例 1、 生产者—消费者问题演变。

情况 1 一个 buffer，一个生产者，一个消费者，生产者只生产一个东西，消费者只进行一次消费，即：生产者只进行一次 putdata 操作，消费者只进行一次 getdata 操作。

解 这是一个同步问题，生产者和消费者分别是 2 个并发的进程。

（1）操作规则

如果 buffer 为空，则消费者只能等待。

（2）操作流程

```
<生产者>
    {
      putdata;
       设置 Buffer 有数据标志  V(S)
     }
<消费者>
     {
          判断 buffer 是否有产品，没有则等待;
          getdata;
     }
```

(3) 信号量

设置 1 个信号量 full，full 表示 buffer 是否有数据，初值为 0。

(4) P、V 操作实现

```
var full:semaphore:=0;
                buffer: array   [1] of item;
                  begin
                      parbegin
                        producer:
                          begin
                                putdata;
                                V(full);
                          end
                        consumer:
                              begin
                                    P(full);
                                    getdata;
                              end
                      parend
                end
```

情况 2 一个 buffer，一个生产者，一个消费者，生产者不断地进行 putdata 操作，消费者不断地进行 getdata 操作，即：生产者不断地生产，消费者不断地消费

(1) 操作规则

只有 buffer 为空时才能进行 putdata 操作；只有 buffer 有数据时才能进行 putdata 操作。

(2) 操作流程

```
<生产者>
   {
```

repeat
判断 buffer 是否为空，不空则等待；
putdata ；
设置 buffer 有数据的标志；
until false
}
<消费者>
{
repeat
判断 buffer 是否有数据，没有数据则等待；
getdata；
设置 buffer 为空标志；
until false
}

(3) 信号量

设置 2 个信号量 full 和 empty。

full 表示 buffer 是否有数据。因为进程在初始状态时，buffer 中没有数据，故初值为 0，变化范围-1~1。

empty 表示 buffer 是否为空。因为进程在初始状态时，buffer 为空，故初值为 0 初值为 1，变化范围-1~1。

(4) P、V 操作实现

```
var full:semaphore:=0;
    emptyl:semaphore:=1;
    buffer: array   [1] of item;
    begin
        parbegin
            producer：
                begin
                    repeat
                        P(empty)；
                        putdata；
                        V(full);
                    until false
                end
            consumer：
                    begin
                        repeat                  P(full)；
                            getdata；
                            V(empty)；
                        until false.
                    end
        parend
    end
```

情况 3　一个 buffer，多个生产者，多个消费者，多个生产者和消费者都在不断地存取 buffer，即生产者不断地进行 putdata 操作，消费者不断地进行 getdata 操作。

(1)　操作规则

只有 buffer 为空时才能进行 putdata 操作；只有 buffer 有数据时才能进行 putdata 操作。

这时 buffer 变成了临界资源，不允许多个进程同时操作 buffer，即不允许多个消费者同时进行 gedata，不允许多个生产者同时进行 putdata 操作。

(2)　操作流程

```
<生产者>
  {
      repeat
          判断 buffer 是否为空，不则等待；
          是否可操作 buffer；
          putdata；
          设置 buffer 可操作标志；
          设置 buffer 有数据的标志；
      until false
  }
  <消费者>
     {
        repeat
            判断 buffer 是否有数据，没有则等待；
            是否可操作 buffer；
            getdata ；
            设置 buffer 可操作标志；
            设置 buffer 为空标志；
        until false
      }
```

(3)　信号量

设置 3 个信号量 full、empty 和 B-M。

full 表示 buffer 是否有数据，初值为 0；

empty 表示 buffer 是否为空，初值为 1；

B-M 表示 buffer 是否可操作，初值为 1。

由于 buffer 只有一个，full 和 empty 可以保证对 buffer 的正确操作，故 B-M 是多余的，可以省略。

(4)　P、V 操作实现

```
<生产者>                          <消费者>
 {                                 {
   repeat                            repeat
     P(empty)；                        P(full)；
     P(B-M)；                          P(B-M)；
     putdata；                         getdata；
     V(B-M)；                          V(B-M)；
     V(full)；                         V(empty)；
```

until false.　　until false

}　　}

情况 4　多个生产者，多个消费者，N 个 buffer，多次循环存取 buffer，即，即多个生产者不断地进行 putdata 操作，多个消费者不断地进行 getdata 操作。

(1) 操作规则

只有 buffer 有空间才能进行 putdata 操作；

只有 buffer 有数据才能进行 putdata 操作；

这时 buffer 变成了临界资源，不允许多个消费者和生产者同时对同一个 buffer 进行 gedata 和 putdata 操作。

(2)　操作流程

<生产者>

{

repeat

判断 buffer 是否有空间，没有则等待；

是否可操作 buffer；

putdata；

设置 buffer 可操作标志；

设置 buffer 有数据的标志；

until　false

}

<消费者>

{

repeat

判断 buffer 是否有数据，没有则等待；

是否可操作 buffer；

getdata；

设置 buffer 可操作标志；

设置 buffer 有空间标志；

until false

}

(3)　信号量

full 表示 buffer 是否有数据，初值为 0;

empty 表示 buffer 是否为空，初值为 N；

B-M 表示 buffer 是否可操作，初值为 1。

(4)　P、V 操作实现

<生产者>　　<消费者>

{　　{

repeat　　repeat

P(empty)；　　P(full)；

P(B-M)；　　P(B-M)；

putdata；　　getdata；

V(B-M)；　　V(B-M)；

V(full)；　　V(empty)；

```
        until false                                    until false
    }                                              }
```

(5) 改进的 P、V 操作实现

在上述的实现中，putdata 和 getdata 操作都在临界区中，因此多个进程对多个 buffer 的操作是不能并发进行的，进程间并行操作的程度很低。实际上只要保证多个进程同时操作不同 buffer 就可以实现对整个 buffer 的并行操作。因此，只要保证为不同的进程分配不同 buffer，putdata 和 getdata 操作是可以同时进行。这样互斥不是发生在对 buffer 的存取操作上，而是发生在对 buffer 的分配上，这个时间与存取 buffer 的时间相比是较短的，因此减少了进程处于临界区的时间。这里引入 2 个函数:

getE_buffer()，返回值是空的 buffer 号；

getD_buffer()，返回值是有数据的 buffer 号。

getE_buffer()和 getD_buffer()通过将 buffer 转换成循环队列的方法来实现对 buffer 的分配。buffer 设有 Pbuff，Pdata 两个指针，分别指向空闲 buffer 和有数据 buffer 的头，每进行一次 getE_buffer()和 getD_buffer()，Pbuff 和 Pdata 两个指针分别向后移动一个位置。

```
GetE_buffer( )
{c=Pbuff
     Pbuff=(Pbuff+1)MOD N;
  Return( c)
 }


getD_data( )
{c=Pdata;
    Pdata=(pdata+1)MOD N;
    Return(c)
 }
```

改进的程序描述如下：

```
var mutex.empty,full:semaphore:=1,n,0;
    buffer: array   [0,…,n-1] of item;
    Pbuff，Pdata: integer:=0,0;
        begin
          parbegin
            producer: begin
                        repeat
                          ⋮
                          P(empty);
                          P(B_M);
                          in:=getE_buffer();
                          V(B_M)
                          putdata(in);
                          V(full);
                        until false;
                      end
            consumer:begin
```

```
            repeat
              P(full);
              P(B_M);
               out:=getD_buffer();
              V(B_M);
              Getdata(out);
              V(empty);
            until false
          end
    parend
end
```