用SetEvent时没有等待的线程，那么这个设置动作依然会起作用，被设置的事件处于被信号通知的状态，所以当后面的那个线程调用等待事件的API时，这个线程将不会等待而直接返回。

Win32的API中关于进程、线程、纤程的个数将近100个，其中大量的是各种形式的处理IPC的函数。对上面讨论的总结和另一些比较重要的内容可以参见图11-26。

Win32 API 函数	描　述
CreateProcess	创建一个新的进程
CreateThread	在已存在的进程中创建一个新的线程
CreateFiber	创建一个新的纤程
ExitProcess	终止当前进程及其全部线程
ExitThread	终止该线程
ExitFiber	终止该纤程
SwichToFiber	在当前线程中运行另一纤程
SetPriorityClass	设置进程的优先级类
SetThreadPriority	设置线程的优先级
CreateSemaphore	创建一个新的信号量
CreateMutex	创建一个新的互斥量
OpenSemaphore	打开一个现有的信号量
OpenMutex	打开一个现有的互斥量
WaitForSingleObject	等待一个单一的信号量、互斥量等
WaitForMultipleObjects	等待一系列已有句柄的对象
PulseEvent	设置事件激活，再变成未激活
ReleaseMutex	释放互斥量使其他线程可以获得它
ReleaseSemaphore	使信号量增加1
EnterCriticalSection	得到临界区的锁
leaveCriticalSection	释放临界区的锁

图11-26　一些管理进程、线程以及纤程的一些Win32调用

可以注意到不是所有的这些都是系统调用。其中有一些是包装器，有一些包含了重要的库代码，这些库代码将Win32的接口映射到本地NT接口。另外一些，例如纤程的API，全部都是用户态下的函数，因为就像我们之前提到的，Windows Vista的内核态中根本没有纤程的概念，纤程完全都是由用户态下的库来实