I/O和虚拟内存部件协作加载设备驱动程序至内核存储器并将它们连接到NTOS和HAL层。I/O管理器提供发现、组织和操作设备的接口，包括安排加载适当的设备驱动程序等。大多数管理设备和驱动器的配置信息都保留在注册表的系统储集中。I/O管理器的即插即用下层部件保留硬件储集内检测出的硬件信息，该储集是保留在内存中的可变储集而非存在于硬盘中，系统每次引导都会重新创建。

以下将详细介绍操作系统的不同部件。

1. 硬件抽象层

正如之前发布的基于NT的Windows系统一样，Windows Vista的目标之一是使得操作系统在不同的硬件平台之间具有可移植性。理想情况下，如果需要在一种新型计算机系统中运行该操作系统，仅仅需要在首次运行时使用新机器编译器重新编译操作系统即可。但实际上并没有那么简单。操作系统各层有大量部件具有很好的可移植性（因为它们主要处理支持编程模式的内部数据结构和抽象，从而支持特定的编成模式），其他层就必须处理设备寄存器、中断、DMA以及机器与机器间显著不同的其他硬件特征。

大多数NTOS内核源代码由C语言编写而非汇编语言（x86中仅2%是汇编语言，比x64少1%）。然而，所有这些C语言代码都不能简单地从x86系统中移植到一个SPARC系统，然后重新编译、重新引导，因为与不同指令集无关并且不能被编译器隐藏的处理机结构及其硬件有很多不同。像C这样的语言难以抽象硬件数据结构和参数，如页表输入格式、物理存储页大小和字长等。所有这些以及大量的特定硬件的优化即使不用汇编语言编写，也将不得不手工处理。

大型服务器的内存如何组织或者何种硬件同步基元是可获得的，与此相关的硬件细节对系统较高层都有比较大的影响。例如，NT的虚拟内存管理器和内核层了解涉及内存和内存位置的硬件细节。在整个系统中，NT使用的是比较和交换同步基元，对于没有这些基元的系统是很难移植上去的。最后，系统对字内的字节分类系统存在很多相关性。在所有NT原来移植到的平台上，硬件是设置为小端（little-endian）模式的。

除了以上这些影响便携性的较大问题外，不同制造商的不同母板还存在大量的小问题。CPU版本的不同会影响同步基元的实现方式。各种支持芯片组也会在硬件中断的优先次序、I/O设备寄存器的存取、DMA转换管理、定时器和实时时钟控制、多处理器同步、BIOS设备(如ACPI)的工作等方面产生差异。微软尝试通过最下端的HAL层隐藏对这些设备类型的依赖。HAL的工作就是对这些硬件进行抽象、隐藏处理器版本、支持芯片集和其他配置变更等具体细节。这些HAL抽象展现为NTOS和驱动可用的独立于机器的服务。

使用HAL服务而不直接写硬件地址，驱动器和内核在与新处理器通信时只需要较小改变，而且在多数情况下，尽管版本和支持芯片集不同但只要有相同的处理器结构，系统中所有部件均无需修改就可运行。

HAL对诸如键盘、鼠标、硬盘等特殊的I/O设备或内存管理单元不提供抽象或服务。这种抽象功能广泛应用于整个内核态的各部件，如果没有HAL，通信时即使硬件间很小的差异也会造成大量代码的重大修改。HAL自身的通信很简单，因为所有与机器相关的代码都集中在一个地方，移植的目标就很容易确定：即实现所有的HAL服务。很多版本中，微软都支持HAL扩展工具包，允许系统制造者生产各自的HAL从而使得其他内核部件在新系统中无需更改即可工作，当然这要在硬件更改不是很大的前提下。

通过内存映射I/O与I/O端口的对比可以更好地了解硬件抽象层是如何工作的。一些机器有内存映射I/O，而有的机器有I/O端口。驱动程序是如何编写的呢？是不是使用内存映射I/O？无需强制做出选择，只需要判断哪种方式使驱动程序可独立于机器运行即可。硬件抽象层为驱动程序编写者分别提供了三种读、写设备寄存器的程序：

```
uc=READ_PORT_UCHAR(port);   WRITE_PORT_UCHAR(port,uc);
us=READ_PORT_USHORT(port);  WRITE_PORT_ USHORT (port,us);
ul=READ_PORT_ULONG(port);   WRITE_PORT_ULONG(port,ul);
```

这些程序各自在指定端口读、写无符号8、16、32位整数，由硬件抽象层决定这是否需要内存映射I/O。这样，驱动程序可以在设备寄存器实现方式有差异的机器间使用而不需要修改。

驱动程序会因为不同目的而频繁存取特定的I/O设备。在硬件层，一个设备在确定的总线上有一个或多个地址。因为现代计算机通常有多个总线（ISA、PCI、PCI-X、USB、1394等），这就可能造成不