徐小龙////xuxl@njupt.edu.cn

Operating Systems

操作系统

南 京 邮 电 大 学
Nanjing University of Posts and Telecommunications

课程回顾

上节课的重点内容

引入进程并发
提高系统效率

进程异步执行
资源共享使用

导致程序结果不可再现

引入进程同步机制

进程有序合作（同步）
资源合理共享（互斥）

结果确定化

信号量与P/V操作

本讲内容

1、信号量(Semaphore)

2、P(Proberen)、V(Verhogen)操作

3、典型问题的解决方法

4、问题思考与拓展学习

- 上节课提到的，基于忙等待方法的进程协商同步机制，给系统带来额外的开销（解决让权等待的问题）
- 信号量和PV是更好解决进程同步与互斥问题的方法

信号量与P/V操作

1、信号量(Semaphore)

（1）信号量的提出

E.W.Dijkstra

荷兰著名科学家、计算机图灵奖获得者**E.W.Dijkstra**提出了一种卓有成效的进程互斥同步工具：信号量（**Semaphore**）机制，广泛应用于现代计算机系统中。

信号量与P/V操作

1、信号量(Semaphore)

（2）信号量的构成

```
Struct semaphore
{
    int value;                  //信号量值
    pointer_PCB queue;          //信号量队列指针
                                //指针指向进程等待队列
```

信号量与P/V操作

本讲内容

1、信号量(Semaphore)

2、P(Proberen)、V(Verhogen)操作

3、典型问题的解决方法

4、问题思考与拓展学习

信号量与P/V操作

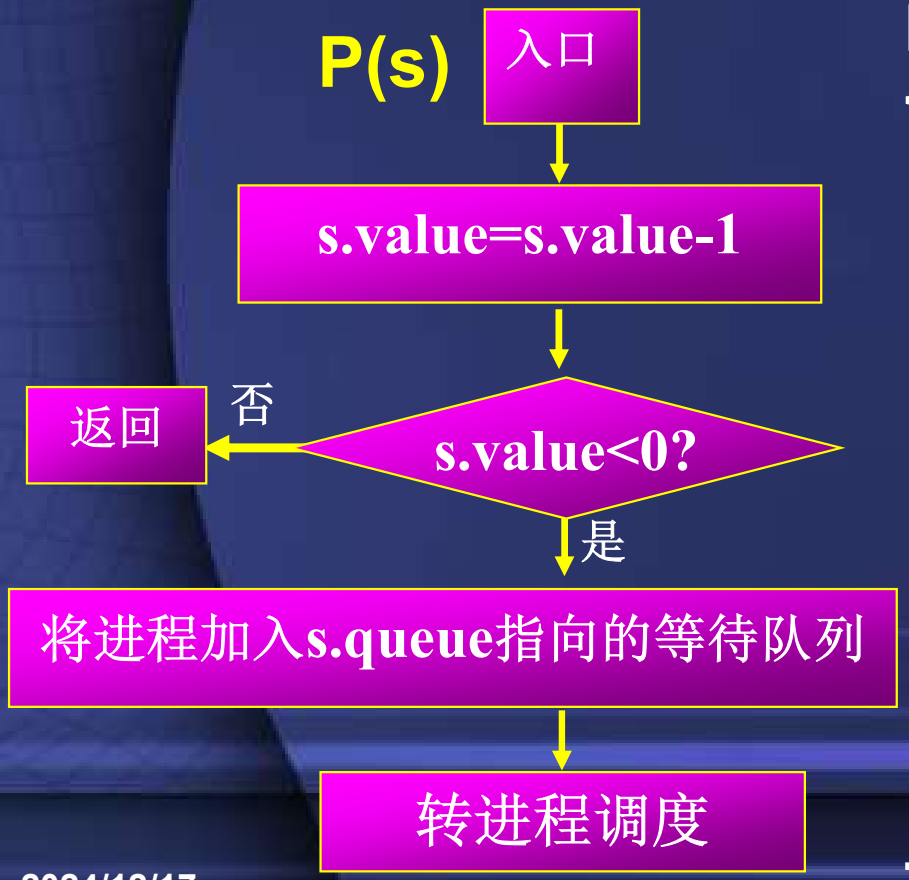
1. **信号量状态分析：**
   - **s > 0：** 表示可用资源的数量，系统处于正常状态，有可用资源供进程使用。
   - **s = 0：** 表示所有资源都已被占用，等待释放资源的进程将被阻塞。
   - **s < 0：** 在某些信号量实现中，可能表示有进程正在等待资源而被阻塞。
2. **P、V操作是否必须成对出现？**
   - **是的，P、V操作通常是成对出现的：**
     - **P（Produce）操作：** 用于申请资源，当资源可用时，P操作将信号量减1；当资源不可用时，P操作会阻塞等待。
     - **V（Vaporize）操作：** 用于释放资源，当资源被释放时，V操作将信号量加1；这通常会唤醒一个等待资源的进程。
3. **P、V操作分散在各个进程中是否会带来隐患？**
   - **是的，分散的P、V操作可能引入竞态条件（Race Condition）：**
     - **竞态条件：** 当多个进程同时访问和修改共享资源时，执行顺序可能导致不确定的结果。
     - **隐患：** 如果P、V操作不加以适当的同步控制，可能会导致信号量的不正确使用，造成资源竞争和数据不一致。

信号量与P/V操作

2、P、V操作

（1）P(Proberen) 操作

信号量的值s. value

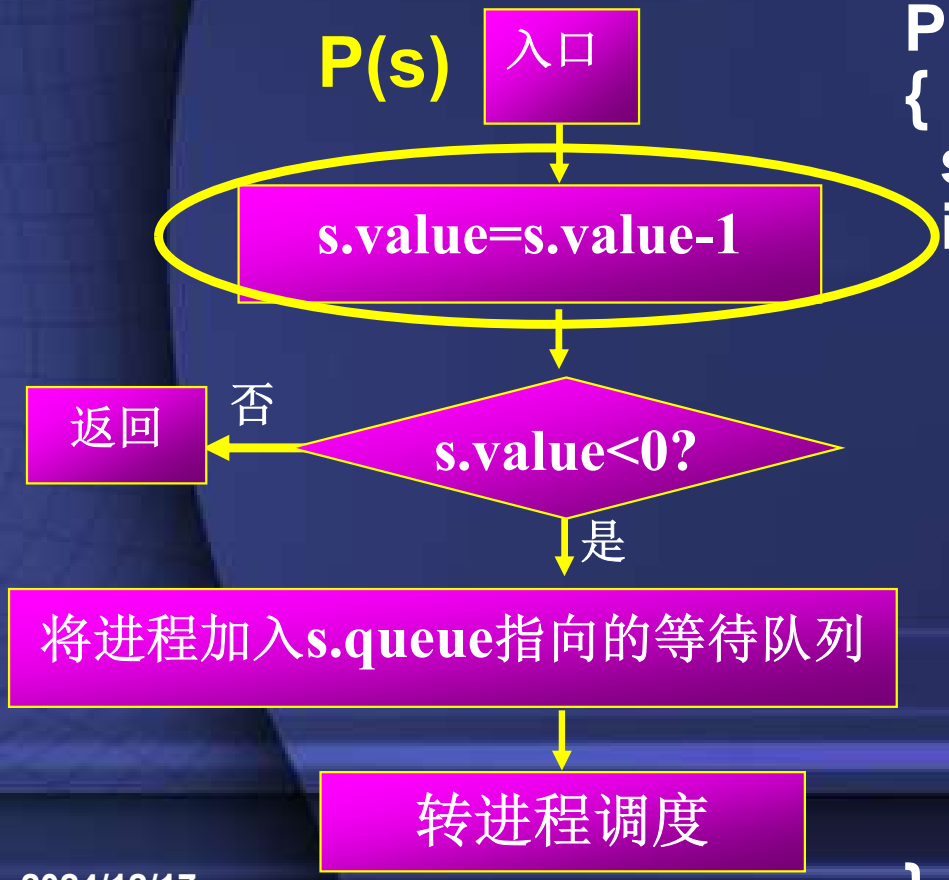



```
P(s)
{
  s.value = s.value－1 ;//s.value减1
  if (s.value < 0)
    //该进程被阻塞，进入相应队列，然
    后转进程调度
  {
    该进程状态置为等待状态;
    将该进程加入相应的等待队列
    s.queue的末尾;
  }
  //若s.value减1后仍大于或等于零，
  则进程继续执行
}
```

信号量与P/V操作

2、P、V操作

（1）P(Proberen) 操作

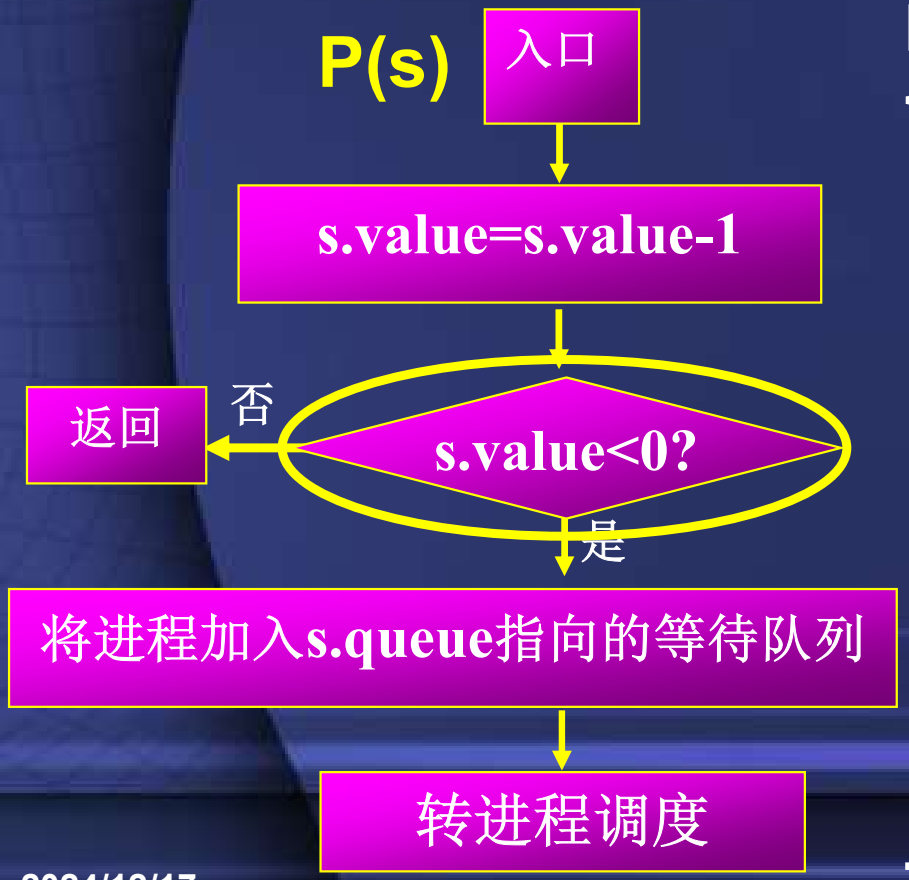



```
P(s)
{
  s.value = s.value－1 ;//s.value减1
if (s.value < 0)
   //该进程被阻塞，进入相应队列，然
   后转进程调度
   {
    该进程状态置为等待状态;
    将该进程加入相应的等待队列
   s.queue的末尾;
   }
  //若s.value减1后仍大于或等于零，
   则进程继续执行
}
```

信号量与P/V操作

2、P、V操作

（1）P(Proberen) 操作

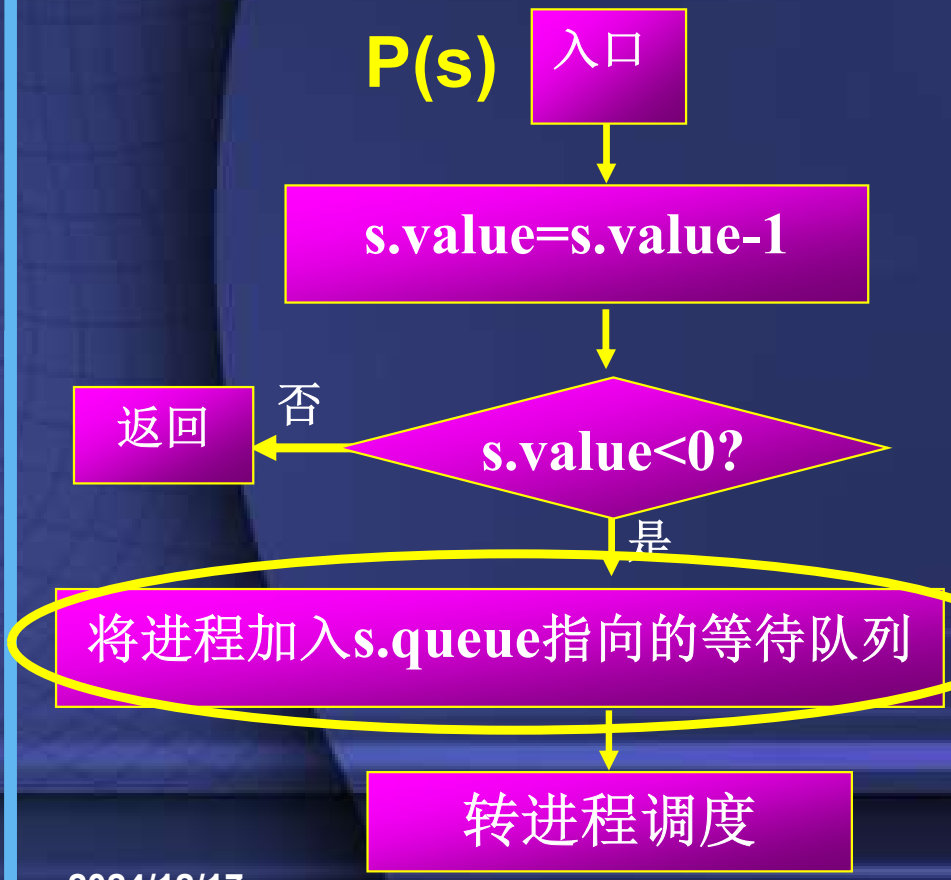



```
P(s)
{
  s.value = s.value－1 ;//s.value减1
  if (s.value < 0)
    //该进程被阻塞，进入相应队列，然
    后转进程调度
  {
    该进程状态置为等待状态;
    将该进程加入相应的等待队列
   s.queue的末尾;
  }
  //若s.value减1后仍大于或等于零，
  则进程继续执行
}
```

信号量与P/V操作

2、P、V操作

（1）P(Proberen) 操作

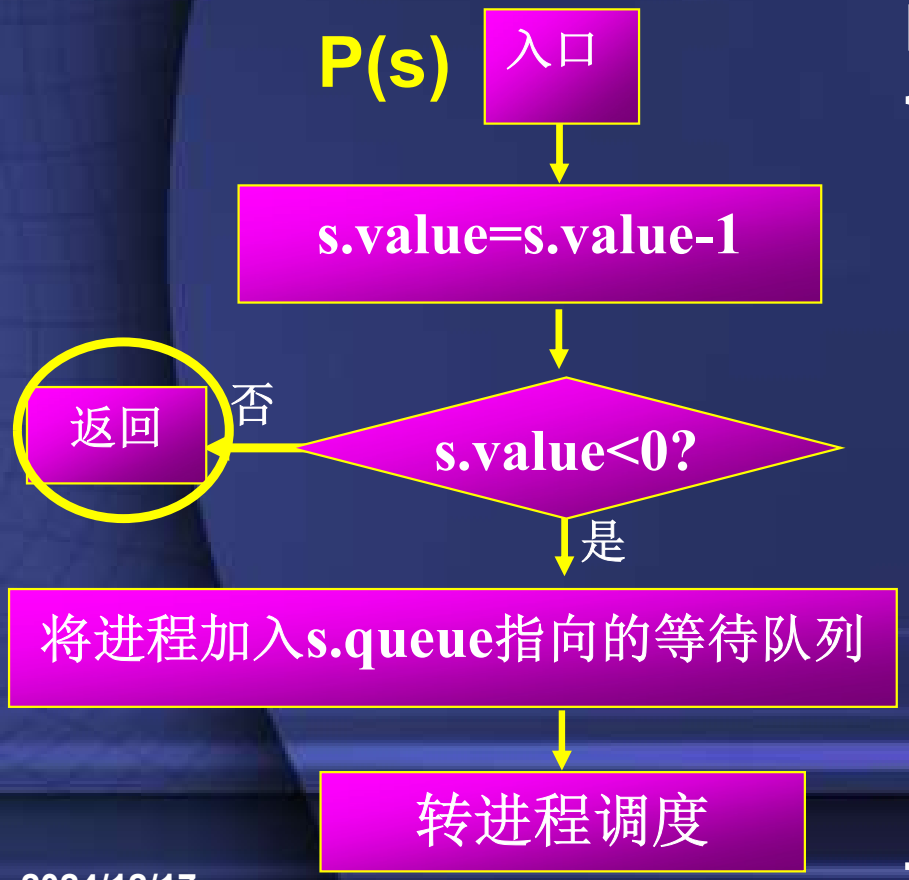



```
P(s)
{
  s.value = s.value－1 ;//s.value减1
  if (s.value < 0)
    //该进程被阻塞，进入相应队列，然
    后转进程调度
  {
    该进程状态置为等待状态;
    将该进程加入相应的等待队列
    s.queue的末尾;
  }
  //若s.value减1后仍大于或等于零，
  则进程继续执行
}
```

信号量与P/V操作

2、P、V操作

（1）P(Proberen) 操作

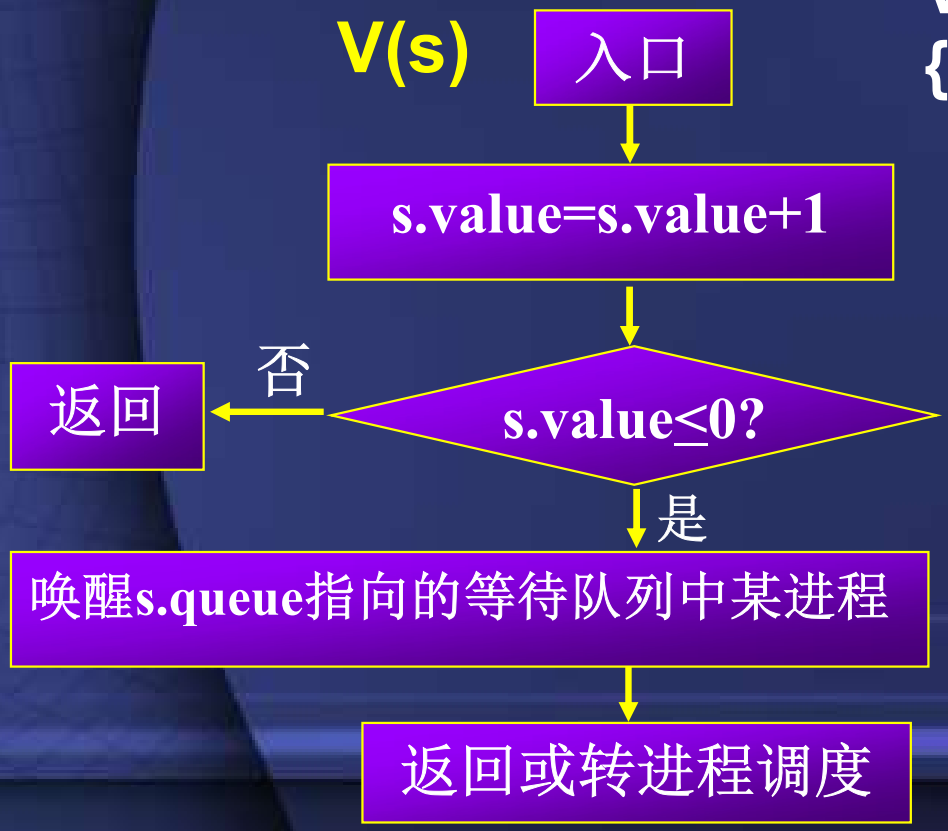



```
P(s)
{
  s.value = s.value－1 ;//s.value减1
  if (s.value < 0)
    //该进程被阻塞，进入相应队列，然
    后转进程调度
  {
    该进程状态置为等待状态;
    将该进程加入相应的等待队列
   s.queue的末尾;
  }
  //若s.value减1后仍大于或等于零，
  则进程继续执行
}
```

信号量与P/V操作

2、P、V操作

（2）V(Verhogen)操作

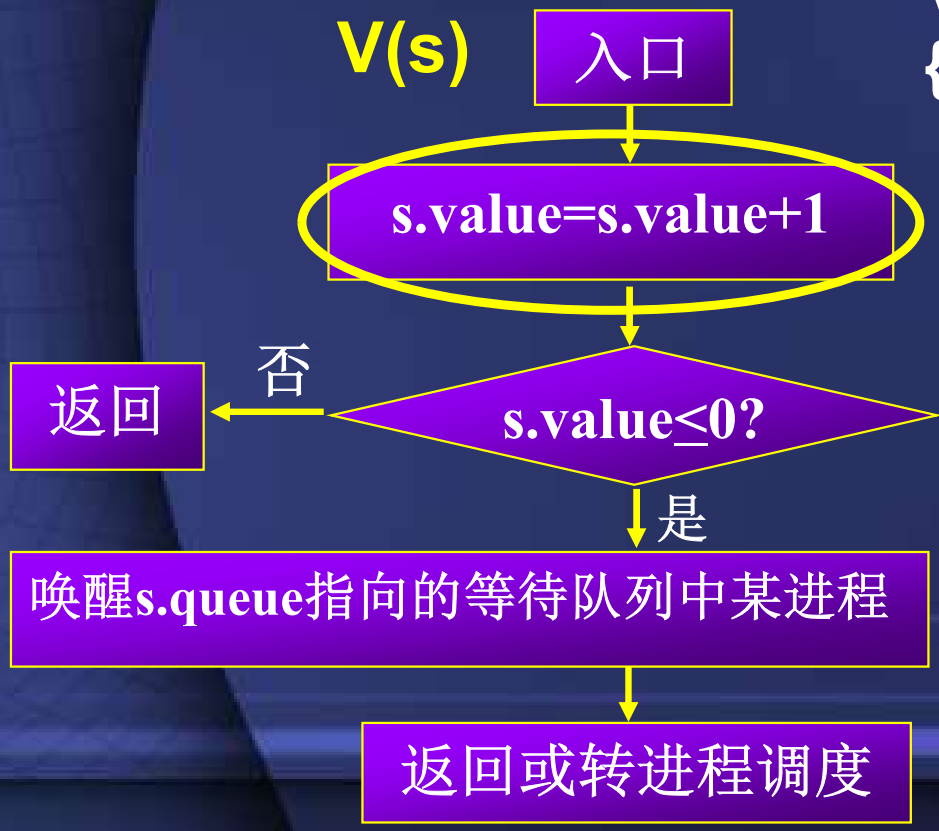



```
V(s)
{
  s.value = s.value +1; //s.value加1
  if (s.value < = 0)
   //从队列中唤醒一等待进程，然后
     继续执行或转进程调度
  {
   唤醒相应等待队列s.queue中等待
   的一个进程;
   将其状态修改为就绪态，并将其插
   入就绪队列;
   }
 //若相加结果大于零，进程继续执行
}
```

信号量与P/V操作

2、P、V操作

（2）V(Verhogen)操作

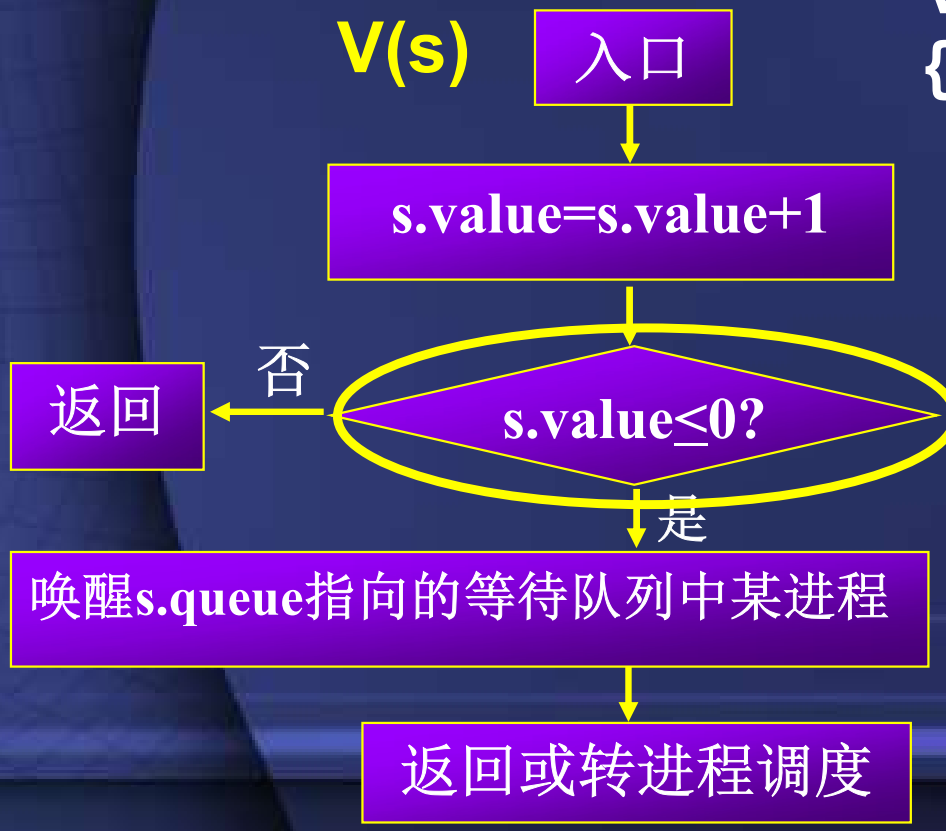



```
V(s)
{
  s.value = s.value +1; //s.value加1
  if (s.value < = 0)
   //从队列中唤醒一等待进程，然后
     继续执行或转进程调度
  {
   唤醒相应等待队列s.queue中等待
   的一个进程;
   将其状态修改为就绪态，并将其插
   入就绪队列;
   }
 //若相加结果大于零，进程继续执行
}
```

信号量与P/V操作

2、P、V操作

（2）V(Verhogen)操作

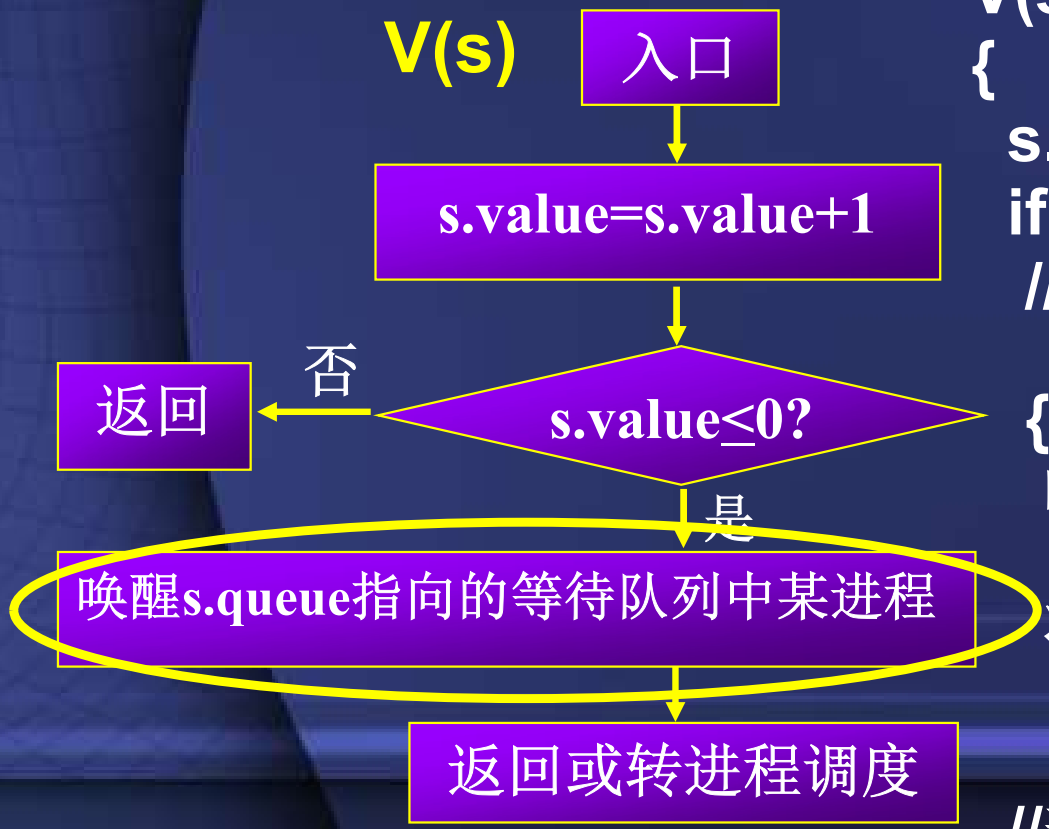



```
V(s)
{
  s.value = s.value +1; //s.value加1
  if (s.value < = 0)
   //从队列中唤醒一等待进程，然后
     继续执行或转进程调度
  {
   唤醒相应等待队列s.queue中等待
   的一个进程;
   将其状态修改为就绪态，并将其插
   入就绪队列;
   }
 //若相加结果大于零，进程继续执行
}
```

信号量与P/V操作

2、P、V操作

（2）V(Verhogen)操作

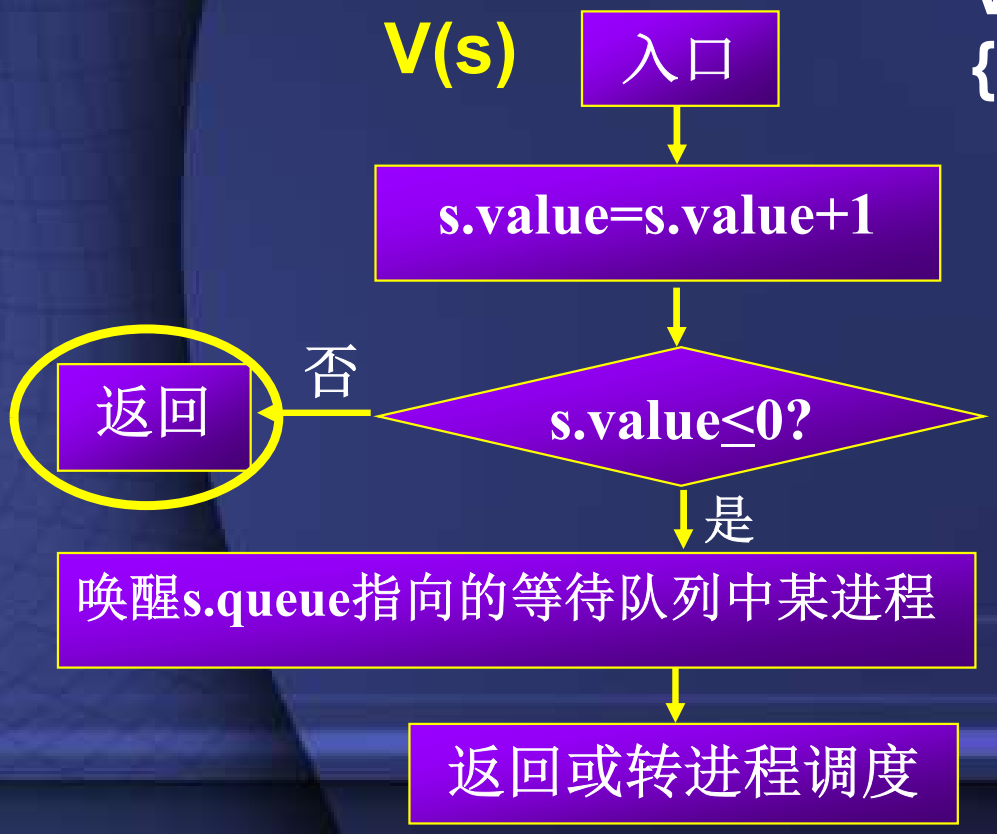



```
V(s)
{
  s.value = s.value +1; //s.value加1
  if (s.value < = 0)
   //从队列中唤醒一等待进程，然后
     继续执行或转进程调度
  {
   唤醒相应等待队列s.queue中等待
   的一个进程;
  将其状态修改为就绪态，并将其插
   入就绪队列;
   }
 //若相加结果大于零，进程继续执行
}
```

信号量与P/V操作

2、P、V操作

（2）V(Verhogen)操作




```
V(s)
{
  s.value = s.value +1; //s.value加1
  if (s.value < = 0)
   //从队列中唤醒一等待进程，然后
     继续执行或转进程调度
  {
   唤醒相应等待队列s.queue中等待
   的一个进程;
   将其状态修改为就绪态，并将其插
   入就绪队列;
   }
 //若相加结果大于零，进程继续执行
}
```

信号量与P/V操作

进程间互斥问题（用信号量实现）

```
s : semaphore;
s.value := 1;

process Pi  // i = 1,2,3,…,n
begin
  …
  P(s);
  临界区;
  V(s);
  …
end
```

s代表临界资源状态，由TS指令控制

```
s : boolean;
s := true;

process Pi  // i = 1,2,…,n
    pi : boolean;
begin
    repeat pi := TS(s) until  pi;
    临界区;
    s := true;
end
```

信号量机制实现进程互斥

1. 分析并发进程的关键活动，划定临界区（如：对临界资源打印机的访问就应放在临界区）
2. 设置互斥信号量 mutex，初值为 1

```
/*信号量机制实现互斥*/
semaphore mutex=1;  //初始化信号量

P1(){
  ...
  P(mutex);       //使用临界资源前需要加锁
  临界区代码段...
  V(mutex);       //使用临界资源后需要解锁
  ...
}

P2(){
  ...
  P(mutex);
  临界区代码段...
  V(mutex);
  ...
}
```

理解：信号量 mutex 表示“进入临界区的名额”

信号量机制实现进程互斥

1. 分析并发进程的关键活动，划定临界区（如：对临界资源打印机的访问就应放在临界区）
2. 设置互斥信号量 mutex，初值为 1
3. 在进入区 P(mutex)——申请资源
4. 在退出区 V(mutex)——释放资源

```
/*信号量机制实现互斥*/
semaphore mutex=1;  //初始化信号量

P1(){
  ...
  P(mutex);       //使用临界资源前需要加锁
  临界区代码段...
  V(mutex);       //使用临界资源后需要解锁
  ...
}

P2(){
  ...
  P(mutex);
  临界区代码段...
  V(mutex);
  ...
}
```

理解：信号量 mutex 表示“进入临界区的名额”

信号量机制实现进程互斥

1. 分析并发进程的关键活动，划定临界区（如：对临界资源打印机的访问就应放在临界区）
2. 设置互斥信号量 mutex，初值为 1
3. 在进入区 P(mutex)——申请资源
4. 在退出区 V(mutex)——释放资源

P、V操作必须成对出现。缺少 P(mutex) 就不能保证临界资源的互斥访问。缺少 V(mutex) 会导致资源永不被释放，等待进程永不被唤醒。

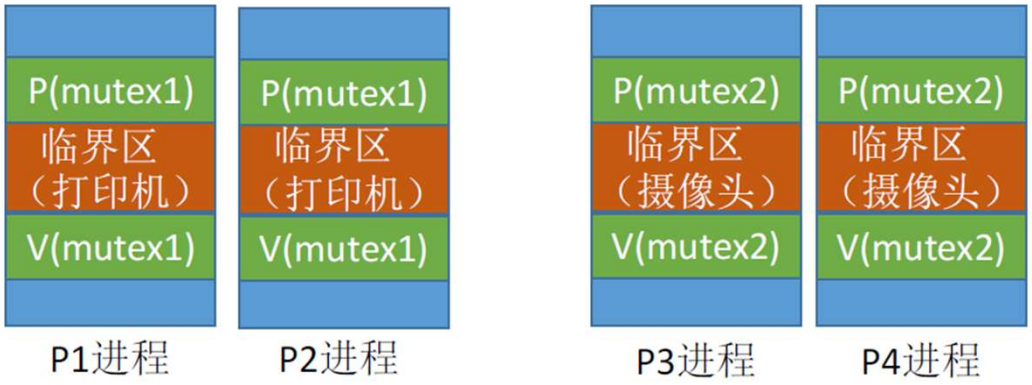
```
/*信号量机制实现互斥*/
semaphore mutex=1;  //初始化信号量

P1(){
  ...
  P(mutex);       //使用临界资源前需要加锁
  临界区代码段...
  V(mutex);       //使用临界资源后需要解锁
  ...
}

P2(){
  ...
  P(mutex);
  临界区代码段...
  V(mutex);
  ...
}
```

理解：信号量 mutex 表示“进入临界区的名额”

信号量机制实现进程互斥

1. 分析并发进程的关键活动，划定临界区（如：对临界资源打印机的访问就应放在临界区）
2. 设置互斥信号量 mutex，初值为 1
3. 在进入区 P(mutex)——申请资源
4. 在退出区 V(mutex)——释放资源

注意：对不同的临界资源需要设置不同的互斥信号量。




```
/*信号量机制实现互斥*/
semaphore mutex=1;  //初始化信号量

P1(){
   ...
   P(mutex);       //使用临界资源前需要加锁
   临界区代码段...
   V(mutex);       //使用临界资源后需要解锁
   ...
}

P2(){
   ...
   P(mutex);
   临界区代码段...
   V(mutex);
   ...
}
```

在生产者-消费者模型中，使用信号量来确保互斥访问的两种方法

方法一：在生产者和消费者中都使用 P(mutex) 和 V(mutex)

实现方式：

- 在生产者的临界区开始时调用 P(mutex)，在结束时调用 V(mutex)。
- 在消费者的临界区开始时同样调用 P(mutex)，在结束时调用 V(mutex)。

```
process producer {
    while (1) {
        P(mutex);       // 请求进入临界区
        // 生产资源
        V(mutex);       // 释放临界区
    }
}

process consumer {
    while (1) {
        P(mutex);       // 请求进入临界区
        // 消费资源
        V(mutex);       // 释放临界区
    }
}
```

优点：

- 清晰明确：每个进程都完全控制其临界区的进入和退出，意图清晰。
- 灵活性：可以在各自的进程中进行不同的临界区管理，便于扩展和维护。

缺点：

- 可能的性能开销：由于每个进程都要请求锁，可能会增加上下文切换和信号量操作的频率。

在生产者-消费者模型中，使用信号量来确保互斥访问的两种方法

方法二：在生产者中使用 P(mutex)，在消费者中使用 V(mutex)

实现方式：

在生产者中使用 P(mutex) 来控制对共享资源的访问。

在消费者中没有对互斥信号量的 P 操作，而通过 V 来释放锁。

```
process producer {
    while (1) {
        P(mutex);       // 请求进入临界区
        // 生产资源
        V(mutex);       // 释放临界区
    }
}

process consumer {
    while (1) {
        // 可能没有 P(mutex)
        // 消费资源
        V(mutex);       // 释放锁
    }
}
```

优点：

- 减少信号量的使用：当没有多个消费者同时访问时，可以减少信号量的使用频率，潜在提高性能。

缺点：

- 互斥保护不完整：如果消费者没有对应的 P(mutex)，那么在生产者放入资源时，消费者可能会在没有适当控制的情况下同时访问临界区。
- 难以管理的并发性：允许生产者和多个消费者并发工作，而不是简单地在同一进程中抢占访问。

信号量机制实现进程同步

进程同步：要让各并发进程按要求有序地推进。

```
P1(){
  代码1;
  代码2;
  代码3;
}

P2(){
  代码4;
  代码5;
  代码6;
}
```

比如，P1、P2 并发执行，由于存在异步性，因此二者交替推进的次序是不确定的。

若 P2 的“代码4”要基于 P1 的“代码1”和“代码2”的运行结果才能执行，那么我们就必须保证“代码4”一定是在“代码2”之后才会执行。

这就是进程同步问题，让本来异步并发的进程互相配合，有序推进。

信号量机制实现进程同步

用信号量实现进程同步：

1. 分析什么地方需要实现“同步关系”，即必须保证“一前一后”执行的两个操作（或两句代码）

信号量机制实现进程同步

用信号量实现进程同步：

1. 分析什么地方需要实现“同步关系”，即必须保证“一前一后”执行的两个操作（或两句代码）
2. 设置同步信号量 S, 初始为 0

```
/*信号量机制实现同步*/
semaphore S=0;   //初始化同步信号量，初始值为0
```

理解：信号量S代表“某种资源”，刚开始是没有这种资源的。P2需要使用这种资源，而又只能由P1产生这种资源

```
P1(){
① 代码1;
② 代码2;
   V(S);
   代码3;
}
```

```
P2(){
   P(S);
③ 代码4;
   代码5;
   代码6;
}
```

若先执行到 V(S) 操作，则 S++ 后 S=1。之后当执行到 P(S) 操作时，由于 S=1，表示有可用资源，会执行 S--，S 的值变回 0，P2 进程不会执行 block 原语，而是继续往下执行代码4。

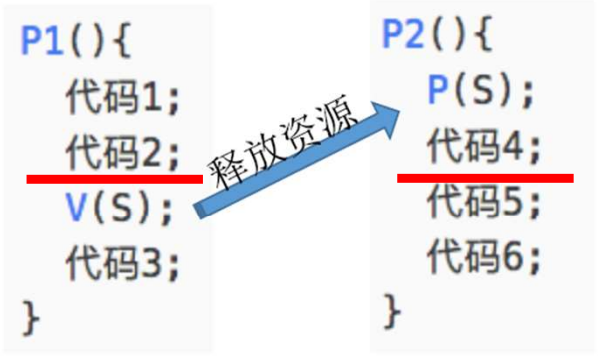
信号量机制实现进程同步

用信号量实现进程同步：

1. 分析什么地方需要实现“同步关系”，即必须保证“一前一后”执行的两个操作（或两句代码）
2. 设置同步信号量 S, 初始为 0

```
/*信号量机制实现同步*/
semaphore S=0;   //初始化同步信号量，初始值为0
```

理解：信号量S代表“某种资源”，刚开始是没有这种资源的。P2需要使用这种资源，而又只能由P1产生这种资源



若先执行到 V(S) 操作，则 S++ 后 S=1。之后当执行到 P(S) 操作时，由于 S=1，表示有可用资源，会执行 S--，S 的值变回 0，P2 进程不会执行 block 原语，而是继续往下执行代码4。

保证了 代码4 一定是在 代码2 之后执行

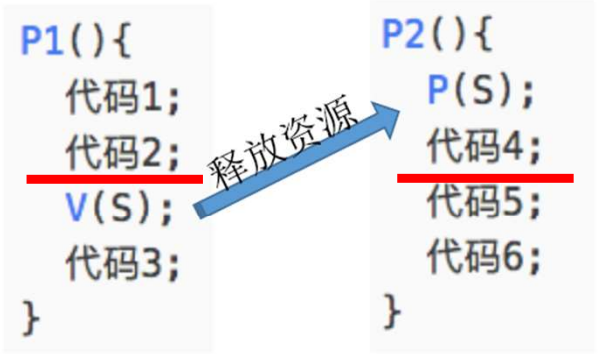
信号量机制实现进程同步

用信号量实现进程同步：

1. 分析什么地方需要实现“同步关系”，即必须保证“一前一后”执行的两个操作（或两句代码）
2. 设置同步信号量 S, 初始为 0

```
/*信号量机制实现同步*/
semaphore S=0;   //初始化同步信号量，初始值为0
```

理解：信号量S代表“某种资源”，刚开始是没有这种资源的。P2需要使用这种资源，而又只能由P1产生这种资源



保证了 代码4 一定是在 代码2 之后执行

若先执行到 V(S) 操作，则 S++ 后 S=1。之后当执行到 P(S) 操作时，由于 S=1，表示有可用资源，会执行 S--，S 的值变回 0，P2 进程不会执行 block 原语，而是继续往下执行代码4。

若先执行到 P(S) 操作，由于 S=0，S-- 后 S=-1，表示此时没有可用资源，因此P操作中会执行 block 原语，主动请求阻塞。之后当执行完代码2，继而执行 V(S) 操作，S++，使 S 变回 0，由于此时有进程在该信号量对应的阻塞队列中，因此会在 V 操作中执行 wakeup 原语，唤醒 P2 进程。这样 P2 就可以继续执行 代码4 了

在“前操作”之后执行 V(S)
在“后操作”之前执行 P(S)

信号量机制实现进程同步

用信号量实现进程同步：

1. 分析什么地方需要实现“同步关系”，即必须保证“一前一后”执行的两个操作（或两句代码）
2. 设置同步信号量 S, 初始为 0
3. 在“前操作”之后执行 V(S)
4. 在“后操作”之前执行 P(S)

技巧口诀：前V后P

理解：信号量S代表“某种资源”，刚开始是没有这种资源的。P2需要使用这种资源，而又只能由P1产生这种资源

```
/*信号量机制实现同步*/
semaphore S=0;   //初始化同步信号量，初始值为0
```

```
P1(){
   代码1;
   代码2;
   V(S);
   代码3;
}
```

释放资源

```
P2(){
   P(S);
   代码4;
   代码5;
   代码6;
}
```

保证了 代码4 一定是在 代码2 之后执行

若先执行到 V(S) 操作，则 S++ 后 S=1。之后当执行到 P(S) 操作时，由于 S=1，表示有可用资源，会执行 S--，S 的值变回 0，P2 进程不会执行 block 原语，而是继续往下执行代码4。

若先执行到 P(S) 操作，由于 S=0，S-- 后 S=-1，表示此时没有可用资源，因此P操作中会执行 block 原语，主动请求阻塞。之后当执行完代码2，继而执行 V(S) 操作，S++，使 S 变回 0，由于此时有进程在该信号量对应的阻塞队列中，因此会在 V 操作中执行 wakeup 原语，唤醒 P2 进程。这样 P2 就可以继续执行 代码4 了

1、操作系统中，程序P1和P2并发运行,伪代码分别如下。

信号量 S 初始值为 0;
int y = 0;

```
void P1( )
{
        P(S); //①
        y = y+3;
        print(y);
}
```

```
void P2( )
 {
    y = y+5:
    V(S) //②

 }
```

（1）请问打印的结果是什么？ 8

（2）如果去掉语句①和语句②。请问可能的打印结果是什么？ 3 8

2、内存中有一组缓冲区被多个生产者进程、多个消费者进程共享使用，总共能存放10个数据，生产者进程把生成的数据放入缓冲区，消费者进程从缓冲区中取出数据使用。缓冲区满时生产者进程就停止将数据放入缓冲区，缓冲区空时消费者进程停止取数据。数据的存入和取出不能同时进行，试用信号量及P、V操作来实现该方案。

```
semaphore mutex, empty, full;
mutex=1;                    //互斥信号量
empty=10;                   //生产者进程的同步信号量
full=0;                     //消费者进程的同步信号量

cobegin
  process Pi  //生产者进程
  {
    while (1) {
      生产数据;
      P(empty); //看看是否还有空间可放
      P(mutex);   //互斥使用放入;
      V(full);    //增1(可能唤醒一个消费者)
      V(mutex);
    }
  }

  process Cj  //消费者进程
  {
    while (1) {
      P(full);     //看看是否有数据
      P(mutex);        //互斥使用取出;
      V(emtpy);    //增1(可能唤醒一个生产者)
      V(mutex);


    }
  }
coend
```

信号量与P/V操作

本讲内容

1、信号量(Semaphore)

2、P(Proberen)、V(Verhogen)操作

3、典型问题的解决方法

4、问题思考与拓展学习

信号量与P/V操作

3、典型问题的解决方法

苹果桔子问题

生产者/消费者问题

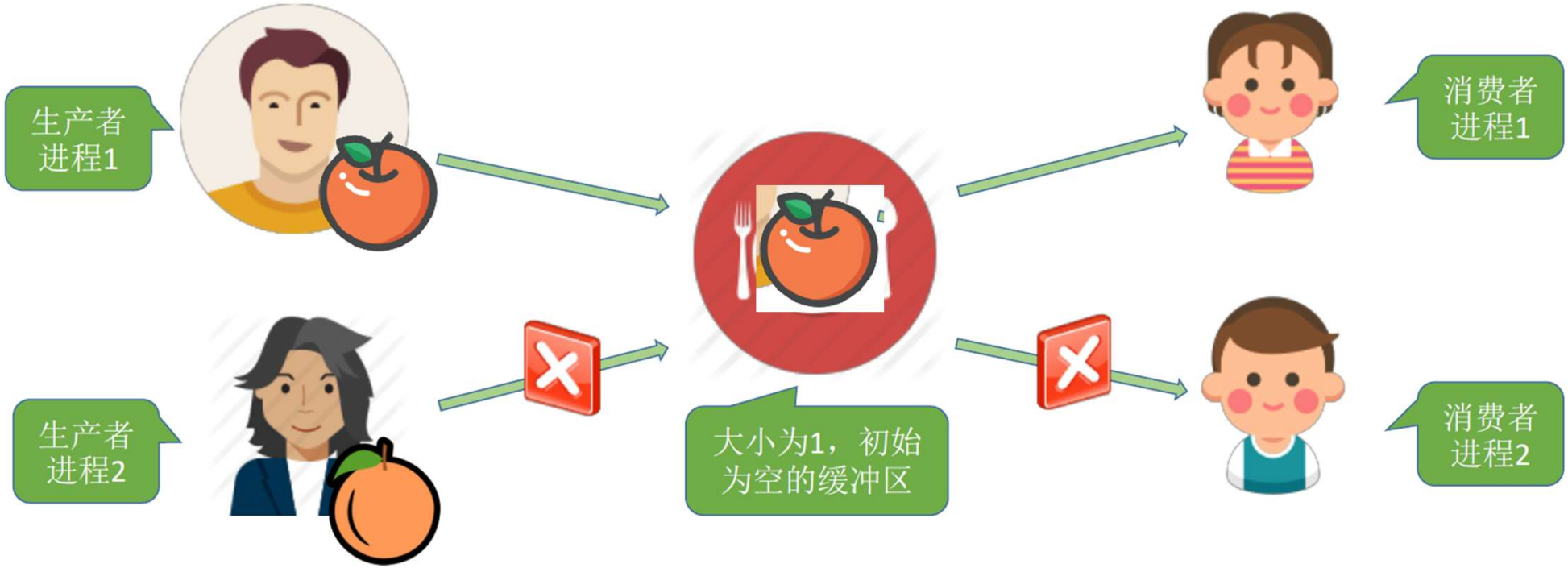
（1）桌上有一只盘子，每次只能放入一只水果；

（2）爸爸专向盘子中放苹果，妈妈专向盘子中放桔子；

（3）儿子专等吃盘子中的桔子，女儿专等吃盘子里的苹果。

问题描述

桌子上有一只盘子，每次只能向其中放入一个水果。爸爸专向盘子中放苹果，妈妈专向盘子中放橘子，儿子专等着吃盘子中的橘子，女儿专等着吃盘子中的苹果。只有盘子空时，爸爸或妈妈才可向盘子中放一个水果。仅当盘子中有自己需要的水果时，儿子或女儿可以从盘子中取出水果。用PV操作实现上述过程。



当盘子中是苹果时，母亲就不能放，儿子也不能吃

多生产者-多消费者问题，并且类型不同

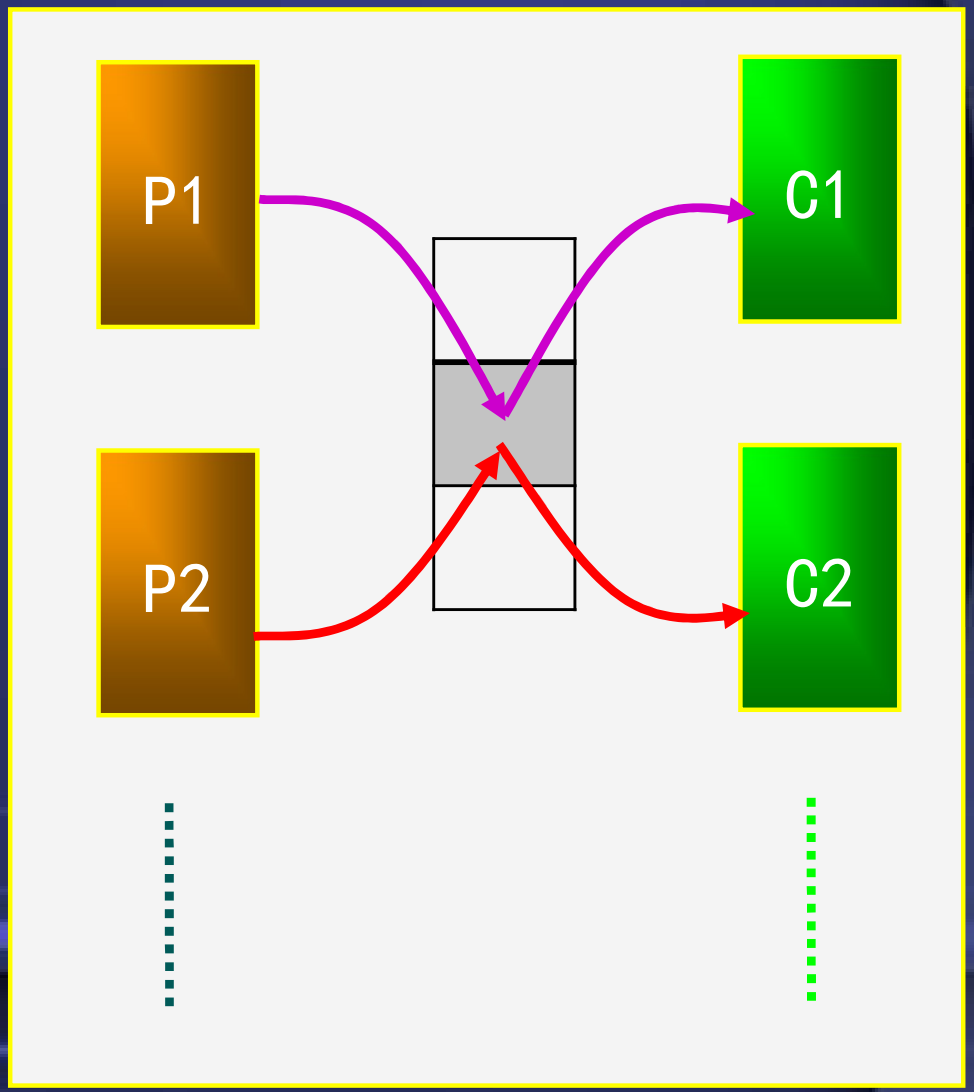
信号量与P/V操作

3、典型问题的解决方法

苹果桔子问题

对应的计算机系统问题：

多个进程共享同一个缓冲区实现定向通信



问题分析

桌子上有一只盘子，每次只能向其中放入一个水果。爸爸专向盘子中放苹果，妈妈专向盘子中放橘子，儿子专等着吃盘子中的橘子，女儿专等着吃盘子中的苹果。只有盘子空时，爸爸或妈妈才可向盘子中放一个水果。仅当盘子中有自己需要的水果时，儿子或女儿可以从盘子中取出水果。

1. 关系分析。找出题目中描述的各个进程，分析它们之间的同步、

互斥关系：
对缓冲区（盘子）的访问要互斥地进行

同步关系（一前一后）：
1. 父亲将苹果放入盘子后，女儿才能取苹果
2. 母亲将橘子放入盘子后，儿子才能取橘子

问题分析

桌子上有一只盘子，每次只能向其中放入一个水果。爸爸专向盘子中放苹果，妈妈专向盘子中放橘子，儿子专等着吃盘子中的橘子，女儿专等着吃盘子中的苹果。只有盘子空时，爸爸或妈妈才可向盘子中放一个水果。仅当盘子中有自己需要的水果时，儿子或女儿可以从盘子中取出水果。

1. 关系分析。找出题目中描述的各个进程，分析它们之间的同步、

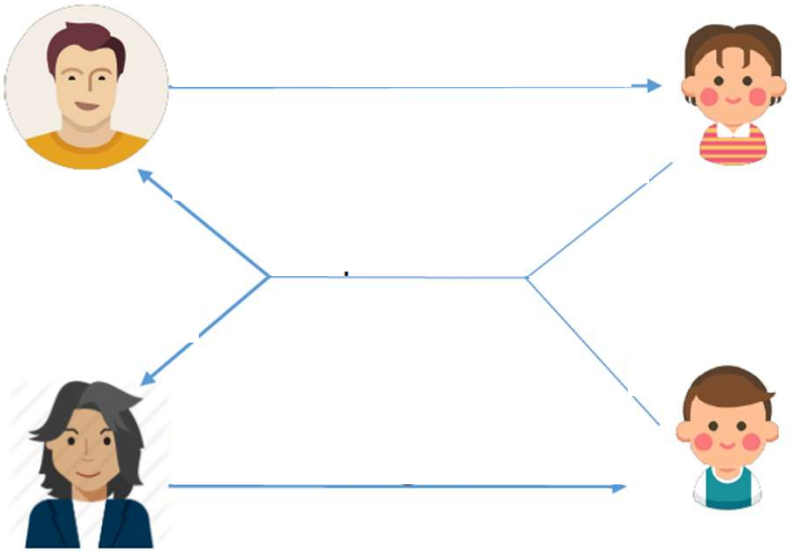
互斥关系：
对缓冲区（盘子）的访问要互斥地进行

同步关系（一前一后）：
1. 父亲将苹果放入盘子后，女儿才能取苹果
2. 母亲将橘子放入盘子后，儿子才能取橘子

问题分析

桌子上有一只盘子，每次只能向其中放入一个水果。爸爸专向盘子中放苹果，妈妈专向盘子中放橘子，儿子专等着吃盘子中的橘子，女儿专等着吃盘子中的苹果。只有盘子空时，爸爸或妈妈才可向盘子中放一个水果。仅当盘子中有自己需要的水果时，儿子或女儿可以从盘子中取出水果。

1. 关系分析。找出题目中描述的各个进程，分析它们之间的同步、



互斥关系：
对缓冲区（盘子）的访问要互斥地进行

同步关系（一前一后）：
1. 父亲将苹果放入盘子后，女儿才能取苹果
2. 母亲将橘子放入盘子后，儿子才能取橘子
3. 只有盘子为空时，父亲或母亲才能放入水果

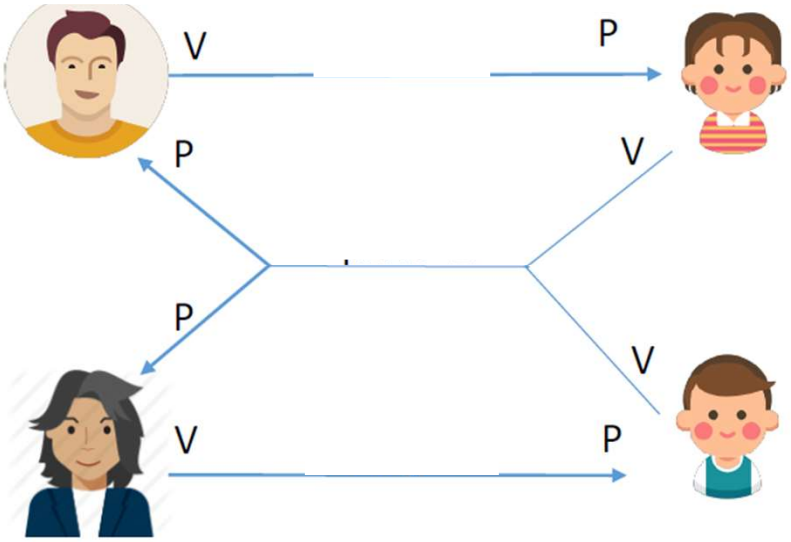
“盘子为空”这个事件可以由儿子或女儿触发，事件发生后才允许父亲或母亲放水果

问题分析

桌子上有一只盘子，每次只能向其中放入一个水果。爸爸专向盘子中放苹果，妈妈专向盘子中放橘子，儿子专等着吃盘子中的橘子，女儿专等着吃盘子中的苹果。只有盘子空时，爸爸或妈妈才可向盘子中放一个水果。仅当盘子中有自己需要的水果时，儿子或女儿可以从盘子中取出水果。

1. 关系分析。找出题目中描述的各个进程，分析它们之间的同步、
2. 整理思路。根据各进程的操作流程确定P、V操作的大致顺序。

互斥：在临界区前后分别PV
同步：前V后P



互斥关系：
对缓冲区（盘子）的访问要互斥地进行

同步关系（一前一后）：
1. 父亲将苹果放入盘子后，女儿才能取苹果
2. 母亲将橘子放入盘子后，儿子才能取橘子
3. 只有盘子为空时，父亲或母亲才能放入水果

“盘子为空”这个事件可以由儿子或女儿触发，事件发生后才允许父亲或母亲放水果

信号量与P/V操作

3、典型问题的解决方法

苹果桔子问题—解决思路—>（1）信号量的设置

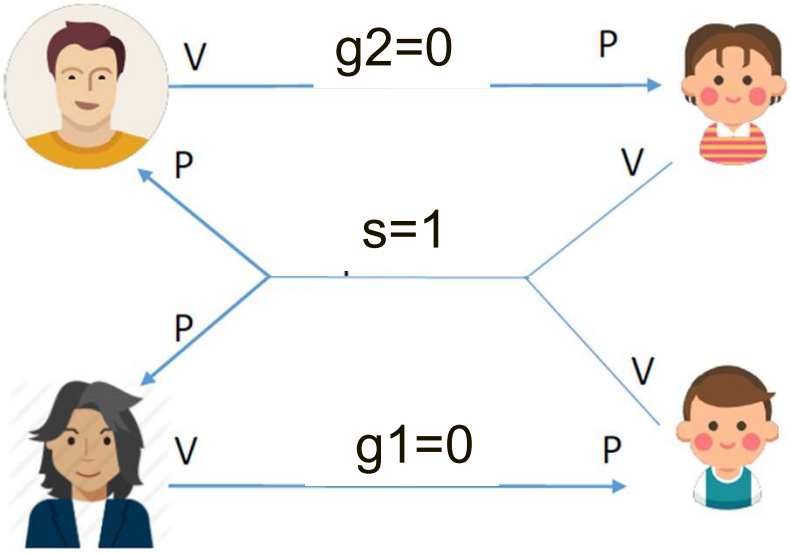
- s代表可用的空盘子数，　初值为1
- g1代表盘子里有无桔子，初值为0
- g2代表盘子里有无苹果，初值为0

问题分析

桌子上有一只盘子，每次只能向其中放入一个水果。爸爸专向盘子中放苹果，妈妈专向盘子中放橘子，儿子专等着吃盘子中的橘子，女儿专等着吃盘子中的苹果。只有盘子空时，爸爸或妈妈才可向盘子中放一个水果。仅当盘子中有自己需要的水果时，儿子或女儿可以从盘子中取出水果。

1. 关系分析。找出题目中描述的各个进程，分析它们之间的同步、
2. 整理思路。根据各进程的操作流程确定P、V操作的大致顺序。
3. 设置信号量。设置需要的信号量，并根据题目条件确定信号量初值。（互斥信号量初值一般为1，同步信号量的初始值要看对应资源的初始值是多少）

互斥：在临界区前后分别PV
同步：前V后P



互斥关系：
对缓冲区（盘子）的访问要互斥地进行

同步关系（一前一后）：
1. 父亲将苹果放入盘子后，女儿才能取苹果
2. 母亲将橘子放入盘子后，儿子才能取橘子
3. 只有盘子为空时，父亲或母亲才能放入水果

“盘子为空”这个事件可以由儿子或女儿触发，事件发生后才允许父亲或母亲放水果

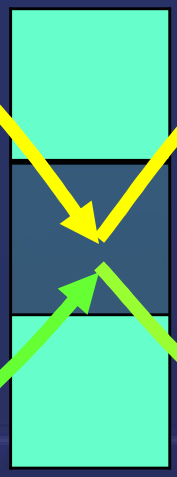
信号量与P/V操作

3、典型问题的解决方法

苹果桔子问题—解决思路—>（2）P、V操作的部署

```
process father
begin
    L1:削一个苹果;
    P(s); 放苹果;
    V(g2);goto L1;
end;
```

```
process daughter
begin
    L4: P(g2);取苹果;
    V(s); 吃苹果;
    goto L4;
end;
```



```
process mother
begin
    L2:剥一个桔子;
    P(s); 放桔子;
    V(g1); goto L2;
end;
```

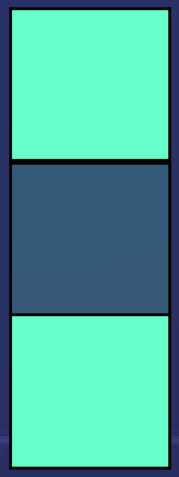
```
process son
begin
    L3: P(g1); 取桔子;
    V(s); 吃桔子;
    goto L3;
end;
```

信号量与P/V操作

3、典型问题的解决方法

苹果桔子问题—解决思路—>（2）P、V操作的部署

```
process father
begin
    L1:削一个苹果;
    P(s); 放苹果;
    V(g2);goto L1;
end;
```

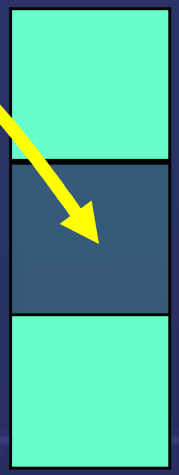


信号量与P/V操作

3、典型问题的解决方法

苹果桔子问题—解决思路—>（2）P、V操作的部署

```
process father
begin
    L1:削一个苹果;
    P(s); 放苹果;
    V(g2);goto L1;
end;
```

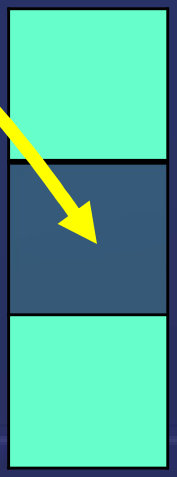


信号量与P/V操作

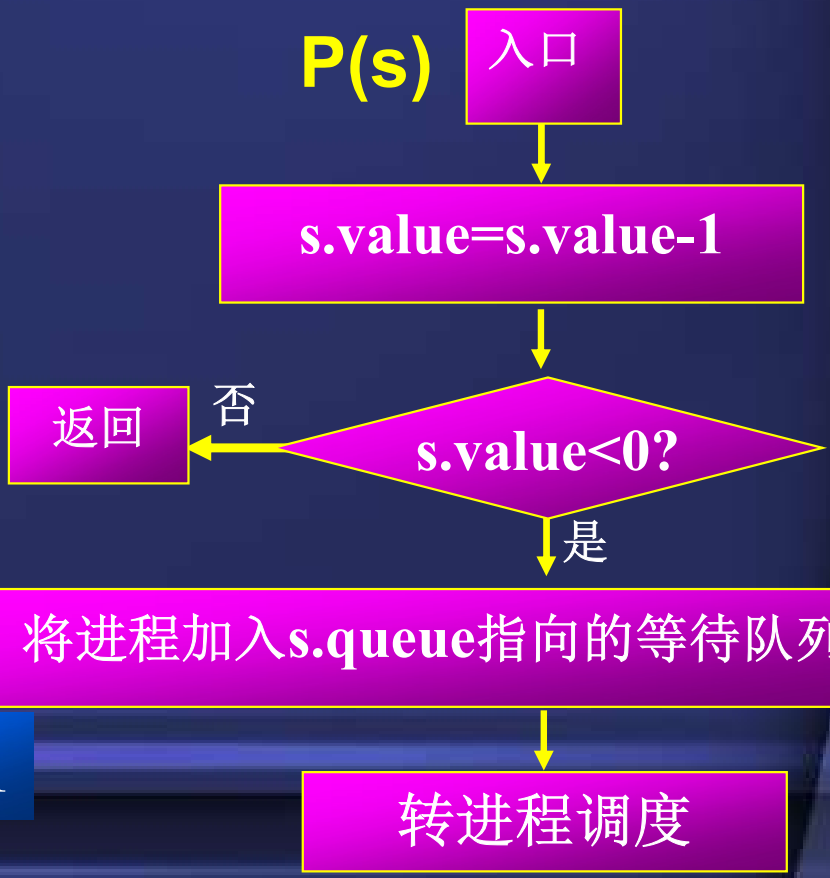
3、典型问题的解决方法

苹果桔子问题—解决思路—>（2）P、V操作的部署

```
process father
begin
    L1:削一个苹果;
    P(s); 放苹果;
    V(g2);goto L1;
end;
```



s.value=1

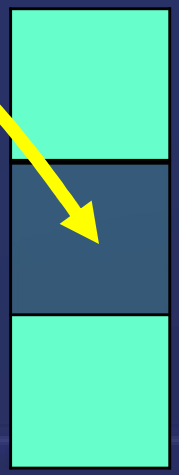


信号量与P/V操作

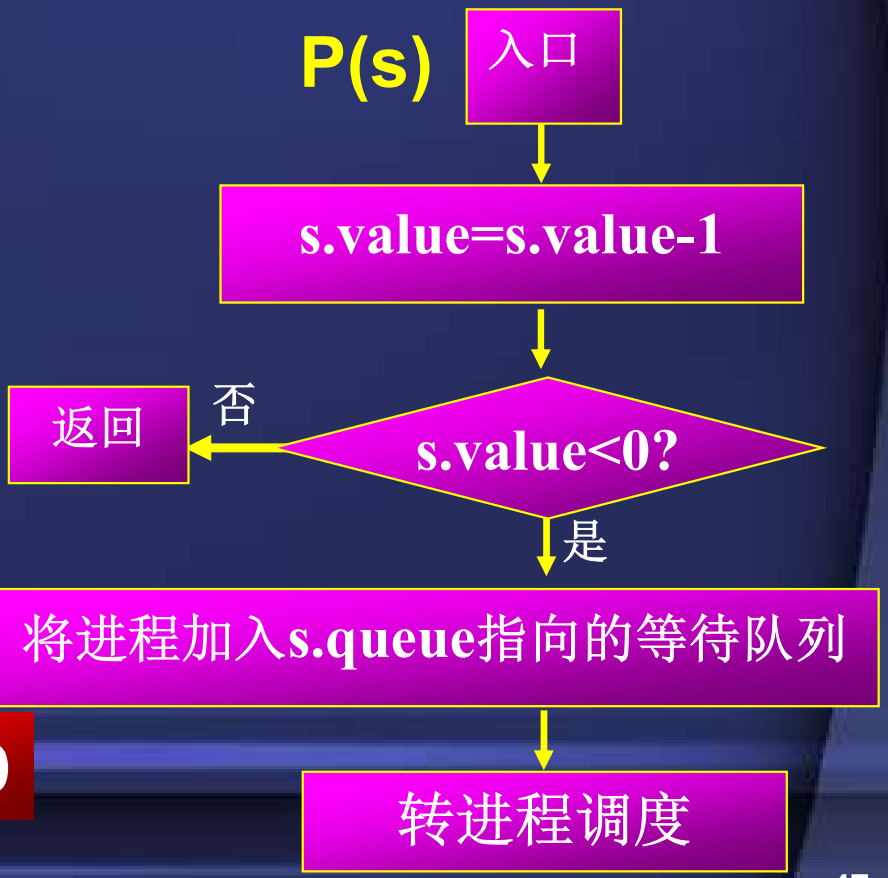
3、典型问题的解决方法

苹果桔子问题—解决思路—>（2）P、V操作的部署

```
process father
begin
    L1:削一个苹果;
    P(s); 放苹果;
    V(g2);goto L1;
end;
```



s.value=0

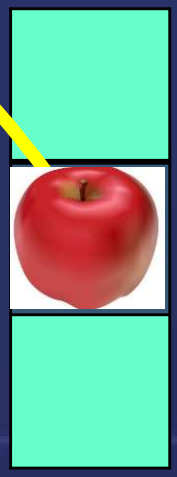


信号量与P/V操作

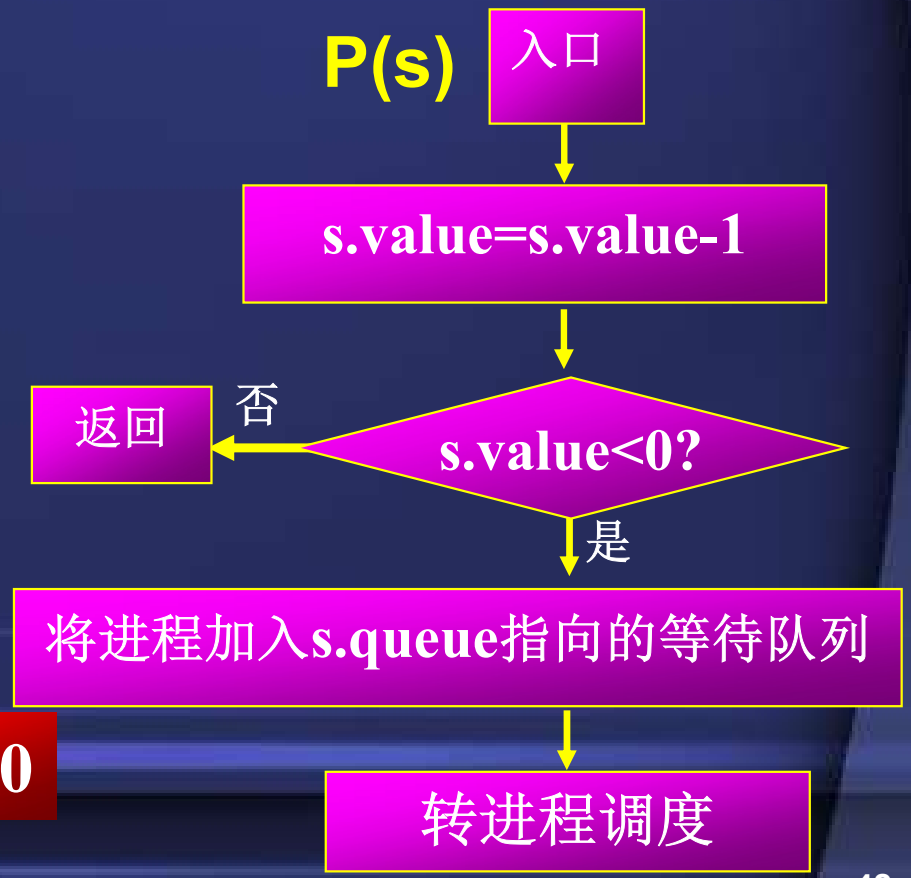
3、典型问题的解决方法

苹果桔子问题—解决思路—>（2）P、V操作的部署

```
process father
begin
    L1:削一个苹果;
    P(s); 放苹果;
    V(g2);goto L1;
end;
```



s.value=0

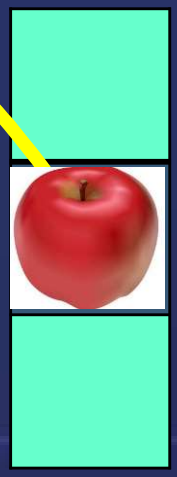


信号量与P/V操作

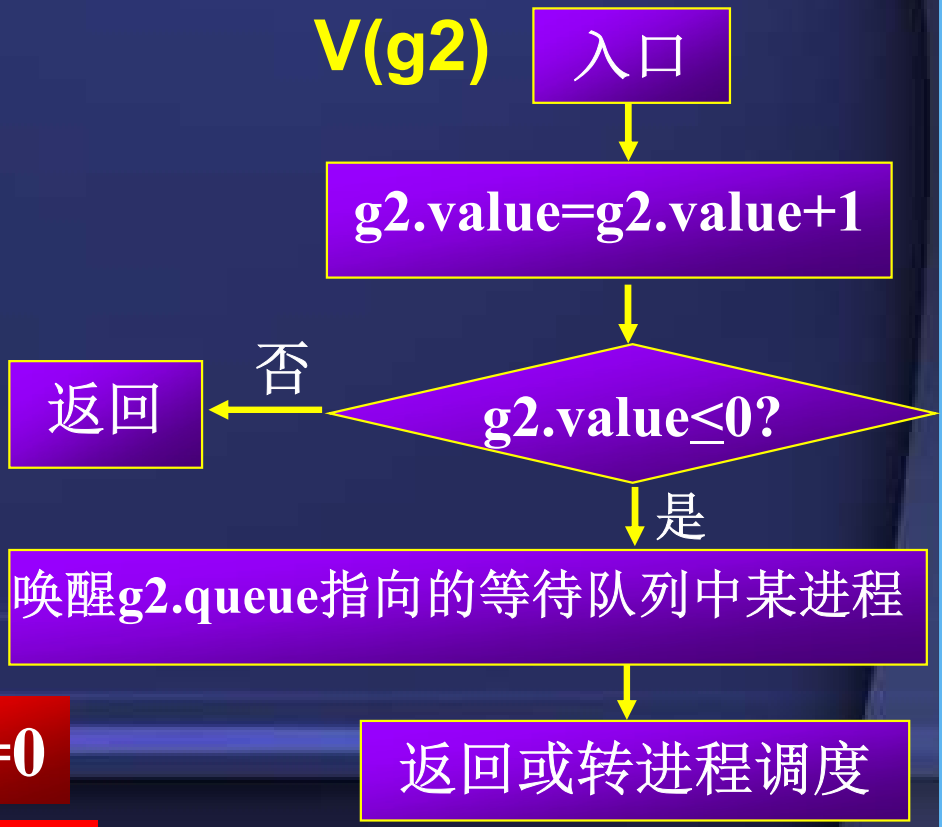
3、典型问题的解决方法

苹果桔子问题—解决思路—>（2）P、V操作的部署

```
process father
begin
    L1:削一个苹果;
    P(s); 放苹果;
    V(g2);goto L1;
end;
```



V(g2)



s.value=0

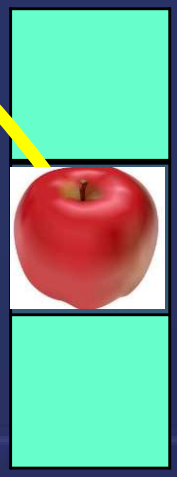
g2.value=0

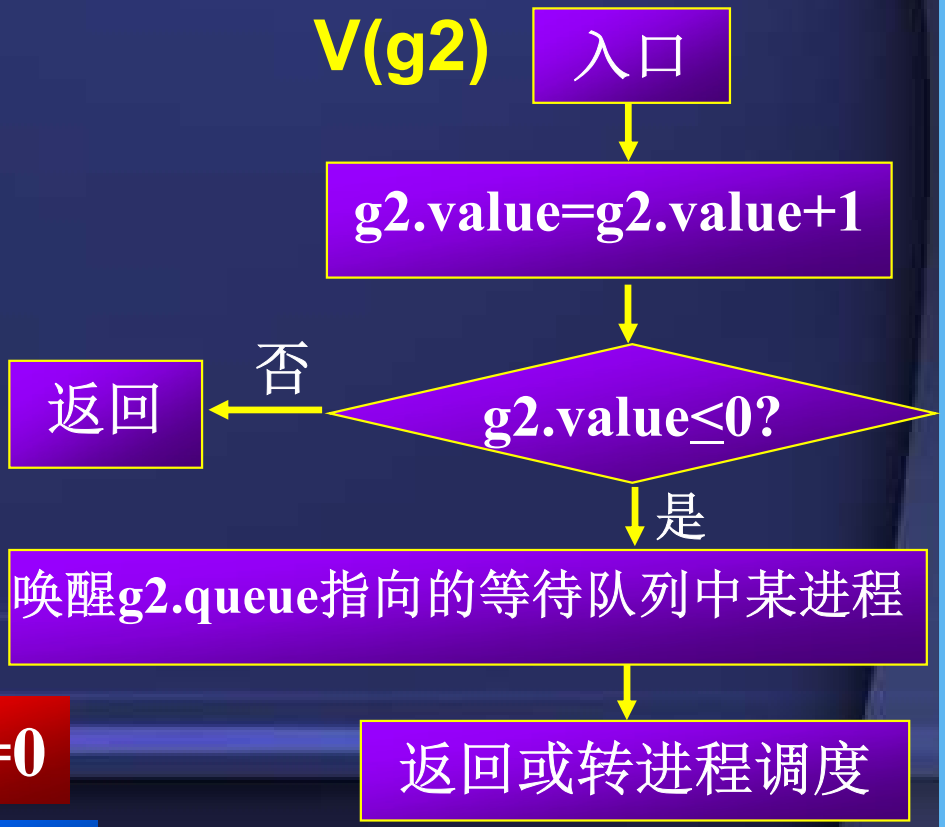
信号量与P/V操作

3、典型问题的解决方法

苹果桔子问题—解决思路—>（2）P、V操作的部署

```
process father
begin
    L1:削一个苹果;
    P(s); 放苹果;
    V(g2);goto L1;
end;
```





s.value=0

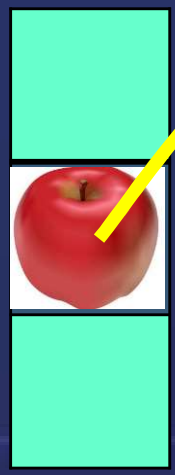
g2.value=1

信号量与P/V操作

3、典型问题的解决方法

苹果桔子问题—解决思路—>（2）P、V操作的部署

```
process daughter
begin
    L4: P(g2);取苹果;
    V(s); 吃苹果;
    goto L4;
end;
```



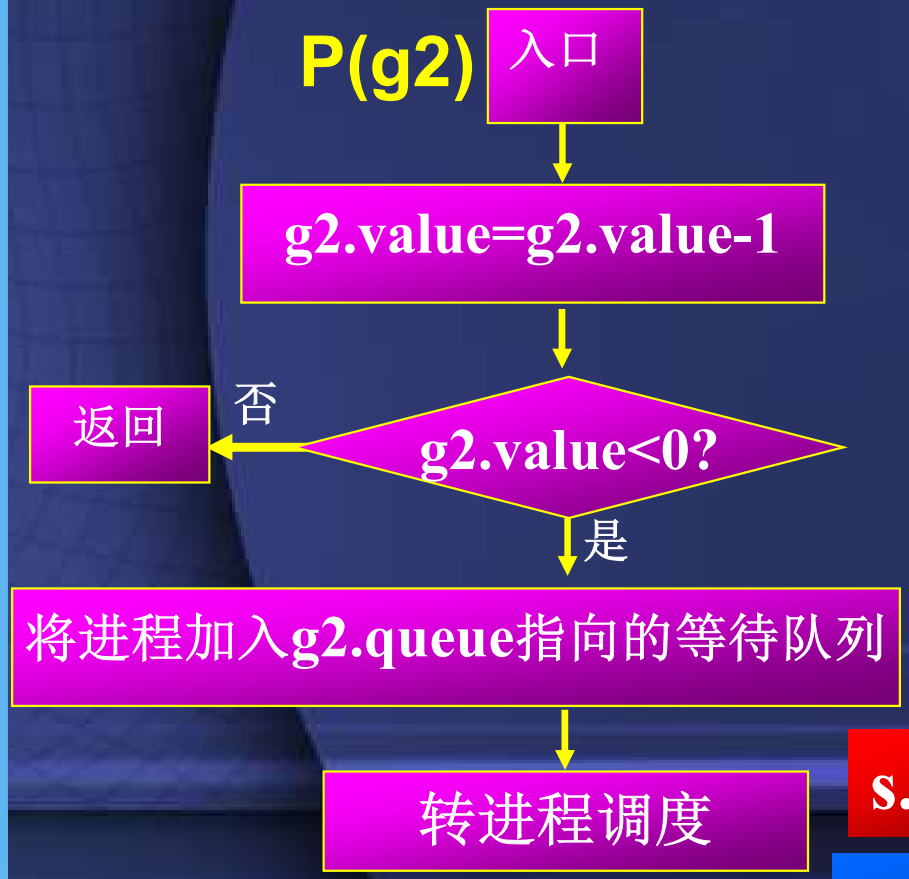
s.value=0

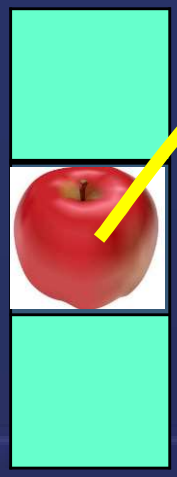
g2.value=1

信号量与P/V操作

3、典型问题的解决方法

苹果桔子问题—解决思路—>（2）P、V操作的部署





```
process daughter
begin
  L4: P(g2);取苹果;
  V(s); 吃苹果;
   goto L4;
end;
```

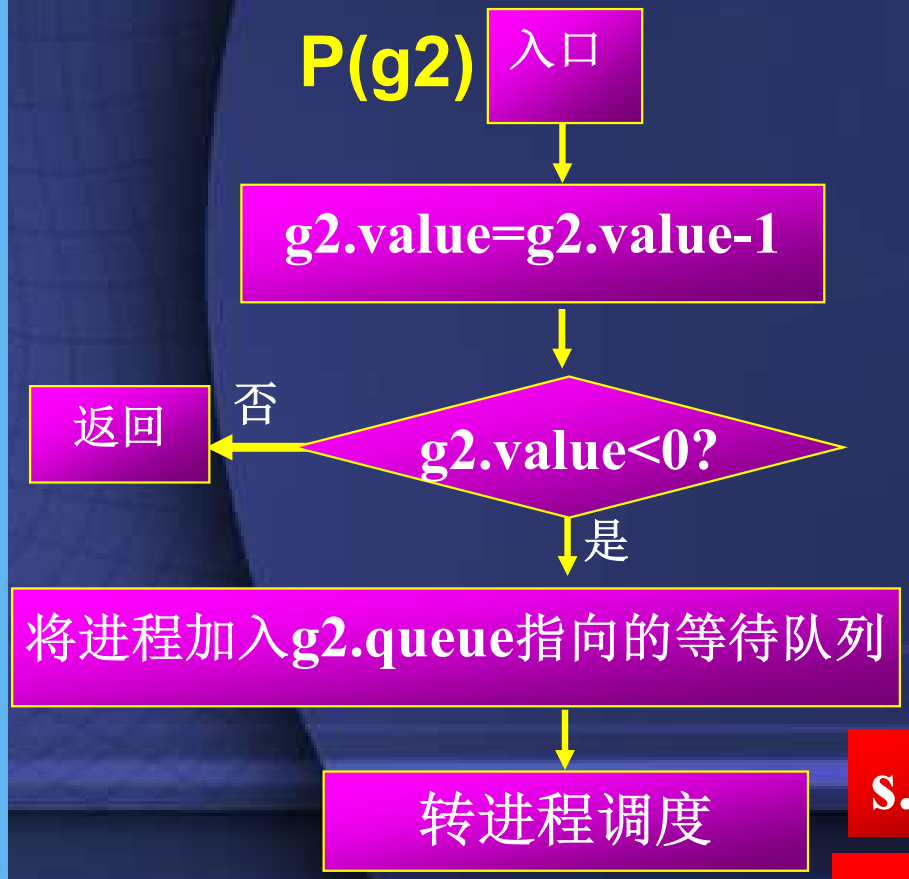
s.value=0

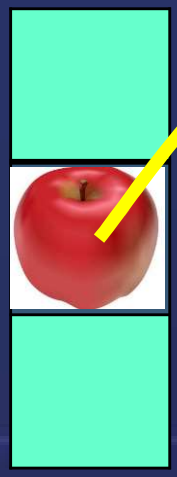
g2.value=1

信号量与P/V操作

3、典型问题的解决方法

苹果桔子问题—解决思路—>（2）P、V操作的部署





```
process daughter
begin
   L4: P(g2);取苹果;
   V(s); 吃苹果;
    goto L4;
end;
```

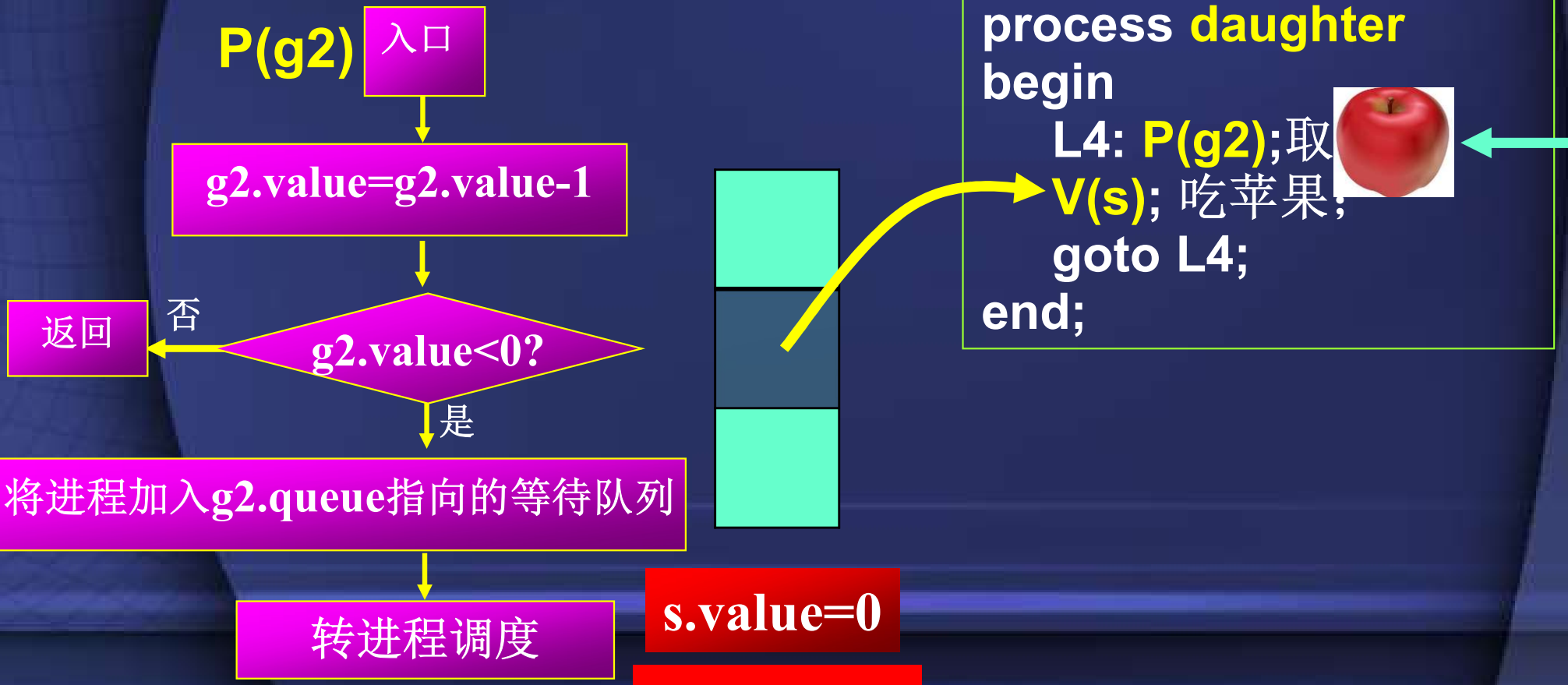
s.value=0

g2.value=0

信号量与P/V操作

3、典型问题的解决方法

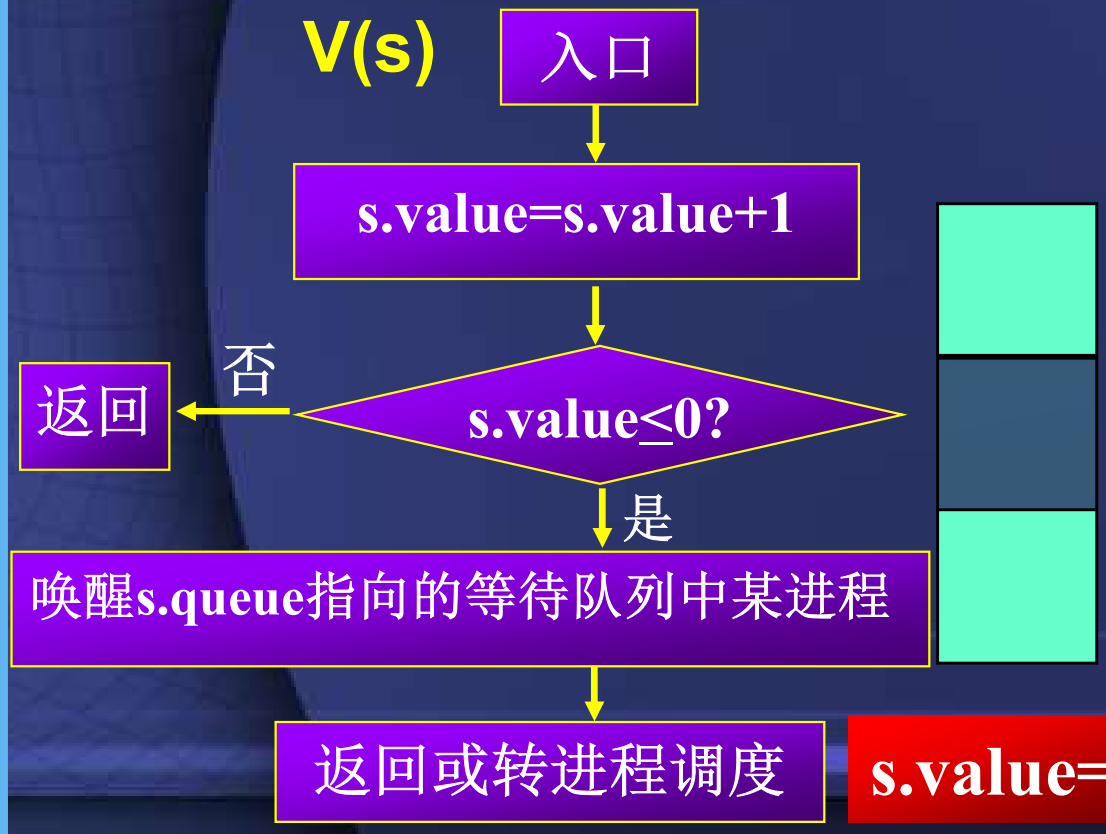
苹果桔子问题—解决思路—>（2）P、V操作的部署



信号量与P/V操作

3、典型问题的解决方法

苹果桔子问题—解决思路—>（2）P、V操作的部署




```
process daughter
begin
    L4: P(g2);取苹果;
    V(s); 吃苹果;
    goto L4;
end;
```

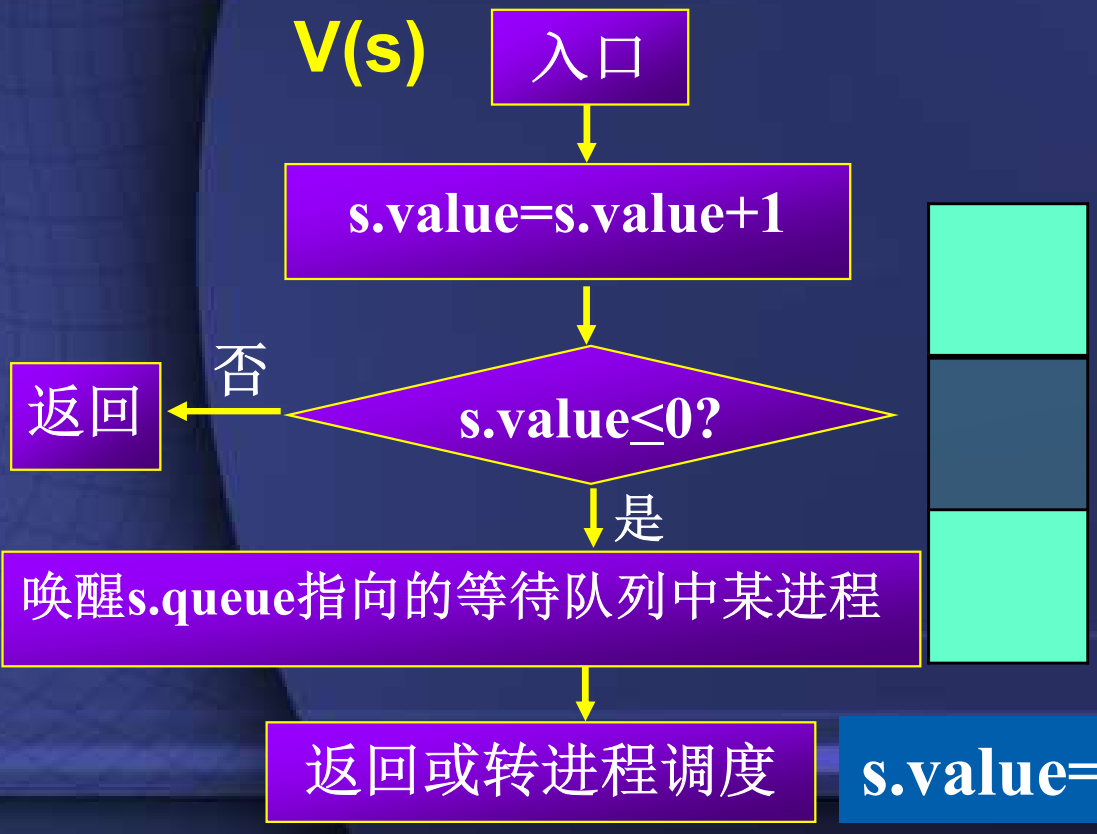
s.value=0

g2.value=0

信号量与P/V操作

3、典型问题的解决方法

苹果桔子问题—解决思路—>（2）P、V操作的部署




```
process daughter
begin
    L4: P(g2);取苹果;
    V(s); 吃苹果;
    goto L4;
end;
```

s.value=1

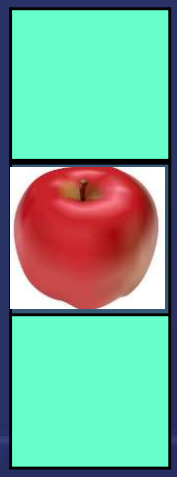
g2.value=0

信号量与P/V操作

3、典型问题的解决方法

苹果桔子问题—解决思路—>（3）异常情况

```
process father
begin
    L1:削一个苹果;
    P(s); 放苹果;
    V(g2);goto L1;
end;
```



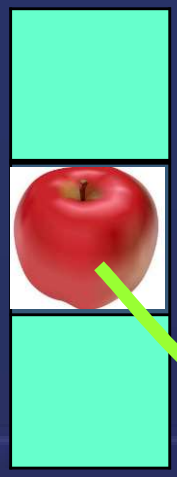
s.value=0

g2.value=1

信号量与P/V操作

3、典型问题的解决方法

苹果桔子问题—解决思路—>（3）异常情况



s.value=0

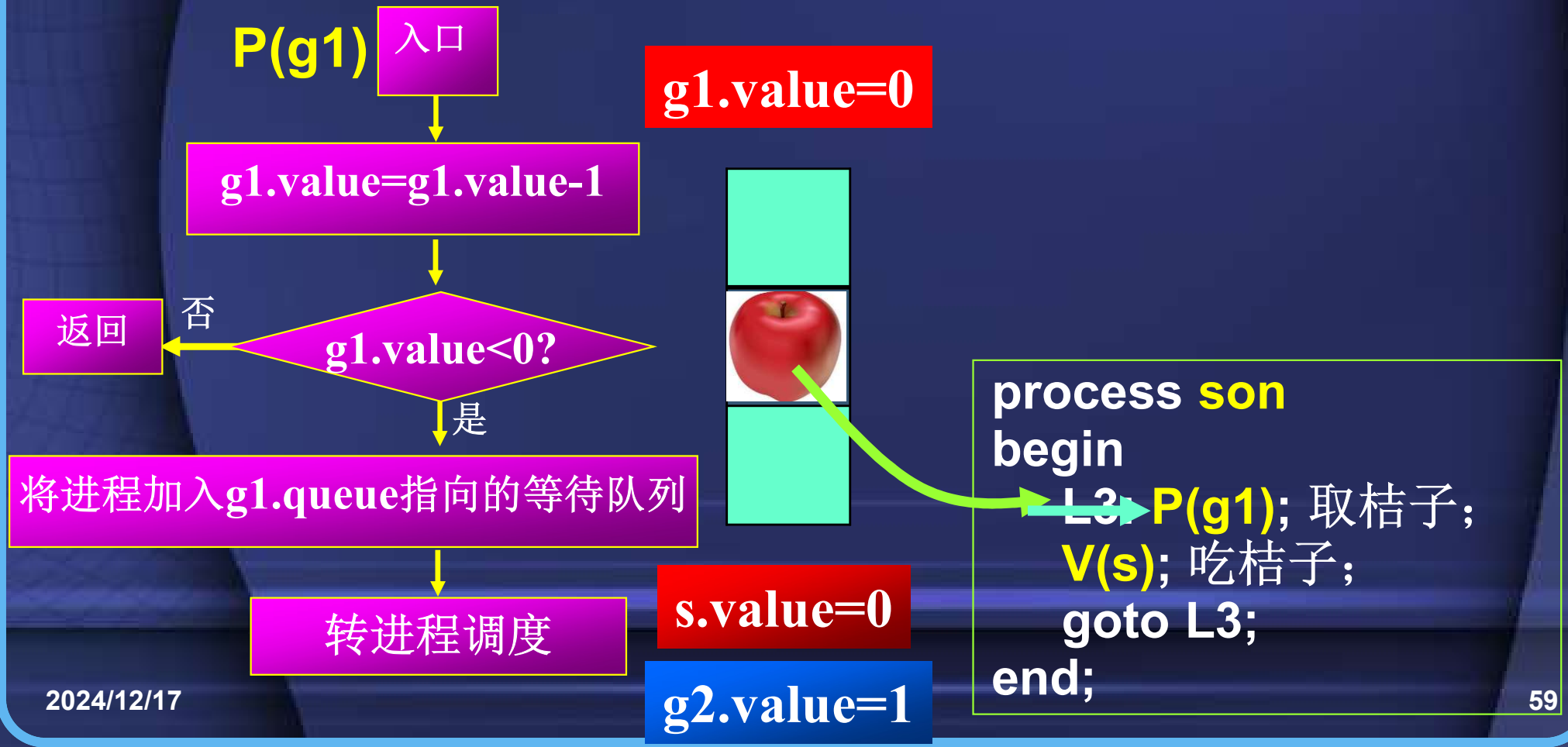
g2.value=1

```
process son
begin
    L3: P(g1); 取桔子;
    V(s); 吃桔子;
    goto L3;
end;
```

信号量与P/V操作

3、典型问题的解决方法

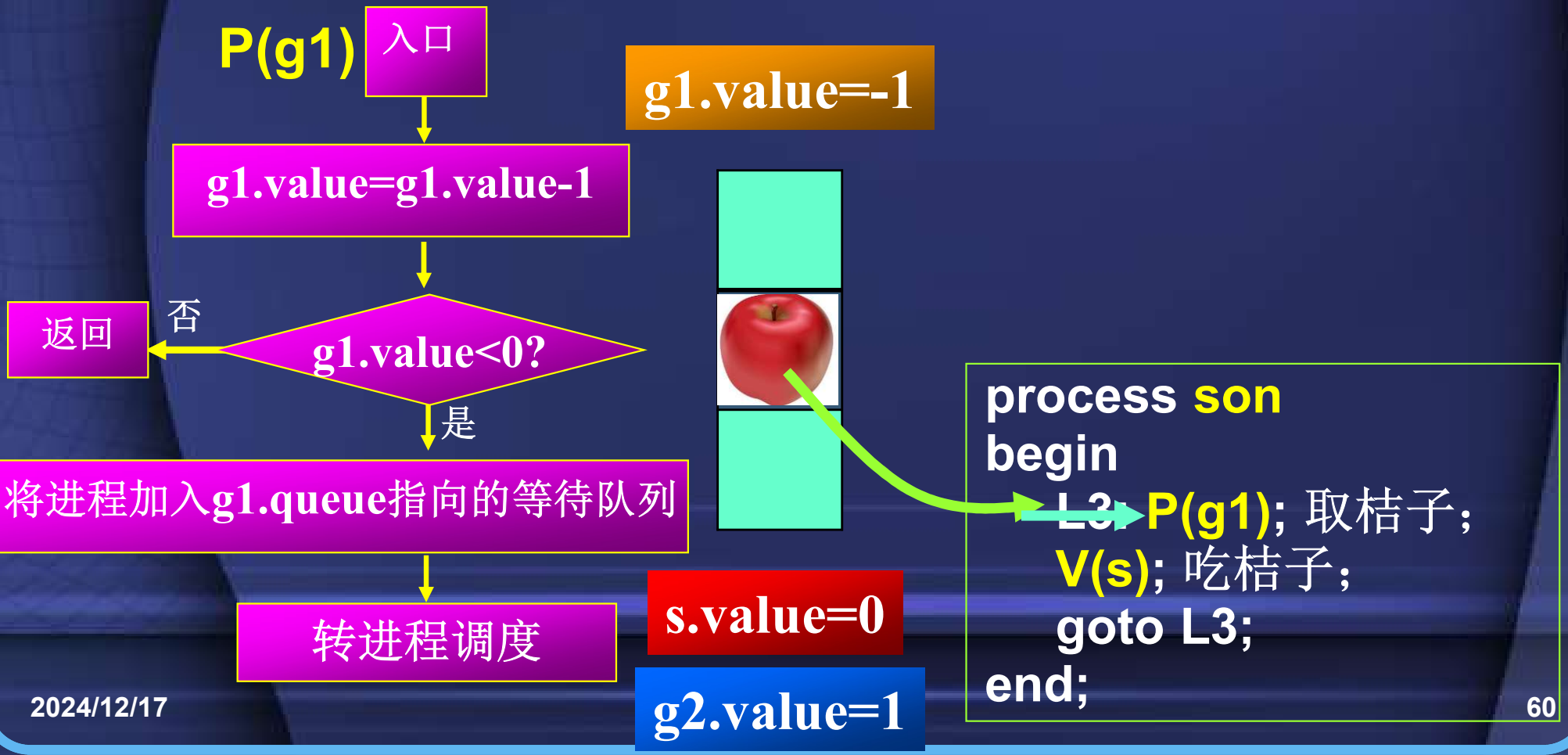
苹果桔子问题—解决思路—>（3）异常情况



信号量与P/V操作

3、典型问题的解决方法

苹果桔子问题—解决思路—>（3）异常情况

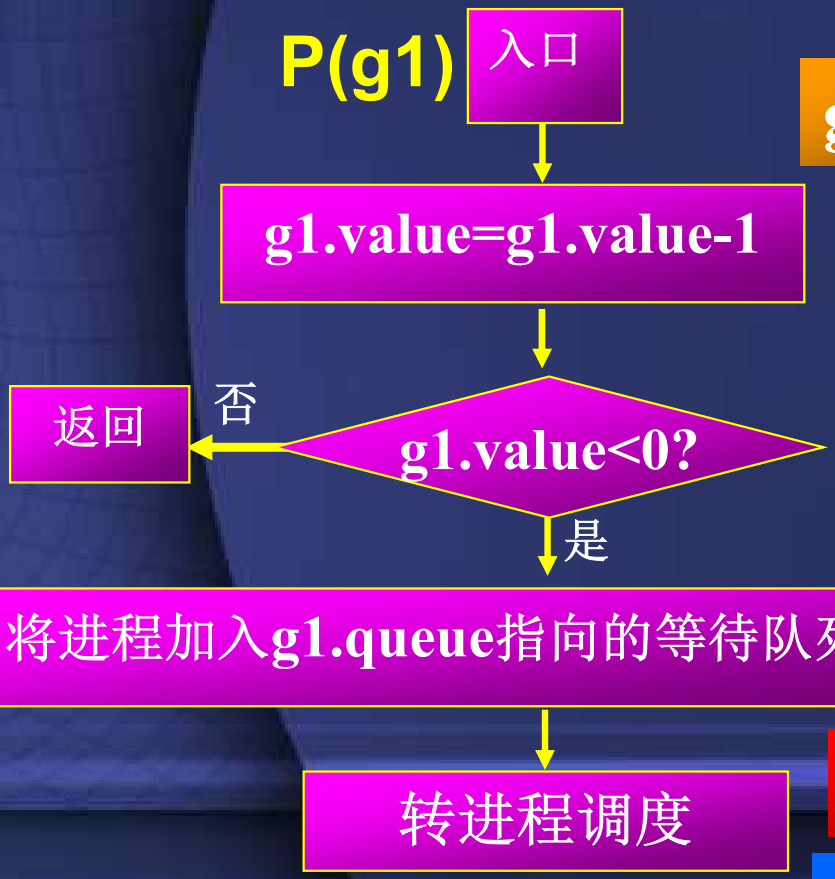


信号量与P/V操作

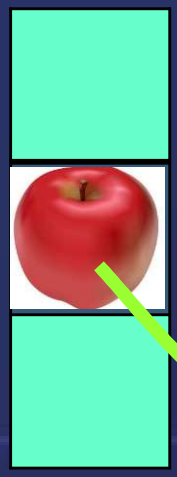
3、典型问题的解决方法

苹果桔子问题—解决思路—>（3）

当盘子里有桔子的时候再进行操作



g1.value=-1



g1.value
g1.queue
→ son

```
process son
begin
  L3: P(g1); 取桔子;
  V(s); 吃桔子;
  goto L3;
end;
```

s.value=0

g2.value=1

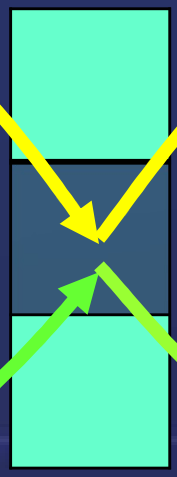
信号量与P/V操作

3、典型问题的解决方法

苹果桔子问题—解决思路—>（2）P、V操作的部署

```
process father
begin
    L1:削一个苹果;
    P(s); 放苹果;
    V(g2);goto L1;
end;
```

```
process daughter
begin
    L4: P(g2);取苹果;
    V(s); 吃苹果;
    goto L4;
end;
```



```
process mother
begin
    L2:剥一个桔子;
    P(s); 放桔子;
    V(g1); goto L2;
end;
```

```
process son
begin
    L3: P(g1); 取桔子;
    V(s); 吃桔子;
    goto L3;
end;
```

2、内存中有一组缓冲区被多个生产者进程、多个消费者进程共享使用，总共能存放10个数据，生产者进程把生成的数据放入缓冲区，消费者进程从缓冲区中取出数据使用。缓冲区满时生产者进程就停止将数据放入缓冲区，缓冲区空时消费者进程停止取数据。数据的存入和取出不能同时进行，试用信号量及P、V操作来实现该方案。

```
semaphore mutex, empty, full;
mutex=1;                    //互斥信号量
empty=10;                   //生产者进程的同步信号量
full=0;                     //消费者进程的同步信号量

cobegin
  process Pi  //生产者进程
  {
    while (1) {
      生产数据;
      P(empty); //看看是否还有空间可放
      P(mutex);   //互斥使用放入;
      V(full);    //增1(可能唤醒一个消费者)
      V(mutex);
    }
  }

  process Cj  //消费者进程
  {
    while (1) {
      P(full);     //看看是否有数据
      P(mutex);        //互斥使用取出;
      V(emtpy);    //增1(可能唤醒一个生产者)
      V(mutex);


    }
  }
coend
```

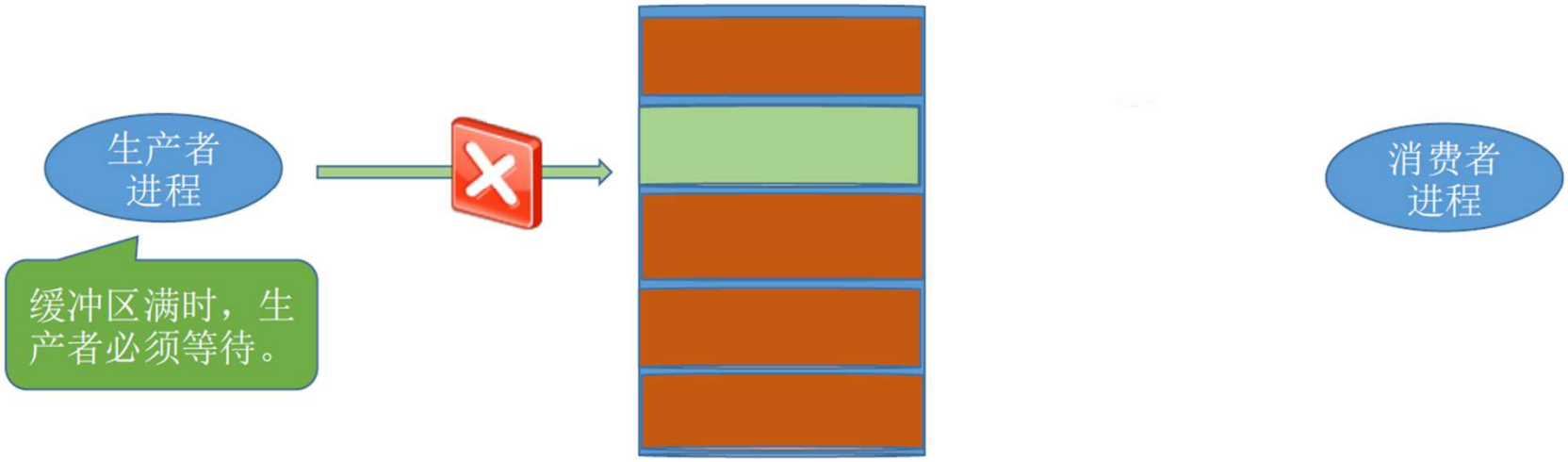
问题描述

系统中有一组生产者进程和一组消费者进程，生产者进程每次生产一个产品放入缓冲区，消费者进程每次从缓冲区中取出一个产品并使用。（注：这里的“产品”理解为某种数据）

生产者、消费者共享一个初始为空、大小为n的缓冲区。

只有缓冲区没满时，生产者才能把产品放入缓冲区，否则必须等待。

缓冲区没满→生产者生产



缓冲区（大小为5）

问题描述

系统中有一组生产者进程和一组消费者进程，生产者进程每次生产一个产品放入缓冲区，消费者进程每次从缓冲区中取出一个产品并使用。（注：这里的“产品”理解为某种数据）

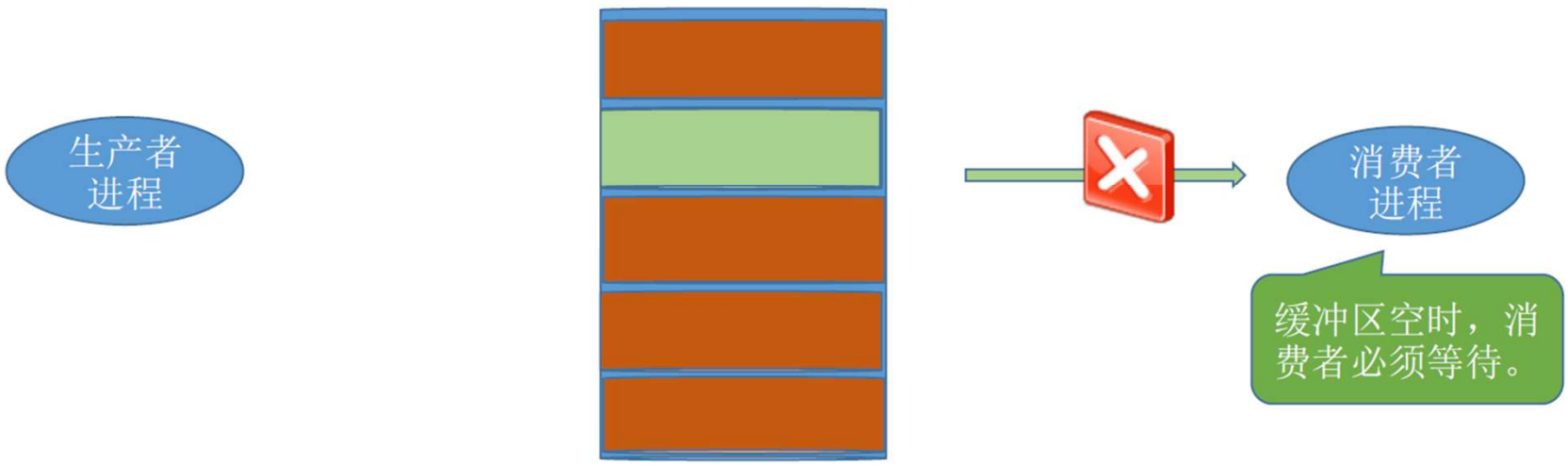
生产者、消费者共享一个初始为空、大小为n的缓冲区。

只有缓冲区没满时，生产者才能把产品放入缓冲区，否则必须等待。

缓冲区没满→生产者生产

只有缓冲区不空时，消费者才能从中取出产品，否则必须等待。

缓冲区没空→消费者消费



缓冲区（大小为5）

问题描述

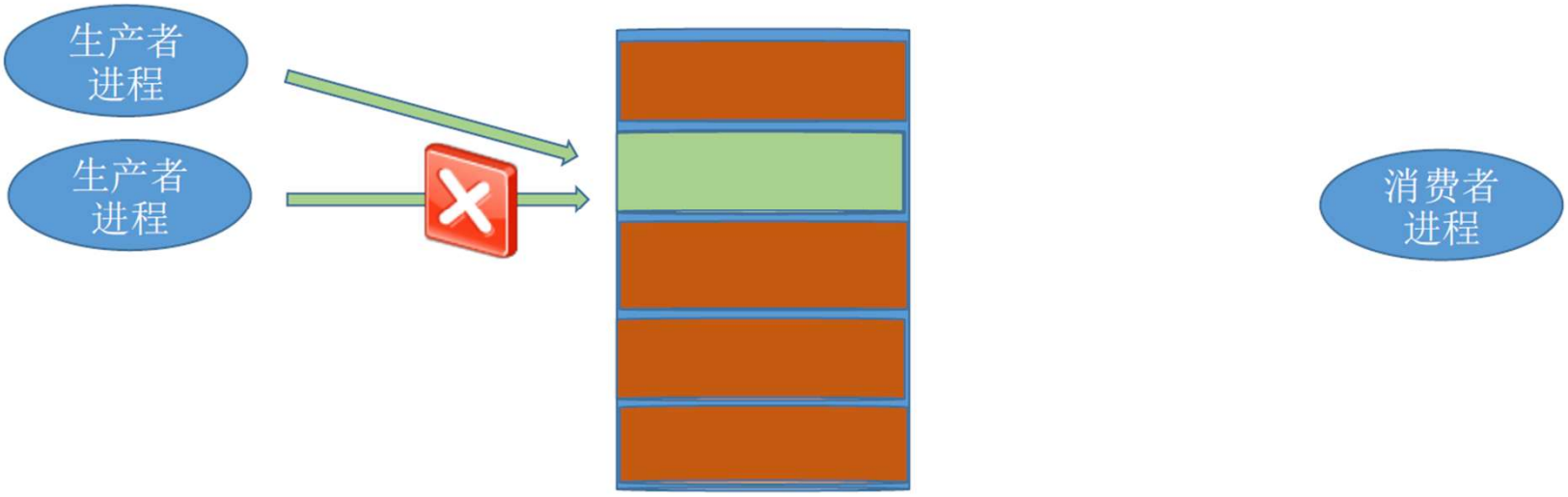
系统中有一组生产者进程和一组消费者进程，生产者进程每次生产一个产品放入缓冲区，消费者进程每次从缓冲区中取出一个产品并使用。（注：这里的“产品”理解为某种数据）

生产者、消费者共享一个初始为空、大小为n的缓冲区。

只有缓冲区没满时，生产者才能把产品放入缓冲区，否则必须等待。 缓冲区没满→生产者生产

只有缓冲区不空时，消费者才能从中取出产品，否则必须等待。 缓冲区没空→消费者消费

缓冲区是临界资源，各进程必须互斥地访问。 互斥关系



缓冲区（大小为5）

问题分析

系统中有一组生产者进程和一组消费者进程，生产者进程每次生产一个产品放入缓冲区，消费者进程每次从缓冲区中取出一个产品并使用。（注：这里的“产品”理解为某种数据）

生产者、消费者共享一个初始为空、大小为n的缓冲区。

只有缓冲区没满时，生产者才能把产品放入缓冲区，否则必须等待。 缓冲区没满→生产者生产

只有缓冲区不空时，消费者才能从中取出产品，否则必须等待。 缓冲区没空→消费者消费

缓冲区是临界资源，各进程必须互斥地访问。 互斥关系

PV操作题目分析步骤：

1. 关系分析。找出题目中描述的各个进程，分析它们之间的同步、互斥关系。
2. 整理思路。根据各进程的操作流程确定P、V操作的大致顺序。
3. 设置信号量。并根据题目条件确定信号量初值。（互斥信号量初值一般为1，同步信号量的初始值要看对应资源的初始值是多少）

在“前操作”之后执行 V(S)
在“后操作”之前执行 P(S)

问题分析

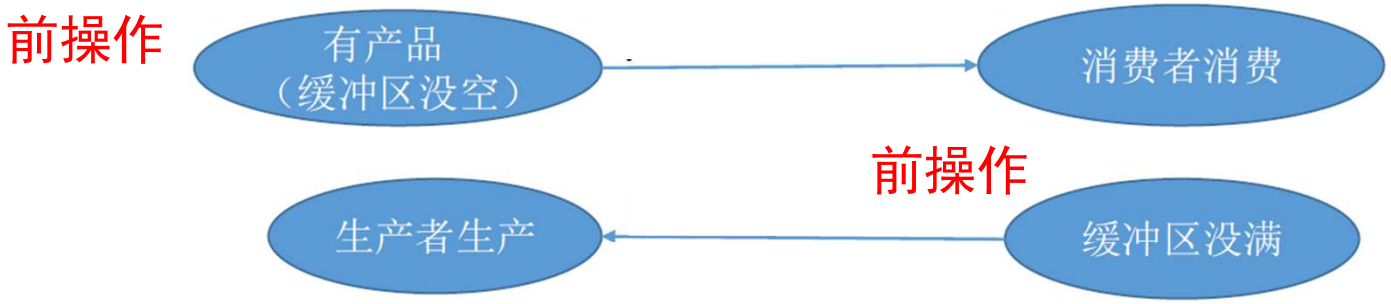
系统中有一组生产者进程和一组消费者进程，生产者进程每次生产一个产品放入缓冲区，消费者进程每次从缓冲区中取出一个产品并使用。（注：这里的“产品”理解为某种数据）

生产者、消费者共享一个初始为空、大小为n的缓冲区。

只有缓冲区没满时，生产者才能把产品放入缓冲区，否则必须等待。 缓冲区没满→生产者生产

只有缓冲区不空时，消费者才能从中取出产品，否则必须等待。 缓冲区没空→消费者消费

缓冲区是临界资源，各进程必须互斥地访问。 互斥关系



PV操作题目分析步骤：

1. 关系分析。找出题目中描述的各个进程，分析它们之间的同步、互斥关系。
2. 整理思路。根据各进程的操作流程确定P、V操作的大致顺序。
3. 设置信号量。并根据题目条件确定信号量初值。（互斥信号量初值一般为1，同步信号量的初始值要看对应资源的初始值是多少）

在“前操作”之后执行 V(S)
在“后操作”之前执行 P(S)

问题分析

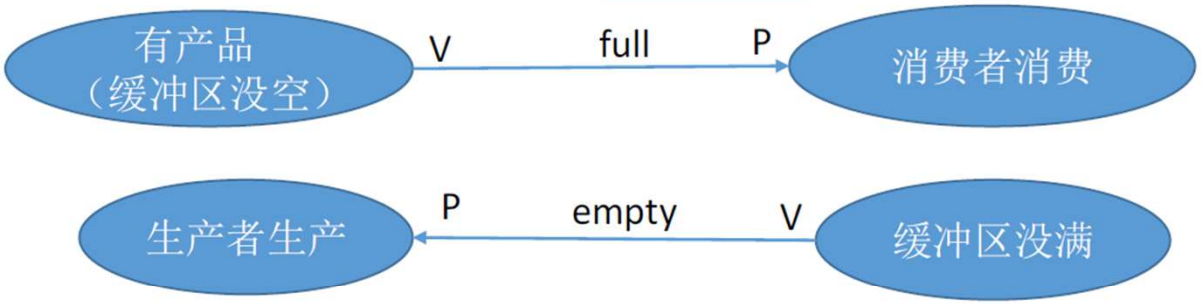
系统中有一组生产者进程和一组消费者进程，生产者进程每次生产一个产品放入缓冲区，消费者进程每次从缓冲区中取出一个产品并使用。（注：这里的“产品”理解为某种数据）

生产者、消费者共享一个初始为空、大小为n的缓冲区。

只有缓冲区没满时，生产者才能把产品放入缓冲区，否则必须等待。 缓冲区没满→生产者生产

只有缓冲区不空时，消费者才能从中取出产品，否则必须等待。 缓冲区没空→消费者消费

缓冲区是临界资源，各进程必须互斥地访问。 互斥关系



PV操作题目分析步骤：

1. 关系分析。找出题目中描述的各个进程，分析它们之间的同步、互斥关系。
2. 整理思路。根据各进程的操作流程确定P、V操作的大致顺序。
3. 设置信号量。并根据题目条件确定信号量初值。（互斥信号量初值一般为1，同步信号量的初始值要看对应资源的初始值是多少）

问题分析

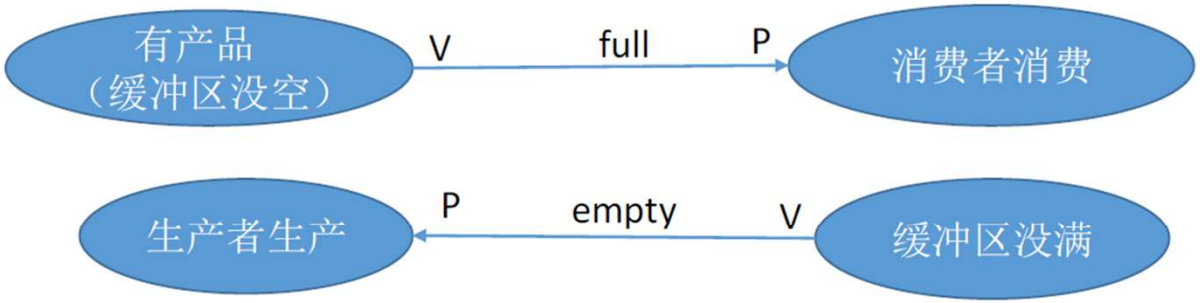
系统中有一组生产者进程和一组消费者进程，生产者进程每次生产一个产品放入缓冲区，消费者进程每次从缓冲区中取出一个产品并使用。（注：这里的“产品”理解为某种数据）

生产者、消费者共享一个初始为空、大小为n的缓冲区。

只有缓冲区没满时，生产者才能把产品放入缓冲区，否则必须等待。 缓冲区没满→生产者生产

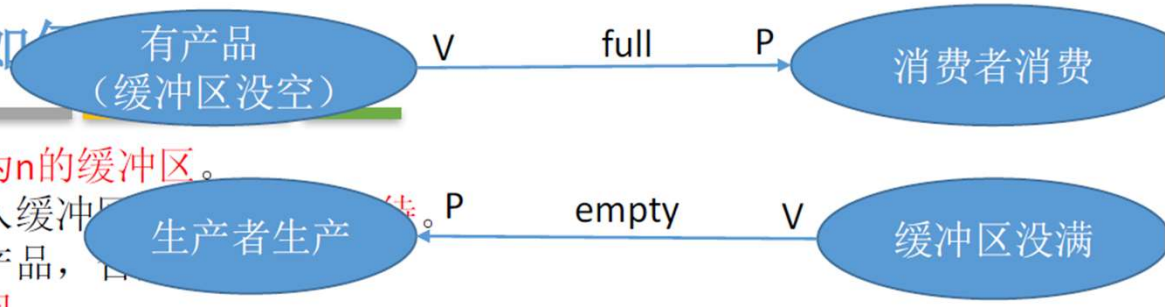
只有缓冲区不空时，消费者才能从中取出产品，否则必须等待。 缓冲区没空→消费者消费

缓冲区是临界资源，各进程必须互斥地访问。 互斥关系




```
semaphore mutex = 1;        //互斥信号量，实现对缓冲区的互斥访问
semaphore empty = n;        //同步信号量，表示空闲缓冲区的数量
semaphore full = 0;         //同步信号量，表示产品的数量，也即非空缓冲区的数量
```

如

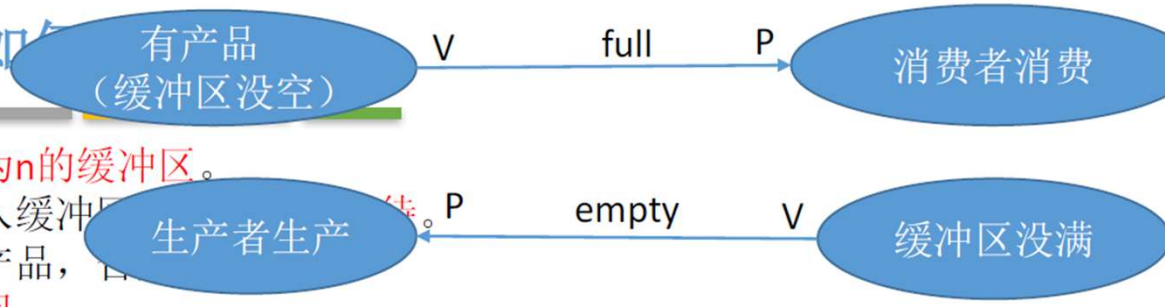


生产者、消费者共享一个初始为空、大小为n的缓冲区。
只有缓冲区没满时，生产者才能把产品放入缓冲区，否则必须等待。
只有缓冲区不空时，消费者才能从中取出产品，否则必须等待。
缓冲区是临界资源，各进程必须互斥地访问。

```
semaphore mutex = 1;          //互斥信号量，实现对缓冲区的互斥访问
semaphore empty = n;          //同步信号量，表示空闲缓冲区的数量
semaphore full = 0;           //同步信号量，表示产品的数量，也即非空缓冲区的数量
```

```
producer (){
    while(1){
        生产一个产品;

        把产品放入缓冲区;

    }
}
```

```
consumer (){
    while(1){

        从缓冲区取出一个产品;

        使用产品;
    }
```

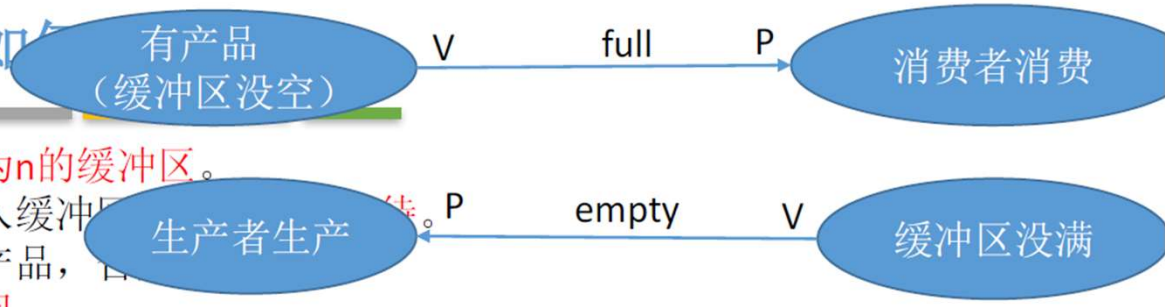


生产者、消费者共享一个初始为空、大小为n的缓冲区。
只有缓冲区没满时，生产者才能把产品放入缓冲区，否则必须等待。
只有缓冲区不空时，消费者才能从中取出产品，否则必须等待。
缓冲区是临界资源，各进程必须互斥地访问。

```
semaphore mutex = 1;          //互斥信号量，实现对缓冲区的互斥访问
semaphore empty = n;          //同步信号量，表示空闲缓冲区的数量
semaphore full = 0;           //同步信号量，表示产品的数量，也即非空缓冲区的数量
```

```
producer (){
    while(1){
        生产一个产品;
        P(empty);          消耗一个空闲缓冲区

        把产品放入缓冲区;


    }
}
```

```
consumer (){
    while(1){


        从缓冲区取出一个产品;


        使用产品;
    }
```

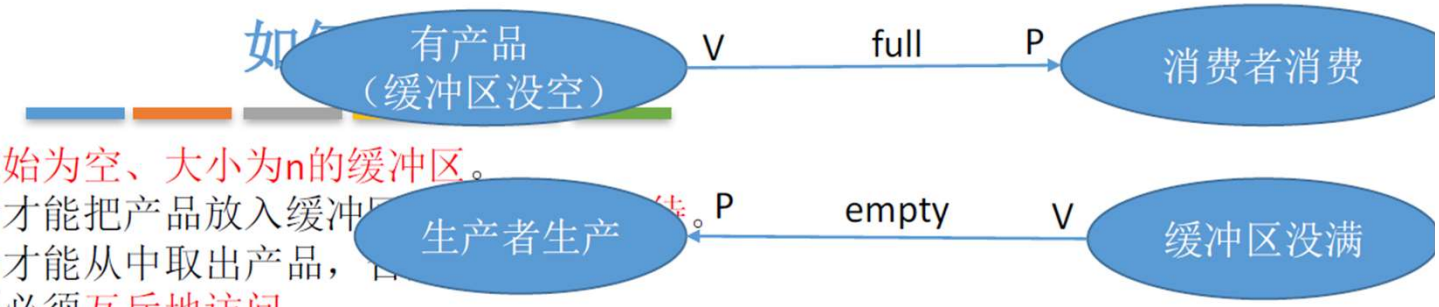


生产者、消费者共享一个初始为空、大小为n的缓冲区。
只有缓冲区没满时，生产者才能把产品放入缓冲区，否则必须等待。
只有缓冲区不空时，消费者才能从中取出产品，否则必须等待。
缓冲区是临界资源，各进程必须互斥地访问。

```
semaphore mutex = 1;          //互斥信号量，实现对缓冲区的互斥访问
semaphore empty = n;          //同步信号量，表示空闲缓冲区的数量
semaphore full = 0;           //同步信号量，表示产品的数量，也即非空缓冲区的数量
```

```
producer (){
    while(1){
        生产一个产品;
        P(empty);          消耗一个空闲缓冲区

        把产品放入缓冲区;

        V(full);
    }                      增加一个产品
}
```

```
consumer (){
    while(1){

        从缓冲区取出一个产品;

        使用产品;
    }
```



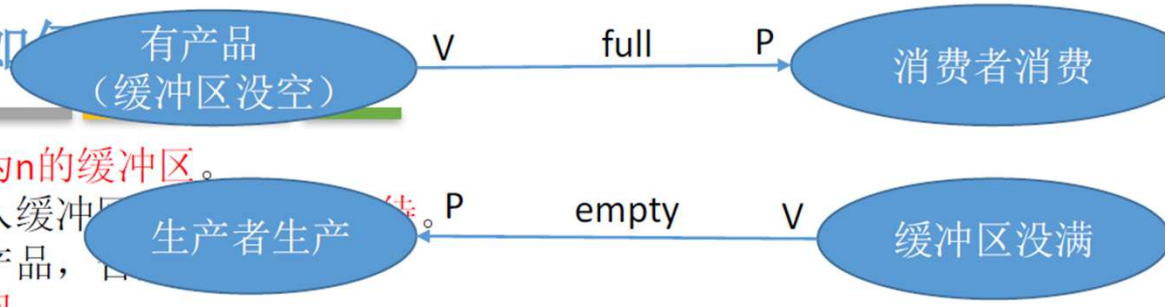
生产者、消费者共享一个初始为空、大小为n的缓冲区。
只有缓冲区没满时，生产者才能把产品放入缓冲区，否则必须等待。
只有缓冲区不空时，消费者才能从中取出产品，否则必须等待。
缓冲区是临界资源，各进程必须互斥地访问。

```
semaphore mutex = 1;          //互斥信号量，实现对缓冲区的互斥访问
semaphore empty = n;          //同步信号量，表示空闲缓冲区的数量
semaphore full = 0;           //同步信号量，表示产品的数量，也即非空缓冲区的数量
```

```
producer (){
    while(1){
        生产一个产品;
        P(empty);          消耗一个空闲缓冲区

        把产品放入缓冲区;

        V(full);
    }                      增加一个产品
}
```

```
consumer (){
    while(1){
        P(full);           消耗一个产品（非空缓冲区）

        从缓冲区取出一个产品;

        使用产品;
    }
```

如何

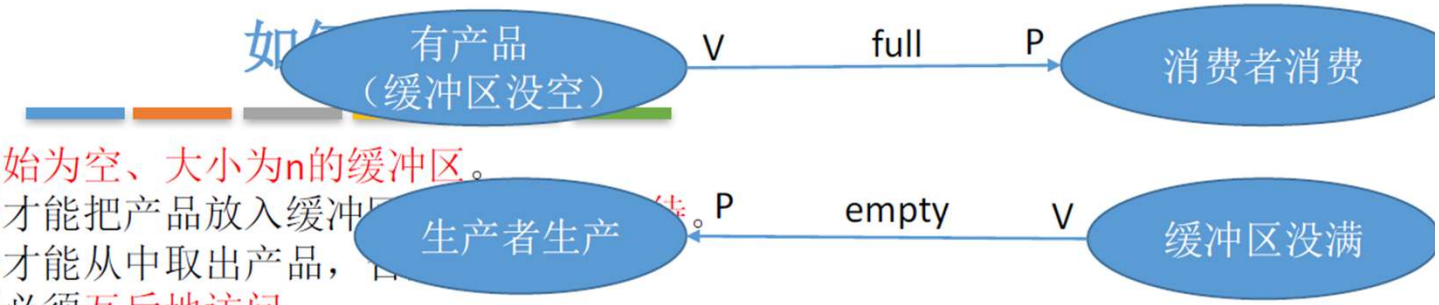


生产者、消费者共享一个初始为空、大小为n的缓冲区。
只有缓冲区没满时，生产者才能把产品放入缓冲区，否则必须等待。
只有缓冲区不空时，消费者才能从中取出产品，否则必须等待。
缓冲区是临界资源，各进程必须互斥地访问。

```
semaphore mutex = 1;          //互斥信号量，实现对缓冲区的互斥访问
semaphore empty = n;          //同步信号量，表示空闲缓冲区的数量
semaphore full = 0;           //同步信号量，表示产品的数量，也即非空缓冲区的数量
```

```
producer (){
    while(1){
        生产一个产品;
        P(empty);          消耗一个空闲缓冲区

        把产品放入缓冲区;

        V(full);
    }                      增加一个产品
}
```

```
consumer (){
    while(1){
        P(full);           消耗一个产品（非空缓冲区）

        从缓冲区取出一个产品;

        V(empty);
        使用产品;          增加一个空闲缓冲区
    }
```



生产者、消费者共享一个初始为空、大小为n的缓冲区。
只有缓冲区没满时，生产者才能把产品放入缓冲区，否则必须等待。
只有缓冲区不空时，消费者才能从中取出产品，否则必须等待。
缓冲区是临界资源，各进程必须互斥地访问。

```
semaphore mutex = 1;          //互斥信号量，实现对缓冲区的互斥访问
semaphore empty = n;          //同步信号量，表示空闲缓冲区的数量
semaphore full = 0;           //同步信号量，表示产品的数量，也即非空缓冲区的数量
```

```
producer (){
    while(1){
        生产一个产品;
        P(empty);
        把产品放入缓冲区;
        V(full);
    }
}
```

消耗一个空闲缓冲区

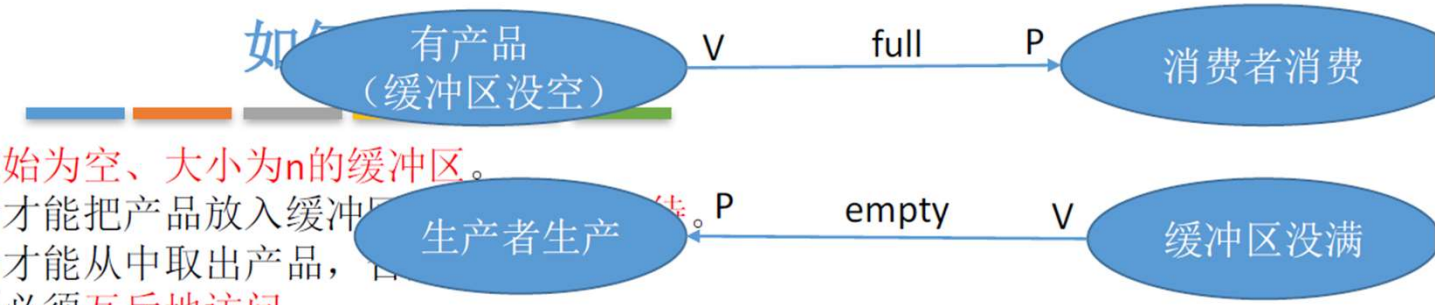
增加一个产品

```
consumer (){
    while(1){
        P(full);
        从缓冲区取出一个产品;
        V(empty);
        使用产品;
    }
}
```

消耗一个产品（非空缓冲区）

增加一个空闲缓冲区

实现两进程的同步关系，是在其中一个进程中执行P，另一进程中执行V



生产者、消费者共享一个初始为空、大小为n的缓冲区。
只有缓冲区没满时，生产者才能把产品放入缓冲区，否则必须等待。
只有缓冲区不空时，消费者才能从中取出产品，否则必须等待。
缓冲区是临界资源，各进程必须互斥地访问。

```
semaphore mutex = 1;          //互斥信号量，实现对缓冲区的互斥访问
semaphore empty = n;          //同步信号量，表示空闲缓冲区的数量
semaphore full = 0;           //同步信号量，表示产品的数量，也即非空缓冲区的数量
```

```
producer (){
    while(1){
        生产一个产品;
        P(empty);
        P(mutex);
        把产品放入缓冲区;
        V(mutex);
        V(full);
    }
}
```

消耗一个空闲缓冲区

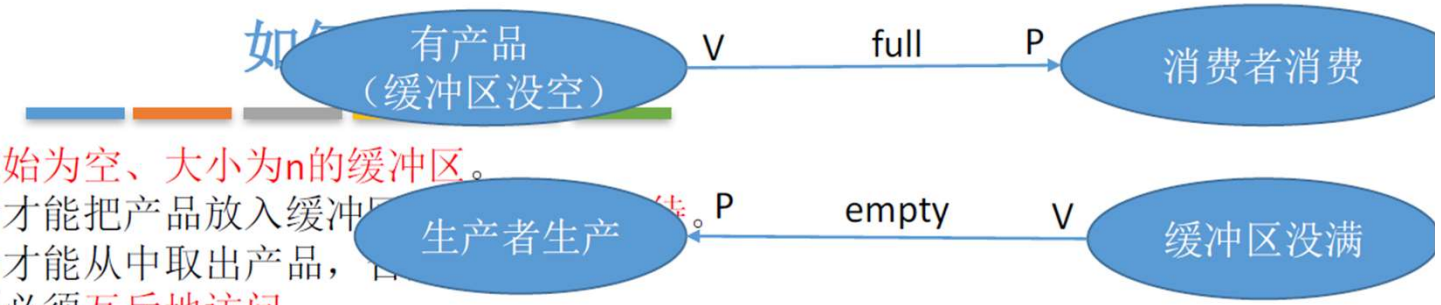
增加一个产品

```
consumer (){
    while(1){
        P(full);
        P(mutex);
        从缓冲区取出一个产品;
        V(mutex);
        V(empty);
        使用产品;
    }
}
```

消耗一个产品（非空缓冲区）

增加一个空闲缓冲区

实现两进程的同步关系，是在其中一个进程中执行P，另一进程中执行V



生产者、消费者共享一个初始为空、大小为n的缓冲区。
只有缓冲区没满时，生产者才能把产品放入缓冲区，否则必须等待。
只有缓冲区不空时，消费者才能从中取出产品，否则必须等待。
缓冲区是临界资源，各进程必须互斥地访问。

```
semaphore mutex = 1;        //互斥信号量，实现对缓冲区的互斥访问
semaphore empty = n;        //同步信号量，表示空闲缓冲区的数量
semaphore full = 0;         //同步信号量，表示产品的数量，也即非空缓冲区的数量
```

```
producer (){
    while(1){
        生产一个产品;
        P(empty);
        P(mutex);
        把产品放入缓冲区;
        V(mutex);
        V(full);
    }
}
```

实现互斥是在同一进程中进行一对PV操作

消耗一个空闲缓冲区

增加一个产品

```
consumer (){
    while(1){
        P(full);
        P(mutex);
        从缓冲区取出一个产品;
        V(mutex);
        V(empty);
        使用产品;
    }
```

消耗一个产品（非空缓冲区）

增加一个空闲缓冲区

当缓冲区大小为1时，可以用empty和full代替互斥信号量mutex

思考：能否改变相邻P、V操作的顺序？

```
producer (){
    while(1){
        生产一个产品;
        P(mutex);        ①
        P(empty);        ②
        把产品放入缓冲区;
        V(mutex);
        V(full);
    }
}
```

mutex 的P操作在前

```
consumer (){
    while(1){
        P(mutex);        ③
        P(full);         ④
        从缓冲区取出一个产品;
        V(mutex);
        V(empty);
        使用产品;
    }
}
```

能否放到PV操作之间？

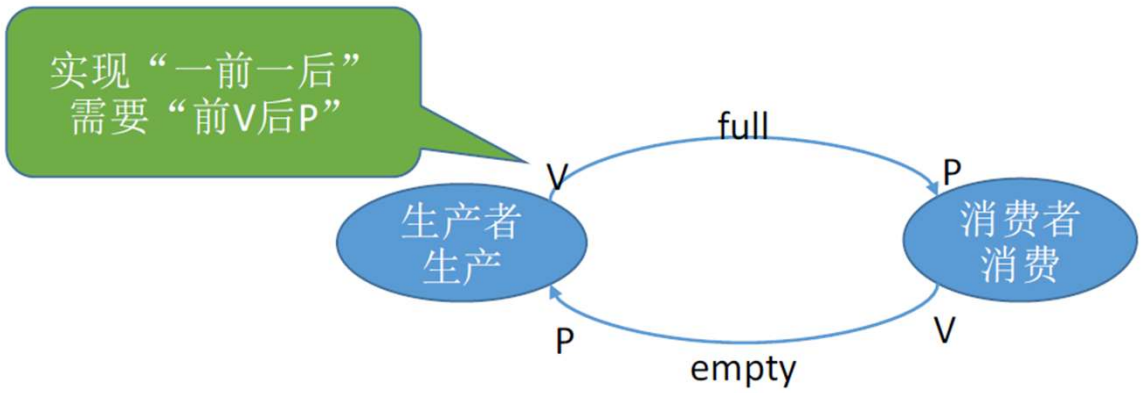
若此时缓冲区内已经放满产品，则 empty=0，full=n。
则生产者进程执行① 使mutex变为0，再执行②，由于已没有空闲缓冲区，因此生产者被阻塞。
由于生产者阻塞，因此切换回消费者进程。消费者进程执行③，由于mutex为0，即生产者还没释放对临界资源的“锁”，因此消费者也被阻塞。
这就造成了生产者等待消费者释放空闲缓冲区，而消费者又等待生产者释放临界区的情况，生产者和消费者循环等待被对方唤醒，出现“死锁”。
同样的，若缓冲区中没有产品，即full=0，empty=n。按③④① 的顺序执行就会发生死锁。
因此，实现互斥的P操作一定要在实现同步的P操作之后。
V操作不会导致进程阻塞，因此两个V操作顺序可以交换。

知识回顾与重要考点

PV 操作题目的解题思路：

1. 关系分析。找出题目中描述的各个进程，分析它们之间的同步、互斥关系。
2. 整理思路。根据各进程的操作流程确定P、V操作的大致顺序。
3. 设置信号量。设置需要的信号量，并根据题目条件确定信号量初值。（互斥信号量初值一般为1，同步信号量的初始值要看对应资源的初始值是多少）

生产者消费者问题是一个互斥、同步的综合问题。
对于初学者来说最难的是发现题目中隐含的两对同步关系。
有时候是消费者需要等待生产者生产，有时候是生产者要等待消费者消费，这是两个不同的“一前一后问题”，因此也需要设置两个同步信号量。



易错点：实现互斥和实现同步的两个P操作的先后顺序（死锁问题）

2、内存中有一组缓冲区被多个生产者进程、多个消费者进程共享使用，总共能存放10个数据，生产者进程把生成的数据放入缓冲区，消费者进程从缓冲区中取出数据使用。缓冲区满时生产者进程就停止将数据放入缓冲区，缓冲区空时消费者进程停止取数据。数据的存入和取出不能同时进行，试用信号量及P、V操作来实现该方案。

```
semaphore mutex, empty, full;
mutex=1;          //互斥信号量
empty=10;         //生产者进程的同步信号量
full=0;       //消费者进程的同步信号量

cobegin
  process Pi  //生产者进程
  {
    while (1) {
      生产数据;
      P(empty)    //看看是否还有空间可放
      P(mutex);   //互斥使用放入;
      V(full);   ///增1(可能唤醒一个消费者)
      V(mutex);

    }
  }

  process Cj  //消费者进程
  {
    while (1) {
      P(full)             //看看是否有数据
      P(mutex);          //互斥使用取出;
      V(emtpy);    //增1(可能唤醒一个生产者)
      V(mutex);

    }
  }
coend
```

3、有三个进程A、B和C完成文件打印任务:
A进程将文件记录从磁盘读入内存的缓冲区1中，每执行一次读一个记录;
B进程将缓冲区1中的内容复制到缓冲区2 中，每执行一次复制一个记录;
C进程将缓冲区2中的内容打印出来，每执行一次打印一个记录。

假设缓冲区的大小与记录大小相同均为1。请使用信号量来实现文件的正确打印。

```
semaphore empty1, full1, empty2, full2;
empty1 = empty2 = n;
full1 = full2 = 0;
void A()
{
 while(1)
 {
   读取磁盘上的记录;

   P(empty1);
   向缓冲区1中放记录;
   V(full1);
  }
}
```

```
void B()
{
while(1)
{
  P(full1);
   从缓冲区1中取记录;
   V(empty1);
   复制;

    P(empty2);
   向缓冲区2中放记录;
   V(full2);
  }
}
```

```
void C()
{
while(1)
 {
   P(full2);
   从缓冲区2中取记录;
    V(empty2);
    打印记录;
  }
}
```

1、有三个进程A、B和C完成文件打印任务:
A进程将文件记录从磁盘读入内存的缓冲区1中，每执行一次读一个记录;
B进程将缓冲区1中的内容复制到缓冲区2 中，每执行一次复制一个记录;
C进程将缓冲区2中的内容打印出来，每执行一次打印一个记录。

假设缓冲区的大小与记录大小相同均为n。请使用信号量来实现文件的正确打印。

4、有两个优先级相同的并发运行的进程 P1 和 P2，各自执行的操作如下，信号量 S1 和 S2 初值均为 0，x、y 和 z 的初值为 0。

Cobegin	

P1：
begin
y:=0;
y:=y+4;
V(S1);
z:=y+3;
P(S2);
y:=z+y
end
	P2：
	begin
	x:=2;
	x:=x+6;
	P(S1);
	x:=x+y;
	V(S2);
	z:=z+x;
	end

Coend	

试问 P1、P2 并发执行后，x、y、z 的值有几种可能，各为多少？

答：

1：x=12，y=11，z=19。

2：x=12，y=23，z=19。

3：x=12，y=11，z=7。

4、有两个优先级相同的并发运行的进程 P1 和 P2，各自执行的操作如下，信号量 S1 和 S2 初值均为 0，x、y 和 z 的初值为 0。

Cobegin	

P1：
begin
y:=0;
y:=y+4;
① V(S1);
⑥ z:=y+3;
⑦ P(S2);
⑧ y:=z+y
end
	P2：
	begin
	x:=2;
	x:=x+6;
	② P(S1);
	③ x:=x+y;
	④ V(S2);
	⑤ z:=z+x;
	end

Coend	

试问 P1、P2 并发执行后，x、y、z 的值有几种可能，各为多少？

答：

1：x=12，y=11，z=19。

2：x=12，y=23，z=19。

3：x=12，y=11，z=7。

4、有两个优先级相同的并发运行的进程 P1 和 P2，各自执行的操作如下，信号量 S1 和 S2 初值均为 0，x、y 和 z 的初值为 0。

Cobegin	

P1：
begin
y:=0;
y:=y+4;
① V(S1);
③ z:=y+3;
⑦ P(S2);
⑧ y:=z+y
end
	P2：
	begin
	x:=2;
	x:=x+6;
	② P(S1);
	④ x:=x+y;
	⑤ V(S2);
	⑥ z:=z+x;
	end

Coend	

试问 P1、P2 并发执行后，x、y、z 的值有几种可能，各为多少？

答：

1：x=12，y=11，z=19。

2：x=12，y=23，z=19。

3：x=12，y=11，z=7。

4、有两个优先级相同的并发运行的进程 P1 和 P2，各自执行的操作如下，信号量 S1 和 S2 初值均为 0，x、y 和 z 的初值为 0。

Cobegin	

P1：
begin
y:=0;
y:=y+4;
① V(S1);
③ z:=y+3;
⑥ P(S2);
⑦ y:=z+y
end
	P2：
	begin
	x:=2;
	x:=x+6;
	② P(S1);
	④ x:=x+y;
	⑤ V(S2);
	⑧ z:=z+x;
	end

Coend	

试问 P1、P2 并发执行后，x、y、z 的值有几种可能，各为多少？

答：

1：x=12，y=11，z=19。

2：x=12，y=23，z=19。

3：x=12，y=11，z=7。

信号量与P/V操作

4、问题思考与拓展学习

（2）拓展学习

- 黄刚, 徐小龙，段卫华. 操作系统教程. 北京: 人民邮电出版社 <4.3.5 POSIX 信号量, 4.3.6 Linux中的信号量机制>
- Wills C E. Process Synchronization. ACM Computing Surveys, 2011
- 高升, 陈月峰. 哲学家就餐问题的算法实现. 计算机工程与科学, 2016
- 徐宝文. 带标记信号量——一种新型同步与互斥机制. 计算机科学, 2001
- 张宝哲, 帖军, 蒋天发. 基于信号量机制的理发师模型在复杂语义下的研究 . 计算机科学, 2012

谢谢大家！