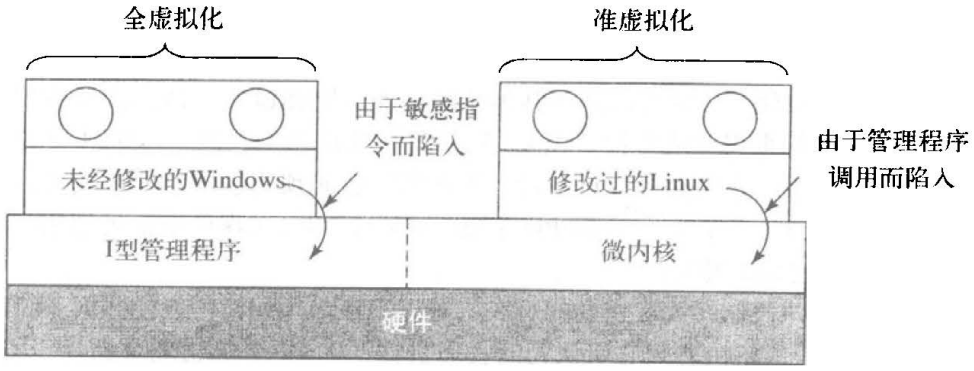

图8-27 支持全虚拟化和准虚拟化的管理程序

Amsden 等人（2006）提出了一个解决方案。在他们的模型当中，当内核需要执行一些敏感操作时会转而调用特殊的例程。这些特殊的例程，称作VMI（虚拟机接口），形成的低层与硬件或管理程序进行交互。这些例程被设计得通用化，不依赖于硬件或特定的管理程序。

这种技术的一个示例如图8-28所示，这是一个准虚拟化的Linux版本，称为VMI Linux（VMIL）。当VMI Linux运行在硬件上的时候，它链接到一个发射敏感指令来完成工作的函数库，如图8-28a所示。当它运行在管理程序上，如VMware或Xen，客户操作系统链接到另一个函数库，该函数库提供对下层管理程序的适当（或不同）例程调用。通过这种方式，操作系统的内核保持了可移植性和高效性，可以适应不同的管理程序。

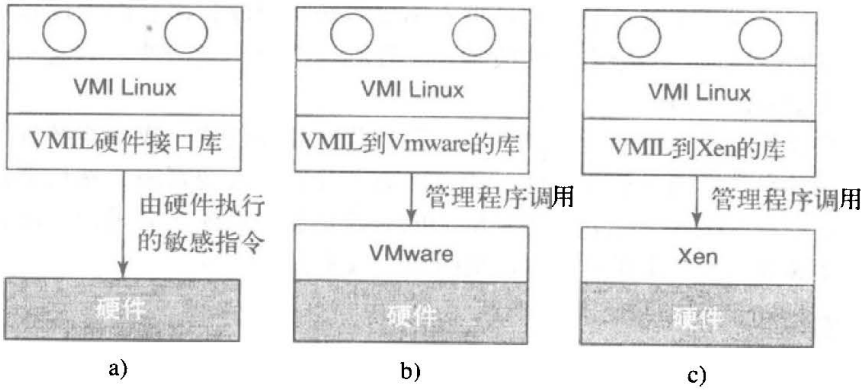

图8-28 VMI Linux运行在：a) 硬件裸机；b) VMware；c) Xen上

关于虚拟机接口还有很多其他的建议。其中比较流行的一个是paravirt ops。它的主要思想与我们上面所介绍的相似，但是在细节上有所不同。

8.3.5 内存的虚拟化

现在我们已经知道了如何虚拟化处理器。但是一个计算机系统不止是一个处理器。它还有内存和I/O设备。它们也需要虚拟化。让我们来看看它们是如何实现的。

几乎全部的现代操作系统都支持虚拟内存，即从虚拟地址空间到物理地址空间的页面映射。这个映射由（多级）页表所定义。通过操作系统设置处理器中的控制寄存器，使之指向顶级页表，从而动态设置页面映射。虚拟化技术使得内存管理更加复杂。

例如，一台虚拟机正在运行，其中的客户操作系统希望将它的虚拟页面7、4、3分别映射到物理页面10、11、12。它建立包含这种映射关系的页表，加载指向顶级页表的硬件寄存器。这条指令是敏感指令。在支持VT技术的处理器上，将会引起陷入；在VMware管理程序上，它将会调用VMware例程；在准虚拟化的客户操作系统中，它将会调用管理程序调用。简单地讲，我们假设它陷入到了I型管理程序中，但实际上在上述三种情况下，问题都是相同的。

那么管理程序会怎么做呢？一种解决办法是把物理页面10、11、12分配给这台虚拟机，然后建立真实的页表使之分别映射到该虚拟机的虚拟页面7、4、3，随后使用这些页面。到目前为止还没有问题。