一个基本块，最后一条指令被一条跳转或调用已翻译好的基本块的指令所代替，从而节省了寻找后续基本块的开销。同样，在用户程序中不需要替换掉敏感指令；因为硬件会直接忽略它们。

讲到这里，即使在不可虚拟化的硬件上，II型管理程序也能正常工作的原因就已经很清楚了：所有的敏感指令被仿真这些指令的过程调用所替代。客户操作系统发射的敏感指令不会被真正的硬件执行。它们转换成了对管理程序的调用，而这些调用仿真了那些敏感指令。

有人可能会天真地认为支持VT技术的处理器在性能上会胜过II型管理程序所使用的软件技术，但是测量结果显示情况并不是这么简单（Adams 和Agesen，2006）。其结果显示，支持VT技术的硬件使用陷入—仿真的方法会引起太多的陷入，而在现代硬件上，陷入的代价是非常昂贵的，它们会清空处理器内的缓存、TLB和分支预测表。相反，当可执行程序中的敏感指令被VMware过程调用所替代，就不会招致这些切换开销。正如Adams 和Agesen所指出的，根据工作负载的不同，软件有的时候会击败硬件。由于这个原因，一些I型管理程序出于对性能的考虑会进行二进制翻译，尽管即使不进行转换，运行于其上的软件也可以正确运行。

8.3.4 准虚拟化

运行在I型和II型管理程序之上的都是没有修改过的客户操作系统，但是这两类管理程序为了获得合理的性能都备受煎熬。另一个逐渐开始流行起来的处理方法是更改客户操作系统的源代码，从而略过敏感指令的执行，转而调用管理程序调用。事实上，对客户操作系统来说就像是用户程序调用操作系统（管理程序）系统调用一样。当采用这种方法时，管理程序必须定义由过程调用集合组成的接口以供客户操作系统使用。这个过程调用集合实际上形成了API（应用程序编程接口），尽管这个接口是供客户操作系统使用，而不是应用程序。

再进一步，从操作系统中移除所有的敏感指令，只让操作系统调用管理程序调用（hypervisor call）来获得诸如I/O操作等系统服务，通过这种方式我们就已经把管理程序变成了一个微内核，如图1-26所示。一些或全部敏感指令有意移除的客户操作系统称为准虚拟化的（paravirtualized）（Barham 等人，2003；Whitaker 等人，2002）。仿真特殊的机器指令是一件让人厌倦的、耗时的工作。它需要调用管理程序，然后仿真复杂指令的精确语义。让客户操作系统直接调用管理程序（或者微内核）完成I/O操作等任务会更好。之前的管理程序都选择模拟完整的计算机，其主要原因在于客户操作系统的源代码不可获得（如Windows）、或源代码种类太多样（如Linux）。也许在将来，管理程序/微内核的API接口可以标准化，然后后续的操作系统都会调用该API接口而不是执行敏感指令。这样的做法将使得虚拟机技术更容易被支持和使用。

全虚拟化和准虚拟化之间的区别如图8-27所示。在这里，我们有两台虚拟机运行在支持VT技术的硬件上。左边，客户操作系统是一个没有经过修改的Windows版本。当执行敏感指令的时候，硬件陷入到管理程序，由管理程序仿真执行它随后返回。右边，客户操作系统是一个经过修改的Linux版本，其中不含敏感指令。当它需要进行I/O操作或修改重要内部寄存器（如指向页表的寄存器）时，它调用管理程序例程来完成这些工作，就像在标准Linux系统中应用程序调用操作系统系统调用一样。

如图8-27所示，管理程序被一条虚线分成两个部分。而在现实中，只有一个程序在硬件上运行。它的一部分用来解释陷入的敏感指令，这种情况下，请参照Windows一边。另一部分用来执行管理程序例程。在图8-27中，后一部分被标记为“微内核”。如果管理程序只是用来运行准虚拟化的客户操作系统，就不需要对敏感指令进行仿真，这样，我们就获得了一个真正的微内核，这个微内核只提供最基本的服务，诸如进程分派、管理MMU等。I型管理程序和微内核之间的界限越来越模糊，当管理程序获得越来越多的功能和例程时，这个界限变得更加不清晰。这个主题是有争议的，但是这一点越来越明确：以内核态运行在硬件上的程序应当短小、可靠，由数千行代码而不是数百万行代码组成。这个话题已经经过很多学者的讨论（Hand 等人，2005；Heiser等人，2006；Hohmuth 等人，2004；Roscoe 等人，2007）。

对客户操作系统进行准虚拟化引起了很多问题。第一，如果所有的敏感指令都被管理程序例程所代替，操作系统如何在物理机器上运行呢？毕竟，硬件不可能理解管理程序例程。第二，如果市场上有很多种管理程序，例如Vmware、剑桥大学开发的开源项目Xen、微软的Viridian，这些管理程序的API接口不同，应该怎么办呢？怎样修改内核使它能够在所有的管理程序上运行？