```
#define N 100                                        /* 缓冲区中的槽数目 */

void producer(void)
{
     int item;
     message m;                                      /* 消息缓冲区 */

     while (TRUE) {
          item = produce_item();                     /* 产生放入缓冲区的一些数据 */
          receive(consumer, &m);                     /* 等待消费者发送空缓冲区 */
          build_message(&m, item);                   /* 建立一个待发送的消息 */
          send(consumer, &m);                        /* 发送数据项给消费者 */
     }
}

void consumer(void)
{
     int item, i;
     message m;

     for (i = 0; i < N; i++) send(producer, &m);    /* 发送N个空缓冲区 */
     while (TRUE) {
          receive(producer, &m);                     /* 接收包含数据项的消息 */
          item = extract_item(&m);                   /* 将数据项从消息中提取出来 */
          send(producer, &m);                        /* 将空缓冲区发送回生产者 */
          consume_item(item);                        /* 处理数据项 */
     }
}
```

图2-36 用N条消息实现的生产者-消费者问题

使用信箱的另一种极端方法是彻底取消缓冲。采用这种方法时，如果send在receive之前执行，则发送进程被阻塞，直到receive发生。在执行receive时，消息可以直接从发送者复制到接收者，不用任何中间缓冲。类似地，如果先执行receive，则接收者会被阻塞，直到send发生。这种方案常被称为会合（rendezvous）。与带有缓冲的消息方案相比，该方案实现起来更容易一些，但却降低了灵活性，因为发送者和接收者一定要以步步紧接的方式运行。

通常在并行程序设计系统中使用消息传递。例如，一个著名的消息传递系统是消息传递接口（Message-Passing Interface，MPI），它广泛应用在科学计算中。有关该系统的更多信息，可参考相关文献（Gropp等人，1994；Snir等人，1996）。

2.3.9 屏障

最后一个同步机制是准备用于进程组而不是用于双进程的生产者-消费者类情形的。在有些应用中划分了若干阶段，并且规定，除非所有的进程都就绪准备着手下一个阶段，否则任何进程都不能进入下一个阶段。可以通过在每个阶段的结尾安置屏障（barrier）来实现这种行为。当一个进程到达屏障时，它就被屏障阻拦，直到所有进程都到达该屏障为止。屏障的操作如图2-37所示。

在图2-37a中可以看到有四个进程接近屏障，这意味着它们正在运算，但是还没有到达每个阶段的结尾。过了一会儿，第一个进程完成了所有需要在第一阶段进行的计算。它接着执行barrier原语，这通常是调用一个库过程。于是该进程被挂起。一会儿，第二个和第三个进程也完成了第一阶段的计算，也接着执行barrier原语。这种情形如图2-37b所示。结果，当最后一个进程C到达屏障时，所有的进程就一起被释放，如图2-37c所示。

作为一个需要屏障的例子，考虑在物理或工程中的一个典型弛豫问题。这是一个带有初值的矩阵。这些值可能代表一块金属板上各个点的温度值。基本想法可以是准备计算如下的问题：要花费多长时间，在一个角上的火焰才能传播到整个板上。

计算从当前值开始，先对矩阵进行一个变换，从而得到第二个矩阵，例如，运用热力学定律考察在ΔT之后的整个温度分布。然后，进程不断重复，随着金属板的加热，给出样本点温度随时间变化的函数。该算法从而随时间变化生成出一系列矩阵。