不要隐藏能力。

如果硬件具有极其高效的方法做某事，它就应该以简单的方法展露给程序员，而不应该掩埋在某些其他抽象的内部。抽象的目的是隐藏不合需要的特性，而不是隐藏值得需要的特性。例如，假设硬件具有一种特殊的方法以很高的速度在屏幕上（也就是视频RAM中）移动大型位图，正确的做法是要有一个新的系统调用能够得到这一机制，而不是只提供一种方法将视频RAM读到内存中并且再将其写回。新的系统调用应该只是移动位而不做其他事情。如果系统调用速度很快，用户总可以在其上建立起更加方便的接口。如果它的速度慢，没有人会使用它。

另一个设计问题是面向连接的调用与无连接的调用。读文件的标准UNIX系统调用和Win32系统调用是面向连接的。首先你要打开一个文件，然后读它，最后关闭它。某些远程文件访问协议也是面向连接的。例如，要使用FTP，用户首先要登录到远程计算机上，读文件，然后注销。

另一方面，某些远程文件访问协议是无连接的，例如Web协议（HTTP）。要读一个Web页面你只要请求它就可以了；不存在事先建立连接的需要（TCP连接是需要的，但是这处于协议的低层；访问Web本身的HTTP协议是无连接的）。

任何面向连接的机制与无连接的机制之间的权衡在于建立连接的机制（例如打开文件）要求的额外开销，以及在后续调用（可能很多）中避免进行连接所带来的好处。对于单机上的文件I/O而言，由于建立连接的代价很低，标准的方法（首先打开，然后使用）可能是最好的方法。对于远程文件系统而言，两种方法都可以采用。

与系统调用接口有关的另一个问题是接口的可见性。POSIX强制的系统调用列表很容易找到。所有UNIX系统都支持这些系统调用，以及少数其他系统调用，但是完全的列表总是公开的。相反，Microsoft从未将Windows Vista系统调用列表公开。作为替代，Win32 API和其他API被公开了，但是这些API包含大量的库调用（超过10 000个），只有很少数是真正的系统调用。将所有系统调用公开的论据是可以让程序员知道什么是代价低廉的（在用户空间执行的函数），什么是代价昂贵的（内核调用）。不将它们公开的论据是这样给实现提供了灵活性，无须破坏用户程序就可以修改实际的底层系统调用，以便使其工作得更好。

13.3 实现

看过用户界面和系统调用接口后，现在让我们来看一看如何实现一个操作系统。在下面8个小节，我们将分析涉及实现策略的某些一般的概念性问题。在此之后，我们将看一看某些低层技术，这些技术通常是十分有益的。

13.3.1 系统结构

实现必须要做出的第一个决策可能是系统结构应该是什么。我们在1.7节分析了主要的可能性，在这里要重温一下。一个无结构的单块式设计实际上并不是一个好主意，除非可能是用于电冰箱中的微小的操作系统，但是即使在这里也是可争论的。

1. 分层系统

多年以来很好地建立起来的一个合理的方案是分层系统。Dijkstra的THE系统（图1-25）是第一个分层操作系统。UNIX和Windows Vista也具有分层结构，但是在这两个系统中分层更是一种试图描述系统的方法，而不是用于建立系统的真正的指导原则。

对于一个新系统，选择走这一路线的设计人员应该首先非常仔细地选择各个层次，并且定义每个层次的功能。底层应该总是试图隐藏硬件最糟糕的特异性，就像图11-7中HAL所做的那样。或许下一层应该处理中断、上下文切换以及MMU，从而在这一层的代码大部分是与机器无关的。在这一层之上，不同的设计人员可能具有不同的口味（与偏好）。一种可能性是让第3层管理线程，包括调度和线程间同步，如图13-2所示。此处的思想是从第4层开始，我们拥有适当的线程，这些线程可以被正常地调度，并且使用标准的机制（例如互斥量）进行同步。

在第4层，我们可能会找到设备驱动程序，每个设备驱动程序作为一个单独的线程而运行，具有自己的状态、程序计数器、寄存器等，可能（但是不必要）处于内核地址空间内部。这样的设计可以大大