同总线上的多个设备有相同的地址，因此需要一些方法来区别它们。HAL把与总线相关的设备地址映射为系统逻辑地址并以此来区分设备。这样，驱动程序就无需知道何种设备与何种总线相关联。这种机制也保护了较高层避免进行总线结构和地址规约的交替。

中断也存在相似的问题——总线依赖性。HAL 同样提供服务在系统范围内命名中断，并且允许驱动程序将中断服务程序附在中断内而无需知道中断向量与总线的关系。中断请求管理也受HAL控制。

HAL提供的另一个服务是在设备无关方式下建立和管理DMA转换，对系统范围和专用I/O卡的DMA引擎进行控制。设备由其逻辑地址指示。HAL实现软件的散布/聚合（从不相邻的物理内存块的地方写或者读）。

HAL也是以用一种可移植的方式来管理时钟和定时器的。定时器是以100纳秒为单位从1601年1月1日开始计数的，因为这是1601年的第一天，简化了闰年的计算。（一个简单测试：1800年是闰年吗？ 答案：不是。）定时器服务和驱动程序中的时钟运行的频率是解耦合的。

有时需要在底层实现内核部件的同步，尤其是为了防止多处理机系统中的竞争环境。HAL提供基元管理同步，如旋转锁，此时一个CPU等待其他CPU释放资源，比较特殊的情况是资源被几个机器指令占有。

最终，系统引导后，HAL和BIOS通信，检查系统配置信息以查明系统所包含的总线、I/O设备及其配置情况，同时该信息被添加进注册表。HAL工作情况摘要如图11-14所示。

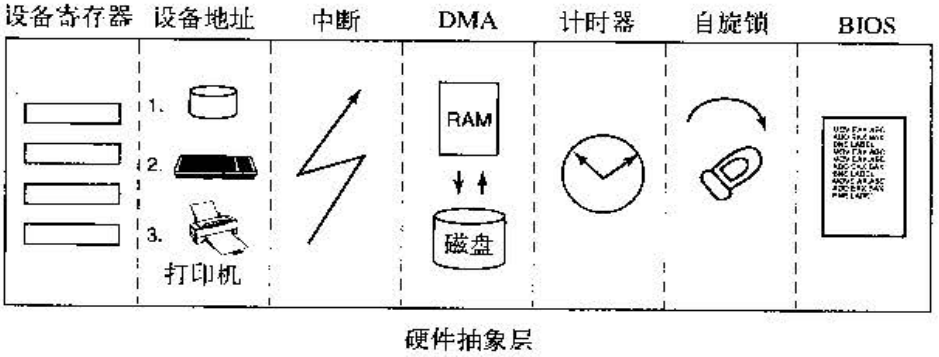


图11-14 一些HAL管理相关的硬件功能

2. 内核层

在硬件抽象层之上是NTOS，包括两层：内核和执行体。“内核”在Windows中是一个易混淆的术语。它可以指运行在处理机内核态下的所有代码，也可以指包含了Windows操作系统核心NTOS的ntoskrnl.exe文件，还可以指NTOS里的内核层，在本章中我们使用这个概念。此外，“内核”甚至用来命名用户态下提供本地系统调用的封装器的Win32库：kernel32.dll

Windows操作系统的内核层（如图11-13所示，执行体之上）提供了一套管理CPU的抽象。最核心的抽象是线程，但是内核也实现了异常处理、陷阱以及各种中断。支持线程的数据结构的创建和终止是在执行体实现的。核心层负责调度和同步线程。在一个单独的层内支持线程，允许执行体在用户态下，可以通过使用用来编写并行代码且相同优先级的多线程模型来执行，但同步原语的执行更专业。

内核线程调度程序负责决定哪些线程执行在系统的每一个CPU上。线程会一直执行，直到产生了一个定时器中断，或者是当线程需要等待一些情况，比如等待一个I/O读写完成或是一个锁定被释放，或者是更高优先级的线程等待运行而需要CPU，这时正在执行的线程会切换到另一个线程（量子过期）。当一个线程向另一个线程转换时，调度程序会在CPU上运行，并确保寄存器及其他硬件状态已保存。然后，调度程序会选择另一个线程在CPU上运行，并且恢复之前所保存的最后一个线程的运行状态。

如果下一个运行的线程是在一个不同的地址空间（例如进程），调度程序也必须改变地址空间。详细的调度算法我们将在本章内谈到进程和线程时讨论。

除了提供更高级别的硬件抽象和线程转换机制，核心层还有另外一项关键功能：提供对下面两种同步机制低级别的支持：control对象和dispatcher对象。Control对象，是核心层向执行体提供抽象的CPU管理的一种数据结构。它们由执行体来分配，但由核心层提供的例程来操作。Dispatcher对象是一种普通执行对象，使用一种公用的数据结构来同步。