从缓冲区中取出这些项目，然后消费(使用)它们。也可能有多个生产者和消费者的变种。

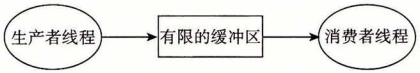

图 12-23　生产者-消费者问题。生产者产生项目并把它们插入到一个有限的缓冲区中。消费者从缓冲区中取出这些项目，然后消费它们

因为插入和取出项目都涉及更新共享变量，所以我们必须保证对缓冲区的访问是互斥的。但是只保证互斥访问是不够的，我们还需要调度对缓冲区的访问。如果缓冲区是满的(没有空的槽位)，那么生产者必须等待直到有一个槽位变为可用。与之相似，如果缓冲区是空的(没有可取用的项目)，那么消费者必须等待直到有一个项目变为可用。

生产者-消费者的相互作用在现实系统中是很普遍的。例如，在一个多媒体系统中，生产者编码视频帧，而消费者解码并在屏幕上呈现出来。缓冲区的目的是为了减少视频流的抖动，而这种抖动是由各个帧的编码和解码时与数据相关的差异引起的。缓冲区为生产者提供了一个槽位池，而为消费者提供一个已编码的帧池。另一个常见的示例是图形用户接口设计。生产者检测到鼠标和键盘事件，并将它们插入到缓冲区中。消费者以某种基于优先级的方式从缓冲区取出这些事件，并显示在屏幕上。

在本节中，我们将开发一个简单的包，叫做 SBUF，用来构造生产者-消费者程序。在下一节里，我们会看到如何用它来构造一个基于预线程化(prethreading)的有趣的并发服务器。SBUF 操作类型为 sbuf_t 的有限缓冲区(图 12-24)。项目存放在一个动态分配的 n 项整数数组(buf)中。front 和 rear 索引值记录该数组中的第一项和最后一项。三个信号量同步对缓冲区的访问。mutex 信号量提供互斥的缓冲区访问。slots 和 items 信号量分别记录空槽位和可用项目的数量。

code/conc/sbuf.h

```
1  typedef struct {
2      int *buf;          /* Buffer array */
3      int n;             /* Maximum number of slots */
4      int front;         /* buf[(front+1)%n] is first item */
5      int rear;          /* buf[rear%n] is last item */
6      sem_t mutex;       /* Protects accesses to buf */
7      sem_t slots;       /* Counts available slots */
8      sem_t items;       /* Counts available items */
9  } sbuf_t;
```

code/conc/sbuf.h

图 12-24　sbuf_t：SBUF 包使用的有限缓冲区

图 12-25 给出了 SBUF 函数的实现。sbuf_init 函数为缓冲区分配堆内存，设置 front 和 rear 表示一个空的缓冲区，并为三个信号量赋初始值。这个函数在调用其他三个函数中的任何一个之前调用一次。sbuf_deinit 函数是当应用程序使用完缓冲区时，释放缓冲区存储的。sbuf_insert 函数等待一个可用的槽位，对互斥锁加锁，添加项目，对互斥锁解锁，然后宣布有一个新项目可用。sbuf_remove 函数是与 sbuf_insert 函数对称的。在等待一个可用的缓冲区项目之后，对互斥锁加锁，从缓冲区的前面取出该项目，对互斥锁解锁，然后发信号通知一个新的槽位可供使用。