计算机操作系统

Operating Systems

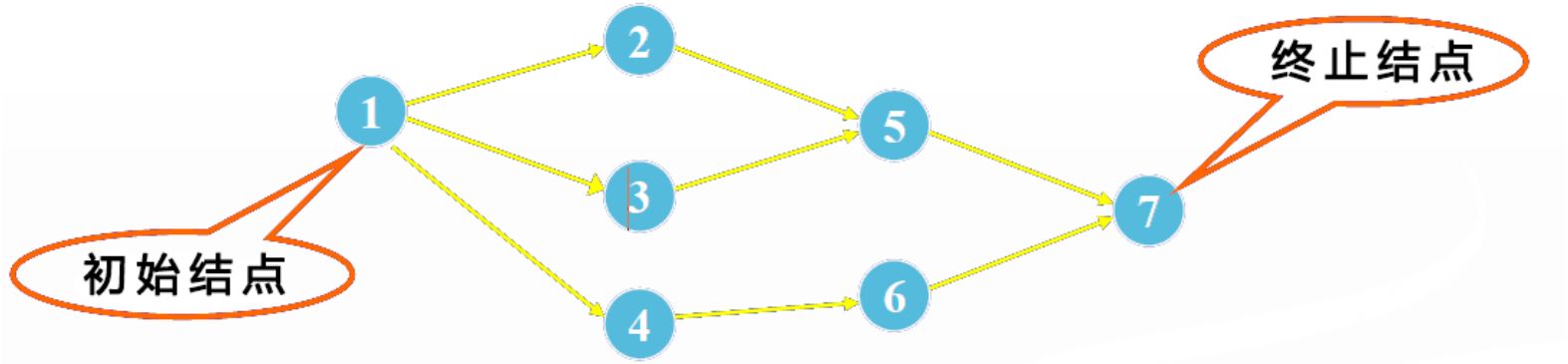
李　琳

第二章 进程的描述与控制

2.4 进程同步

引子：前趋图

- 前趋图：用于描述实体(进程、程序段)之间执行次序的**DAG**。



程序段级顺序执行

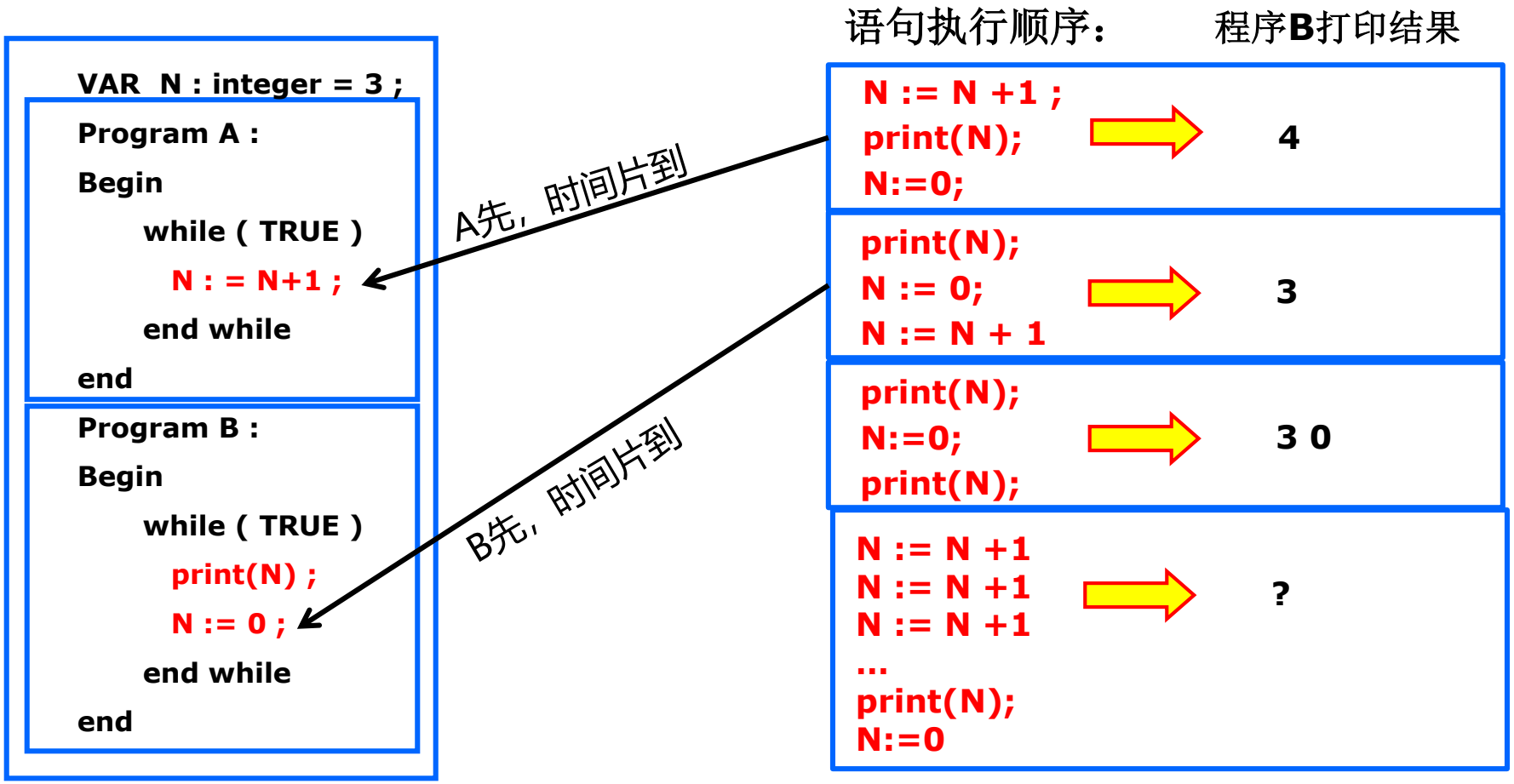
输入程序段: I
计算程序段: **C**
打印程序段: **P**

语句级顺序执行

S1: a := x + y
S2: b := a – 5
S3: c := b + 1

2.4 进程同步

引子：并发执行的问题



结论：程序B因为受到程序A的影响，失去了封闭性，运行结果出现了不可再现性。

2.4 进程同步

2.4.1 基本概念

- 产生的原因

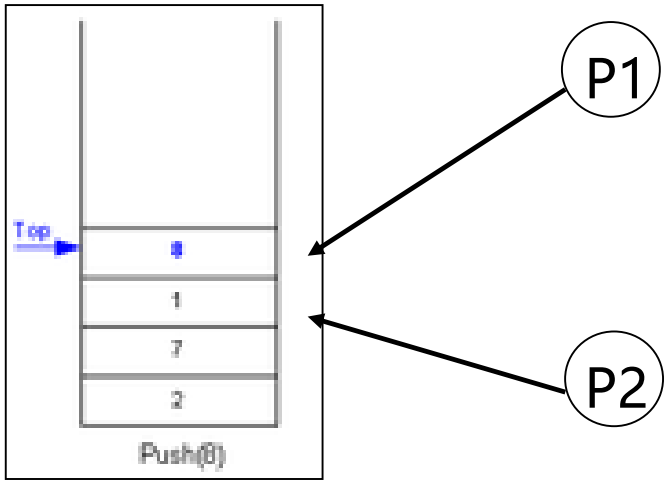
在多道程序的环境中，系统中的多个进程可以并发执行，同时它们又要共享系统中的资源（硬件资源、内存变量、系统信号等）。由此将会产生错综复杂的进程间相互制约的关系。

- 两种制约关系

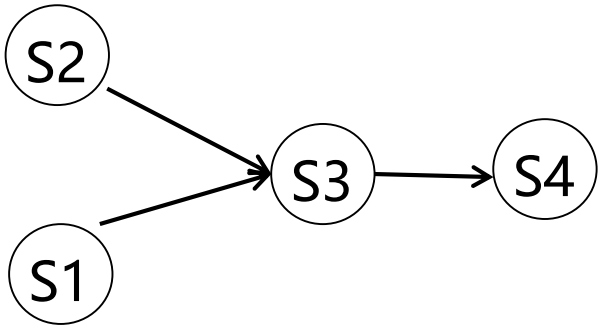
间接相互制约关系：共享某种系统资源。 → 互斥关系

直接相互制约关系：主要源于进程间合作。 → 同步关系

两个进程并发进行入栈操作



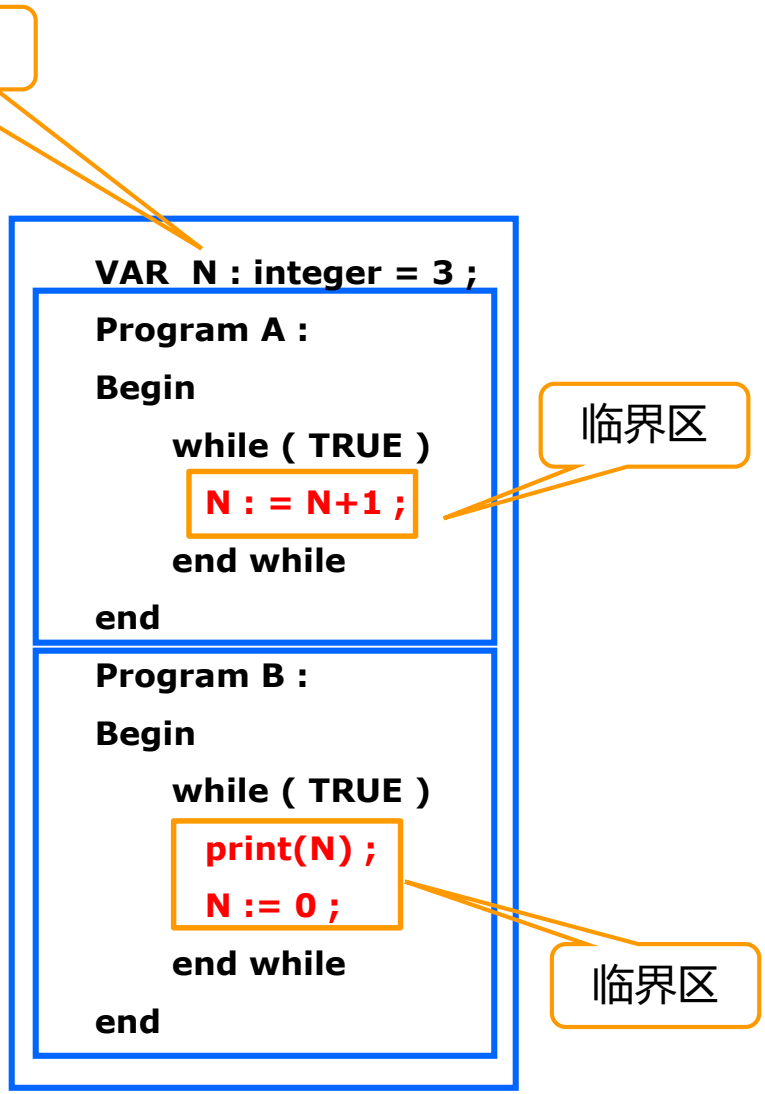
多个进程中的语句要求先后顺序



2.4 进程同步

2.4.1 基本概念

- 一次仅能为一个进程所使用的资源称为临界资源。
  - 硬件资源：打印机；
  - 软件资源：内存变量、指针、数组
- 进程中访问临界资源的代码段称为临界区。



如何实现临界资源的互斥访问?

临界资源访问互斥

↓

临界区访问互斥

2.4 进程同步

2.4.1 基本概念

- 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

- 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

2.4 进程同步

2.4.2 早期方法

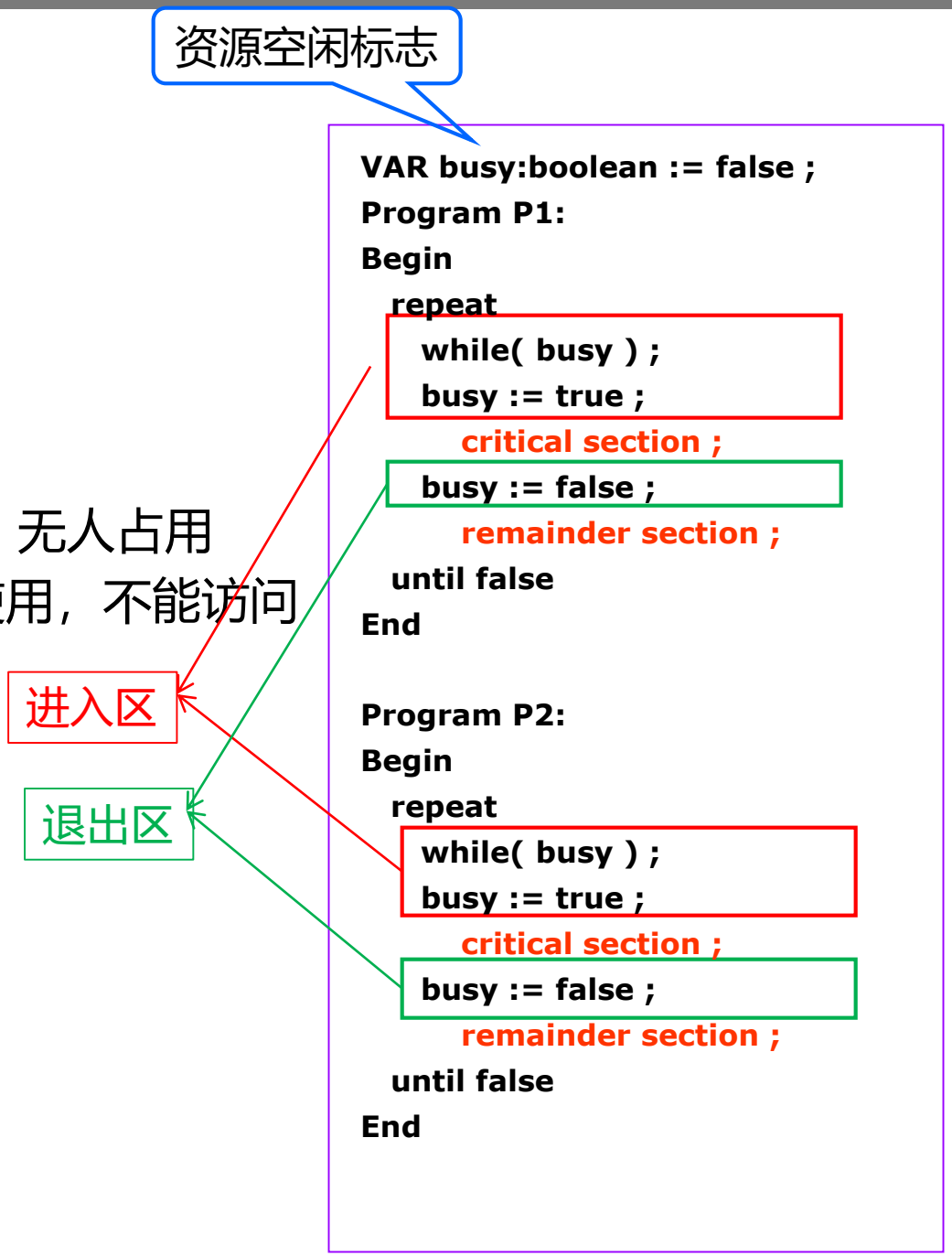
- 软件解法1：按需访问

✓资源空闲标志: busy

- busy = false : 资源空闲，无人占用
- busy = true : 资源正在使用，不能访问

✓违背什么原则?

忙则等待，让权等待



2.4 进程同步

2.4.2 早期方法

- 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

轮转标志

```
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
```

2.4 进程同步

2.4.2 早期方法

- 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

愿望标志

```
VAR pturn,qturn:boolean := 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
```

2.4 进程同步

2.4.2 早期方法

- 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 P0:
Begin
        repeat
                interested[0] = TRUE;// 0想进入临界区
                turn = 0; // 轮到0
                while( turn == 0 && interested[1] == TRUE);
                //轮到0并且1想进入临界区2个条件同时成立就等待
                   critical section ;
                interested[0] = FALSE; // 0不想进入临界区
                   remainder section ;
        until false
End
```

```
Program P1:
Begin
        repeat
                interested[1] = TRUE;// 1想进入临界区
                 turn = 1; // 轮到1
                while( turn == 1 && interested[0] == TRUE);
                //轮到1并且0想进入临界区2个条件同时成立就等待
                   critical section ;
                interested[1] = FALSE; // 本进程已离开临界区
                   remainder section ;
        until false
End
```

要点：

（1）turn设为自己并判断是为了另一个进程无意愿的情况下才能进入。否则等于把机会让给对方。Turn后赋值的需要等待。忙则等待。

（2）临界区进入的条件只要打破一个就行。而turn必然只有一个值，必然有一个进程条件被打破，空闲让进 。

（3）turn后赋值的那个进程，总会等到另一个进程不感兴趣了打破另一个条件，进入临界区，有限等待。

2.4 进程同步

2.4.2 早期方法

- 硬件解法：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
```

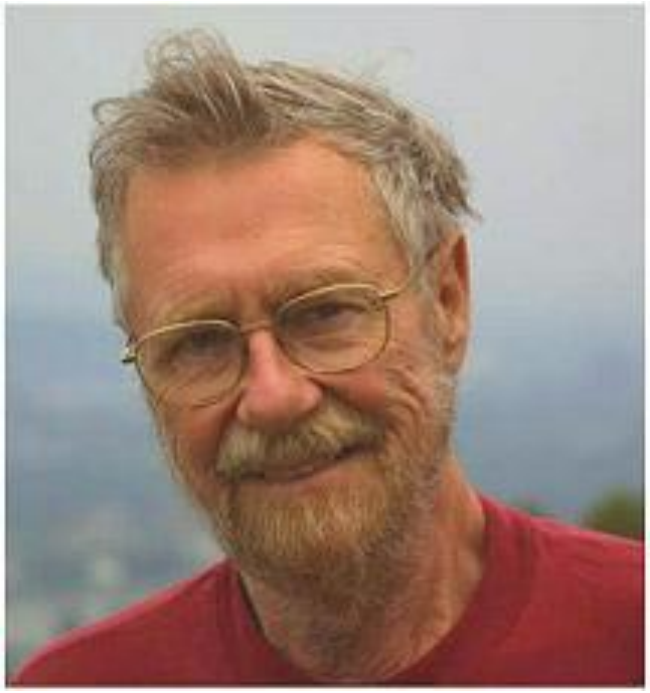
```
VAR lock :boolean := false ;
Program P1:
Begin
  repeat
    enter_region(lock );
       critical section ;
    leave_region(lock);
       remainder section ;
  until false
End

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

2.4 进程同步

2.4.3 信号量方法（OS提供）

- 软件解法
  - ✓忙等待
  - ✓实现需要很高的编程技巧，一事一议
- 硬件解法
  - ✓忙等待
  - ✓只能处理互斥关系
- 操作系统需要提供一种可靠方法
  - ✓信号量机制
  - ✓荷兰计算机科学家Edsger Wybe Dijkstra; 1972年ACM 图灵奖获得者



2.4 进程同步

2.4.3 信号量方法

- 整数型信号量
  - ✓一个整数信号量类型
  - ✓两个原子操作：wait/signal(P/V)

忙等待

操作系统代码

```
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
```

解决互斥关系：用户程序代码

```
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
```

2.4 进程同步

2.4.3 信号量方法

- 记录型信号量
  - ✓一个记录型信号量类型**semaphore**
  - ✓两个原子操作：**wait/signal(P/V)**

操作系统代码

```
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
```

解决互斥关系：用户程序代码

```
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
```

2.4 进程同步

2.4.3 信号量方法

- 整型信号量的物理含义
  - ✓S>0　资源的数目
  - ✓S=0　没有资源
  - ✓S<0　程序出错

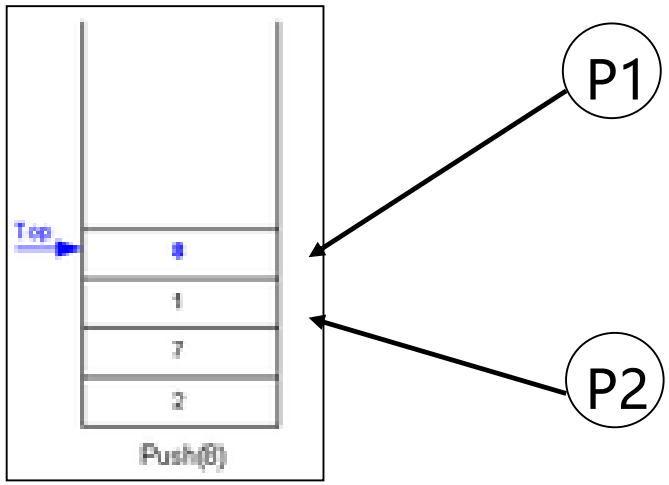
- 记录型信号量的物理含义
  - ✓S.value >0　资源的数目
  - ✓S.value =0　没有资源，阻塞临界
  - ✓S.value <0　绝对值是因为得不到这个资源而被阻塞的进程数目（S.L队列中进程数目）

2.4 进程同步

2.4.4 信号量的应用

- 解决互斥问题

两个进程并发进行入栈操作



要点:

(1) 首先写出解决问题的一般程序
(2) 发现进程间需要互斥
(3) 在临界区前后加上PV操作

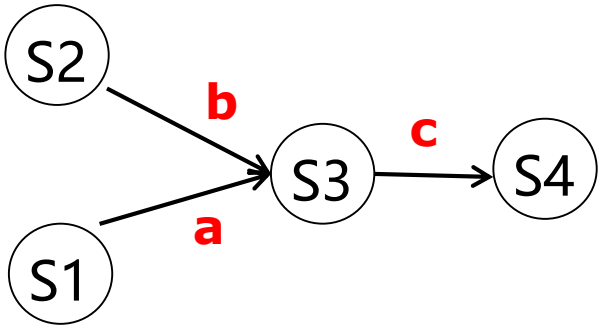
```
Parbegin
  VAR  top : integer = -1 ;
       stack: array[0..n-1] of item;
       mutex: semaphore :=1 ;
  Process P1 :
  Begin
     while ( TRUE )
       wait( mutex ) ;
        top : = (top+1) mod n;
        stack[top] := x ;
       signal(mutex ) ;
     end while
  end
  Process P2 :
  Begin
 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

2.4 进程同步

2.4.4 信号量的应用

- 解决同步问题

多个进程中的语句要求先后顺序



Parbegin

VAR a,b,c: semaphore :=1 ;

Process P1 :

Begin … S1; signal(a) ; … End

Process P2 :

Begin … S2; signal(b) ; … End

Process P3 :

Begin … wait(a) ; wait(b) ; S3; signal(c) ; … End

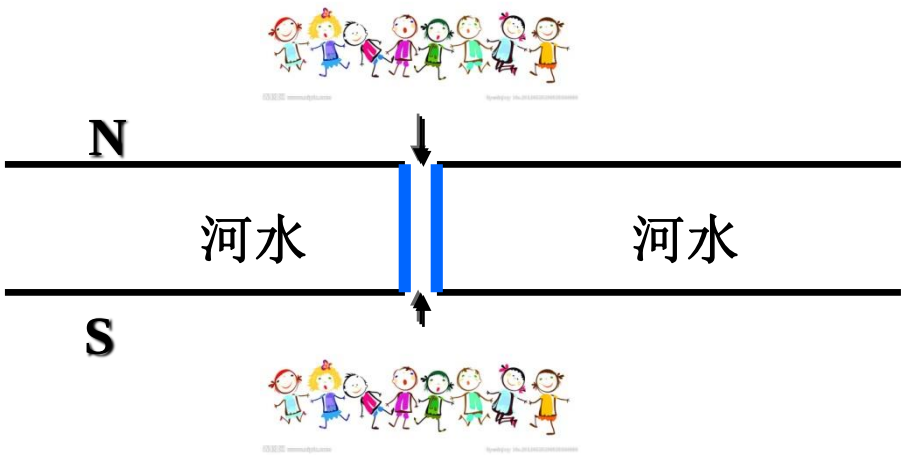
Process P4 :

Begin … wait(c) ; S2; … End

Parend

试一试

- 过独木桥问题，一次只能过一个人



能否每个人是一个进程?

```
Parbegin
   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
```