



操作系统

L10 进程同步3

胡燕

大连理工大学 软件学院



6.7-生产者消费者问题

回顾练习

- 问题模式(1:1:n)
 - 1 producer, 1 consumer, warehouse with capacity n(buffer size=n)

```
P:
  i = 0;
  while (true) {
    生产产品;
    P(empty);
    往Buffer [i]放产品;
    i = (i+1) % n;
    V(full);
  }
```

```
C:
  j = 0;
  while (true) {
    P(full);
    从Buffer[j]取产品;
    j = (j+1) % n;
    V(empty);
    消费产品;
  }
```

需不需要加互斥锁？为什么？

6.7-生产者消费者问题

回顾练习

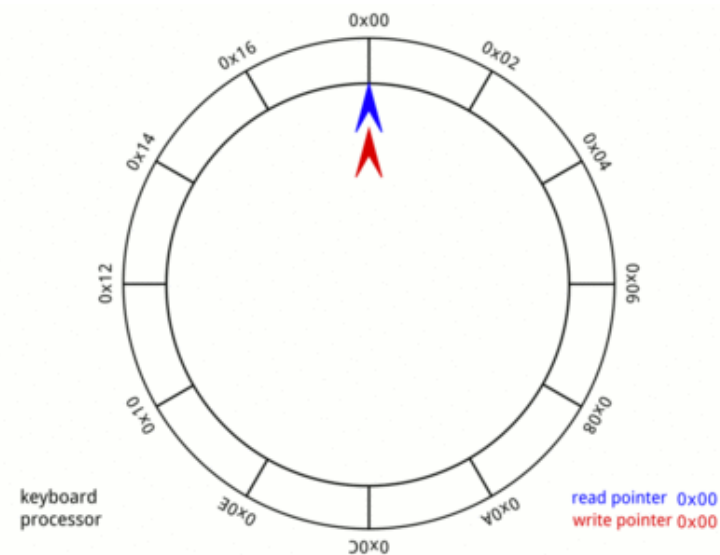
- 问题模式(1:1:n)
 - 1 producer, 1 consumer, warehouse with capacity n(buffer size=n)

P:

```
i = 0;
while (true) {
    生产产品;
    P(empty);
    往Buffer [i]放产品;
    i = (i+1) % n;
    V(full);
};
```

C:

```
j = 0;
while (true) {
    P(full);
    从Buffer[j]取产品;
    j = (j+1) % n;
    V(empty);
    消费产品;
};
```





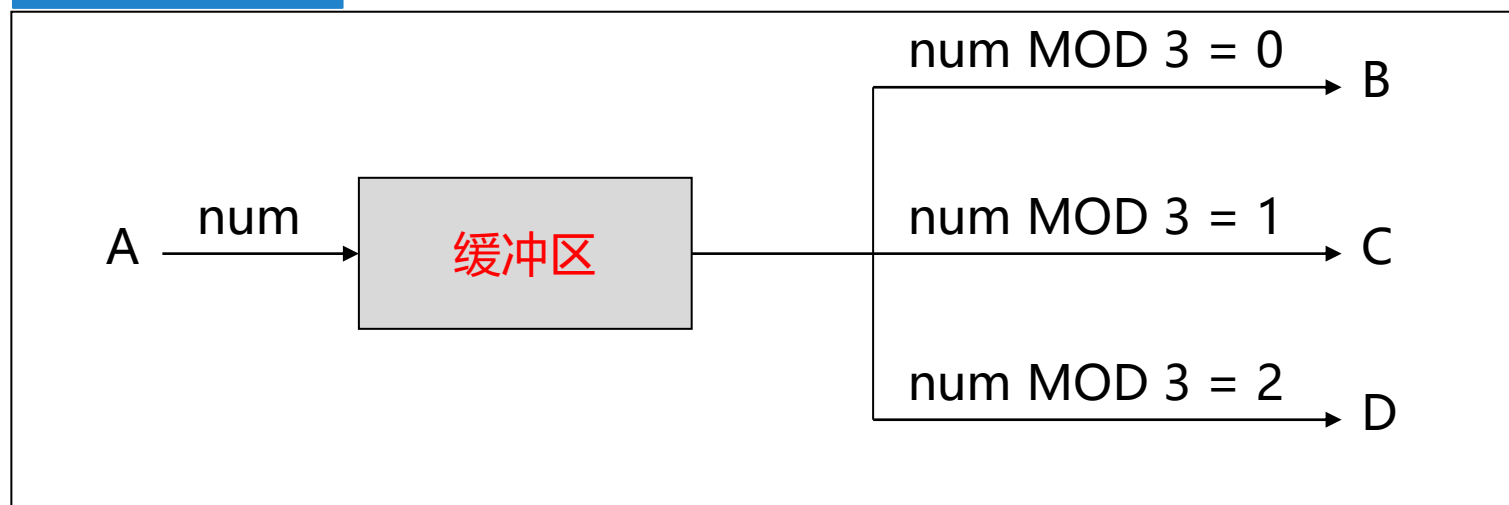
• 练习

设有4个进程A、B、C、D共享一个大小为1的缓冲区，其中：

- 进程A负责循环地从文件读一个整数并放入缓冲区，
- 进程B从缓冲区循环读入MOD 3为0的整数并累计求和；
- C从缓冲区循环地读入MOD 3为1的整数并累计求和；
- D从缓冲区中循环地读入MOD 3为2的整数并累计求和。

请分析其中的同步关系，并用信号量对该问题进行正确实现。

同步关系分析



信号量:

$S_{\text{empty}} = 1$

$S_{\text{fullB}} = 0$

$S_{\text{fullC}} = 0$

$S_{\text{fullD}} = 0$

设有4个进程A、B、C、D共享一个大小为1的缓冲区，其中：

- 进程A负责循环地从文件读一个整数并放入缓冲区，
- 进程B从缓冲区循环读入MOD 3为0的整数并累计求和；
- C从缓冲区循环地读入MOD 3为1的整数并累计求和；
- D从缓冲区中循环地读入MOD 3为2的整数并累计求和。

请分析其中的同步关系，并用信号量对该问题进行正确实现。

作答



6.7-生产者消费者问题

回顾练习

• 练习

设有4个进程A、B、C、D共享一个大小为1的缓冲区，其中：

- 进程A负责循环地从文件读一个整数并放入缓冲区，
- 进程B从缓冲区循环读入MOD 3为0的整数并累计求和；
- C从缓冲区循环地读入MOD 3为1的整数并累计求和；
- D从缓冲区中循环地读入MOD 3为2的整数并累计求和。

请分析其中的同步关系，并用信号量对该问题进行正确实现。

信号量：

$S_{empty} = 1$

$S_{fullB} = 0$

$S_{fullC} = 0$

$S_{fullD} = 0$

```
Process PA
Begin
  P(Sempty);
  <读入num至缓冲区>
  if (num MOD 3 = 0)
    V(SfullB);
  if (num MOD 3 = 1)
    V(SfullC);
  else
    V(SfullD);
end
```

```
Process PB
Begin
  P(SfullB);
  <从缓冲区读入num并累计求和>
  V(Sempty);
end
```

```
Process PC
Begin
  P(SfullC);
  <从缓冲区读入num并累计求和>
  V(Sempty);
end
```

```
Process PD
Begin
  P(SfullD);
  <从缓冲区读入num并累计求和>
  V(Sempty);
end
```



某寺庙，有小和尚、老和尚若干。庙内有一水缸，由小和尚提水入缸，供老和尚饮用。水缸可容纳 30 桶水，每次入水、取水仅为1桶，不可同时进行。水取自同一井中，水井径窄，每次只能容纳一个水桶取水。设水桶个数为5个，试用信号灯和PV操作给出老和尚和小和尚的活动。



和尚取水问题: 分析进程间协作关系

```
young_monk(){
```

```
    while(1){
```

```
        go to the well;
```

```
        get water;
```

```
        go to the temple;
```

```
        pour the water into the big jar;
```

```
    }
```

```
}
```

```
old_monk(){
```

```
    while(){
```

```
        get water;
```

```
        drink water;
```

```
    }
```

```
}
```

分析其中的协作关系...



和尚取水问题: 分析资源, 设定信号量

```
young_monk(){
    while(1){
        go to the well;

        get water;

        go to the temple;

        pour the water into the big jar;
    }
}

old_monk(){
    while(){
        get water;

        drink water;
    }
}
```

semaphore empty=30; // 表示缸中目前还能装多少桶水, 初始时能装**30**桶水
semaphore full=0; // 表示缸中有多少桶水, 初始时缸中没有水
semaphore buckets=5; // 表示有多少只空桶可用, 初始时有**5**只桶可用
semaphore mutex_well=1; // 用于实现对井的互斥操作
semaphore mutex_bigjar=1; // 用于实现对缸的互斥操作



和尚取水问题: solution

```
young_monk(){
```

```
  while(1){
```

```
    P(empty);
```

```
    P(buckets);
```

```
    go to the well;
```

```
    P(mutex_well);
```

```
    get water;
```

```
    V(mutex_well);
```

```
    go to the temple;
```

```
    P(mutex_bigjar);
```

```
    pure the water into the big jar;
```

```
    V(mutex_bigjar);
```

```
    V(buckets);
```

```
    V(full);
```

```
  }
```

```
}
```

```
old_monk(){
```

```
  while(){
```

```
    P(full);
```

```
    P(buckets);
```

```
    P(mutex_bigjar);
```

```
    get water;
```

```
    V(mutex_bigjar);
```

```
    drink water;
```

```
    V(buckets);
```

```
    V(empty);
```

```
  }
```

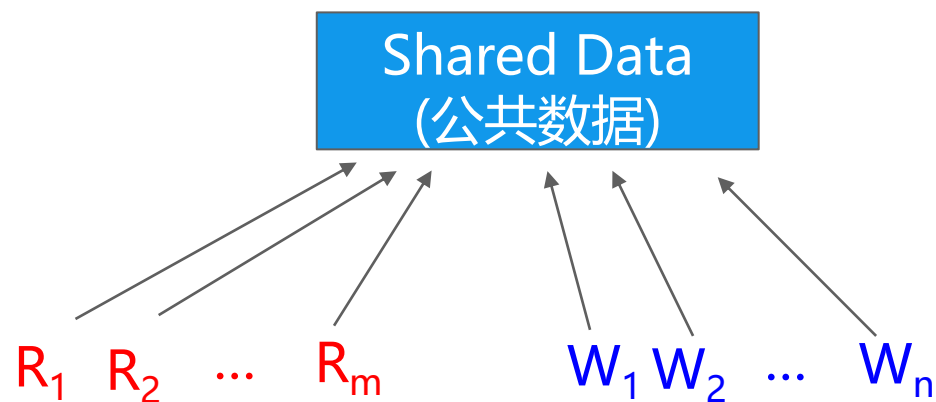
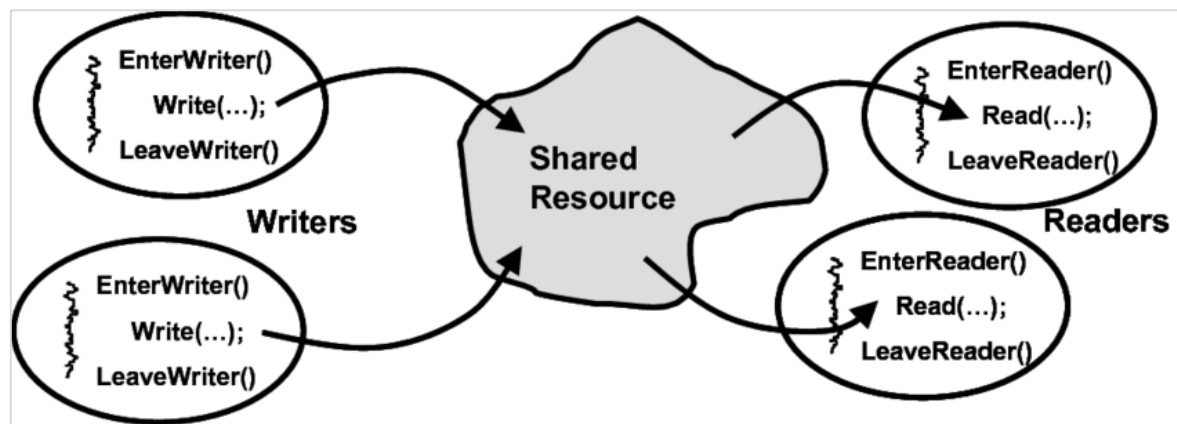
```
}
```

读者写者问题

Reader Writer Problem

01

读者写者问题



6.8-读者写者问题

问题表示 & 同步分析

读者写者问题同步分析

步骤1：读者-写者进程表示







R:

```
while (true) {
    read();
}
```

W:




```
while (true) {
    write();
}
```

步骤2：进程协作关系分析

Process 1	Process 2	Not allowed
		
Process 1	Process 2	Not allowed
		




有读者在读时，不可以写
有写者在写时，不可以读

R-W互斥

Process 1	Process 2	Not allowed
		

不允许两个写着同时写共享数据

W-W互斥

Process 1	Process 2	Not allowed
		

允许两个读者同时读共享数据

共享读

6.8-读者写者问题

问题求解: Try1

Try1: 使用单个mutex对读写操作进行互斥保护



mutex=1

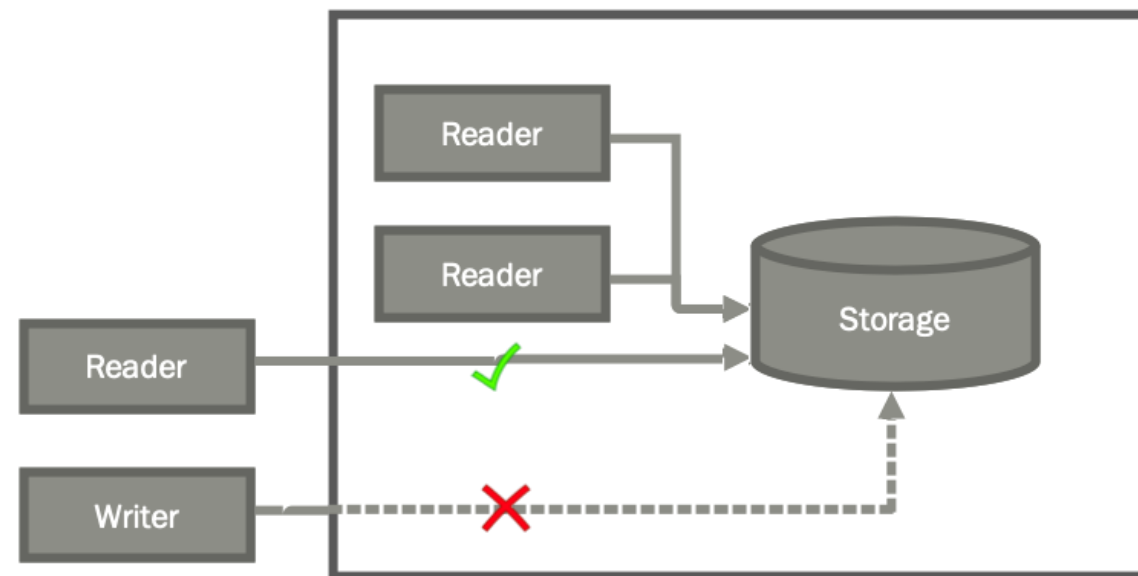
```
R:
while (true) {
    P(mutex);
    read();
    V(mutex);
}
```

```
W:
while (true) {
    P(mutex);
    write();
    V(mutex);
}
```

问题: 施加了过于严厉的互斥管制



改进目标: 实现读共享





Try2: 通过引入引用计数实现读者间共享

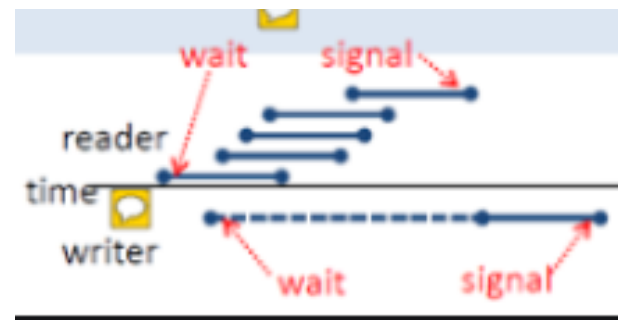
Reader:

```
while (true) {  
    P(r_mutex);  
    r_cnt++;  
    if(r_cnt==1)  
        P(mutex);  
    V(r_mutex);  
    read();  
    P(r_mutex);  
    r_cnt--;  
    if(r_cnt==0)  
        V(mutex);  
    V(r_mutex);  
}
```

Writer:

```
while (true) {  
    P(mutex);  
    write();  
    V(mutex);  
};
```

问题: Writer Starvation





Try3: 通过引入额外的**rw_mutex**用于R-W竞争

R:

```
while (true) {  
    P(rw_mutex);  
    P(r_mutex);  
    r_cnt++;  
    if(r_cnt==1)  
        P(mutex);  
    V(r_mutex);  
    V(rw_mutex);  
    read();  
    P(r_mutex);  
    r_cnt--;  
    if(r_cnt==0)  
        V(mutex);  
    V(r_mutex);  
}
```

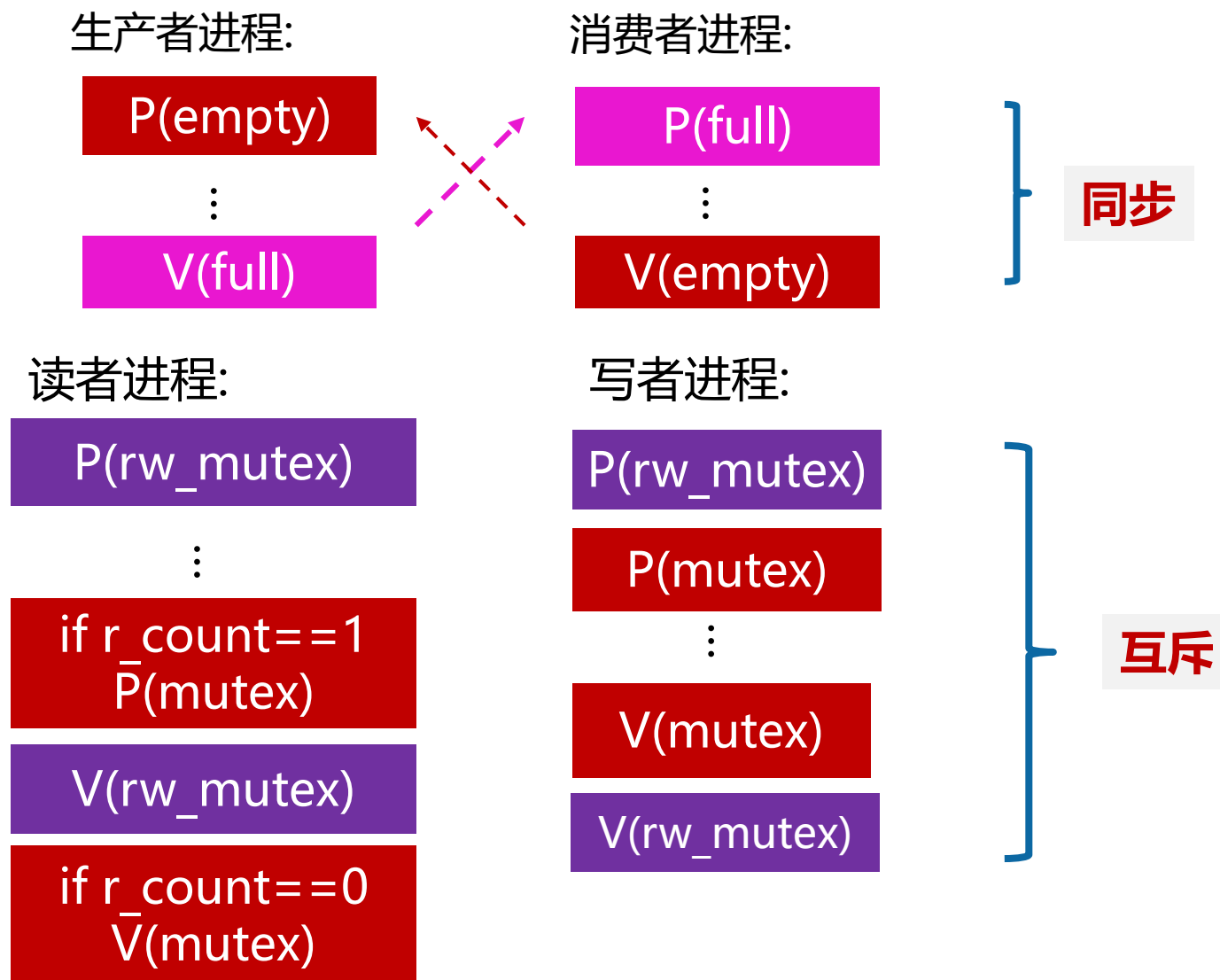
W:

```
while (true) {  
    P(rw_mutex);  
    P(mutex);  
    write();  
    V(mutex);  
    V(rw_mutex);  
};
```


信号量应用问题求解步骤

- 进程结构表示
- 同步关系分析
- 确定信号量
- 明确信号量初值
- 使用P/V操作实施同步控制

P/V操作安排



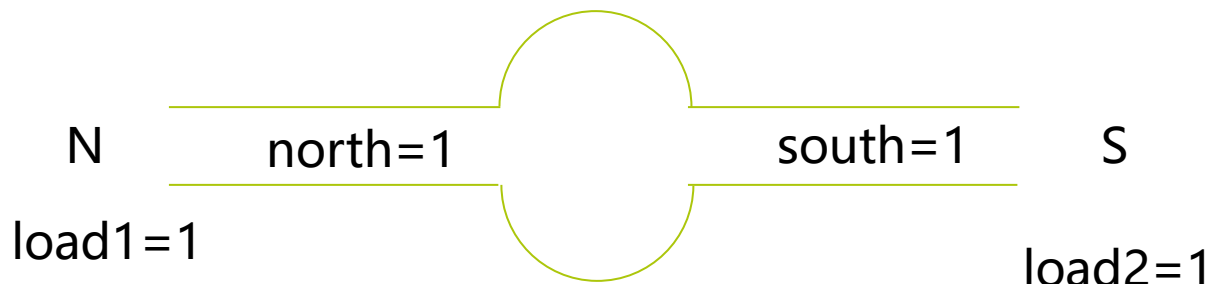


• 练习

一座小桥(最多只能承重两个人)横跨南北两岸。
任意时刻同一方向只允许一人过桥;
南侧桥段和北侧桥段较窄只能通过一人;
桥中央一处宽敞, 允许两个人通过或歇息。
试用信号量和PV操作写出南、北两岸过桥的同步算法。

同步关系分析

- 1-对桥的北入口的互斥
- 2-对桥的南入口的互斥
- 3-对桥的北小段的双向互斥控制
- 4-对桥的南小段的双向互斥控制



一座小桥(最多只能承重两个人)横跨南北两岸。

任意时刻同一方向只允许一人过桥；

南侧桥段和北侧桥段较窄只能通过一人；

桥中央一处宽敞，允许两个人通过或歇息。

试用信号量和PV操作写出南、北两岸过桥的同步算法。

作答



相关能力提升途径

- 利用Linux环境的sysvipc对象，如共享内存区、信号量等，编程实现典型的同步问题

实验检验能力提升

- 分析Linux中的信号量实现

提升系统分析能力

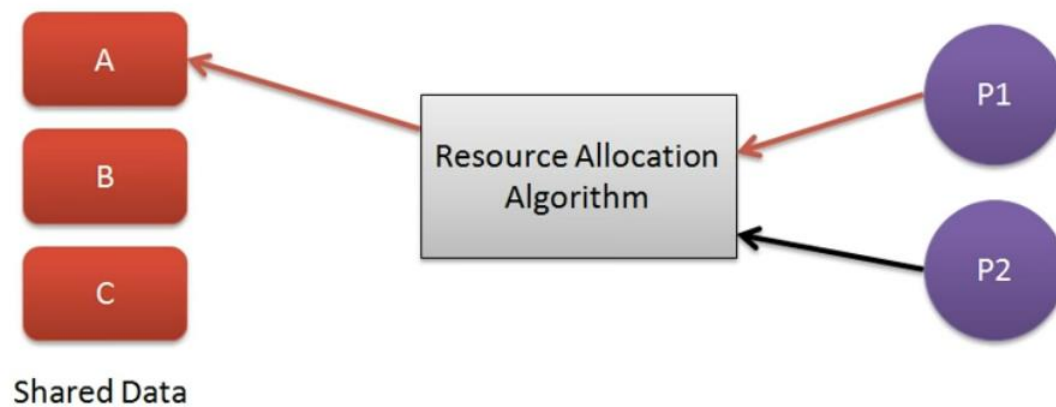
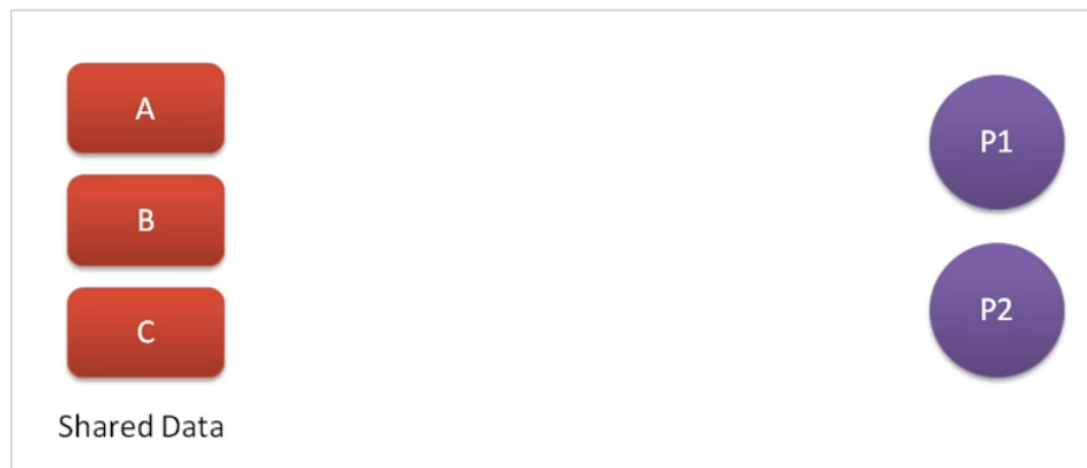
- 了解并发问题的分析与测试方法
 - 阅读文献，了解分布式系统中的并发问题

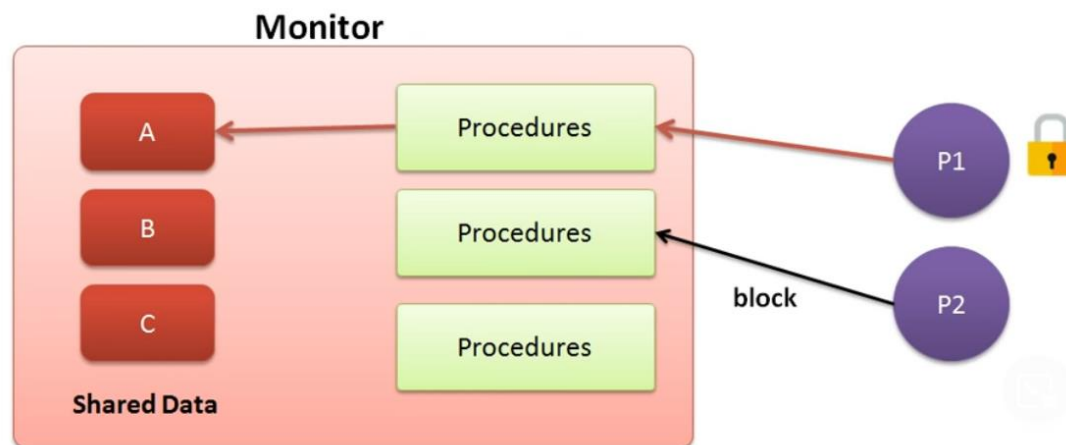
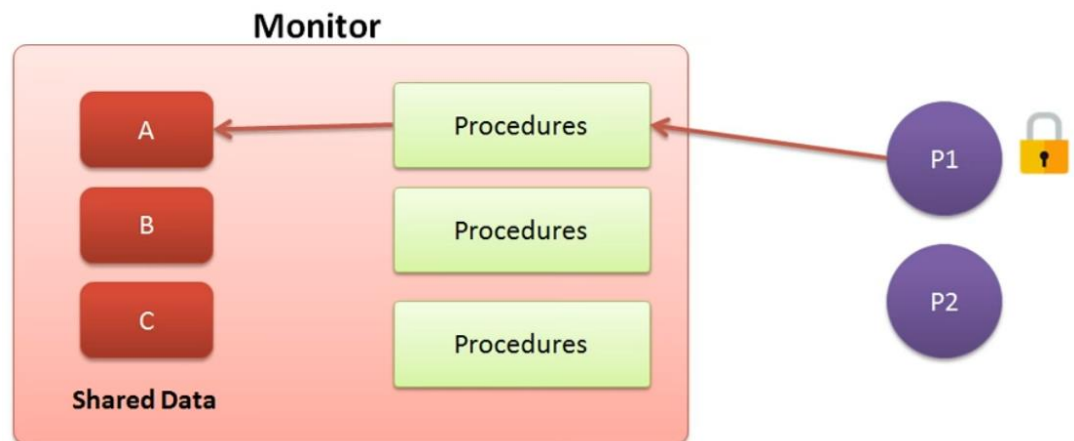
了解并发控制相关研究

管程

Monitor

02



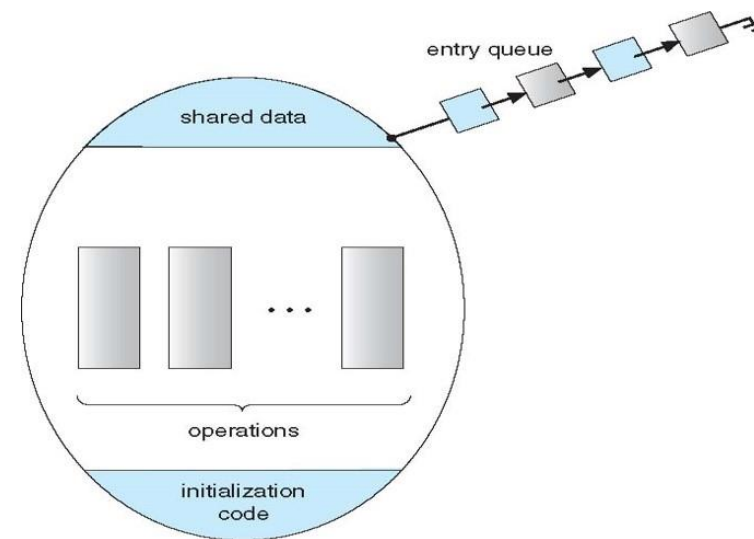


管程的概念

- 提供对共享资源的统一管理
- 提供一组对资源操作的接口
- 对并发进程对管程中资源访问的控制，由管程统一实现

monitor理念：把分散在各进程中的临界区集中起来进行管理

管程实施：建立一个“秘书”程序来管理对临界资源的访问
“秘书”每次仅允许一个进程来访，实现对资源的互斥访问



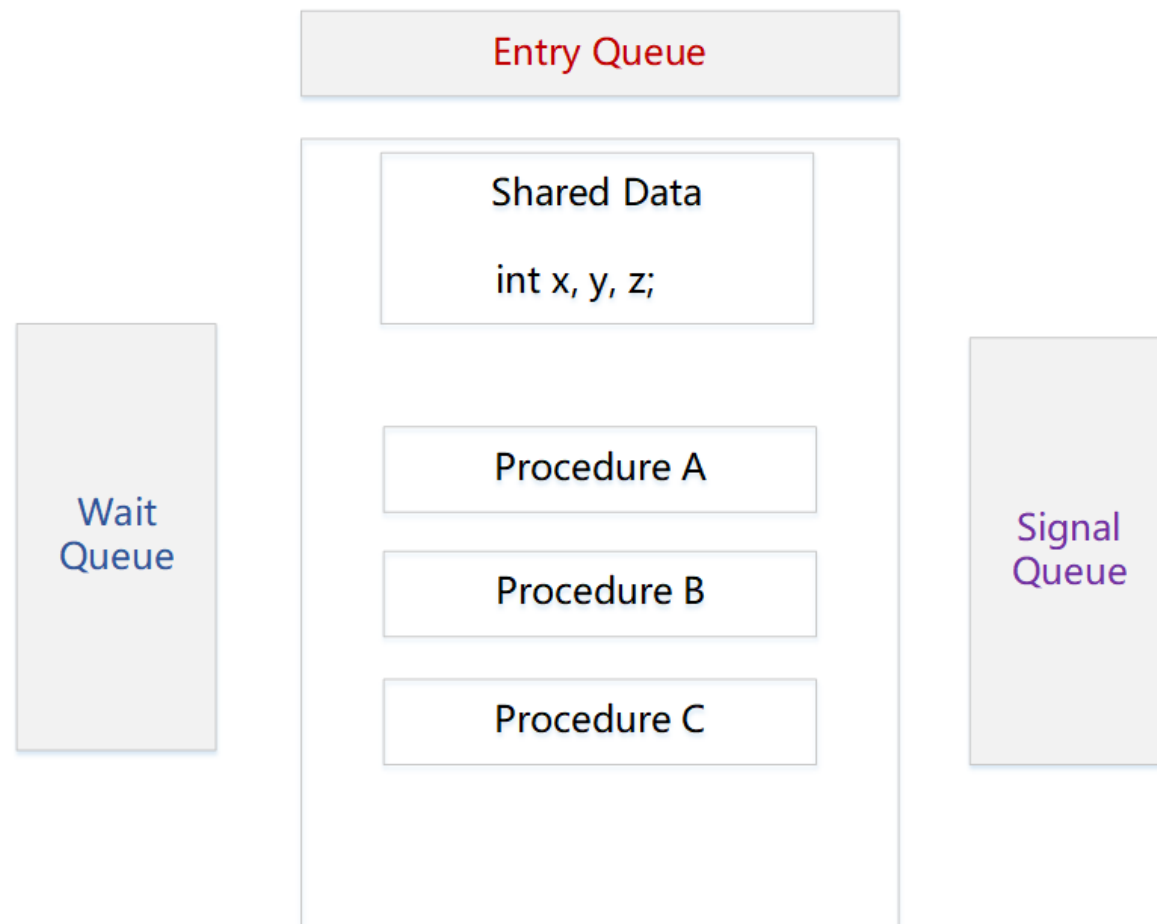


管程实现中的关键队列

Entry Queue (保证管程操作互斥, 必须)

Wait Queue

Signal Queue

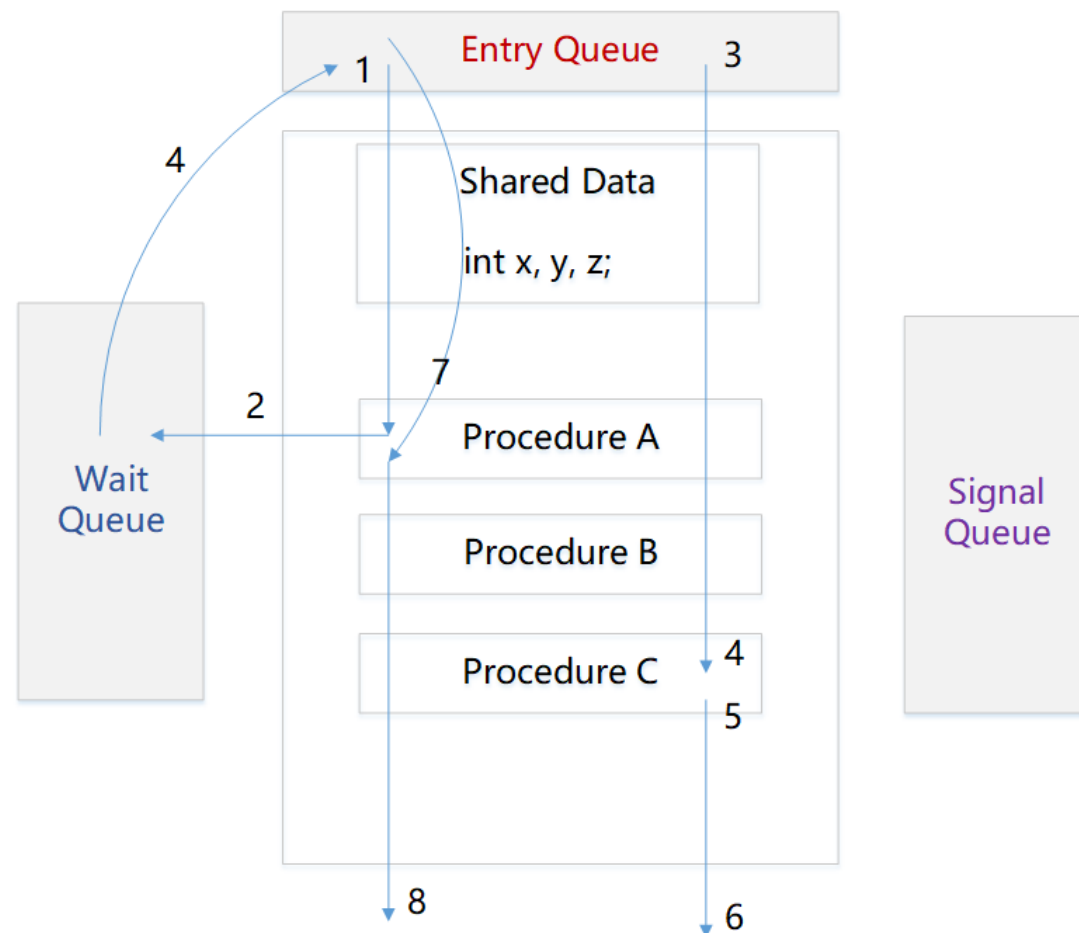




管程的三种不同实现语义

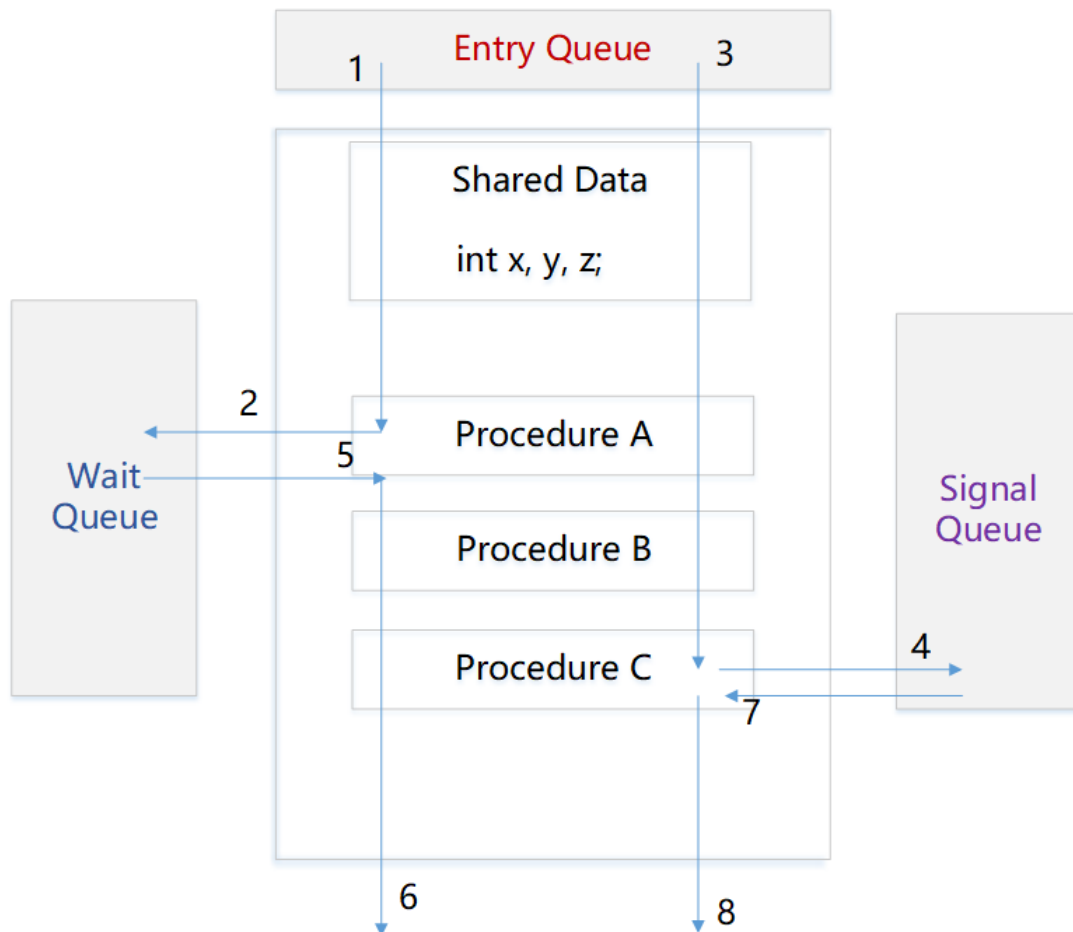
- Mesa语义
- Hoare语义
- Brinch Hanson语义

Mesa Semantics:



1. 线程A进入管程
2. 线程A等待某个资源
3. 线程B进入管程
4. 线程B释放线程A等待的资源，线程A被转到Entry Queue，而线程B继续执行
5. 线程B继续在管程内执行
6. 线程B离开管程
7. 线程A重新进入管程
8. 线程A离开管程
9. 其他线程可以继续进入管程

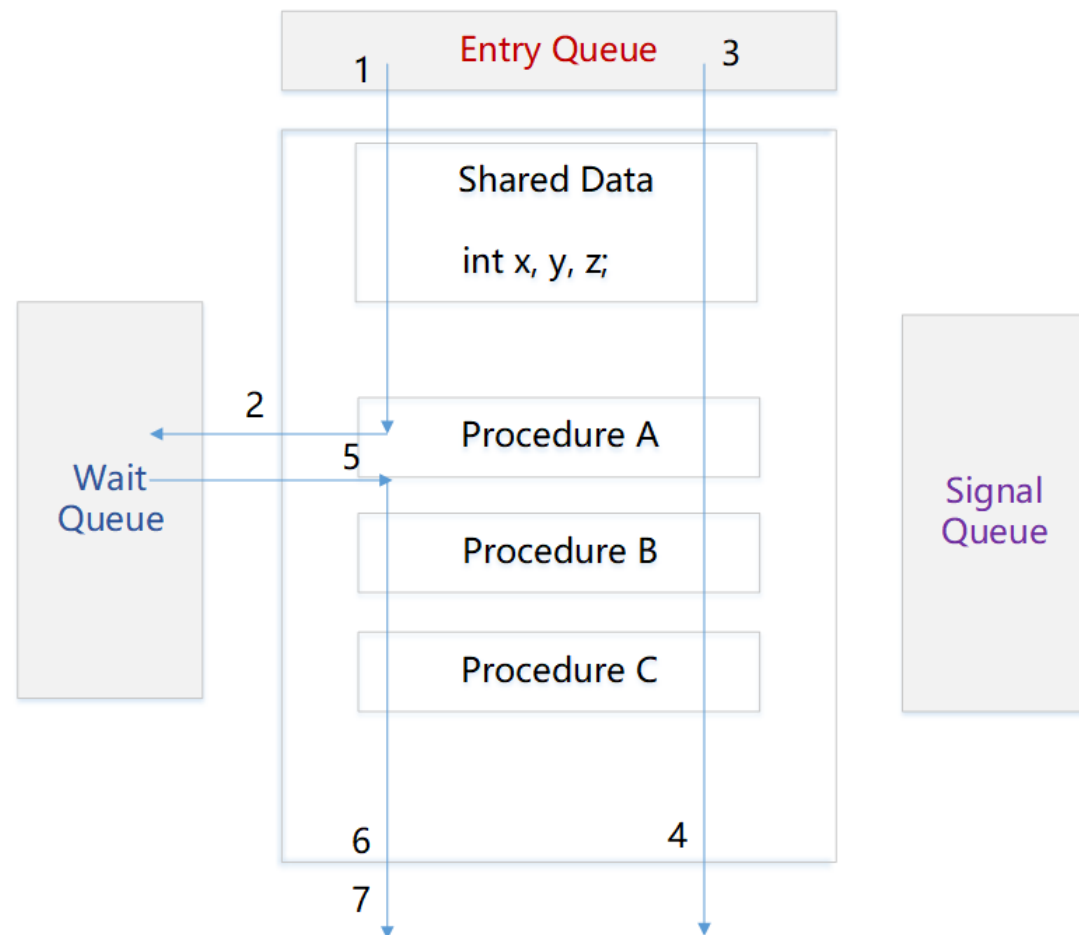
Hoare Semantics:



1. 线程A进入管程
2. 线程A等待某个资源
3. 线程B进入管程
4. 线程B释放线程A等待的资源，唤醒线程A，而线程B进入Signal Queue
5. 线程A重新进入管程继续执行
6. 线程A离开管程
7. 线程B重新进入管程
8. 线程B离开管程
9. 其他线程可以继续进入管程



Hansen Semantics:



1. 线程A进入管程
2. 线程A等待某个资源
3. 线程B进入管程
4. 线程B发资源已释放信号给线程A, 随后线程B离开管程
5. 线程A重新进入管程继续执行
6. 线程A离开管程
7. 其他线程可以继续进入管程



条件变量full, empty

处理生产者消费者问题

```
monitor PC {  
    condition : full, empty;  
    int : count = 0;  
  
    entry put {  
        if (count==max) wait (full);  
        insert item  
        count = count+1;  
        if (count==1) signal (empty);  
    }  
  
    entry get {  
        if (count==0) wait (empty);  
        remove item  
        count = count-1;  
        if (count==max-1) signal (full);  
    }  
}
```

```
producer process  
    while (TRUE) {  
        produce item  
        PC.put;  
    }
```

```
consumer process  
    while (true) {  
        PC.get;  
        consume item  
    }
```

小结:  生产者消费者问题回顾

 读者写者问题

 管程



进程独立的地址空间，异步执行

又要付出进程同步的成本



现实生活中的多任务，如何设计，怎样同步。



公交车司机与售票员需要协作，两个进程分别表示为：
Driver: while(true) { 启动车辆； 正常运行； 到站停车； }
TicketSeller: while(true) {关门； 售票； 开门； }
请用信号量来解决该同步协作问题。



设有一个可以装A、B两种物品的仓库，其容量无限大，但要求仓库中A、B两种物品的数量满足下述不等式：

$$-M \leq A \text{物品数量} - B \text{物品数量} \leq N$$

其中M和N为正整数。试用信号量和PV操作描述A、B两种物品的入库过程。



有一座东西向的独木桥，用信号量P/V操作实现：

- (1) 每次只允许一个人过桥；
- (2) 当独木桥上有行人时，同方向的行人可以过桥，相反方向的人必须等待。



有一个理发师，一把理发椅和N把供等候理发的顾客坐的椅子。
如果没有顾客，则理发师便在理发师的椅子上睡觉；
当一个顾客到来时，必须唤醒理发师进行理发；如果理发师正在理发时又有顾客来到，则如果有空椅子可做，他就坐下来等，如果没有空椅子，他就离开。
请为理发师和顾客各编写一段程序（伪码）描述他们的行为，并用信号量正确控制他们之间的并发协作。



理发师问题的信号量解决方案：

Barber

```
While (true) {
```

```
    P(clients);
```

```
    V(barbers);
```

```
}
```

Client

```
While (true) {
```

```
    n_waiting++;  
    if (n_waiting > 5) {  
        return;
```

```
    }
```

```
    V(clients);
```

```
    P(barbers);
```

```
    n_waiting--;  
    接受理发;
```

```
}
```

信号量：

clients=0

barbers=1

整数：

n_waiting=0



一座小桥(最多只能承重两个人)横跨南北两岸, 任意时刻同一方向只允许一人过桥, 南侧桥段和北侧桥段较窄只能通过一人, 桥中央一处宽敞, 允许两个人通过或歇息。试用信号量和PV操作写出南、北两岸过桥的同步算法。



桥上可能没有人，也可能有一人，也可能有两人。

共需要四个信号量：

load1来控制桥上可向南人数，初值为1；

load2来控制桥上可向北人数，初值为1；

north用来控制北段桥的使用，初值为1，用于对北段桥互斥；

south用来控制南段桥的使用，初值为1，用于对南段桥互斥。



```
tosouth(){  
    过北段桥;  
    到桥中间;  
  
    过南段桥;  
    到达南岸  
}
```

```
tonorth(){  
    过南段桥;  
    到桥中间  
  
    过北段桥;  
    到达北岸  
}
```

```
semaphore load1=1,load 2 =1;  
semaphore north=1;  
semaphore south=1;
```



```
tosouth(){  
    P(load1);  
    P(north);  
    过北段桥;  
    到桥中间;  
    V(north);  
    P(south);  
    过南段桥;  
    到达南岸  
    V(south);  
    V(load1);  
}
```

```
tonorth(){  
    P(load2);  
    P(south);  
    过南段桥;  
    到桥中间  
    V(south);  
    P(north);  
    过北段桥;  
    到达北岸  
    V(north);  
    V(load2);  
}
```



有一座东西向的独木桥，用信号量P/V操作实现：

- (1) 每次只允许一个人过桥；
- (2) 当独木桥上有行人时，同方向的行人可以过桥，相反方向的人必须等待。



n 个并发进程，信号量初始值为 1，当 n 个进程都执行 P 操作后，信号量的值为()。



信号量初值为 4，多次 PV 操作后变为 -2，那么当前时刻已经顺利获得资源的进程数目 = ()。



5 个并发进程，信号量初始值为 3，那么信号量取值范围是整数区间为 $[(), ()]$ 。



谢谢!
Thank you!