计算机操作系统

Operating Systems

田卫东

March, 2014

第2章 进程管理

2.3 进程同步

2.3.1 进程同步的基本概念

(1) 什么是进程同步

- 进程同步问题的产生

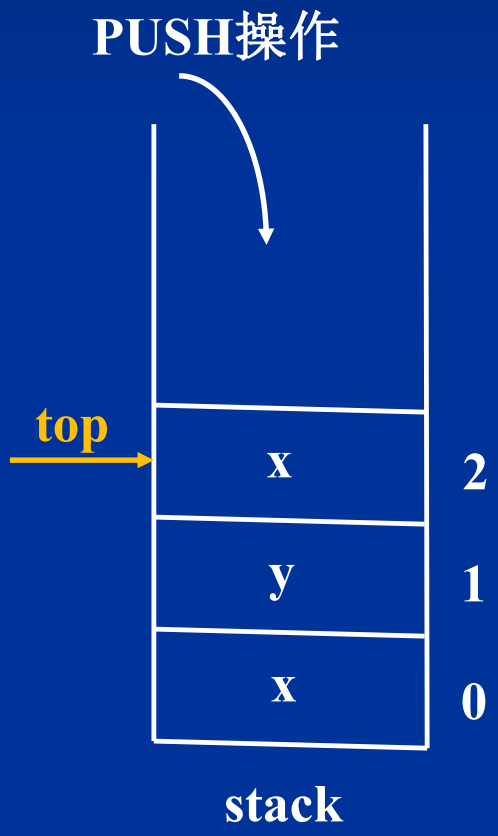
  在多道程序的环境中，系统中的多个进程可以并发执行，同时它们又要共享系统中的资源，这些资源有些是可共享使用的，如磁盘，有些是以独占方式使用的，如打印机。由此将会产生错综复杂的进程间相互制约的关系。

- 两种形式的制约关系：
  - 间接相互制约关系：共享某种系统资源。
  - 直接相互制约关系：主要源于进程间合作。

2.3 进程同步

(2) 临界资源

- 入栈操作




```
Parbegin
  VAR  top : integer := -1 ;
       stack: array[0..n-1] of item;
  Program A :
  Begin
     while ( TRUE )
        top : = (top+1) mod n;
        stack[top] := x ;
     end while
  end
  Program B :
  Begin
     while ( TRUE )
        top : = (top+1) mod n;
        stack[top] := y ;
     end while
  End
Parend
```

PUSH操作 1

2

PUSH操作 3

4

2.3 进程同步

(2) 临界资源

- **互斥共享的资源：临界资源**

  虽然计算机系统中多个进程可以共享系统中的各种资源，然而有些资源一次只能为一个进程使用。

  一次仅能为一个进程所使用的资源称为临界资源。

- **临界资源实例**

  硬件资源：打印机；

  软件资源：内存变量、指针、数组。

如何实现临界资源的互斥访问？

临界资源

```
Parbegin
  VAR  top : integer := -1 ;
       stack: array[0..n-1] of item;
  Program A :
  Begin
     while ( TRUE )
       top : = (top+1) mod n;
       stack[top] := x ;
     end while
  end
  Program B :
  Begin
     while ( TRUE )
       top : = (top+1) mod n;
       stack[top] := y ;
     end while
  End
Parend
```

2.3 进程同步

(3) 临界区

- **临界区的定义**

  进程中访问临界资源的代码段称为临界区。

- **策略**

  临界资源访问互斥

  ↓

  临界区访问互斥

```
Parbegin
  VAR  top : integer := -1 ;
       stack: array[0..n-1] of item;
  Program A :
  Begin
     while ( TRUE )
        top : = (top+1) mod n;
        stack[top] := x ;
     end while
  end
  Program B :
  Begin
     while ( TRUE )
        top : = (top+1) mod n;
        stack[top] := y ;
     end while
  End
Parend
```

临界资源

临界区

临界区

2.3 进程同步

(4) 临界区访问控制模型

```
repeat
        critical section ;
        remainder section ;
until false
```

```
repeat
    entry section;
        critical section ;
    exit section ;
        remainder section ;
until false
```

进入区

退出区

(5) 同步机制遵循的原则

- **空闲让进：** 当无进程处于临界区时，请求进入临界区的进程可立即进入
- **忙则等待：** 当已有进程进入临界区时，其他试图进入临界区进程须等待
- **有限等待：** 对要求访问临界资源进程，保证能在有限时间内进入临界区
- **让权等待：** 当进程不能进入临界区时，应释放处理机。

2.3 进程同步
2.3.2 实现进程同步的早期方

(1) 软件解法1：按需访问

- 资源空闲标志: **busy**
  - busy = false : idle;
  - busy = true : busy;

- 违背什么原则？

  忙则等待,让权等待,有限等待

资源空闲标志

```
VAR busy:boolean := false ;
Program P1:
Begin
  repeat
    while( busy ) ;
    busy := true ;
      critical section ;
    busy := false ;
      remainder section ;
  until false
End

Program P2:
Begin
  repeat
    while( busy ) ;
    busy := true ;
      critical section ;
    busy := false ;
      remainder section ;
  until false
End
```

2.3 进程同步
2.3.2 实现进程同步的早期方

(2) 软件解法2：轮询

- 轮转标志: turn
  - turn = 1 :轮到 P1 ;
  - turn = 2 :轮到 P2 ;

- 违背什么原则？

严格限制资源访问顺序;让权等待,有限等待

轮流标志

```
VAR turn:integer := 1 ;
Program P1:
Begin
  repeat
    while( turn = 2) ;

      critical section ;
    turn := 2 ;
      remainder section ;
  until false
End

Program P2:
Begin
  repeat
    while( turn = 1 ) ;

      critical section ;
    turn := 1 ;
      remainder section ;
  until false
End
```

2.3 进程同步
2.3.2 实现进程同步的早期方

(3) 软件解法3：访前先看

- 愿望标志: pturn,qturn
  - pturn = true : P想访问；
  - qturn = true : Q想访问；

- 违背什么原则？

空闲让进，让权等待,有限等待

```
VAR pturn,qturn:boolean := false,false ;
Program P:
Begin
  repeat
   pturn := true ;
   while( qturn ) ;
       critical section ;
   pturn := false ;
       remainder section ;
  until false
End

Program Q:
Begin
  repeat
   qturn := true ;
   while( pturn ) ;
       critical section ;
   qturn := false ;
       remainder section ;
  until false
End
```

2.3 进程同步
2.3.2 实现进程同步的早期方法

(4) 软件解法4：Peterson算法,1981

```
#define  FALSE  0
#define  TRUE   1
#define   N      2                    // 进程的个数
int  turn;                           // 轮到谁？
int  interested[N];                  // 兴趣数组，初始值均为FALSE

void  enter_region ( int  process)   // process = 0 或 1
{
     int  other;                          // 另外一个进程的进程号
     other  =  1  -  process;
     interested[process]  =  TRUE;// 表明本进程感兴趣
     turn  =  process;                    // 设置标志位
     while( turn == process && interested[other] == TRUE);
}

void  leave_region ( int  process)
{
    interested[process]  =  FALSE; // 本进程已离开临界区
}
```

```
Program P1:
Begin
  repeat
    enter_region(0 );
        critical section ;
    leave_region(0);
        remainder section ;
  until false
End

Program P2:
Begin
  repeat
    enter_region(1 );
        critical section ;
    leave_region(1);
        remainder section ;
  until false
End
```

2.3 进程同步

2.3.2 实现进程同步的早期方法

(6) 硬件解法2：SWAP指令

```
// 交换锁lock和key的值
function SWAP(lock, key  ) {
    var tmp : boolean := lock ;
    lock := key ;
    key := tmp ;
}
```

```
function enter_region( var lock : boolean )
Var key:boolean ;  //局部变量
Begin
   key := true;
   while(key)
      SWAP(lock,key);
end

function leave_region( var lock : boolean )
Begin
    lock := false;
end
```

```
VAR lock :boolean := false ;
Program P1:
Begin
   repeat
    enter_region(lock );
        critical section ;
    leave_region(lock);
        remainder section ;
   until false
End

Program P2:
Begin
   repeat
    enter_region(lock);
        critical section ;
    leave_region(lock);
        remainder section ;
   until false
End
```

2.3 进程同步
2.3.2 实现进程同步的早期方法

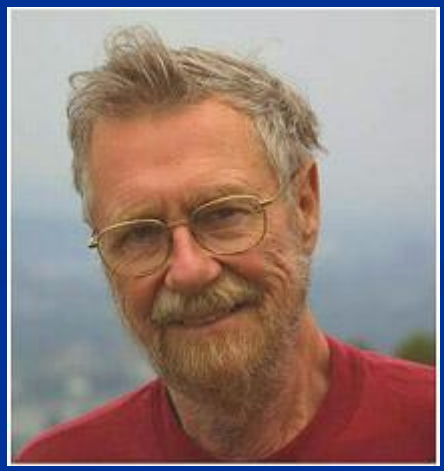
(7) 小结

- 软件解法
  - 忙等待
  - 实现过于复杂，需要高的编程技巧

- 硬件解法
  - 简单、有效，特别适用于多处理机
  - 缺点：忙等待

信号量机制的提出者

- Dijkstra
  荷兰计算机科学家Edsger Wybe Dijkstra;
  1972年ACM 图灵奖获得者
- 贡献:
  提出“goto有害论”;
  提出信号量和PV原语;
  解决了有趣的“哲学家聚餐”问题;
  最短路径算法(SPF)的创造者;
  第一个Algol 60编译器的设计者和实现者;
  THE操作系统的设计者和开发者;



2.3.3 信号量机制

(1) 整数型信号量，Dijkstra,1965

- 定义
  - 整数信号量S;
  - 两个原子操作：wait/signal(P/V)
- 原则：

  除初始化外，S只能由wait/signal访问

```
VAR S:integer := 1 ;
function Wait ( var S: integer )
Begin
    while ( S <= 0 ) do no_op();
    S := S-1;
end

function Signal( var S : integer )
Begin
     S := S+1;
end
```

```
VAR S:integer := 1 ;
Program P1:
Begin
   repeat
     wait(S);
          critical section ;
     signal(S);
          remainder section ;
   until false
End

Program P2:
Begin
   repeat
     wait(S);
          critical section ;
     signal(S);
          remainder section ;
   until false
End
```

(2) 记录型信号量

- 定义
  - 记录型信号量S;
  - 两个原子操作：wait/signal(P/V)
- 原则：

  除初始化外，S只能由wait/signal访问

```
type semaphore = record
    value: integer ;                  // 资源数量
    L: list of process ;              //  阻塞进程队列
end

function wait ( var S ： Semaphore )
Begin
   S.value := S.value-1;
   if ( S.value < 0 ) then block (S.L);
end
function signal(var S ： Semaphore )
Begin
   S.value := S.value+1;
   if ( S.value <= 0 ) then wakeup (S.L);
end
```

```
VAR S:semaphore:= 1 ;
Program P1:
Begin
  repeat
   wait(S );
       critical section ;
   signal(S);
       remainder section ;
  until false
End

Program P2:
Begin
  repeat
   wait(S);
       critical section ;
   signal(S);
       remainder section ;
  until false
End
```

2.3 进程同步

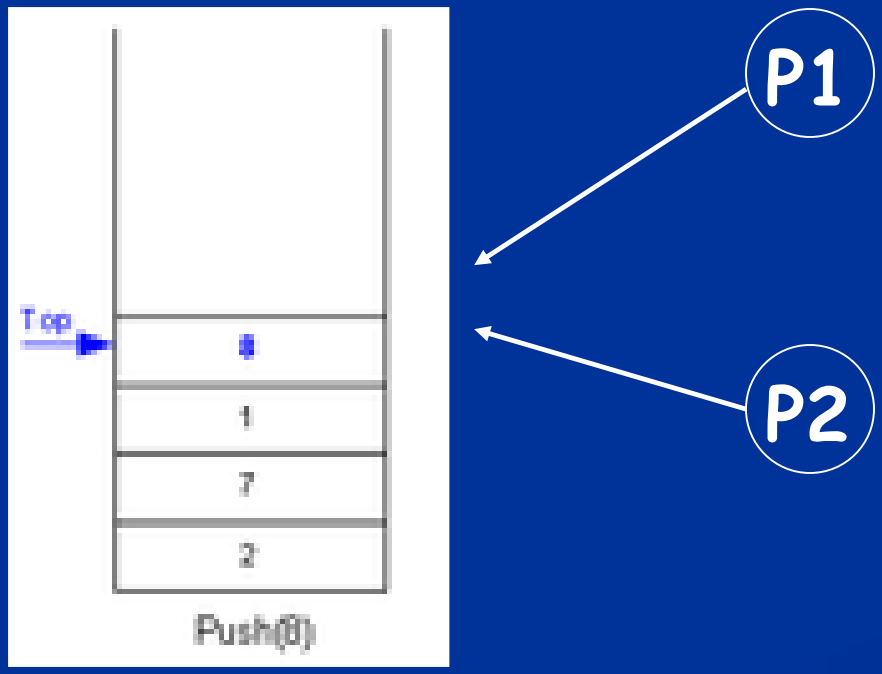
(2) 记录型信号量

- 信号量S的物理含义
  - 对信号量的每次wait操作，意味着进程请求一个单位的该类资源，因此描述为S.value:= S.value-1；当S.value＜0时，表示该类资源已分配完毕，因此进程应进行自我阻塞，放弃处理机，并插入到信号量链表S.L中。
  - 对信号量的每次signal操作，表示执行进程释放一个单位资源，故S.value:= s.value+1操作表示资源数目加1。若加1后仍是S.value＜=0，则表示在该信号量链表中，仍有等待该资源的进程被阻塞，将S.L链表中的第一个等待进程唤醒。
  - S. value:
    - S. value>=0: 资源数量;
    - S. value<0 : |S. value|为阻塞进程的数量;

2.3 进程同步
2.3.4 信号量的应用

(1) 信号量用于互斥

例1：2个进程进行堆栈的入栈

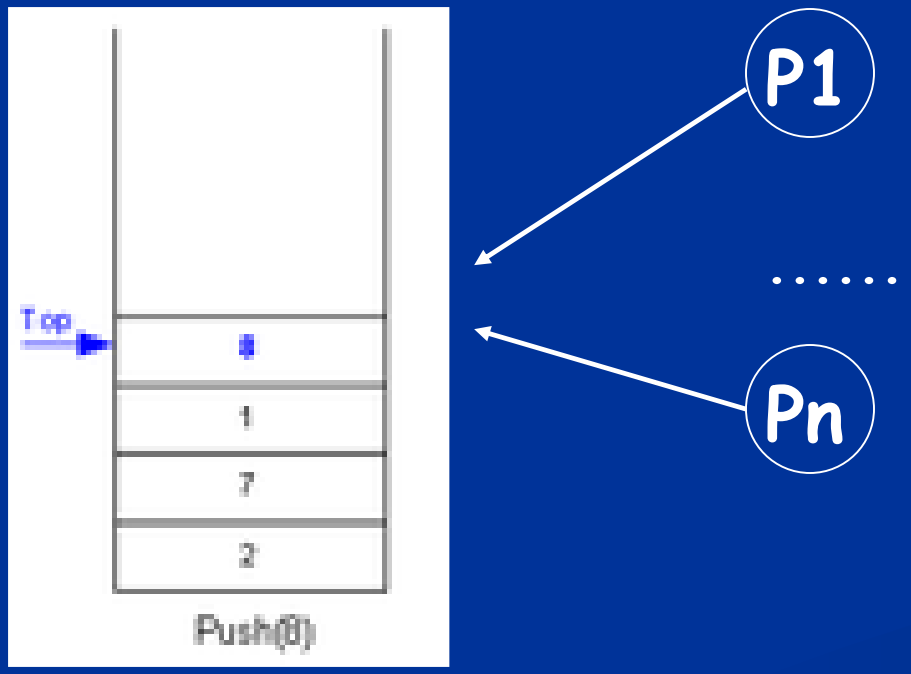



```
Parbegin
  VAR  top : integer = -1 ;
       stack: array[0..n-1] of item;
       mutex: semaphore :=1 ;
  Process P1 :
  Begin
     while ( TRUE )
       wait( mutex ) ;
       top : = (top+1) mod n;
       stack[top] := x ;
       signal(mutex ) ;
     end while
  end
  Process P2 :
  Begin
     while ( TRUE )
       wait( mutex ) ;
       top : = (top+1) mod n;
       stack[top] := y ;
       signal(mutex ) ;
     end while
  End
Parend
```

2.3 进程同步
2.3.4 信号量的应用

(1) 信号量用于互斥

例1-扩展： n个进程进行堆栈的入栈

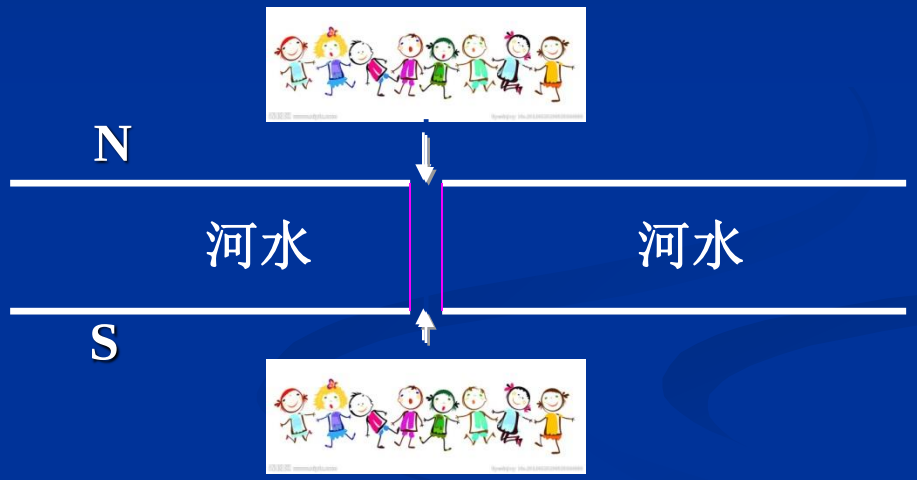



```
Parbegin
  VAR  top : integer = -1 ;
       stack: array[0..n-1] of item;
       mutex: semaphore :=1 ;
  Process P1 :
  Begin
     while ( TRUE )
       wait( mutex ) ;
       top : = (top+1) mod n;
       stack[top] := x ;
       signal(mutex ) ;
     end while
  end
  ......
Process Pn :
  Begin
     while ( TRUE )
       wait( mutex ) ;
       top : = (top+1) mod n;
       stack[top] := y ;
       signal(mutex ) ;
     end while
  End
Parend
```

2.3 进程同步

(1) 信号量用于互斥

例2：过独木桥问题，一次只能过1人.



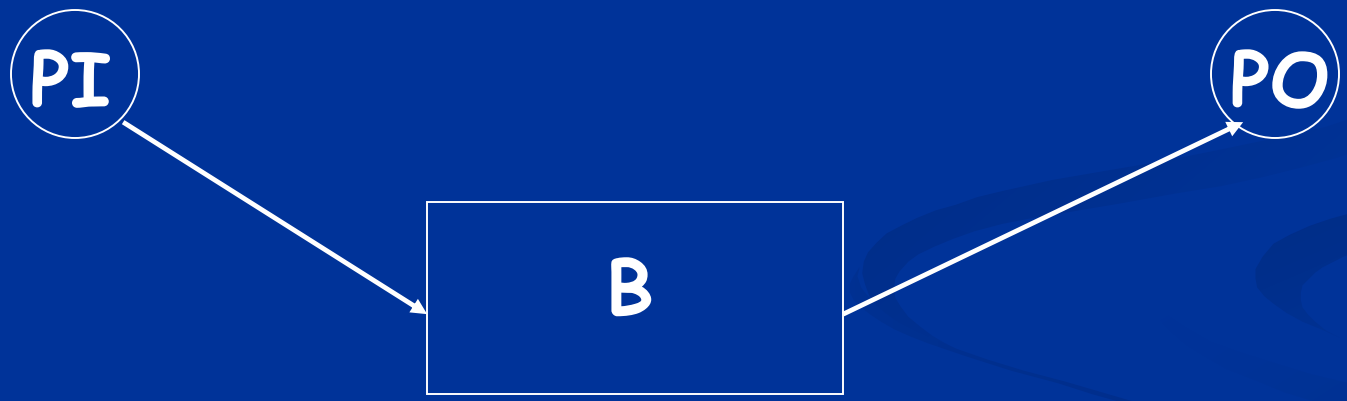
```
Parbegin

  Process S2N :
  Begin
     repeat

        go across the bridge ;

     until false
  end
  Process N2S :
  Begin
     repeat

        go across the bridge ;

     until false
  End
Parend
```

```
Parbegin
  VAR  mutex : semaphore := 1 ;
  Process S2N :
  Begin
     repeat
        wait( mutex ) ;
        go across the bridge ;
        signal(mutex ) ;
     until false
  end
  Process N2S :
  Begin
     repeat
        wait( mutex ) ;
        go across the bridge ;
        signal(mutex ) ;
     until false
  End
Parend
```

2.3 进程同步

(2) 信号量用于同步

例3：PI,PO两进程共享缓冲区B,PI负责读入数据并保存在B中，PO负责打印B中数据，要求每个数据必须且只打印一次，不许漏打或重复打印。



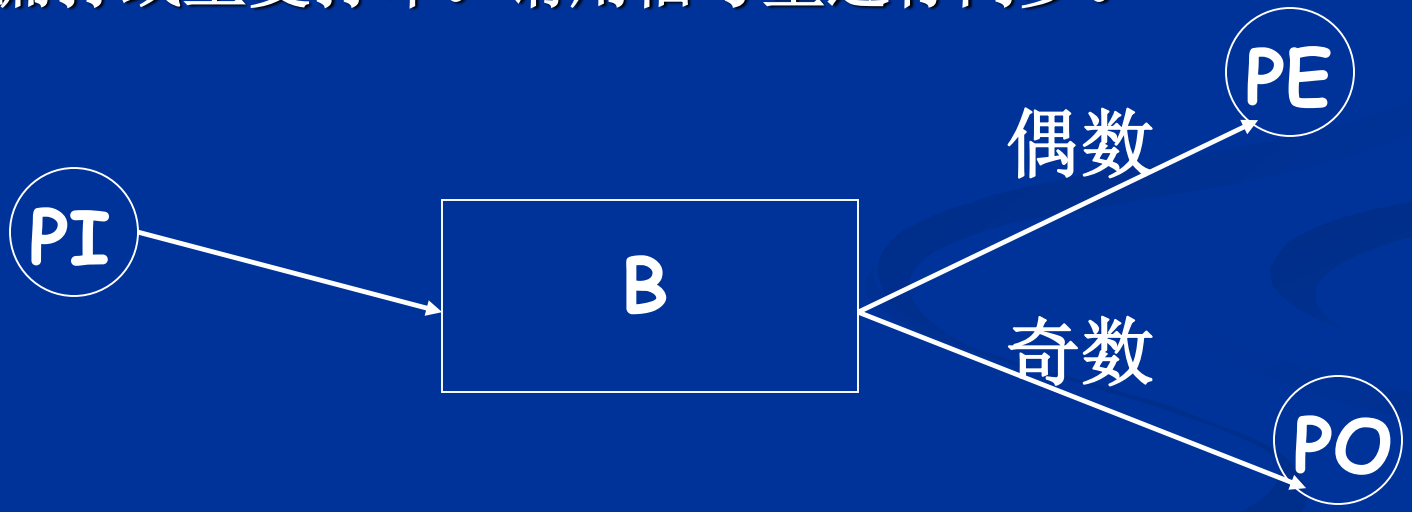
```
Parbegin
  Process PI :
  Begin
     repeat
        read x from I/O ;
        B := x ;
     until false
  end
  Process PO :
  Begin
     repeat
        y := B ;
        print( y) ;
     until false
  End
Parend
```

```
Parbegin
 VAR  S1,S2 : semaphore := 1, 0 ;
  Process PI :
  Begin
     repeat
        read x from I/O ;
        wait( S1 ) ;
        B := x ;
        signal(S2 ) ;
     until false
  end
  Process PO :
  Begin
     repeat
        wait( S2 ) ;
        y := B ;
        signal(S1 ) ;
        print( y) ;
     until false
  End
Parend
```

2.3 进程同步

(2) 信号量用于同步

例4：PI,PE,PO三进程通过共享缓冲区B协作,PI负责产生随机数并保存在B中，当B中数据为奇数时由PO负责打印，当B中数据为偶数时由PE负责打印，要求每个数据必须且只打印一次，不许漏打或重复打印。请用信号量进行同步。



```
Parbegin

  Process PI :
  Begin
     repeat
        generate x randomly;

        B := x ;

     until false
  end
  Process PE :
  Begin
     repeat

        y := B ;

        print( y) ;
     until false
  End
Process PO :
  Begin
     repeat

        z := B ;

        print( z) ;
     until false
  End
Parend
```

```
Parbegin
  var S,SO,SE:semaphore :=1,0,0;
  Process PI :
  Begin
     repeat
        generate x randomly;
        wait(S);
        B := x ;
        if ( x is odd ) signal( SO );
           else         signal( SE ) ;
     until false
  end
  Process PE :
  Begin
     repeat
        wait(SE);
        y := B ;
        signal( S );
        print( y) ;
     until false
  End
Process PO :
  Begin
     repeat
        wait(SO);
        z := B ;
        signal( S );
        print( z) ;
     until false
  End
Parend
```