更好的模型是从概率的角度来看CPU的利用率。假设一个进程等待I/O操作的时间与其停留在内存中时间的比为p。当内存中同时有n个进程时，则所有n个进程都在等待I/O（此时CPU空转）的概率是p^n。CPU的利用率由下面的公式给出：

$$\text{CPU利用率} = 1-p^n$$

图2-6以n为变量的函数表示了CPU的利用率，n称为多道程序设计的道数（degree of multiprogramming）。

从图2-6中可以清楚地看到，如果进程花费80%的时间等待I/O，为使CPU的浪费低于10%，至少要有10个进程同时在内存中。当读者认识到一个等待用户从终端输入的交互式进程是处于I/O等待状态时，那么很明显，80%甚至更多的I/O等待时间是普遍的。即使是在服务器中，做大量磁盘I/O操作的进程也会花费同样或更多的等待时间。

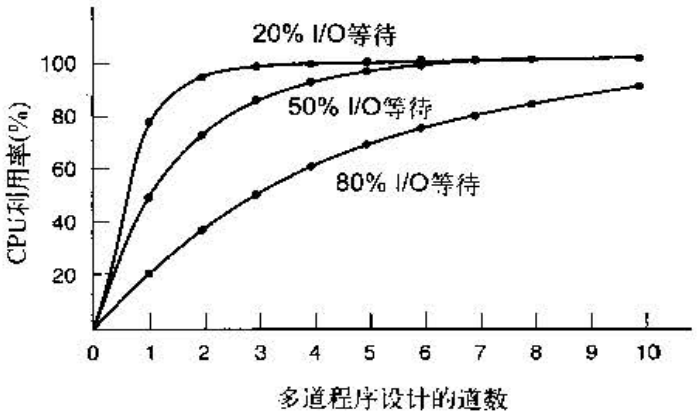

图2-6 CPU利用率是内存中进程数目的函数

从完全精确的角度考虑，应该指出此概率模型只是描述了一个大致的状况。它假设所有n个进程是独立的，即内存中的5个进程中，3个运行，2个等待，是完全可接受的。但在单CPU中，不能同时运行3个进程，所以当CPU忙时，已就绪的进程也必须等待CPU。因而，进程不是独立的。更精确的模型应该用排队论构造，但我们的模型（当进程就绪时，给进程分配CPU，否则让CPU空转）仍然是有效的，即使图2-6的真实曲线会与图中所画的略有不同。

虽然图2-6的模型很简单，很粗略，它依然对预测CPU的性能很有效。例如，假设计算机有512MB内存，操作系统占用128MB，每个用户程序也占用128MB。这些内存空间允许3个用户程序同时驻留在内存中。若80%的时间用于I/O等待，则CPU的利用率（忽略操作系统开销）大约是$1-0.8^3$，即大约49%。在增加512MB字节的内存后，可从3道程序设计提高到7道程序设计，因而CPU利用率提高到79%。换言之，第二个512MB内存提高了30%的吞吐量。

增加第三个512MB内存只能将CPU利用率从79%提高到91%，吞吐量的提高仅为12%。通过这一模型，计算机用户可以确定第一次增加内存是一个合算的投资，而第二个则不是。

2.2 线程

在传统操作系统中，每个进程有一个地址空间和一个控制线程。事实上，这几乎就是进程的定义。不过，经常存在在同一个地址空间中准并行运行多个控制线程的情形，这些线程就像（差不多）分离的进程（共享地址空间除外）。在下面各节中，我们将讨论这些情形及其实现。

2.2.1 线程的使用

为什么人们需要在一个进程中再有一类进程？有若干理由说明产生这些迷你进程（称为线程）的必要性。下面我们来讨论其中一些理由。人们需要多线程的主要原因是，在许多应用中同时发生着多种活动。其中某些活动随着时间的推移会被阻塞。通过将这些应用程序分解成可以准并行运行的多个顺序线程，程序设计模型会变得更简单。

在前面我们已经进行了有关讨论。准确地说，这正是之前关于进程模型的讨论。有了这样的抽象，我们才不必考虑中断、定时器和上下文切换，而只需考察并行进程。类似地，只是在有了多线程概念之后，我们才加入了一种新的元素：并行实体共享同一个地址空间和所有可用数据的能力。对于某些应用而言，这种能力是必需的，而这正是多进程模型（它们具有不同地址空间）所无法表达的。

第二个关于需要多线程的理由是，由于线程比进程更轻量级，所以它们比进程更容易（即更快）创建，也更容易撤销。在许多系统中，创建一个线程较创建一个进程要快10~100倍。在有大量线程需要动态和快速修改时，具有这一特性是很有用的。