当一个进程在逻辑上不能继续运行时，它就会被阻塞，典型的例子是它在等待可以使用的输入。还可能有这样的情况：一个概念上能够运行的进程被迫停止，因为操作系统调度另一个进程占用了CPU。这两种情况是完全不同的。在第一种情况下，进程挂起是程序自身固有的原因（在键入用户命令行之前，无法执行命令）。第二种情况则是由系统技术上的原因引起的（由于没有足够的CPU，所以不能使每个进程都有一台它私用的处理器）。在图2-2中可以看到显示进程的三种状态的状态图。这三种状态是：

1) 运行态（该时刻进程实际占用CPU）。

2) 就绪态（可运行，但因为其他进程正在运行而暂时停止）。

3) 阻塞态（除非某种外部事件发生，否则进程不能运行）。

前两种状态在逻辑上是类似的。处于这两种状态的进程都可以运行，只是对于第二种状态暂时没有CPU分配给它。第三种状态与前两种状态不同，处于该状态的进程不能运行，即使CPU空闲也不行。

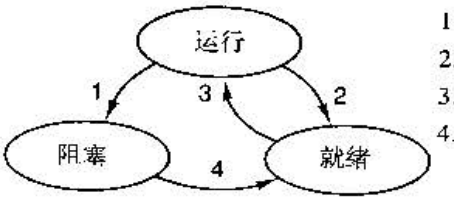

图2-2 一个进程可处于运行态、阻塞态和就绪态，图中显示出各状态之间的转换

进程的三种状态之间有四种可能的转换关系，如图2-2所示。在操作系统发现进程不能继续运行下去时，发生转换1。在某些系统中，进程可以执行一个诸如pause的系统调用来进入阻塞状态。在其他系统中，包括UNIX，当一个进程从管道或设备文件（例如终端）读取数据时，如果没有有效的输入存在，则进程会被自动阻塞。

转换2和3是由进程调度程序引起的，进程调度程序是操作系统的一部分，进程甚至感觉不到调度程序的存在。系统认为一个运行进程占用处理器的时间已经过长，决定让其他进程使用CPU时间时，会发生转换2。在系统已经让所有其他进程享有了它们应有的公平待遇而重新轮到第一个进程再次占用CPU运行时，会发生转换3。调度程序的主要工作就是决定应当运行哪个进程、何时运行及它应该运行多长时间，这是很重要的一点，我们将在本章的后面部分进行讨论。已经提出了许多算法，这些算法力图在整体效率和进程的竞争公平性之间取得平衡。我们将在本章稍后部分研究其中的一些问题。

当进程等待的一个外部事件发生时（如一些输入到达），则发生转换4。如果此时没有其他进程运行，则立即触发转换3，该进程便开始运行。否则该进程将处于就绪态，等待CPU空闲并且轮到它运行。

使用进程模型使得我们易于想象系统内部的操作状况。一些进程正在运行执行用户键入命令所对应的程序。另一些进程是系统的一部分，它们的任务是完成下列一些工作：比如，执行文件服务请求、管理磁盘驱动器和磁带机的运行细节等。当发生一个磁盘中断时，系统会做出决定，停止运行当前进程，转而运行磁盘进程，该进程在此之前因等待中断而处于阻塞态。这样，我们就可以不再考虑中断，而只是考虑用户进程、磁盘进程、终端进程等。这些进程在等待时总是处于阻塞状态。在已经读入磁盘或键入字符后，等待它们的进程就被解除阻塞，并成为可调度运行的进程。

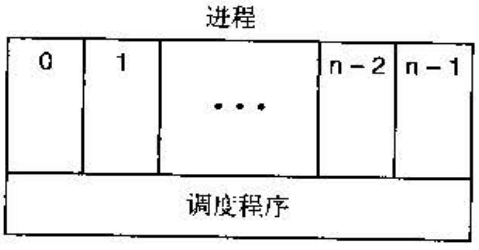

图2-3 以进程构造的操作系统最底层处理中断和调度，在该层之上是顺序进程

从这个观点引出了图2-3所示的模型。在图2-3中，操作系统的最底层是调度程序，在它上面有许多进程。所有关于中断处理、启动进程和停止进程的具体细节都隐藏在调度程序中。实际上，调度程序是一段非常短小的程序。操作系统的其他部分被简单地组织成进程的形式。不过，很少有真实的系统是以这样的理想方式构造的。

2.1.6 进程的实现

为了实现进程模型，操作系统维护着一张表格（一个结构数组），即进程表（process table）。每个进