陷入内核态之后不需要改变内存映射。所有要做的只是切换到线程的内核栈。由于进程在用户态下的页面仍然是可访问的，内核态下的代码在读取参数和访问缓冲时，就不用在地址空间之间来回切换、或者临时将页面进行两次映射。这里的权衡是通过用较小的进程私有地址空间，来换取更快的系统调用。

当运行在内核态的时候，Windows允许线程访问其余的地址空间。这样该线程就可以访问所有用户态的地址空间，以及对该进程来说通常不可访问的内核地址空间中的区域，例如页表的自映射区域。在线程切换到用户态之前，必须切换到它最初的地址空间。

1. 虚拟地址分配

虚拟地址的每页处于三种状态之一：无效、保留或提交。*无效页面*（invalid page）是指一个页面没有被映射到一个*内存区对象*（section object），对它的访问会引发一个相应的页面失效。一旦代码或数据被映射到虚拟页面，就说一个页面处于*提交*（committed）状态。在提交的页上发生页面失效会导致如下情况：将一个包含了引起失效的虚拟地址的页面映射到这样的页面——由内存区对象所表示，或被保存于页面文件之中。这种情况通常发生在需要分配物理页面，以及对内存区对象所表示的文件进行I/O来从硬盘读取数据的时候。但是页面失效的发生也可能是页表正在更新而造成的，即物理页面仍在内存的高速缓存中，这种情况下不需要进行I/O。这些叫做*软异常*（soft fault），稍后我们会更详细地讨论它们。

虚拟页面还可以处于保留的（reserved）状态。保留的虚拟页是无效的，但是这些页面不能被内存管理器用于其他目的而分配。例如，当创建一个新线程时，用户态栈空间的许多页保留于进程的虚拟地址空间，仅有一个页面是提交的。当栈增长时，虚拟内存管理器会自动提交额外的页面，直到保留页面耗尽。保留页面的功效是：可以保证栈不会太长而覆盖其他进程的数据。保留所有的虚拟页意味着栈最终可以达到它的最大尺寸；而栈所需要的连续虚拟地址空间的页面，也不会有用于其他用途的风险。除了无效、保留、提交状态，页面还有其他的属性：可读、可写及可运行（在AMD64兼容的处理器下）。

2. 页面文件

关于后备存储器的分配有一个有趣的权衡，已提交页面没有被映射于特定文件。这些页使用了*页面文件*（pagefile）。问题是该如何以及何时把虚拟页映射到页面文件的特定位置。一个简单的策略是：当一个页被提交时，为虚拟页分配一个硬盘上页面文件中的页。这会确保对于每一个有必要换出内存的已提交页，都有一个确定的位置写回去。

Windows使用一个适时（just-in-time）策略。直到需要被换出内存之前，在页面文件中的具体空间不会分配给已提交的页面。硬盘空间当然不需要分配给永远不换出的页面。如果总的虚拟内存比可用的物理内存少，则根本不需要页面文件。这对基于Windows的嵌入式系统是很方便的。这也是系统启动时的方式，因为页面文件是在第一个用户态进程smss.exe启动之后才初始化的。

在预分配策略下，用于私有数据（如栈、写时复制代码页）的全部虚拟内存受到页面文件大小的限制。通过适时分配的策略，总的虚拟内存大小是物理内存和页面文件大小的总和。既然相对物理内存来说硬盘足够大与便宜，提升性能的需求自然比空间的节省更重要。

有关请求调页，需要马上进行初始化从硬盘读取页的请求——因为在页入（page-in）操作完成之前，遇到页面失效的线程无法继续运行下去。对于失效页面的一个可能的优化是：在进行一次I/O操作时预调入一些额外的页面。然而，对于修改过的页写回磁盘和线程的执行一般并不是同步的。对于分配页面文件空间的适时策略便是利用这一点，在将修改过的页面写入页面文件时提升性能：修改过的页面被集中到一起，统一进行写入操作。由于只有当页面被写回时页面文件的空间才真正被分配，可以通过排列使页面文件中的页面较为接近甚至连续，来对大批写回页面时的寻找次数进行优化。

当存储在页面文件中的页被读取到内存中时，直到它们第一次被修改之前，这些页面一直保持它们在页面文件中的位置。如果一个页面从没被修改过，它将会进入到一个空闲物理页面的列表中去——这个表称作*后备链表*（standby list），这个表中的页面可以不用写回硬盘而再次被使用。如果它被修改，内存管理器将会释放页面文件中的页，并且内存将保留这个页的惟一副本。这是内存管理器通过把一个加载后的页标识为只读来实现的。线程第一次试图写一个页时，内存管理器检测到它所处的情况并释放页面文件中的页，再授权写操作给相应的页，之后让线程再次进行尝试。

Windows支持多达16个页面文件，通常覆盖到不同的磁盘来达到较高的I/O带宽。每一个页面文件