18) 最后，调用NtResumeThread挂起线程，并把这个结构返回给包含所创建的进程和线程的ID、句柄的调用者。

调度

Windows内核没有任何中央调度线程。所以，当一个线程不能够再执行时，线程将进入内核态，调度线程再决定转向的下一个线程。在下面这些情况下，当前正在执行的线程会执行调度程序代码：

1. 当前执行的线程发生了信号量、互斥、事件、I/O等类型的阻塞。

2. 线程向一个对象发信号（如发一个信号或者是唤醒一个事件）时。

3. 配额过期。

第一种情况，线程已经在内核态运行并开始对调度器或输入输出对象执行操作了。它将不能继续执行，所以线程会请求调度程序代码寻找装载下一个线程的CONTEXT记录去恢复其执行。

第二种情况，线程也是在内核中运行。但是，在向一些对象发出信号后，它肯定还能够继续执行，因为发信号对象从来没有受到阻塞。然而，线程必须请求调度程序，来观测它的执行结果是否释放了一个具有更高调度优先级的正准备运行的线程。如果是这样，因为Windows完全是可抢占式的，所以就会发生一个线程切换（例如，线程切换可以发生在任何时候，不仅仅是在当前线程结束时）。但是，在多处理器的情况下，处于就绪状态的线程会在另一个CPU上被调度，那么，即使原来线程拥有较低的调度优先级，也能在当前的CPU上继续执行。

第三种情况，内核态发生中断，这时线程执行调度程序代码找到下一个运行的线程。由于取决于其他等待的线程，可能会选择同样的线程，这样线程就会获得新的配额，可以继续执行。否则发生线程切换。

在另外两种情况下，调度程序也会被调度：

1) 一个输入输出操作完成时。

2) 等待时间结束时。

在第一种情况下，线程可能处于等待输入输出时被释放然后执行。如果不保证最小执行时间，必须检查是否可以事先对运行的线程进行抢占。调度程序不会在中断处理程序中运行（因为那使中断关闭保持太久）。相反，中断处理发生后，DPC会排队等待一会儿。第二种情况下，线程已经对一个信号量进行了down操作或者因一些其他对象而被阻塞，但是定时器已经过期。对于中断处理程序来说，有必要让DPC再一次排队等待，以防止它在定时器中断处理程序时运行。

如果一个线程在这个时刻已到就绪，则调度程序将会被唤醒并且如果新的可运行线程有较高的优先级，那么和情形1的情况类似，当前的线程会被抢占。

现在让我们来看看具体的调度算法。Win32 API提供两个API来影响线程调度。首先，有一个叫SetPriorityClass的用来设定被调用进程中所有线程的优先级。其等级可以是：实时、高、高于标准、标准、低于标准和空闲的。优先级决定进程的先后顺序。（在Vista系统中，进程优先级等级也可以被一个进程用来临时地把它自己标记为后台运行（background）状态，即它不应该被任何其他的活动进程所干扰。）注意优先级是对进程而言的，但是实际上会在每个线程被创建的时候通过设置每个线程开始运行的基本优先级可以影响进程中每条线程的实际优先级。

第二个就是SetThreadPriority。它根据进程的优先级类来设定进程中每个线程的相对优先级（可能地，但是不必然地，调用线程）。可划分如下等级：紧要的、最高的、高于标准的、标准的、低于标准的、最低的和休眠的。时间紧急的线程得到最高的非即时的调度优先，而空闲的线程不管其优先级类别都得到最低的优先级。其他优先级的值依据优先级的等级来定，依次为（+2, +1, 0, −1, −2）。进程优先级等级和相对线程优先级的使用使得能够更容易地确定应用程序的优先级。

调度程序按照下列方式进行调度。系统有32个优先级，从0到31。依照图11-27的表格，进程优先级和相对线程优先级的组合形成32个绝对线程优先级。在表格的数字决定了线程的基本优先级（base priority）。除此之外，每条线程都有当前优先级（current priority），这个当前的优先级可能会高于（但是不低于）前面提到的基本优先级，关于这一点我们稍后将会讨论。