统是根本不行的。至少可能需要重新定义系统调用的语义，并且不得不重写库。而且所有这些工作必须与在一个进程中有一个线程的原有程序向后兼容。有关线程的其他信息，可以参阅（Hauser 等人，1993；Marsh等人，1991）。

2.3 进程间通信

进程经常需要与其他进程通信。例如，在一个shell 管道中，第一个进程的输出必须传送给第二个进程，这样沿着管道传递下去。因此在进程之间需要通信，而且最好使用一种结构良好的方式，不要使用中断。在下面几节中，我们就来讨论一些有关进程间通信（Inter Process Communication，IPC）的问题。

简要地说，有三个问题。第一个问题与上面的叙述有关，即一个进程如何把信息传递给另一个。第二个要处理的问题是，确保两个或更多的进程在关键活动中不会出现交叉，例如，在飞机订票系统中的两个进程为不同的客户试图争夺飞机上的最后一个座位。第三个问题与正确的顺序有关（如果该顺序是有关联的话），比如，如果进程A产生数据而进程B打印数据，那么B在打印之前必须等待，直到A已经产生一些数据。我们将从下一节开始考察所有这三个问题。

有必要说明，这三个问题中的两个问题对于线程来说是同样适用的。第一个问题（即传递信息）对线程而言比较容易，因为它们共享一个地址空间（在不同地址空间需要通信的线程属于不同进程之间的通信情形）。但是另外两个问题（需要梳理清楚并保持恰当的顺序）同样适用于线程。同样的问题可用同样的方法解决。下面开始讨论进程间通信的问题，不过请记住，同样的问题和解决方法也适用于线程。

2.3.1 竞争条件

在一些操作系统中，协作的进程可能共享一些彼此都能读写的公用存储区。这个公用存储区可能在内存中（可能是在内核数据结构中），也可能是一个共享文件。这里共享存储区的位置并不影响通信的本质及其带来的问题。为了理解实际中进程间通信如何工作，我们考虑一个简单但很普遍的例子：一个假脱机打印程序。当一个进程需要打印一个文件时，它将文件名放在一个特殊的假脱机目录（spooler directory）下。另一个进程（打印机守护进程）则周期性地检查是否有文件需要打印，若有就打印并将该文件名从目录下删掉。

设想假脱机目录中有许多槽位，编号依次为0，1，2，…，每个槽位存放一个文件名。同时假设有两个共享变量：out，指向下一个要打印的文件；in，指向目录中下一个空闲槽位。可以把这两个变量保存在一个所有进程都能访问的文件中，该文件的长度为两个字。在某一时刻，0号至3号槽位空（其中的文件已经打印完毕），4号至6号槽位被占用（其中存有排好队列的要打印的文件名）。几乎在同一时刻，进程A和进程B都决定将一个文件排队打印，这种情况如图2-21所示。

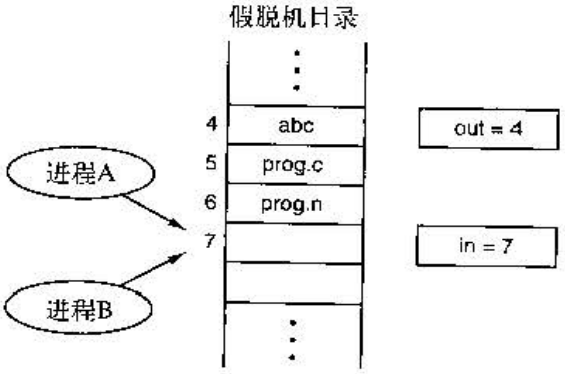

图2-21 两个进程同时想访问共享内存

在Murphy法则（任何可能出错的地方终将出错）生效时，可能发生以下的情况。进程A读到in的值为7，将7存在一个局部变量next_free_slot中。此时发生一次时钟中断，CPU认为进程A已运行了足够长的时间，决定切换到进程B。进程B也读取in，同样得到值为7，于是将7存在B的局部变量next_free_slot中。在这一时刻两个进程都认为下一个可用槽位是7。

进程B现在继续运行，它将其文件名存在槽位7中并将in的值更新为8。然后它离开，继续执行其他操作。

最后进程A接着从上次中断的地方再次运行。它检查变量next_free_slot，发现其值为7，于是将打印文件名存入7号槽位，这样就把进程B存在那里的文件名覆盖掉。然后它将next_free_slot加1，得到值为8，就将8存到in中。此时，假脱机目录内部是一致的，所以打印机守护进程发现不了任何错误，但进程B却永远得不到任何打印输出。类似这样的情况，即两个或多个进程读写某些共享数据，而最后的结果取决于进程运行的精确时序，称为竞争条件（race condition）。调试包含有竞争条件的程序是一件很头痛的事。大多数的测试运行结果都很好，但在极少数情况下会发生一些无法解释的奇怪现象。