**深度探索I/O完成端口**

应用程序将IOCompletion执行体对象当作与多个文件句柄相关的I/O完成的核心。一旦一个文件与一个完成端口相关联，任何在此文件上异步I/O操作的完成都会导致一个完成通知包（completion notification packet）加入到完成端口队列。一个线程只需简单的等待一个完成通知包被排队到此完成端口上，就可以等待在多个文件上的所有正在进行之中的I/O操作的完成事件。Windows API中的WaitForMultipleObjects 提供了类似的功能，但完成端口的优点在于在系统的协助下发挥高效的并发性。这里的并发性可以理解为应用程序主动处理客户请求的线程的数量的多少。

当应用程序创建一个完成端口时，需要设定并发量。该数值指示了在任何给定时候正在运行的与该端口相关联的线程的最大数量。正如前面所提到的，理想情况是在任何给定的时刻，系统中每个处理器都有一个线程在运行。Windows利用与一个端口相关联的并发值参数来控制一个应用程序中活动线程的数量。如果与一个端口相关的活动线程数达到并发值，那么，在这个端口上等待的线程将不允许再运行了。相反，它将等待某个活动线程处理完当前操作并检查是否有别的包正在该端口上等待。如果有的话，该线程只是简单的抓获该包然后处理。在这个过程中，没有上下文切换，CPU得到最大限度的利用。

创建完成端口需要调用Windows API CreateIoCompletionPort：

HANDLE CreateIoCompletionPort(

HANDLE FileHandle,

HANDLE ExistingCompletionPort,

DWORD CompletionKey,

DWORD NumberOfConcurrentThreads

);

创建一个完成端口时，通常对参数ExistingCompletionPort赋值NULL， NumberOfConcurrentThreads参数定义了在完成端口上同时允许执行的线程数量。如果有文件句柄传递给FileHandle参数，则该文件与完成端口关联在了一起。当这个文件上的I/O请求完成时，一个完成通知包将被投递到完成端口消息队列中。另外一个API GetQueuedCompletionStatus是用来获取排队完成状态，它使调用线程挂起，直到收到一个完成通知包。

BOOL GetQueuedCompletionStatus(

HANDLE CompletionPort,

LPDWORD lpNumberOfBytesTransferred,

LPDWORD CompletionKey,

LPOVERLAPPED\* lpOverlapped,

DWORD dwMiillisecondTimeout

);

完成端口实际上是在管理一个线程池，它会记录当前活动（即没有被I/O等事件阻塞）的线程数。当有完成通知包到达该端口时，在该端口上等待的线程按照后进先出（LIFO）的次序被唤醒，因此最近（most recently）被阻塞的线程就是获得下一个完成通知包的线程。那些长时间得不到响应的的线程的堆栈将会被从内存调到磁盘交换区去等待，当与一个端口关联的线程太多超过了当前的处理能力时，就可以将长时间阻塞的线程占用的内存减到最少。

服务器应用程序往往通过网络端点来接受客户请求，而这些网络端点是由文件句柄来表示的。这样的例子包括Windows Sockets 2(Winsock2)套接字或者命名管道。当服务器创建它的通信端点时，它将这些通信端点与一个完成端口关联起来，并且它的线程通过调用GetQueuedCompletionStatus来等待此端口上进来的完成通知。当一个线程在此完成端口上得到一个I/O完成通知包时，它便不再等待，开始处理I/O结果数据，从而变成一个活动的线程。一个线程在处理过程中可能将阻塞很多次，比如当它需要从磁盘上的文件读取数据时，或者当它需要与其他的线程同步时。Windows NT检测到这些活动，并且识别出该完成端口上至少已经有一个活动线程。因此，当活动线程由于I/O请求而阻塞时，如果在队列中存在一个包，则唤醒另一个正在此完成端口上等待的线程提供处理服务。

微软的指导原则是，将并发值设置成大约等于该系统中处理器的数目。但是要注意，一个完成端口上实际活动线程数量有可能超过设置的并发值。考虑并发值被设置为1的情况，一个客户请求进来了，某个线程因为被调度来处理该请求而变成活动的。下一个请求到达时，正在该端口上等待的另一个线程却不允许执行，因为活动的线程数已经达到了设置的并发上限值。然后，当活动线程需要等待I/O而阻塞时，等待的线程将被激活，当它尚在活动时，上一个线程的I/O完成了，这使得它继续保持活动状态（继续执行数据处理服务）。此刻，一直到两个线程中有一个被阻塞，并发值始终是2，高于设置的并发上限值1。大多数时候，活动线程数将维持在设置的并发限制值上，或者超过一点。

应用程序通过调用PostQueuedCompletionStatus这个API向完成端口投递一个自定义的完成通知包。服务器一般通过该函数发送消息通知线程有外部事件发生，例如需要温和的关机。

完成端口内部机制

当传递NULL值给ExistingCompletionPort参数来调用CreateIoCompletionPort来创建完成端口时，将调用同名的NtCreateIoCompletion系统服务。实质上，IOCompletion对象是建立在一个称为队列的内核同步对象基础上。系统创建一个完成端口的同时，在完成端口所分配到的内存中初始化一个队列对象（指向完成端口的指针同时指向了此队列对象，因为队列对象位于完成端口对象内存的开始处）。当一个线程调用CreateIoCompletionPort来创建完成端口时，第四个参数NumberOfConcurrentThreads即为队列的并发值。NtCreateIoCompletion函数将调用KeInitializeQueue系统服务来初始化该端口的消息队列。

当应用程序再次调用CreateIoCompletionPort时，将调用NtSetInformationFile服务来使参数一（文件句柄）与参数二（一个已有的完成端口）关联起来。完成通知包FileCompletionInformation包含的信息：CreateIoCompletionPort的参数二ExistingCompletionPort（已有的完成端口句柄）和参数三CompletionKey（完成键）。NtSetInformationFile通过解引用操作从该文件句柄获得对应的文件对象，并且申请一个记录完成上下文的数据结构。这个数据结构在NTDDK.H定义如下：

typedef struct \_IO\_COMPLETION\_CONTEXT {

PVOID Port;

ULONG Key;

} IO\_COMPLETION\_CONTEXT, \*PIO\_COMPLETION\_CONTEXT;

最后，将调用NtSetInformationFile系统服务设置文件对象中CompletionContext域的值。当一个异步I/O在一个文件对象上完成时，系统内部执行具有I/O管理功能的IopCompleteRequest系统服务，检查文件对象中的CompletionContext域是否为非NULL。如果是，则I/O管理器生成一个完成通知包，通过调用KeInsertQueue系统服务将完成通知包投递到完成端口队列（注意，完成端口对象和队列对象是同义的）。

当一个服务器线程调用GetQueuedCompletionStatus时，它将调用NtRemoveIoCompletion系统服务。在验证参数后，并且将完成端口句柄转换成一个指向该端口的指针后，NtRemoveIoCompletion调用KeRemoveQueue。

正如你所看到的，KeRemoveQueue和KeInsertQueue是完成端口模型的两个引擎级函数，它们决定阻塞在完成端口上等待I/O完成通知包的线程什么时候被唤醒。在系统内部，队列对象维护了完成端口上当前活动线程的计数值，以及最大的并发活动线程的数量。当一个线程调用KeRemoveQueue并且当前活动线程数大于或等于并发数上限时，那么该线程将被投放到一个阻塞线程队列（按LIFO顺序）中，等待系统调度来获取并处理完成通知包。此线程列表挂在队列对象的外面，线程的控制块数据结构中有一个指针引用了一个与之相关的队列对象；如果这个指针为NULL，则该线程没有与队列关联。

Windows依赖与线程控制块中的队列指针来跟踪和记录那些“由于被阻塞在除了完成端口之外的其他事情上而变成不活动”的线程。那些有可能会导致一个线程阻塞的调度例程（例如KeWaitForSingleObject，KeDelayExecutionThread等等）要检查该线程的队列指针。如果该指针不为NULL，则这些函数调用KiActivateWaiterQueue—一个与队列相关的函数，它会递减与该队列相关联的活动线程的计数值。如果计数值递减到小于设置的并发值，并且此时至少有一个完成通知包在该队列中，那么处于该队列的线程列表最前面的那个线程被唤醒，并且把最老的（the oldest）完成通知包交给它处理。相反，无论何时，与一个队列相关联的线程在阻塞之后被唤醒时，调度程序执行KiUnwaitThread函数来增加该队列上活动线程的计数值。

最后，PostQueuedCompletionStatus这个Windows API将调用NtSetIoCompletion服务。该函数只是简单的调用KeInsertQueue将自定义的完成通知包插入到完成端口的队列中。

Windows NT的完成端口API提供了一种易于使用和高效的方法最大限度地发挥服务器的性能——最大限度的减少上下文切换的同时最大限度的提高系统并发量。这些API使我们能够调用I/O管理器和内核提供的一些服务功能。队列对象可以被设备驱动程序调用（这些接口尽管没有公开，但还是很容易查询到的），不过完成端口的API没有提供相关访问功能。但是，如果队列接口被继承，我们完全可以通过编写队列处理程序并通过手动设置CompletionContext的值来模拟完成端口模型。