该解决方案使用了三个信号量：一个称为full，用来记录充满的缓冲槽数目；一个称为empty，记录空的缓冲槽总数；一个称为mutex，用来确保生产者和消费者不会同时访问缓冲区。full的初值为0，empty的初值为缓冲区中槽的数目，mutex初值为1。供两个或多个进程使用的信号量，其初值为1，保证同时只有一个进程可以进入临界区，称作二元信号量（binary semaphore）。如果每个进程在进入临界区前都执行一个down操作，并在刚刚退出时执行一个up操作，就能够实现互斥。

在有了一些进程间通信原语之后，我们再观察一下图2-5中的中断顺序。在使用信号量的系统中，隐藏中断的最自然的方法是为每一个I/O设备设置一个信号量，其初值为0。在启动一个I/O设备之后，管理进程就立即对相关联的信号量执行一个down操作，于是进程立即被阻塞。当中断到来时，中断处理程序随后对相关信号量执行一个up操作，从而将相关的进程设置为就绪状态。在该模型中，图2-5中的第5步包括在设备的信号量上执行up操作，这样在第6步中，调度程序将能执行设备管理程序。当然，如果这时有几个进程就绪，则调度程序下次可以选择一个更为重要的进程来运行。在本章的后续内容中，我们将看到调度算法是如何进行的。

在图2-28的例子中，我们实际上是通过两种不同的方式来使用信号量，两者之间的区别是很重要的。信号量mutex用于互斥，它用于保证任一时刻只有一个进程读写缓冲区和相关的变量。互斥是避免混乱所必需的操作。在下一节中，我们将讨论互斥量及其实现方法。

```
#define N 100                                   /* 缓冲区中的槽数目 */
typedef int semaphore;                          /* 信号量是一种特殊的整型数据 */
semaphore mutex = 1;                            /* 控制对临界区的访问 */
semaphore empty = N;                            /* 计数缓冲区的空槽数目 */
semaphore full = 0;                             /* 计数缓冲区的满槽数目 */

void producer(void)
{
    int item;

    while (TRUE) {                              /* TRUE是常量1 */
        item = produce_item( );                 /* 产生放在缓冲区中的一些数据 */
        down(&empty);                           /* 将空槽数目减1 */
        down(&mutex);                           /* 进入临界区 */
        insert_item(item);                      /* 将新数据项放到缓冲区中 */
        up(&mutex);                             /* 离开临界区 */
        up(&full);                              /* 将满槽的数目加1 */
    }
}


void consumer(void)
{
    int item;

    while (TRUE) {                              /* 无限循环 */
        down(&full);                            /* 将满槽数目减1 */
        down(&mutex);                           /* 进入临界区 */
        item = remove_item( );                  /* 从缓冲区中取出数据项 */
        up(&mutex);                             /* 离开临界区 */
        up(&empty);                             /* 将空槽数目加1 */
        consume_item(item);                     /* 处理数据项 */
    }
}
```

图2-28 使用信号量的生产者—消费者问题

信号量的另一种用途是用于实现同步（synchronization）。信号量full和empty用来保证某种事件的顺序发生或不发生。在本例中，它们保证当缓冲区满的时候生产者停止运行，以及当缓冲区空的时候消费者停止运行。这种用法与互斥是不同的。