只有当内核中的目标线程被阻塞和被标示为准备接收APC时，用户态下的APC才可调用用户态下的应用程序。但随着用户态堆栈和寄存器的修改，为了执行在ntdll.dll系统库中的APC调度算法，内核将等待中的线程中断，并返回到用户态。APC调度算法调用和I/O操作相关的用户态应用程序。除了一些I/O完成后，作为一种执行代码方法的用户态下的APC外，Win32 API中的QueueUserAPC允许将APC用于任意目的。

执行体也使用除了I/O完成之外的一些APC操作。由于APC机制精心设计为只有当它是安全的时候才提供APC，它可以用来安全地终止线程。如果这不是一个终止线程的好时机，该线程将宣布它已进入一个临界区，并延期交付APC直至得到许可。在获得锁或其他资源之前，内核线程会标记自己已进入临界区并延迟APC，这时，它们不能被终止，并仍然持有资源。

5. 调度对象

另一种同步对象是调度对象。这是常用的内核态对象（一种用户可以通过句柄处理的类型），它包含一个称为dispatcher_header的数据结构，如图11-15所示。

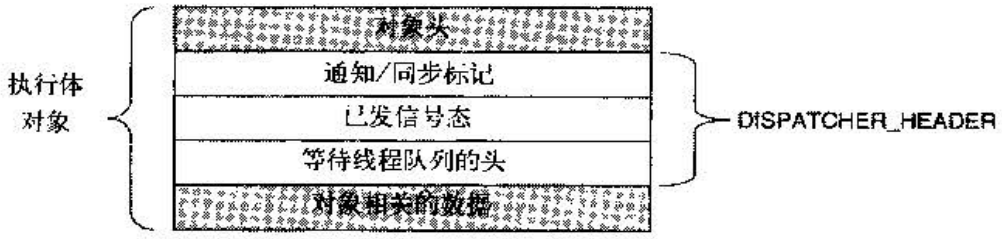

图11-15 执行对象中嵌入的dispatcher_header数据结构

它们包括信号器、互斥体、事件、可等待定时器和其他一些可以等待其他线程同步执行的对象。它们还包括表示打开的文件的对象、进程、线程和IPC端口。调度数据结构包含了表示对象状态的标志，和等待被标记的对象的线程队列。

同步原语，如信号器，是标准的调度对象。另外定时器、文件、端口线程和进程使用调度对象机制去通知。当一个定时器开启、一个文件I/O完成、一个端口正在传输数据或是一个线程或进程终止时，相关的调度对象会被通知，并唤醒所有等待该事件的线程。

由于Windows使用了一个单一的标准机制去同步内核态对象，一些专门的API就无需再等待事件，例如在UNIX中用来等待子进程的wait3。而通常情况下，线程要一次等待多个事件。在UNIX中，通过“select”系统调用，一个进程可以等待任何一个64位网络接口可以获得的数据。在Windows中亦有一个类似的APIWaitForMultipleObjects，但是它允许一个线程等待任何类型的有句柄的调度对象。超过64个句柄可以指定WaitForMultipleObjects，以及一个可选择的超时值。线程随时准备运行任何一个和句柄标记相关的事件或发生超时。

内核使用两个不同的程序使得线程等待调度对象运行。发出一个通知对象信号使每一个等待的线程可以运行。同步对象仅使第一个等待的线程可以运行，用于调度对象，实施锁元，如互斥体。当一个线程等待一个锁再次开始运行，它做的第一件事就是再次尝试请求锁。如果一次仅有一个线程可以保留锁，其他所有可运行的线程可能立刻被阻塞，从而产生许多不必要的现场交换。使用同步机制和使用通知机制的分派对象（dispatcher object）之间的差别是dispatcher_header结构中的一个标记。

另外，在Windows代码中互斥体称为“变体”（mutant）。因为当一个线程保留一个出口时，它们需要执行OS/2语义中的非自动解锁，看来这是Cutler奇特的考虑。

6. 执行体

如图11-13所示，在NTOS的内核层以下是执行体。执行体是用C语言编写的，在结构上最为独立（内存管理是一个明显的例外），并且经过少量的修改已经移植到新的处理器上（MIPS、x86、PowerPC、Alpha、IA64和x64）。执行体包括许多不同的组件，所有的组件都通过内核层提供的抽象控制器来运行。

每个组件分为内部和外部的数据结构和接口。每个组件的内部方法是隐藏的，只有组件自己可以调用，而外部方法可以由执行体的所有其他组件调用。外部接口的一个子集由一个ntoskrnl.exe提供，而且设备驱动可以链接到它们就好像执行体是一个库。微软称许多执行体组件为“管理器”，因为每一个组