当我们转向网络服务器时，情况略微有些改变。这里，多个进程经常竞争CPU，因此调度功能再一次变得至关重要。例如，当CPU必须在运行一个收集每日统计数据的进程和服务用户需求的进程之间进行选择的时候，如果后者首先占用了CPU，用户将会更高兴。

另外，为了选取正确的进程运行，调度程序还要考虑CPU的利用率，因为进程切换的代价是比较高的。首先用户态必须切换到内核态；然后要保存当前进程的状态，包括在进程表中存储寄存器值以便以后重新装载。在许多系统中，内存映像（例如，页表内的内存访问位）也必须保存；接着，通过运行调度算法选定一个新进程；之后，应该将新进程的内存映像重新装入MMU；最后新进程开始运行。除此之外，进程切换还要使整个内存高速缓存失效，强迫缓存从内存中动态重新装入两次（进入内核一次，离开内核一次）。总之，如果每秒钟切换进程的次数太多，会耗费大量CPU时间，所以有必要提醒注意。

1. 进程行为

几乎所有进程的（磁盘）I/O请求或计算都是交替突发的，如图2-38所示。典型地，CPU不停顿地运行一段时间，然后发出一个系统调用以便读写文件。在完成系统调用之后，CPU又开始计算，直到它需要读更多的数据或写更多的数据为止。请注意，某些I/O活动可以看作是计算。例如，当CPU向视频RAM复制数据以更新屏幕时，因为使用了CPU，所以这是计算，而不是I/O活动。按照这种观点，当一个进程等待外部设备完成工作而被阻塞时，才是I/O活动。

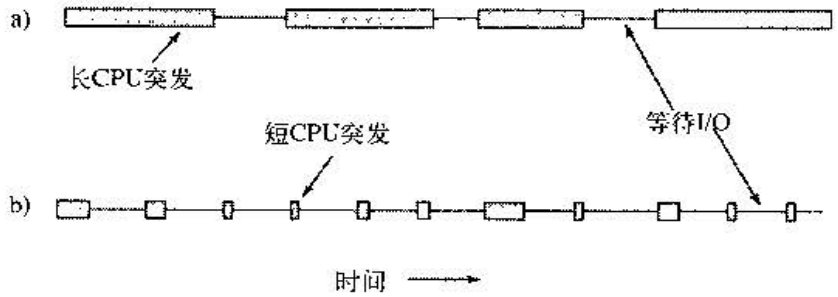

图2-38 CPU的突发使用和等待I/O的时期交替出现：a) CPU密集型进程；b) I/O密集型进程

图2-38中有一件值得注意的事，即某些进程（图2-38a的进程）花费了绝大多数时间在计算上，而其他进程（图2-38b的进程）则在等待I/O上花费了绝大多数时间。前者称为计算密集型（compute-bound），后者称为I/O密集型（I/O-bound）。典型的计算密集型进程具有较长时间的CPU集中使用和较小频度的I/O等待。I/O密集型进程具有较短时间的CPU集中使用和频繁的I/O等待。它是I/O类的，因为这种进程在I/O请求之间较少进行计算，并不是因为它们有特别长的I/O请求。在I/O开始后无论处理数据是多还是少，它们都花费同样的时间提出硬件请求读取磁盘块。

有必要指出，随着CPU变得越来越快，更多的进程倾向为I/O密集型。这种现象之所以发生是因为CPU的改进比磁盘的改进快得多，其结果是，未来对I/O密集型进程的调度处理似乎更为重要。这里的基本思想是，如果需要运行I/O密集型进程，那么就应该让它尽快得到机会，以便发出磁盘请求并保持磁盘始终忙碌。从图2-6中可以看到，如果进程是I/O密集型的，则需要多运行一些这类进程以保持CPU的充分利用。

2. 何时调度

有关调度处理的一个关键问题是何时进行调度决策。存在着需要调度处理的各种情形。第一，在创建一个新进程之后，需要决定是运行父进程还是运行子进程。由于这两种进程都处于就绪状态，所以这是一种正常的调度决策，可以任意决定，也就是说，调度程序可以合法选择先运行父进程还是先运行了进程。

第二，在一个进程退出时必须做出调度决策。一个进程不再运行（因为它不再存在），所以必须从就绪进程集中选择另外某个进程。如果没有就绪的进程，通常会运行一个系统提供的空闲进程。

第三，当一个进程阻塞在I/O和信号量上或由于其他原因阻塞时，必须选择另一个进程运行。有时，阻塞的原因会成为选择的因素。例如，如果A是一个重要的进程，并正在等待B退出临界区，让B随后运行将会使得B退出临界区，从而可以让A运行。不过问题是，通常调度程序并不拥有做出这种相关考虑