计算机考研系列书课包

玩转操作系统

主讲人 刘财政

第二讲　进程管理

本讲内容

考点一: 进程的基本概念 ★★

考点二: 进程管理和状态切换 ★★★★★

考点三: 进程和线程 ★★★★

考点四: 进程同步 ★★★★★

考点五: 管程机制 ★★

考点六: 进程通信 ★★

考点七: 经典同步问题 ★★★★★

考点四:

进程同步

考点框架

- 进程同步的概念
- 进程同步软件实现方法
- 进程硬件软件实现方法
- 锁机制
- 信号量机制
- 信号量的应用

考点一: 进程的基本概念

进程同步的概念

考点四: 进程同步

进程同步的概念

- 进程具有异步性的特征。异步性是指，各并发执行的进程以各自独立的、不可预知的速度向前推进。

考点四: 进程同步

进程同步的概念

两个进程P1和P2并发执行，共享初值为0的全局变量x。P1和P2实现对全局变量x加1的机器级代码描述如下：

P1	P2
①Mov R1,x //(x)->R1 ②Inc R1 //(R1) + 1->R1 ③Mov x,R1 //(R1)->x	④Mov R2,x //(x)->R2 ⑤Inc R2 //(R2) + 1->R2 ⑥Mov x,R2 //(R2)->x

在所有可能的指令执行序列中，x的值为（）

（1）当按照①②③④⑤⑥的顺序执行，x=2

①Mov R1,x //(x)->R1
②Inc R1 //(R1) + 1->R1
③Mov x,R1 //(R1)->x
④Mov R2,x //(x)->R2
⑤Inc R2 //(R2) + 1->R2
⑥Mov x,R2 //(R2)->x

考点四: 进程同步

进程同步的概念

两个进程P1和P2并发执行，共享初值为0的全局变量x。P1和P2实现对全局变量x加1的机器级代码描述如下：

P1	P2
①Mov R1,x //(x)->R1 ②Inc R1 //(R1) + 1->R1 ③Mov x,R1 //(R1)->x	④Mov R2,x //(x)->R2 ⑤Inc R2 //(R2) + 1->R2 ⑥Mov x,R2 //(R2)->x

在所有可能的指令执行序列中，x的值为（）

（2）当按照①④②③⑤⑥的顺序执行，x=1

①Mov R1,x //(x)->R1
④Mov R2,x //(x)->R2
②Inc R1 //(R1) + 1->R1
③Mov x,R1 //(R1)->x
⑤Inc R2 //(R2) + 1->R2
⑥Mov x,R2 //(R2)->x

考点四: 进程同步

进程同步的概念

两个进程P1和P2并发执行，共享初值为0的全局变量x。P1和P2实现对全局变量x加1的机器级代码描述如下：

P1	P2
①Mov R1,x //(x)->R1 ②Inc R1 //(R1) + 1->R1 ③Mov x,R1 //(R1)->x	④Mov R2,x //(x)->R2 ⑤Inc R2 //(R2) + 1->R2 ⑥Mov x,R2 //(R2)->x

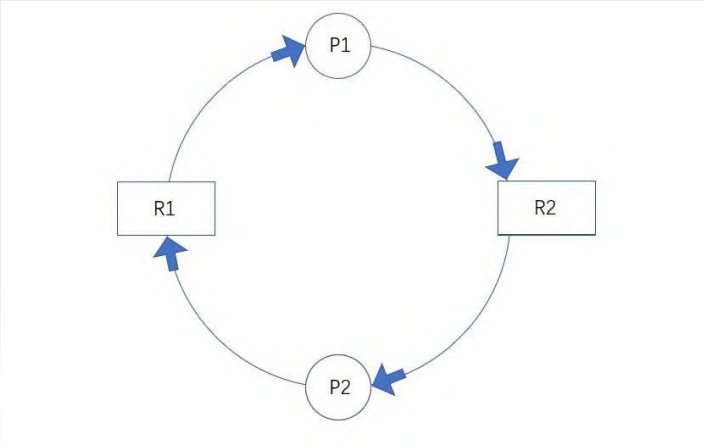
- 当进程共享变量时，进程的执行顺序不同，最后的结果是不一样的，所以就需要对进程的执行顺序进行控制，这就是进程同步。
- 所谓进程同步就是指协调这些完成某个共同任务的并发线程/进程，在某些位置上指定线程/进程的先后执行次序、传递信号或消息。

考点四: 进程同步

进程同步的概念

资源竞争出现了两个控制问题：

- 一个是死锁 （deadlock ）问题，一组进程如果都获得了部分资源，还想要得到其他进程所占有的资源，最终所有的进程将陷入死锁。
- 另一个是饥饿（starvation ）问题，这是指这样一种情况：一个进程由于其他进程总是优先于它而被无限期拖延。



考点四: 进程同步

进程同步的几个术语

①进程间的两种制约关系：

- 进程同步，本质上是由于进程之间共享资源导致的。进程间的两种制约关系：
- 间接相互制约（互斥）：因为进程在并发执行的时候共享临界资源而形成的相互制约的关系，需要对临界资源互斥地访问；这种关系也叫进程互斥。
- 直接制约关系（同步）：多个进程之间为完成同一任务而相互合作而形成的制约关系。这种关系也叫进程合作（协作）。

考点四: 进程同步

进程同步的几个术语

②临界资源：

指同一时刻只允许一个进程可以该问的资源称之为临界资源，诸进程之间采取互斥方式，实现对临界资源的共享。

考点四: 进程同步

进程同步的几个术语

③临界区

进程中访问临界资源的那段代码。每个进程在进入临界区之前，应先对欲访问的临界资源的“大门”状态进行检测，如果“大门”敞开，进程便可进入临界区，并将临界区的“大门”关上；否则就表示有进程在临界区内，当前进程无法进入临界区。

可把一个访问临界资源的循环进程描述如下：

```
repeat
        entry section;      //进入区
        critical section;     //临界区
        exit section;    // 退出去
        remainder section;  //剩余区
until false;
```

负责检查是否可进入临界区，若可进入，则应设置正在访问临界资源的标志，以阻止其他进程同时进入临界区

访问临界资源的那段代码

负责解除正在访问临界资源的标志

扫尾工作

考点四: 进程同步

进程同步的几个术语

④ 原语

由若干条指令组成的，用户完成一定功能的一个过程。原语操作的一个特点就是“原子操作”，因此原语在执行的过程中不允许被中断。原子操作在系统态下执行，常驻内存。

考点四: 进程同步

进程同步遵循的规则

- 空闲让进。当无进程处于临界区时，表明临界资源处于空闲状态，应允许一个请求进入临界区的进程立即进入自己的临界区，以有效地利用临界资源。

考点四: 进程同步

进程同步遵循的规则

- 忙则等待。当已有进程进入临界区时，表明临界资源正在被访问，因而其它试图进入临界区的进程必须等待，以保证对临界资源的互斥访问。

考点四: 进程同步

进程同步遵循的规则

- 有限等待。对要求访问临界资源的进程，应保证在有限时间内能进入自己的临界区，以免陷入“死等”状态。

考点四: 进程同步

进程同步遵循的规则

- 让权等待。当进程不能进入自己的临界区时，应立即释放处理机，以免进程陷入“忙等”状态。

考点四: 进程同步

进程同步遵循的规则

- 四个准则必须满足是空闲让进（不能做到空闲让进将会导致死等或者死锁）；忙则等待（不能做到忙则等待则会导致错误）；有限等待（不能做到有限等待将会导致死等）。
- 而让权等待则不是必须的，如果不能做到让权等待，充其量就是浪费了处理器资源，不会带来死锁或者错误，例如后面我们讲的Peterson和TestAndSet方法，可以很好地做到进程同步，但是不满足让权等待的要求。

考点四: 进程同步

进程同步遵循的规则

- 请注意：忙等和死等都是没能进入临界区，它们的区别如下：
- 死等： 对行死等的进程来说，这个进程可能是处于阻塞状态，等着别的进程将其唤醒（signal 原语），但是如果唤醒原语一直无法执行，对于阻塞的进程来说，就是一直处于死等的状态，是无法获得处理机的。
- 忙等：忙等状态比较容易理解，处于忙等状态的进程是一直占有处理机去不断的判断临界区是否可以进入，在此期间，进程一直在运行，这就是忙等状态。因为处于忙等的进程会一直霸占处理机的，相当于陷入死循环了。

考点四: 进程同步


进程同步
两种形式的制约关系
间接相互制约关系
互斥——竞争
直接相互制约关系
同步——协作
临界资源
分区
进入区enter section
临界区critical section
退出区exit section
剩余区remainder section
同步机制应遵循的规则
1.空闲让进
2.忙则等待
3.有限等待
4.让权等待

考点四: 进程同步

进程同步软件实现方法

考点四: 进程同步

考点四: 进程同步

单标志法

int turn = 0;

P0进程:

```
while(turn!=0);
critical section;
turn=1;
remainder section;
```

P1进程:

```
while(turn!=1);
critical section;
turn=0;
remainder section;
```

- 设置一个公共整型变量turn，代表是否允许进入临界区的进程编号。

考点四: 进程同步

单标志法

int turn = 0;

P0进程:

```
while(turn!=0);
critical section;
turn=1;
remainder section;
```

P1进程:

```
while(turn!=1);
critical section;
turn=0;
remainder section;
```

- 设置一个公共整型变量turn，代表是否允许进入临界区的进程编号。
- 如果P0进入临界区离开后，P1却没有进入临界区的打算，则turn始终无法修改为0，P0无法再次进入临界区。

考点四: 进程同步

单标志法

```
int  turn = 0;
```

P0进程:

```
while(turn!=0);
critical section;
turn=1;
remainder section;
```

P1进程:

```
while(turn!=1);
critical section;
turn=0;
remainder section;
```

- 设置一个公共整型变量turn，代表是否允许进入临界区的进程编号。
- 如果turn=0，则P0可以进入临界区，只要P0没有退出临界区，而另外一个进程将会一直停留在while循环，无法访问临界区资源。

考点四: 进程同步

单标志法

int turn = 0;

P0进程:

```
while(turn!=0);
critical section;
turn=1;
remainder section;
```

P1进程:

```
while(turn!=1);
critical section;
turn=0;
remainder section;
```

- 如果P0进程无法进入临界区，P1则也会无法进入（turn值未修改），出现临界区资源空闲的状态，违背了「空闲让进」原则。

考点四: 进程同步

双标志法先检查

- 在每个进程访问临界区资源之前，先检查一下临界资源是否被其他进程占用，
- 如果flag为true，则当前进程需要等待。
- flag[i]代表i进程是否访问临界资源，true代表正在访问，false代表未进入临界区。

bool flag[2]; flag[0] = false; flag[1]=false;

P0进程：	P1进程：
while(flag[1]);①	while(flag[0]);②
flag[0]=TRUE;③	flag[1]=TRUE;④
critical section;	critical section;
flag[0]=FALSE;	flag[1]=FALSE;
remainder section;	remainder section;

- 优点：不用交替的进入，两个进程在时间片轮转下并发执行，可以连续使用。

考点四: 进程同步

双标志法先检查

- 在每个进程访问临界区资源之前，先检查一下临界资源是否被其他进程占用，
- 如果flag为true，则当前进程需要等待。
- flag[i]代表i进程是否访问临界资源，true代表正在访问，false代表未进入临界区。

bool flag[2]; flag[0] = false; flag[1]=false;

P0进程：	P1进程：
while(flag[1]);①	while(flag[0]);②
flag[0]=TRUE;③	flag[1]=TRUE;④
critical section;	critical section;
flag[0]=FALSE;	flag[1]=FALSE;
remainder section;	remainder section;

- 缺点：如果并发过程中按照①②③④的顺序进行，此时有没进程访问临界区资源，则进程代码块会跳过while循环等待，则两个进程会同时进入临界区，违背了忙则等待的原则。

考点四: 进程同步

双标志后检查

双标志前检查是先检查对方的状态，再设置自己的状态。而双标志后检查是先设置自己的状态，表明自己想要进临界区的意愿，然后再进行检查其他进程的状态。

bool flag[2]; flag[0] = false; flag[1]=false;

P0进程：	P1进程：
flag[0]=TRUE; ①	flag[1]=TRUE; ②
while(flag[1]); ③	while(flag[0]); ④
critical section;	critical section;
flag[0]=FALSE;	flag[1]=FALSE;
remainder section;	remainder section;

- 在该算法下，两个进程都会表明自己想要进入临界区的意愿，如果并发过程中按照①②③④的顺序进行,但是在检查过程中发现，大家都想进去，两个进程都处于等待的状态，导致「饥饿」现象。

考点四: 进程同步

Peterson算法

- 为了防止两个进程为了进入临界区而出现无限等待的现象，在设置一个变量turn，在设置该进程是否进入临界区的标志后，再设置turn标志。

bool flag[2]; flag[0] = false; flag[1]=false; int turn = 0;

P0进程：	P1进程：
flag[0]=TRUE; turn=1; ①	flag[1]=TRUE; turn=0;②
while(flag[1] && turn==1);③	while(flag[0]&&turn==0);④
critical section;	critical section;
flag[0]=FALSE;	flag[1]=FALSE;
remainder section;	remainder section;

考点四: 进程同步

Peterson算法

- while()循环中，同时考虑另一个进程状态和当前进程的不允许进入临界标志。

bool flag[2]; flag[0] = false; flag[1]=false; int turn = 0;

P0进程：	P1进程：
flag[0]=TRUE; turn=1; ①	flag[1]=TRUE; turn=0;②
while(flag[1] && turn==1);③	while(flag[0]&&turn==0);④
critical section;	critical section;
flag[0]=FALSE;	flag[1]=FALSE;
remainder section;	remainder section;

考点四: 进程同步

Peterson算法

- while()循环中，同时考虑另一个进程状态和当前进程的不允许进入临界标志。

P0进程：	P1进程：
flag[0]=TRUE; turn=1; ①	flag[1]=TRUE; turn=0;②
while(flag[1] && turn==1);③	while(flag[0]&&turn==0);④
critical section;	critical section;
flag[0]=FALSE;	flag[1]=FALSE;
remainder section;	remainder section;

考点四: 进程同步

Peterson算法

- while()循环中，同时考虑另一个进程状态和当前进程的不允许进入临界标志。

P0进程：	P1进程：
flag[0]=TRUE; turn=1; ①	flag[1]=TRUE; turn=0;②
while(flag[1] && turn==1);③	while(flag[0]&&turn==0);④
critical section;	critical section;
flag[0]=FALSE;	flag[1]=FALSE;
remainder section;	remainder section;

- 假如按照①②③④的顺序执行，则执行完①②后，flag[0]=TRUE, flag=TRUE, turn =0，此时遇到④是会一直处于循环状态的，而③可以顺利跳出while循环，进入临界区。

考点四: 进程同步

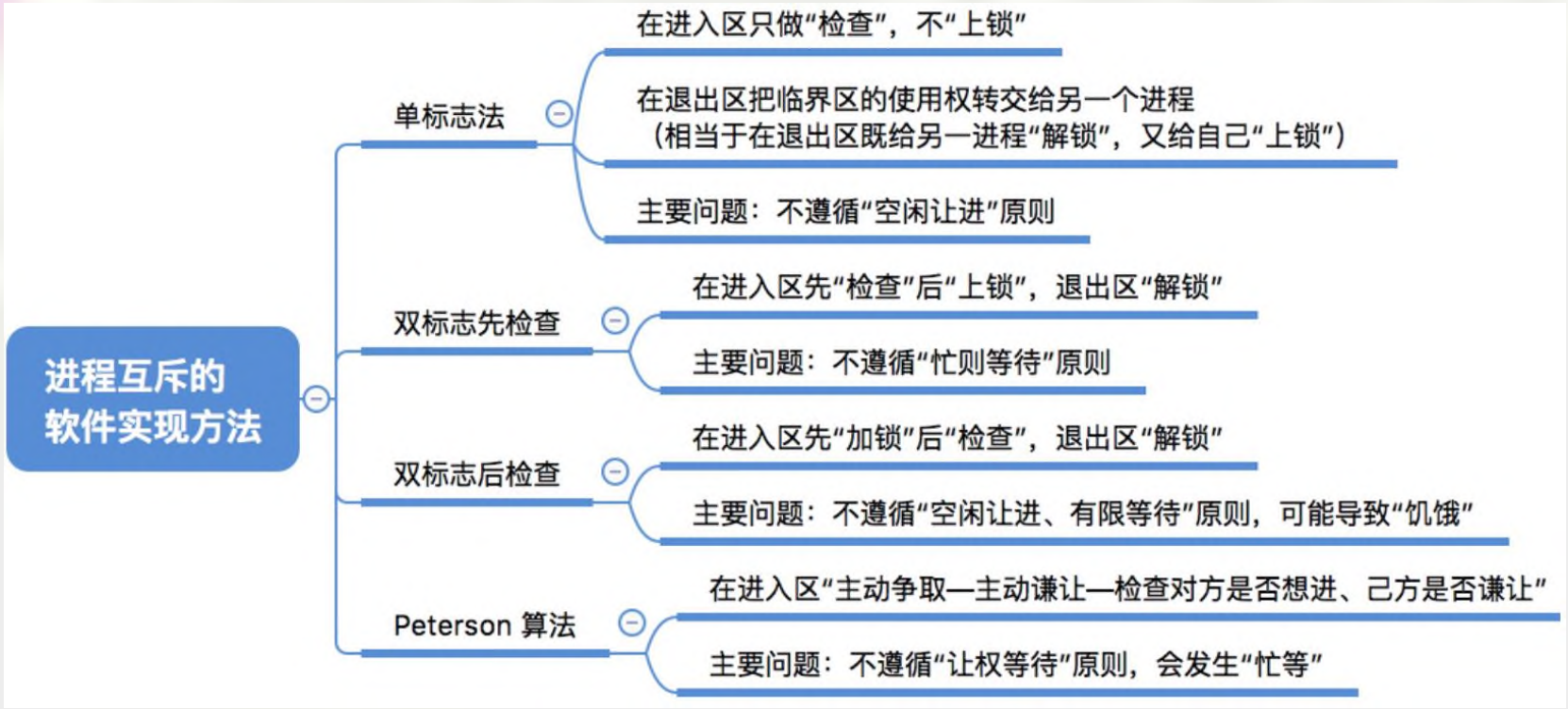
Peterson算法

- while()循环中，同时考虑另一个进程状态和当前进程的不允许进入临界标志。

P0进程：	P1进程：
flag[0]=TRUE; turn=1; ①	flag[1]=TRUE; turn=0;②
while(flag[1] && turn==1);③	while(flag[0]&&turn0=0);④
critical section;	critical section;
flag[0]=FALSE;	flag[1]=FALSE;
remainder section;	remainder section;

- 利用turn值可以很好的解决「双标志后检查」算法引起的饥饿现象，Peterson算法是「单标志法」和「双标志后检查」算法的结合，利用flag解决临界资源的互斥访问，利用turn解决饥饿现象。但是Peterson算法会处于忙等状态。

考点四: 进程同步


进程互斥的
软件实现方法
单标志法
在进入区只做“检查”，不“上锁”
在退出区把临界区的使用权转交给另一个进程
（相当于在退出区既给另一进程“解锁”，又给自己“上锁”）
主要问题：不遵循“空闲让进”原则
双标志先检查
在进入区先“检查”后“上锁”，退出区“解锁”
主要问题：不遵循“忙则等待”原则
双标志后检查
在进入区先“加锁”后“检查”，退出区“解锁”
主要问题：不遵循“空闲让进、有限等待”原则，可能导致“饥饿”
Peterson 算法
在进入区“主动争取—主动谦让—检查对方是否想进、己方是否谦让”
主要问题：不遵循“让权等待”原则，会发生“忙等”

考点一: 进程的基本概念

进程硬件软件实现方法

考点四: 进程同步

考点四: 进程同步

中断屏蔽方法

关中断 ----> 关闭中断后，就不允许当前的进程被中断，也不会发生进程切换

临界区

开中断 ----> 当前进程访问完临界区之后，开启中断，经过处理机调度，给需要访问临界资源的进程提供机会

- 该方法简单高效，
- 但是不适合多处理机，只适合操作系统内核进程，不适合用户进程（开/关中断指令只能在内核态运行）
- 如果这组指令让用户随意使用会很危险，可能会影响整个CPU的调度。

考点四: 进程同步

中断屏蔽方法

- 利用“开/关中断指令”的方式实现。和原语的实现思想类似，在整个指令执行过程中，不允许发生进程切换，因此不可能存在两个进程同时都在临界区的情况。

```
...
关中断 ----> 关闭中断后，就不允许当前的进程被中断，也不会发生进程切换
临界区
开中断    ---->  当前进程访问完临界区之后，开启中断，经过处理机调度，给需要访问临界资源的进程提供机会
...
```

考点四: 进程同步

TestAndSet指令

```
// 布尔型变量lock表示该临界区是否被加锁
// true表示加锁，false表示未加锁
bool TestAndSet(bool *lock){
    bool old;
    old = *lock; // old用来存放lock原来的值
    *lock = true; // 给临界区上锁，给lock设置为true
    return old; // 返回lock原来的值
}
```

```
// 使用TSL指令实现进程互斥的算法逻辑
while(TestAndSet(&lock));
临界区代码段...
lock = false;  // 解锁
剩余区代码段...
```

考点四: 进程同步

TestAndSet指令

```
// 布尔型变量lock表示该临界区是否被加锁
// true表示加锁，false表示未加锁
bool TestAndSet(bool *lock){
    bool old;
    old = *lock; // old用来存放lock原来的值
    *lock = true; // 给临界区上锁，给lock设置为true
    return old; // 返回lock原来的值
}
```

```
// 使用TSL指令实现进程互斥的算法逻辑
while(TestAndSet(&lock));
临界区代码段...
lock = false;  // 解锁
剩余区代码段...
```

- 简称TS指令，或者TestAndSetLock指令，或者TSL指令。TSL指令是通过硬件实现的，执行过程为原子操作，不允许被中断。

考点四: 进程同步

TestAndSet指令

```
// 布尔型变量lock表示该临界区是否被加锁
// true表示加锁，false表示未加锁
bool TestAndSet(bool *lock){
    bool old;
    old = *lock; // old用来存放lock原来的值
    *lock = true; // 给临界区上锁，给lock设置为true
    return old; // 返回lock原来的值
}
```

```
// 使用TSL指令实现进程互斥的算法逻辑
while(TestAndSet(&lock));
临界区代码段...
lock = false;  // 解锁
剩余区代码段...
```

- 如果lock初始为false，while循环条件不满足，直接跳出循环，当前进程进入临界区。
- 如果lock初始为true，说明当前临界资源被其余的进程访问，等到正在运行的进程在退出区将lock设置为false对临界区进行解锁。

考点四: 进程同步

TestAndSet指令

```
// 布尔型变量lock表示该临界区是否被加锁
// true表示加锁，false表示未加锁
bool TestAndSet(bool *lock){
    bool old;
    old = *lock; // old用来存放lock原来的值
    *lock = true; // 给临界区上锁，给lock设置为true
    return old; // 返回lock原来的值
}
```

```
// 使用TSL指令实现进程互斥的算法逻辑
while(TestAndSet(&lock));
临界区代码段...
lock = false;  // 解锁
剩余区代码段...
```

- 优点：实现简单，不需要想软件实现方法一样严格检查逻辑漏洞；
- TSL指令适用于多处理机环境。

考点四: 进程同步

TestAndSet指令

```
// 布尔型变量lock表示该临界区是否被加锁
// true表示加锁，false表示未加锁
bool TestAndSet(bool *lock){
    bool old;
    old = *lock; // old用来存放lock原来的值
    *lock = true; // 给临界区上锁，给lock设置为true
    return old; // 返回lock原来的值
}
```

```
// 使用TSL指令实现进程互斥的算法逻辑
while(TestAndSet(&lock));
临界区代码段...
lock = false;  // 解锁
剩余区代码段...
```

- 缺点：不满足“让权等待”原则，对于暂时无法进入临界区的进程会占用CPU并且循环执行TSL指令，从而导致“忙等”

考点四: 进程同步

Swap指令

```
// Swap 指令的作用是交换两个变量的值
Swap(bool *a, bool *b){
    bool temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

```
// 以下是用Swap指令实现互斥的算法逻辑
// lock表示当前临界区是否被加载
bool old = true;
while(old == true)
    Swap(&lock, &old);
临界代码段...
lock = false;
剩余代码段...
```

- 逻辑上Swap指令和TSL指令并无太大的区别。old变量先设置为true，进入while循环，交换old值和lock值。

考点四: 进程同步

Swap指令

```
// Swap 指令的作用是交换两个变量的值
Swap(bool *a, bool *b){
    bool temp;
    temp = *a;
    *a = *b;
    *b = temp;
}
```

```
// 以下是用Swap指令实现互斥的算法逻辑
// lock表示当前临界区是否被加载
bool old = true;
while(old == true)
    Swap(&lock, &old);
临界代码段...
lock = false;
剩余代码段...
```

- 当前临界区资源被占用，则lock值为true，此时交换后，依然进入while循环。
- 如果当前临界资源空闲，则old值被替换为false，跳出循环，该进程进入临界区代码。
- 优缺点方面与TSL指令类似。

考点四: 进程同步

锁机制

考点四: 进程同步

LockOne类

- 这个类有一个标志数组flag，继续来个比喻，这个flag就相当于一个旗帜。LockOne类遵循这样的协议：如果线程想进入临界区，首先把自己的旗帜升起来(flag相应位置1)，表示感兴趣。然后等对方的旗帜降下来就可以进入临界区了。
- 如果线程离开临界区，则把自己的旗帜降下来。
- LockOne类的协议看起来挺朴实的，但是存在一个问题：当两个线程都把旗帜升起来，然后等待对方的旗帜降下来就会出现死锁的状态

考点四: 进程同步

LockTwo类

- 观察LockOne类存在的问题，就是在两个线程同时升起旗帜的时候，需要有一个线程妥协吧，这样就需要指定一个牺牲品，因此LockTwo类解决这个问题。
- 当两个线程进行竞争的时候，总有一个牺牲品（较晚对victim赋值的线程），因此可以避免死锁。但是，当没有竞争时，只有一个线程想进入临界区，那么牺牲品一直是自己，直到等待别人来替换自己才行。

考点四: 进程同步

Perterson锁

- 通过上面两个类可以发现，LockOne类适合没有竞争的场景，LockTwo类适合有竞争的场景。那么将LockOne类和LockTwo类结合起来，就可以构造出一种很好的锁算法。

考点四: 进程同步

Barkey锁

- Bakery锁，是一种最简单也最为人们锁熟知的n线程锁算法。每个线程想进入临界区之前都会升起自己的旗帜，并得到一个序号。然后升起旗帜的线程中序号最小的线程才能进入临界区。 每个线程离开临界区的时候降下自己的旗帜

考点四: 进程同步

信号量机制

考点四: 进程同步

考点四: 进程同步

一些约定

- 用户进程可以通过使用操作系统提供的一对原语来对信号量进行操作，从而很方便的实现了进程互斥、进程同步。
- 信号量其实就是一个变量，可以用一个信号量来表示系统中某种资源的数量，比如：系统中只有一台打印机，就可以设置一个初值为1的信号量。

考点四: 进程同步

一些约定

- 原语是一种特殊的程序段，其执行只能一气呵成，不可被中断。
- 原语是由关中断/开中断指令实现的。因此如果能把进入区、退出区的操作都用“原语”实现，使这些操作能“一气呵成”就能避免问题。

考点四: 进程同步

一些约定

- 一对原语：wait(S)原语和signal(S)原语，可以把原语理解为我们自己写的函数，函数名分别为wait和signal，括号里的信号量S其实就是函数调用时传入的一个参数。
- wait、signal原语常简称为P、V操作（来自荷兰语proberen和verhogen）。因此，做题的时候常把
- wait(S)、signal(S)两个操作分别写为P(S)、V(S)

考点四: 进程同步

整型信号量

最初由Dijkstra把整型信号量定义为一个整型量，除初始化外，仅能通过两个标准的原子操作(Atomic Operation) wait(S)和signal(S)来访问。

这两个操作一直被分别称为**P、V**操作。 wait和signal操作可描述为：

wait(S)：while S≤0 do no-op

S：=S-1;

signal(S)：S：=S+1;

在整型信号量机制中的wait操作，只要是信号量**S≤0**， 就会**不断地测试**。因此，该机制并未遵循“让权等待”的准则， 而是使进程处于“**忙等**”的状态。

考点四: 进程同步

记录型信号量

- 在信号量机制中，除了需要一个用于代表资源数目的整型变量value外，还应**增加一个进程链表L**，用于链接上述的所有**等待进程**。记录型信号量是由于它采用了记录型的数据结构而得名的。

所包含的上述两个数据项可描述为：

```
// 记录信号量的定义
typedef struct{
    int value;                  // 剩余资源数
    struct process *L;          // 等待队列
} semaphore;
```

考点四: 进程同步

记录型信号量

```
procedure wait(S)
    var S: semaphore;
    begin
      S.value :    = S.value-1;
      if S.value < 0 then block(S,L)
    end
```

如果剩余资源数不够，
使用block原语使进程从运行态进入阻塞态，并把挂到信号量S的等待队列（即阻塞队列）中

考点四: 进程同步

记录型信号量

```
procedure signal(S)
   var S: semaphore;
   begin
    S.value :    =S.value+1;
    if S.value≤0 then wakeup(S,L);
   end
```

释放资源后，若还有别的进程在等待这种资源，则使用wakeup原语唤醒等待队列中的一个进程，该进程从阻塞态变为就绪态

考点四: 进程同步

记录型信号量

```
// 记录信号量的定义
typedef struct{
    int value;              // 剩余资源数
    struct process *L;      // 等待队列
} semaphore;
```

- 在记录型信号量机制中，S.value的初值表示系统中某类资源的数目， 因而又称为**资源信号量**，这个初值取决于系统中有多少可用资源数量；

考点四: 进程同步

记录型信号量

```
procedure wait(S)
    var S: semaphore;
    begin
        S.value :    = S.value-1;
        if S.value < 0 then block(S,L)
    end
```

- 对**资源信号量**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，因此进程应调用block原语，进行**自我阻塞**，放弃处理机，并插入到信号量链表S.L中。该机制遵循了“**让权等待**”准则。
- 此时S.value的绝对值表示在该信号量链表中已阻塞进程的数目。

考点四: 进程同步

记录型信号量

```
procedure signal(S)
    var S: semaphore;
    begin
        S.value :    =S.value+1;
        if S.value≤0 then wakeup(S,L);
    end
```

- 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，故还应调用wakeup原语，将S.L链表中的第一个等待**进程唤醒。**

考点四: 进程同步

记录型信号量

如果S.value的初值为1，表示只允许一个进程访问临界资源，此时的信号量转化为互斥信号量。

```
procedure wait(S)
    var S: semaphore;
    begin
        S.value :    = S.value-1;
        if S.value < 0 then block(S,L)
    end
```

考点四: 进程同步

记录型信号量

两类信号量

- 互斥信号量：初值为1，用于实现进程互斥
- 资源信号量：初值大于1，用于实现进程合作

考点四: 进程同步

信号量的应用

考点四: 进程同步

信号量实现互斥

利用信号量实现进程互斥的进程可描述如下:

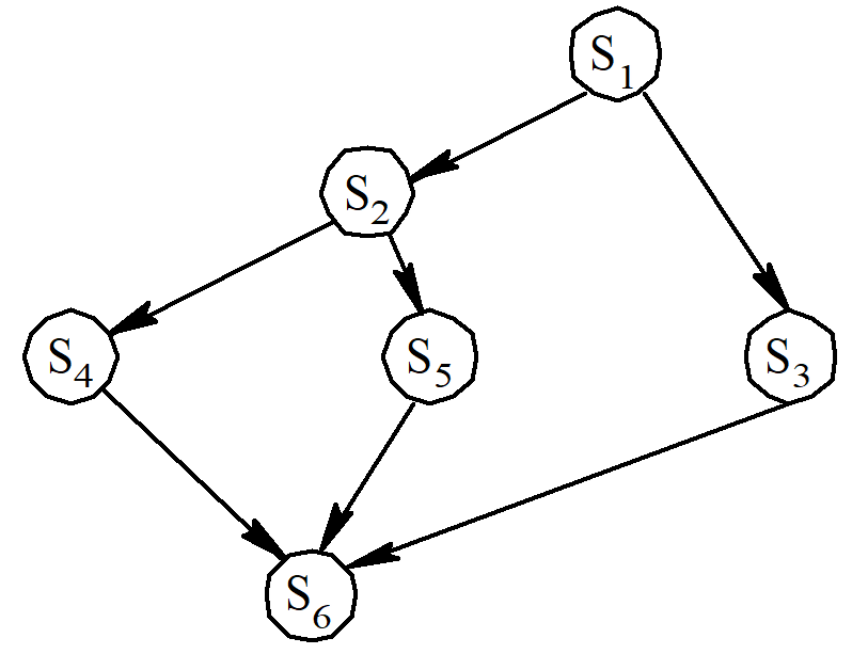
```
var mutex:semaphore :   =1;
process 1:
   begin
       repeat
           wait(mutex);
           critical section
           signal(mutex);
           remainder seetion
       until false;
   end
```

```
process 2:
    begin
        repeat
            wait(mutex);
            critical section
             signal(mutex);
              remainder section
         until false
    end
```

考点四: 进程同步

信号量实现合作

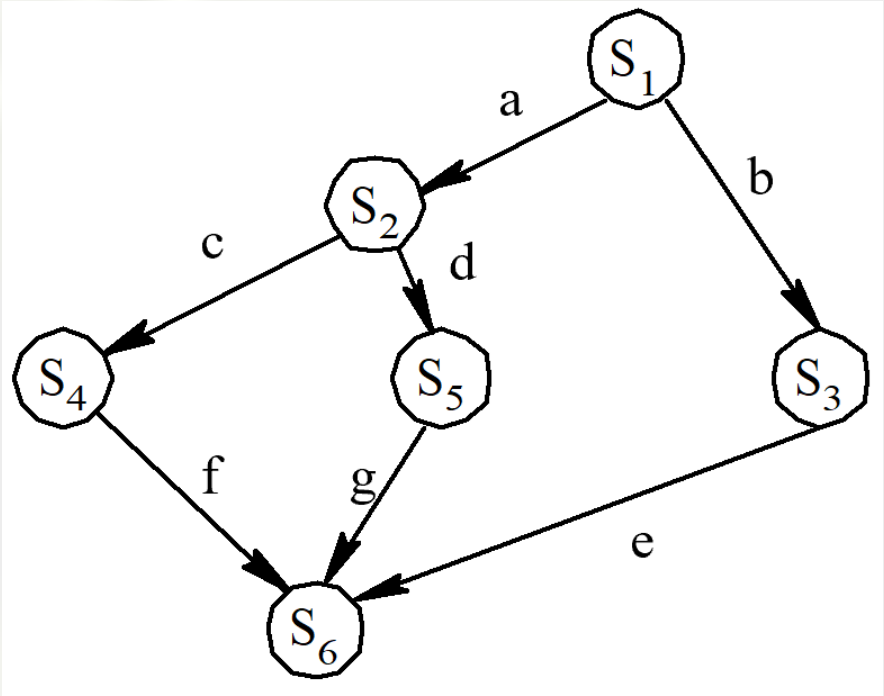
并发执行的进程，如何写这6个进程?



考点四: 进程同步

信号量实现合作

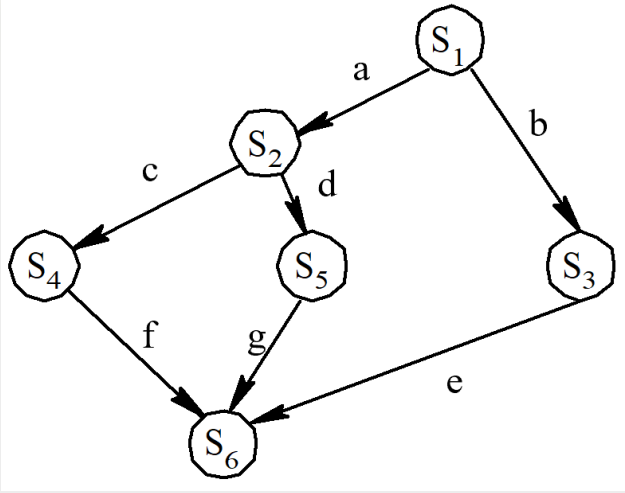
并发执行的进程，如何写这6个进程?



考点四: 进程同步

信号量实现合作

```
Var a,b,c,d,e,f,g; semaphore : = 0,0,0,0,0,0,0;
    begin
      parbegin
        begin S1; signal(a); signal(b); end;
        begin wait(a); S2; signal(c); signal(d); end;
        begin wait(b); S3; signal(e); end;
        begin wait(c); S4; signal(f); end;
        begin wait(d); S5; signal(g); end;
        begin wait(e); wait(f); wait(g); S6; end;
      parend
end
```



【牛刀小试】

1．【广东工业大学 2014】临界区是指并发进程中访问共享变量的()段。

A．管理信息　　B．信息存储　　C．数据　　D．程序

D【解析】临界区

进程中访问临界资源的那段代码。每个进程在进入临界区之前，应先对欲访问的临界资源的“大门”状态进行检测，如果“大门”敞开，进程便可进入临界区，并将临界区的“大门”关上；否则就表示有进程在临界区内，当前进程无法进入临界区。

可把一个访问临界资源的循环进程描述如下：

```
repeat
        entry section;      //进入区
        critical section;   //临界区
        exit section;    // 退出去
        remainder section;  //剩余区
until false;
```

负责检查是否可进入临界区，若可进入，则应设置正在访问临界资源的标志，以阻止其他进程同时进入临界区

访问临界资源的那段代码

负责解除正在访问临界资源的标志

扫尾工作

2．【燕山大学　2013】设与某资源相关联的信号量初值为4，当前值为-2，若M表示该资源的可用个数，N表示等待该资源的进程数，则M，N分别是( )。

A．0,2　　B．2,0　　C．2,2　　D．1,0

- 对**资源信号量**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，因此进程应调用block原语，进行**自我阻塞**，放弃处理机，并插入到信号量链表S.L中。该机制遵循了“**让权等待**”准则。
- 此时S.value的绝对值表示在该信号量链表中已阻塞进程的数目。

2．【燕山大学　2013】设与某资源相关联的信号量初值为4，当前值为-2，若M表示该资源的可用个数，N表示等待该资源的进程数，则M，N分别是( )。

A．0,2　　B．2,0　　C．2,2　　D．1,0

A【解析】S=-2表示等待该资源的进程数N=2，则可用资源M=0，因此选择A。

【经典总结】信号量的物理含义：S. value>0表示有S. value个资源可用；S. value=0表示无资源可用；S. value<0时，S. value表示等待队列中的进程个数。信号量的当前值为1，表示该资源的可用个数为1，没有等待该资源的进程。

3．【南京理工大学 2013】在对记录型信号量的P操作的定义中，当信号量的值(　　)时，执行P操作的进程变为阻塞态。

A．大于0　　B．小于0　　C．等于0　　D．小于等于0

- 对**资源信号量**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，因此进程应调用block原语，进行**自我阻塞**，放弃处理机，并插入到信号量链表S.L中。该机制遵循了“**让权等待**”准则。
- 此时S.value的绝对值表示在该信号量链表中已阻塞进程的数目。

3．【南京理工大学 2013】在对记录型信号量的P操作的定义中，当信号量的值(　　)时，执行P操作的进程变为阻塞态。

A．大于0　　B．小于0　　C．等于0　　D．小于等于0

B【解析】对于记录型信号量，在执行一次P操作时，信号量S减1；当S<0时，进程应阻塞，因此选择B。

4．有3个进程共享同一程序段，而每次只允许两个进程进入该程序段，若用P，V操作同步机制，则信号量S的取值范围是( )。

A．2,1,0,-1　　B．3,2,1,0

C．2,1,0,-1,　　D．1,0,-1,-2

A【**解析**】程序段最多允许两个进程进入，则表示资源的数目为2，因此，信号量初值为2。

- 每进入一个进程，信号量的值减1，
- 当信号量的值减为0时，表示两个进程均进入程序段，
- 此时若再有一个进程请求进入执行P操作，则信号量的值减为-1，进程阻塞。

因此选择A。

5．在9个生产者、6个消费者共享容量为8的缓冲器的生产者消费者问题中，保证进程互斥使用缓冲器的信号量初值为( )。

A．1　　B．6　　C．8　　D．9

A【**解析**】互斥使用即每次只允许一个进程使用，故信号量初值为1，因此选择A。

6．若一个系统中共有5个并发进程涉及某个相同的变量A，则变量A的相关临界区是由( )个临界区构成的（假设每个进程对于变量A的操作都只有一段代码）。

A．1　　B．5　　C．与资源数量有关　　D．与进程功能有关

进程中访问临界资源的那段代码。每个进程在进入临界区之前，应先对欲访问的临界资源的“大门”状态进行检测，如果“大门”敞开，进程便可进入临界区，并将临界区的“大门”关上；否则就表示有进程在临界区内，当前进程无法进入临界区。

可把一个访问临界资源的循环进程描述如下：

```
repeat
        entry section;      //进入区
        critical section;    //临界区
        exit section;    // 退出去
        remainder section;  //剩余区
until false;
```

负责检查是否可进入临界区，若可进入，则应设置正在访问临界资源的标志，以阻止其他进程同时进入临界区

访问临界资源的那段代码

负责解除正在访问临界资源的标志

扫尾工作

6．若一个系统中共有5个并发进程涉及某个相同的变量A，则变量A的相关临界区是由( )个临界区构成的（假设每个进程对于变量A的操作都只有一段代码）。

A．1　　B．5　　C．与资源数量有关　　D．与进程功能有关

B【**解析**】临界区就是每个进程中访问临界资源的那段代码。5个并发进程都涉及变量A，每一个进程中都有访问变量A的代码，所以每个进程中都有相关临界区，因此变量A的相关临界区是由5个临界区构成的，因此选择B。

7．【汕头大学　2016】( )中的两条语句并发执行，不会出现与时间有关的错误。

A．y=x+a；和 z=x-b；　　B．y=x+a；和x=b；

C．z=x-a；和 z=y+b；　　D．以上答案都不对

A【**解析**】对于A，x的值不变，y=x+a；和z=x-b；并发执行，不会出现与时间有关的错误。

对于B，第一条语句中的x和第二条语句中的x共享且x值会变，会出现与时间有关的错误。

对于C，z是共享变量且会改变，会出现与时间有关的错误。因此选择A。

8．进程A在执行过程中要使用临界资源，但要先获得进程B的计算结果，而此时进程B正忙于I/O操作，则此时进程A应遵循同步机制的(　)准则。

A．让权等待　　B．空闲让进　　C．忙则等待　　D．有限等待

A【**解析**】进程A在执行过程中要使用临界资源，但要先获得进程B的计算结果，即A要在B之后，此时进程A应遵循同步机制的让权等待准则，因此选择A。

【经典总结】进程同步机制：

- 空闲让进：当无进程处于临界区时，应允许一个请求进入临界区的进程立即进入自己的临界区。
- 忙则等待：当已有进程进入其临界区时，其他试图进入临界区的进程必须等待。
- 有限等待：对要求访问临界资源的进程，应保证能在有限时间内进入自己的临界区。
- 让权等待：当进程不能进入自己的临界区时，应立即释放处理机。

9、有两个并发进程如下所示，对于这段程序的运行，正确的说法是（　）。

```
int  x, y, z, t, u;
P1 (){                          P2 (){
while (true){                       while (true){
    x=1; ①                              x=0 ;⑤
    y=0; ②                              t=0;⑥
    if (x>=1)y=y+1;③                    if (x<=1)t=t+2;⑦
    z=y; ④                              u=t;⑧
  }                                   }
}                                   }
```

A．程序能正确运行，结果唯一　　B．程序不能正确运行，可能有两种结果

C．程序不能正确运行，结果不确定　　D．程序不能正确运行，可能会死锁

C【解析】

- 执行顺序为①②③④⑤⑥⑦⑧，此时结果为x=0；y=1；z=1；t=2；u=2。
- 执行顺序为①②⑤⑥③④⑦⑧，此时结果为x=0；y=0；z=0；t=2；u=2。
- 执行顺序为⑤⑥⑦⑧①②③④，此时结果为x=1；y=1；z=1；t=2；u=2。
- 一共有三种结果，程序不能正确运行。因此选择C。

10、有两个优先级相同的并发程序P1和P2，它们的执行过程如下所示，假设当前信号量s1=0，s2=0。当前的z=2，进程运行结束后，x、y和z的值分别是（ ）。

进程P1	进程P2
…	…
y:=1;	x:1=1
y:=y+2;	x:=x+1;
z:=y+1;	P (s1);
V(s1);	x:=x+y;
P(s2);	z:=x+z;
y:=z+y;	V(s2);

A．5，9，9　　B．5，9，4　　C．5，12，9　　D．5，12，4

C【解析】由于并发进程，进程的执行具有不确定性，

在P1，P2执行到第一个P，V操作前，应该是相互无关的。现在考虑第一个对1的P，V操作，由于进程P2是P(s1)操作，因此，它必须等待P1执行完V(s1)操作后才可继续运行，此时x，y，z的值分别为3，3，4，

当进程P1执行完V(s1)后便在P(s2)上阻塞，此时P2可以运行直到V(s2)，此时x，y，z的值分别为5，3，9，进程P1继续运行直到结束，最终的x，y，2值分别为5，12，9。因此选择C。

11．【广东工业大学　2014】有两个并发进程都要使用一台打印机，打印机对应的信号量是S，若S=0，则表示()。

A．没有进程在用打印机

B．有一进程在用打印机

C．有一进程在用打印机，另一进程正等待使用打印机

D．两个进程都在用打印机

B【解析】S=0表示有一进程在用打印机，因此选择B。

【经典总结】

- P(S)：将信号量S减1，若结果≥0，则该进程继续执行；若结果<0，则该进程被阻塞，并将其插入该信号量的等待队列中，然后转去调度另一进程。
- V(S)：将信号量S加1，若结果>0，则该进程继续执行；若结果≤0，则从该信号量的等待队列中移出一个进程，使其从阻塞状态变为就绪状态，并插入就绪队列中，然后返回当前进程继续执行。

【经典总结】

- 信号量S值的大小表示某类资源的数量。
  - 当S>0时，其值表示当前可供分配的资源数目；
  - 当S<0时，其绝对值表示S信号量的等待队列中的进程数目。每执行一次P操作，S值减1，表示请求分配一个资源。
  - 若S≥0，表示可以为进程分配资源，即允许进程进入其临界区；
  - 若S<0，表示已没有资源可供分配，申请资源的进程被阻塞，并插入S的等待队列中，S的绝对值表示等待队列中进程的数目，此时CPU将重新进行调度。

【经典总结】

- 信号量S值的大小表示某类资源的数量。
- 每执行一次V操作，S值加1，表示释放一个资源。若S>0，表示等待队列为空；若S≤0，则表示等待队列中有因申请不到相应资源而被阻塞的进程，于是唤醒其中一个进程，并将其插入就绪队列。
- 无论以上哪种情况，执行V操作的进程都可继续运行。

12．【广东工业大学　2014】我们把在一段时间内，只允许一个进程访问的资源称为临界资源，下列论述正确的为( )。

A．对临界资源是不能实现资源共享的

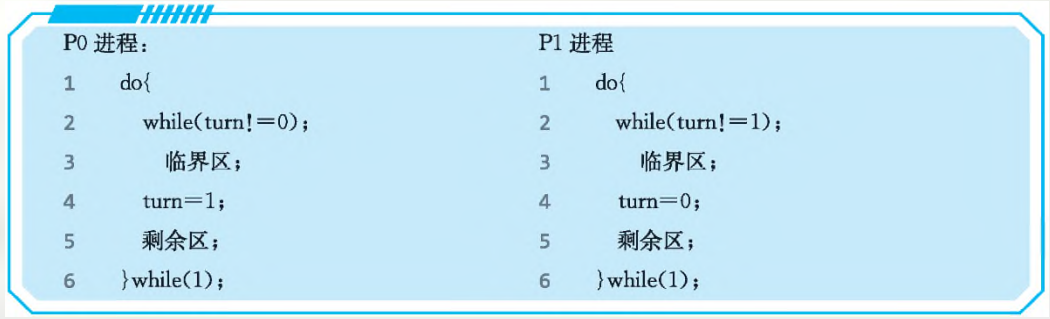
B．只要能使程序并发执行，这些并发执行的程序便可对临界资源实现共享

C．为临界资源配上相应的设备控制块后，便能被共享

D．对临界资源，应采取互斥访问方式来实现共享

D【**解析**】各进程采取互斥的方式，实现共享的资源称作临界资源。属于临界资源的硬件有打印机、磁带机等，软件有消息缓冲队列、变量、数组、缓冲区等。诸进程间应采取互斥方式，实现对这种资源的共享。进程中用于实现进程互斥的那段代码称为临界区。显然，若能保证诸进程互斥地进入自己的临界区，便可实现诸进程对临界资源的互斥访问。为此，每个进程在进入临界区之前，应先对欲访问的临界资源进行检查，看它是否正被访问。如果此刻该临界资源未被访问，进程便可进入临界区对该资源进行访问。因此选择D。

13、【南京理工大学　2018】进程P0和P1共享某个临界资源，下图是解决两个进程P0和P1临界区问题的一个算法，该算法(　)。



```
P0 进程：                          P1 进程
1   do{                            1   do{
2     while(turn!=0);              2     while(turn!=1);
3       临界区；                   3       临界区；
4     turn=1;                      4     turn=0;
5     剩余区；                     5     剩余区；
6   }while(1);                     6   }while(1);
```

A．不能保证进程互斥进入，也不能保证有空让进

B．能保证进程互斥进入，但不能保证有空让进

C．既能保证进程互斥进入，也能保证有空让进

D．不能保证进程互斥进入，但能保证有空让进

B【**解析**】初始化是设置的等待P0对进程进行访问，但是如果P1先申请访问就不能执行，违背了空闲让进的准则。当turn值为1时，如果P0进程想要访问临界资源，会在while语句中执行空循环等待，直到turn变为0。所以当P0不能拿到访问权时，并没有放弃等待，而是一直占着CPU做空循环等待，违背了让权等待的准则。

14．【中国计量大学　2020】对两个并发进程，其互斥信号量为mutex，初值为1，若mutex=0，则表明(　)。

A．有一个进程进入临界区而另一个进程正处于阻塞状态

B．有一个进程进入临界区且没有进程处于阻塞状态

C．没有进程进入临界区

D．有两个进程进入临界区

B【**解析**】mutex=0表示有一个进程进入临界区且没有进程处于阻塞状态，因此选择B。

15．【吉林大学　2016】互斥算法的三个正确性条件是什么？下面基于交换指令的硬件互斥算法是否满足上述条件？若不满足请改进。

```
int lock;//全局变量,初始值为 0
int key;//进程局部变量
do{
   key=1;
   do{
        swap(&lock,&key);
   }while(key==1);
   临界区;
   lock=0;
    其余代码;
 }while(1);
```

实现互斥，就是保证同一时刻最多只有一个进程处于临界区内，即实现对临界区的管理。需要满足如下三个正确性原则：

(1)互斥性原则：任意时刻至多只能有一个进程处于关于同一组共享变量的临界区之中。

(2)进展性原则：临界区空闲时，只有执行了临界区入口及出口部分代码的进程参与下一个进入临界区的决策，该决策不可无限期延迟。

(3)有限等待性原则：请求进入临界区的进程应该在有限的等待时间内获得进入临界区的机会。

15．【吉林大学　2016】互斥算法的三个正确性条件是什么？下面基于交换指令的硬件互斥算法是否满足上述条件？若不满足请改进。

```
1   int lock;//全局变量,初始值为 0
2   int key;//进程局部变量
3   do{
4     key=1;
5     do{
6           swap(&lock,&key);
7     }while(key==1);
8     临界区;
9     lock=0;
10     其余代码;
11   }while(1);
```

TSL指令把“上锁”和“检查”操作用硬件的方式变成了一气呵成的原子操作。

- 缺点：不满足“让权等待”原则，暂时无法进入临界区的进程会占用CPU并循环执行TSL指令，从而导致“忙等”。
- 改进：可以采用记录型信号量机制来解决进程同步问题。

谢谢大家

考点五:

管程机制

考点框架

- 管程的定义
- 管程的组成
- 管程的特性
- 管程和进程的对比
- 条件变量

考点五: 管程机制

管程的定义

- 在信号量机制中，每个要访问临界资源的进程都必须自备同步的PV操作，大量分散的同步操作会给系统管理带来麻烦，且容易因为同步操作不当而导致系统死锁。于是便产生了一种新的进程同步工具——**管程（Monitors）**。

考点五: 管程机制

管程的定义

- 代表共享资源的数据结构，以及由对该共享数据结构实施操作的一组过程所组成的资源管理程序，共同构成了一个操作系统的资源管理模块，我们称之为管程。

考点五: 管程机制

管程的组成

- 管程由四部分组成:

① 管程的名称;

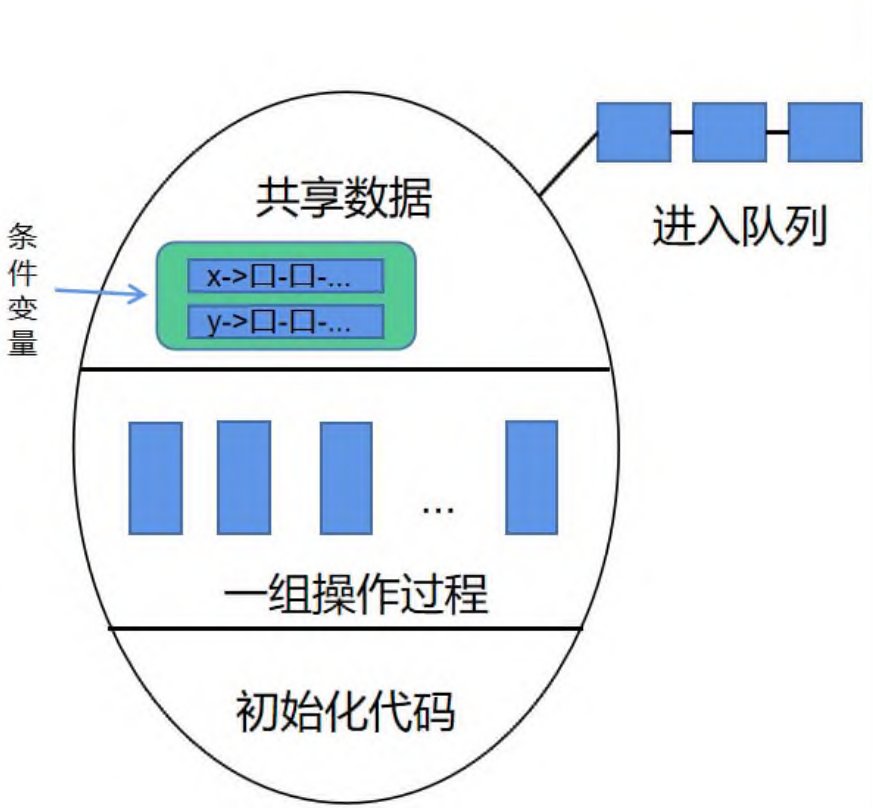
② 局部于管程内部的共享数据结构说明;

③ 对该数据结构进行操作的一组过程;

④ 对局部于管程内部的共享数据设置初始值的语句

考点五: 管程机制

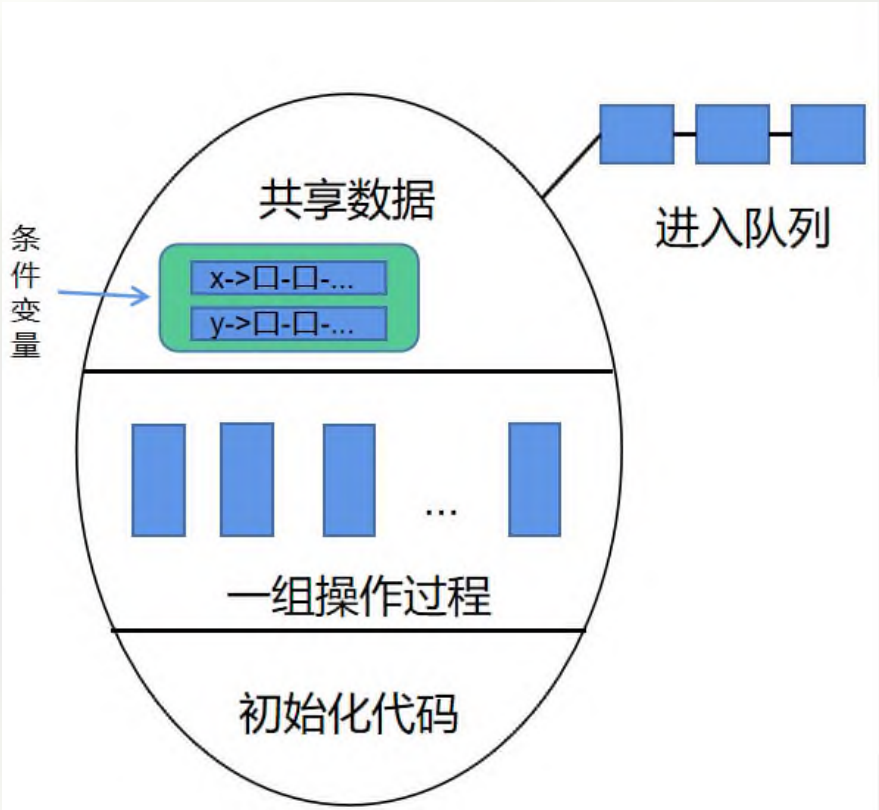
管程的组成



- 局部于管程内部的数据结构，仅能被局部于管程内部的过程所访问，任何管程外的过程都不能访问它；反之，局部于管程内部的过程也仅能访问管程内的数据结构。

考点五: 管程机制

管程的组成



- 管程相当于围墙，它把共享变量和对它进行操作的若干过程围了起来，所有进程要访问临界资源时，都必须经过管程(相当于通过围墙的门)才能进入，
- 管程每次只准许一个进程进入管程，从而实现了进程互斥。

考点五: 管程机制

管程的特性

管程是一种程序设计语言结构成分，它和信号量有同等的表达能力，从语言的角度看，管程主要有以下特性：

(1) 模块化。管程是一个基本程序单位，可以单独编译。

(2) 抽象数据类型。管程中不仅有数据，而且有对数据的操作。

(3) 信息掩蔽。管程中的数据结构只能被管程中的过程访问，这些过程也是在管程内部定义的，供管程外的进程调用，而管程中的数据结构以及过程(函数)的具体实现外部不可见。

考点五: 管程机制

管程和进程的对比

- 虽然二者都定义了数据结构，但进程定义的是私有数据结构PCB，管程定义的是公共数据结构，如消息队列等；
- 二者都存在对各自数据结构上的操作，但进程是由顺序程序执行有关的操作，而管程主要是进行同步操作和初始化操作；
- 设置进程的目的在于实现系统的并发性，而管程的设置则是解决共享资源的互斥使用问题；
- 进程通过调用管程中的过程对共享数据结构实行操作，该过程就如通常的子程序一样被调用，因而管程为被动工作方式，进程则为主动工作方式；
- 进程之间能并发执行，而管程则不能与其调用者并发；
- 进程具有动态性，由“创建”而诞生，由“撤销”而消亡，而管程则是操作系统中的一个资源管理模块，供进程调用。

考点五: 管程机制

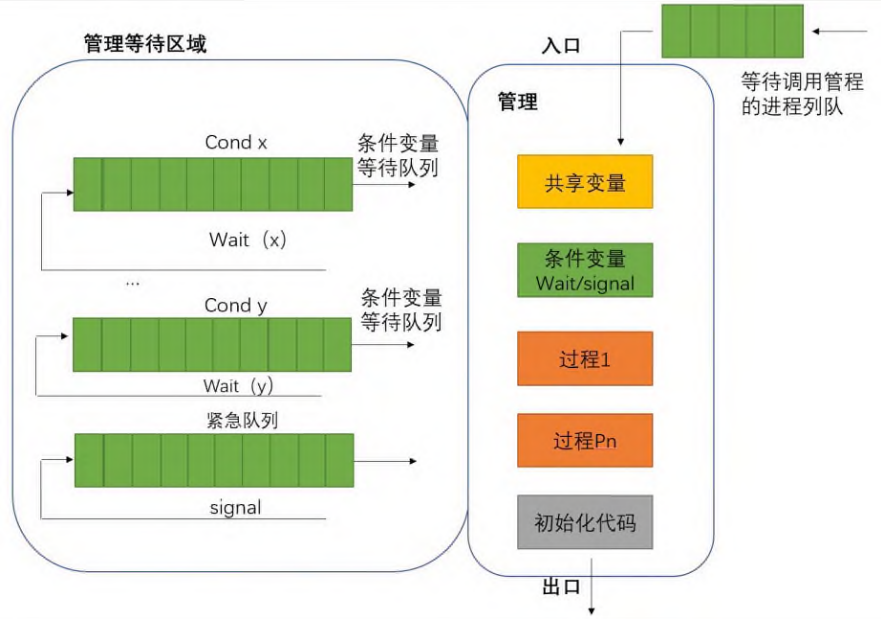
条件变量

- 当进程调用管程，在管程过程中被阻塞或挂起，若一直不释放管程，则其他进程无法进入，被迫长时间等待。
- 引入了条件变量（condition）。
- **一个进程被阻塞或挂起的条件(原因)可能多个，因此管程中设置多个条件变量，这些条件变量也只能在管程中访问。**

考点五: 管程机制

条件变量

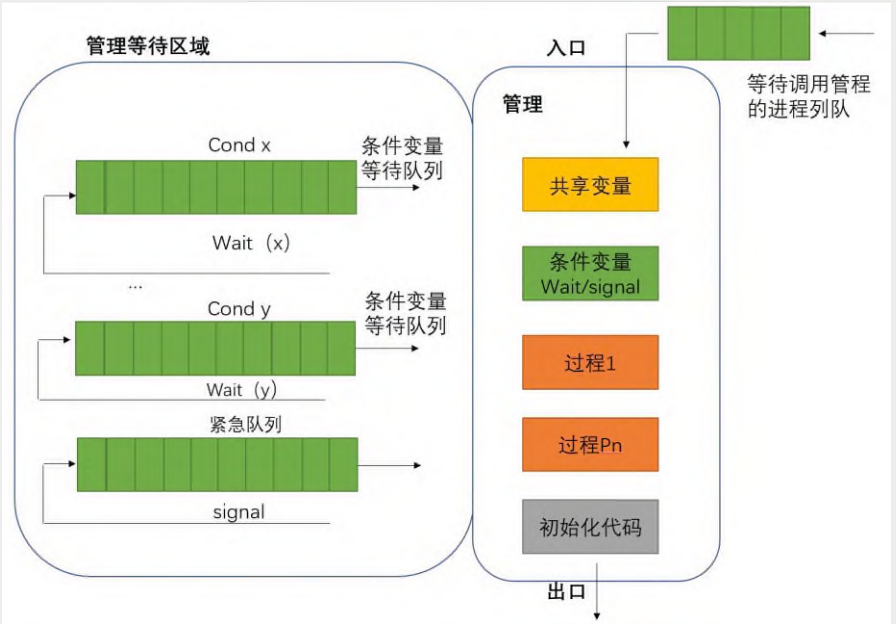
- 管程中每次只允许一个进程进入管程。当调用管程的进程因为某原因阻塞或者挂起时，把这个原因定义为一个条件变量
- 管程中对每个条件变量都须予以说明，其形式为：var x，y：condition。



考点五: 管程机制

条件变量

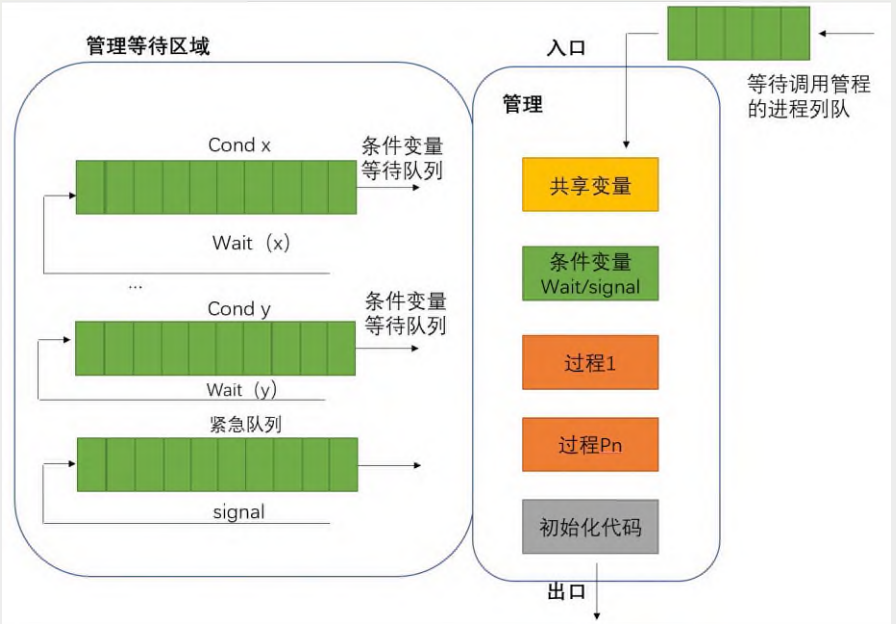
- 管程中对每个条件变量都须予以说明，其形式为：Var x，y：condition。
- 对条件变量的操作仅仅是 wait 和 signal，**因此条件变量也是一种抽象数据类型，每个条件变量保存了一个链表，用于记录因该条件变量而阻塞的所有进程，同时提供的两个操作即可表示为 x.wait和 x.signal。**



考点五: 管程机制

条件变量

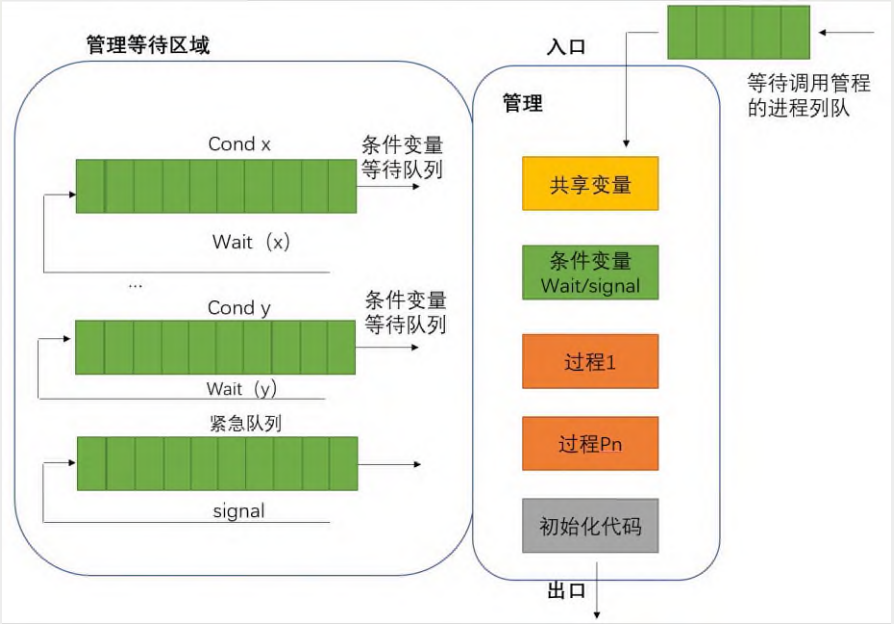
- x.wait：正在调用管程的进程因x条件需要被阻塞或挂起，则调用x.wait将自己插入到x条件的等待队列中，并释放管程，直至x条件发生变化。



考点五: 管程机制

条件变量

- x.signal：正在调用管程的进程因为x条件发生了变化（资源使用完，归还），则调用x.signal，重新启动一个因x条件而阻塞或挂起的进程，如果存在多个，则选择其中的一个，如果没有，继续执行原进程，不产生任何唤醒操作。



考点五: 管程机制

条件变量

- 应当指出，**x.signal**操作的作用，是**重新启动一个被阻塞的进程**，但如果没有被阻塞的进程，则x.signal操作不产生任何后果。
- 与信号量机制中的signal操作不同。因为，后者总是要执行s =s+1操作，因而总会改变信号量的状态。

【牛刀小试】

1．下列选项中不属于管程的组成部分的是(　　)。

A．局限于管程的共享数据结构

B．对管程内的数据结构进行操作的一组过程

C．管程外过程调用管程内数据结构的说明

D．对局限于管程的数据结构设置初始值的语句

C【**解析**】管程由管程的名称、局限于管程的共享数据结构、对管程内的数据结构进行操作的一组过程以及对局限于管程的数据结构设置初始值的语句组成。因此选择C。

【经典总结】管程由四部分组成：

(1)管程的名称；

(2)局部于管程内部的共享数据结构说明；

(3)对该数据结构进行操作的一组过程；

(4)对局部于管程内部的共享数据设置初始值的语句。

管程的引入是为了解决临界区分散所带来的管理和控制问题。在没有管程之前，对临界区的访问分散在各个进程之中，不易发现和纠正分散在用户程序中的不正确地使用P，V操作等问题。管程将这些分散在各进程中的临界区集中起来，并加以控制和管理，一次只允许一个进程进入管程内，既便于系统管理共享资源，又能保证互斥。

2．若x是管程内的条件变量，则当进程执行x.wait()时所做的工作是(　　)。

A．实现对变量x的互斥访问

B．唤醒一个在x上阻塞的进程

C．根据x的值判断该进程是否进入阻塞状态

D．阻塞该进程，并将之插入x的阻塞队列中

D【**解析**】“条件变量”是管程内部说明和使用的一种特殊变量，其作用类似于信号量机制中的“信号量”，都是用于实现进程同步的。需要注意的是，在同一时刻，管程中只能有一个进程在执行。

- 如果进程A执行了x.wait()操作，那么该进程会被阻塞，并挂到条件变量x对应的阻塞队列上，这样，管程的使用权会被释放，另一个进程就可以进入管程。
- 如果进程B执行了x.signal()操作，那么会唤醒条件变量x对应的阻塞队列队头进程。在Pascal语言的管程中，规定只有一个进程要离开管程时才能调用signal()操作。

谢谢大家

考点六:

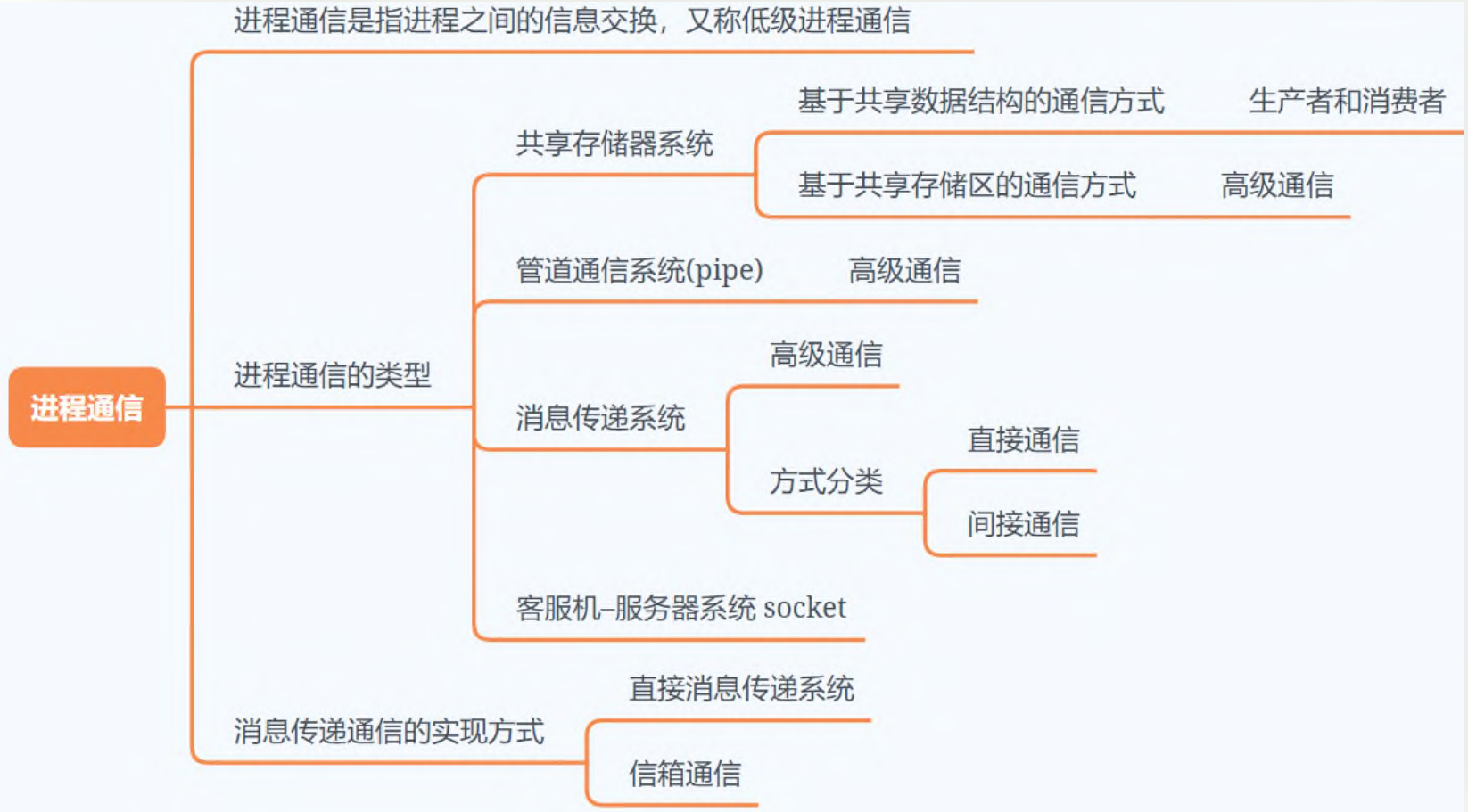
进程通信

考点框架

- 进程通信的概念
- 管道通信
- 消息传递系统
- 共享存储器系统

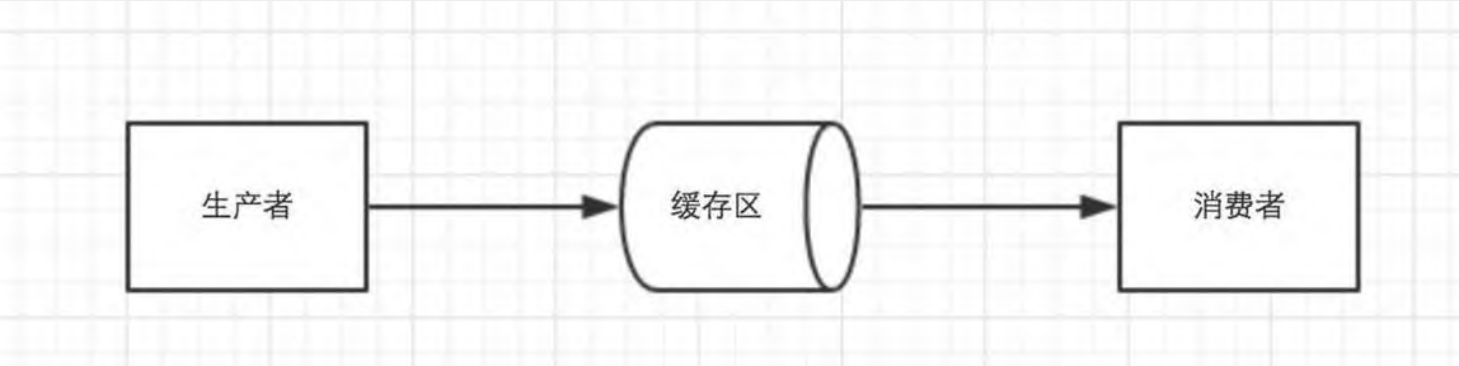
考点六: 进程通信

进程通信的概念



考点六: 进程通信

进程通信的概念



考点六: 进程通信

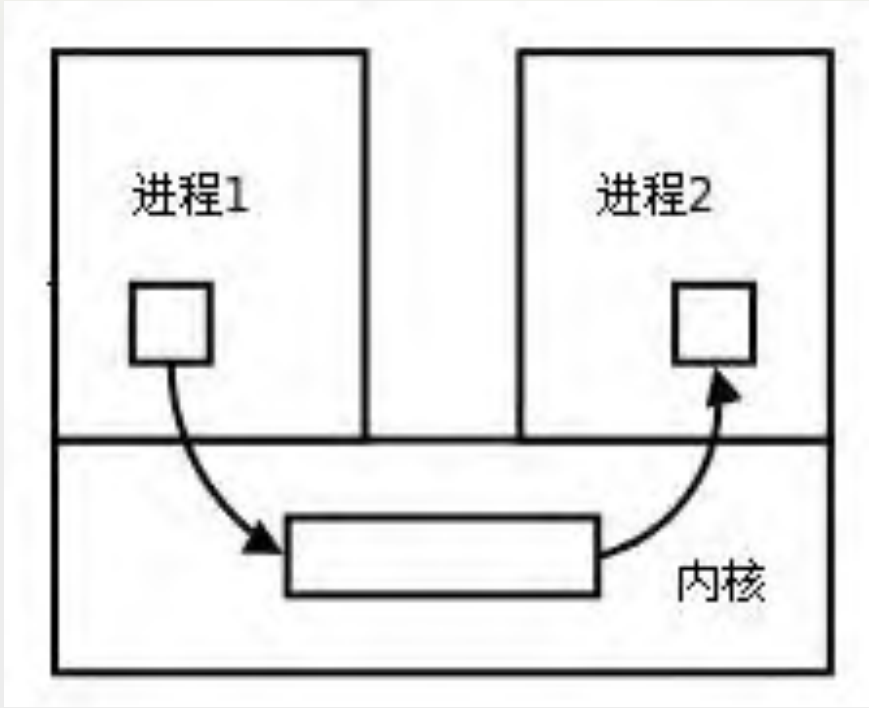
进程通信的概念

- 进程之间可能会存在特定的协同工作的场景，而协同就必须要进行进程间通信，进程通信（InterProcess Communication，IPC）就是指进程之间的信息交换，协同工作可能有以下场景。
- 数据传输：一个进程需要将它的数据发送给另一个进程
- 资源共享：多个进程之间共享同样的资源。
- 通知事件：一个进程需要向另一个或一组进程发送消息，通知它发生了某种事件。
- 进程控制：有些进程希望完全控制另一个进程的执行（如Debug进程），此时控制进程希望能够拦截另 一个进程的所有陷入和异常，并能够及时知道它的状态改变。

考点六: 进程通信

进程通信的概念

- 进程之间要交换数据必须通过内核，在内核中开辟一块缓冲区，进程1把数据从用户空间拷到内核缓冲区，进程2再从内核缓冲区把数据读走，
- 内核提供的这种机制称为**进程间通信（IPC，InterProcess Communication）**



考点六: 进程通信

进程通信的概念

低级通信:

- 信号量机制
- 管程

效率问题表现在下述两方面:

- 效率低，生产者每次只能向缓冲池投放一个产品(消息)，消费者每次只能从缓冲区中取得一个消息;
- 通信对用户不透明。

考点六: 进程通信

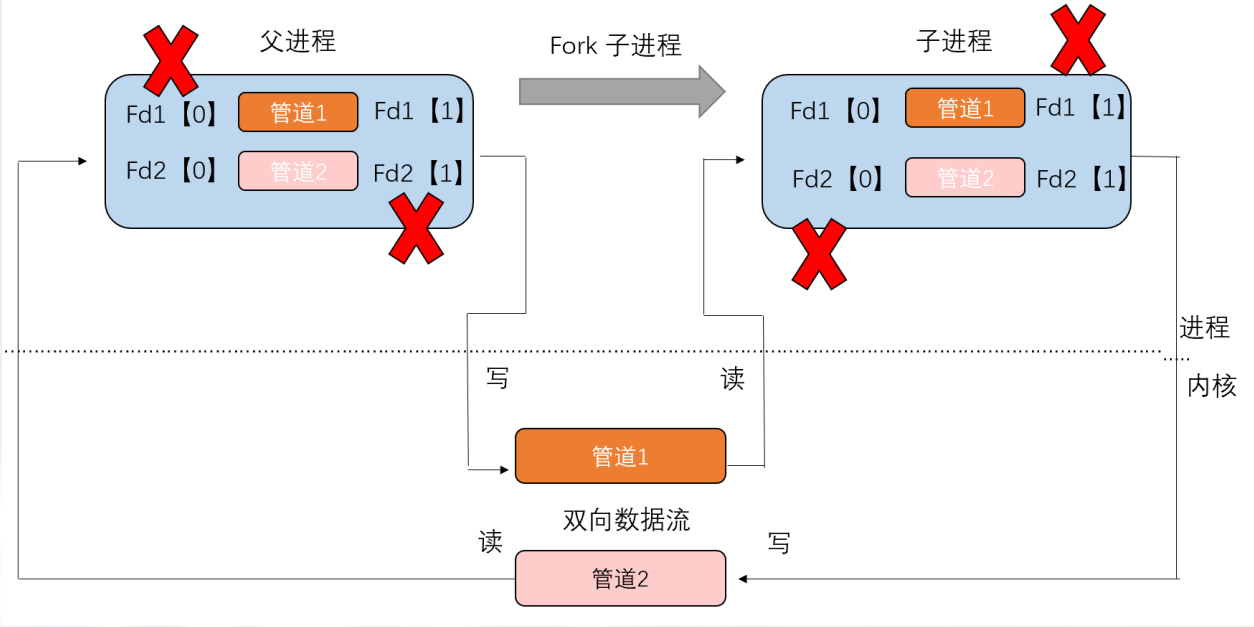
管道通信

- 所谓“管道”，是指用于连接一个读进程和一个写进程以实现他们之间通信的一个共享文件，又名pipe文件（但是请大家注意，管道文件不属于任何文件系统），管道都是半双工，单向的数据通信，也就说任意时刻只能有一个进程对管道进行操作。

考点六: 进程通信

管道通信

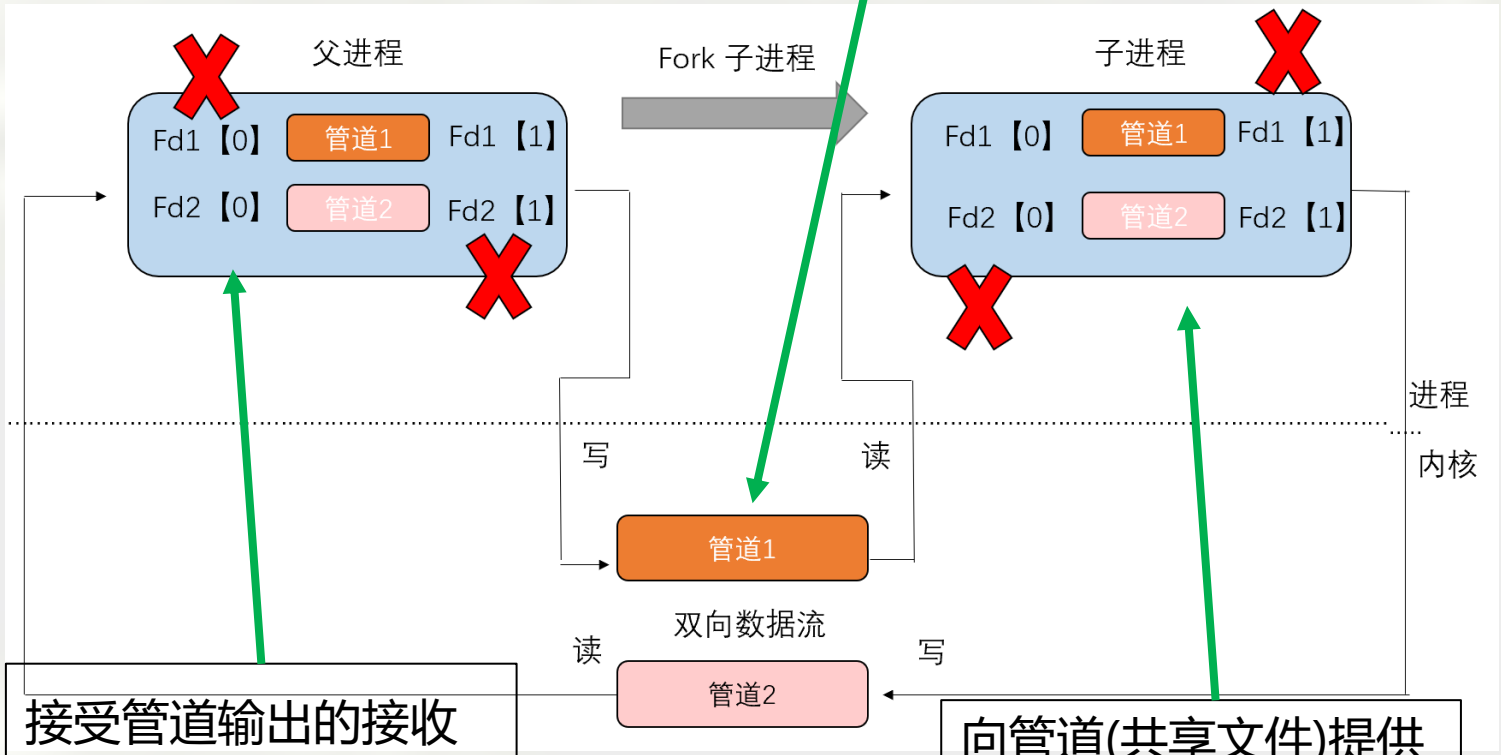
- 向管道(共享文件)提供输入的发送进程(即写进程)， 以字符流形式将大量的数据送入管道;
- 而接受管道输出的接收进程(即读进程)，则从管道中接收(读)数据。



考点六: 进程通信

管道通信

管道是指用于连接一个读进程和一个写进程以实现他们之间通信的一个共享文件，又名pipe文件。



接受管道输出的接收进程(即读进程)，则从管道中接收(读)数据。

向管道(共享文件)提供输入的发送进程(即写进程)， 以字符流形式将大量的数据送入管道;

考点六: 进程通信

管道通信

为了协调双方的通信，管道机制必须提供以下三方面的协调能力：

① 互斥，即当一个进程正在对pipe执行读/写操作时，其它(另一)进程必须等待。

② 同步，指当写(输入)进程把一定数量(如4 KB,管道是有参数定义的大小)的数据写入pipe，便去睡眠等待， 直到读(输出)进程取走数据后，再把他唤醒。当读进程读一空pipe时，也应睡眠等待，直至写进程将数据写入管道后，才将之唤醒。

③ 确定对方是否存在，只有确定了对方已存在时，才能进行通信。

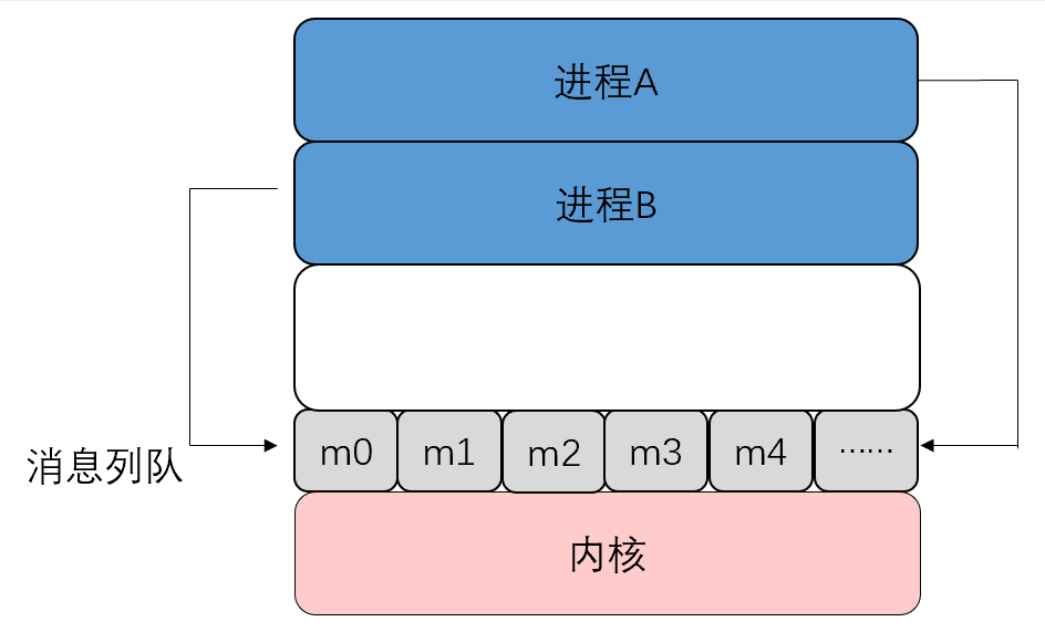
考点六: 进程通信

消息传递系统

在消息传递系统中，进程间的数据交换，是以格式化的消息(message)为单位的；

在计算机网络中，又把message称为报文。

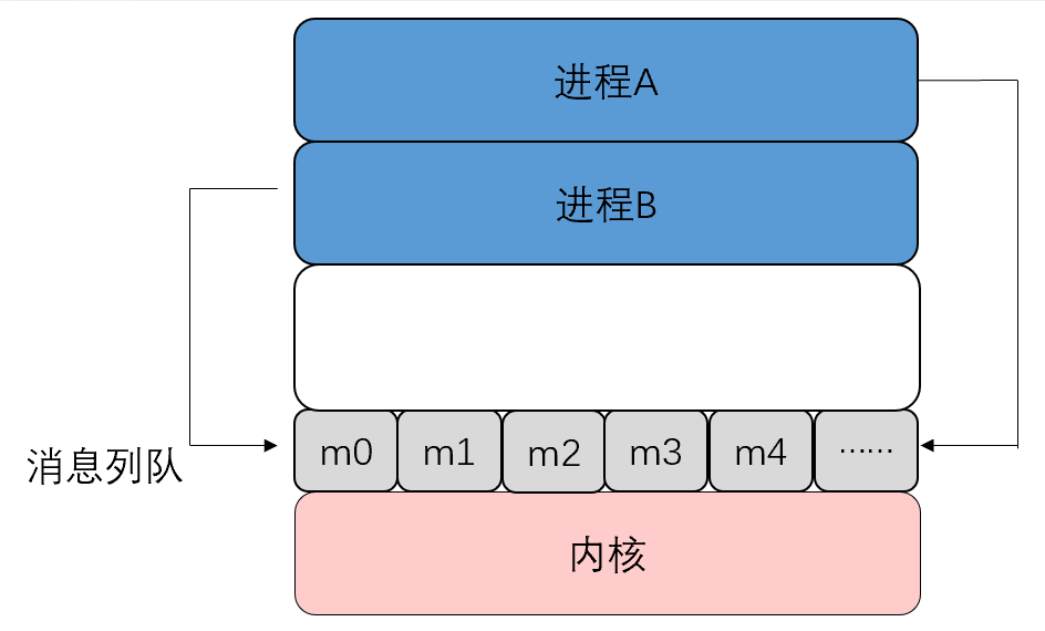
程序员直接利用系统提供的一组通信命令(原语)进行通信。



考点六: 进程通信

消息传递系统

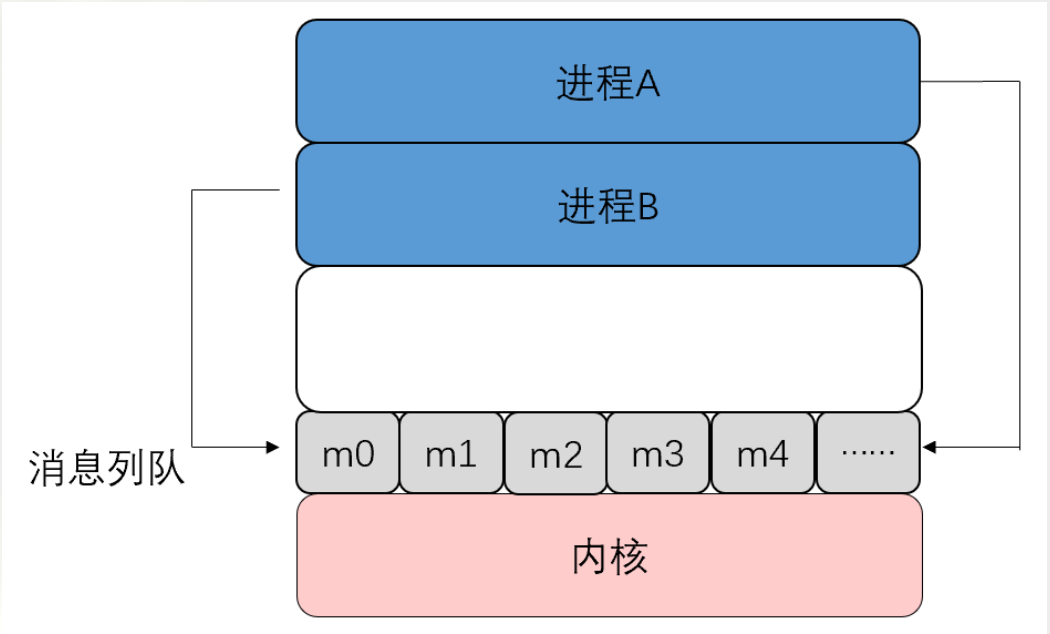
A 进程要给 B 进程发送消息，A 进程把数据放在对应的消息队列后就可以正常返回了，B 进程在需要的时候自行去消息队列中读取数据就可以了。同样的，B 进程要给 A 进程发送消息也是如此，如图所示。



考点六: 进程通信

消息传递系统

操作系统隐藏了通信的实现细节，大大减化了通信程序编制的复杂性，而获得广泛的应用。其实现方式的不同而进一步分成直接通信方式和间接通信方式两种。



考点六: 进程通信

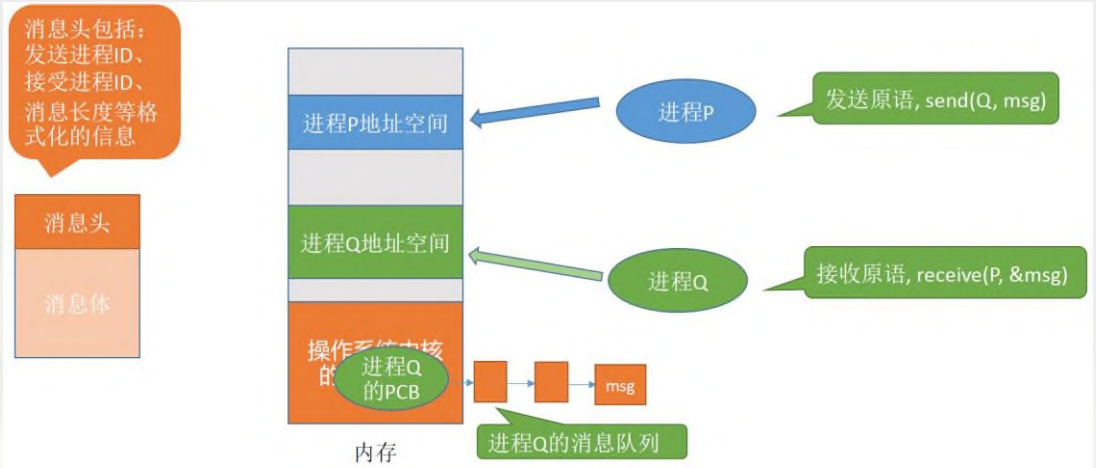
消息传递系统

- **直接通信方式:**这是指发送进程利用OS所提供的发送命令，直接把消息发送给目标进程。此时，要求发送进程和接收进程都以显式方式提供对方的标识符。通常，系统提供下述两条通信命令(原语):

  Send(Receiver, message); 发送一个消息给接收进程;

  Receive(Sender, message); 接收Sender发来的消息;

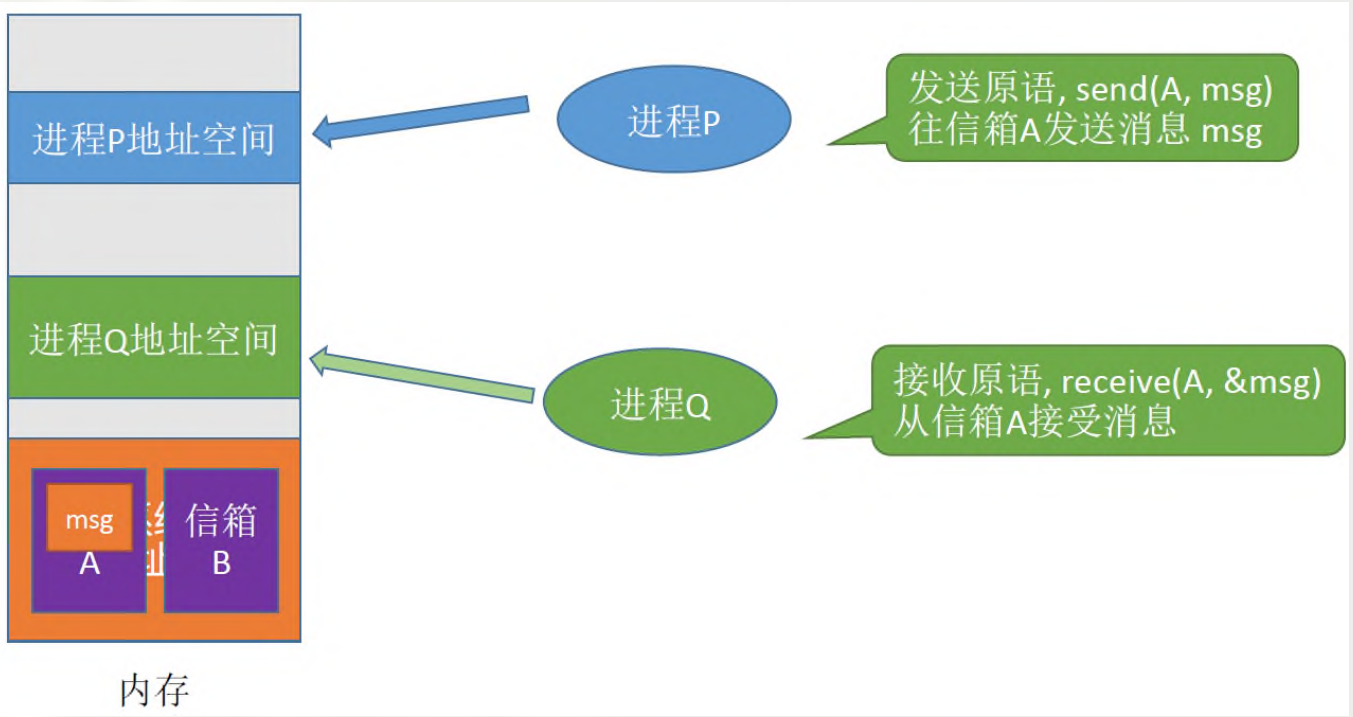
例如，原语Send(P_2, m_1)表示将消息m_1发送给接收进程P_2; 而原语Receive(P_1, m_1)则表示接收由P_1发来的消息m_1。



考点六: 进程通信

消息传递系统

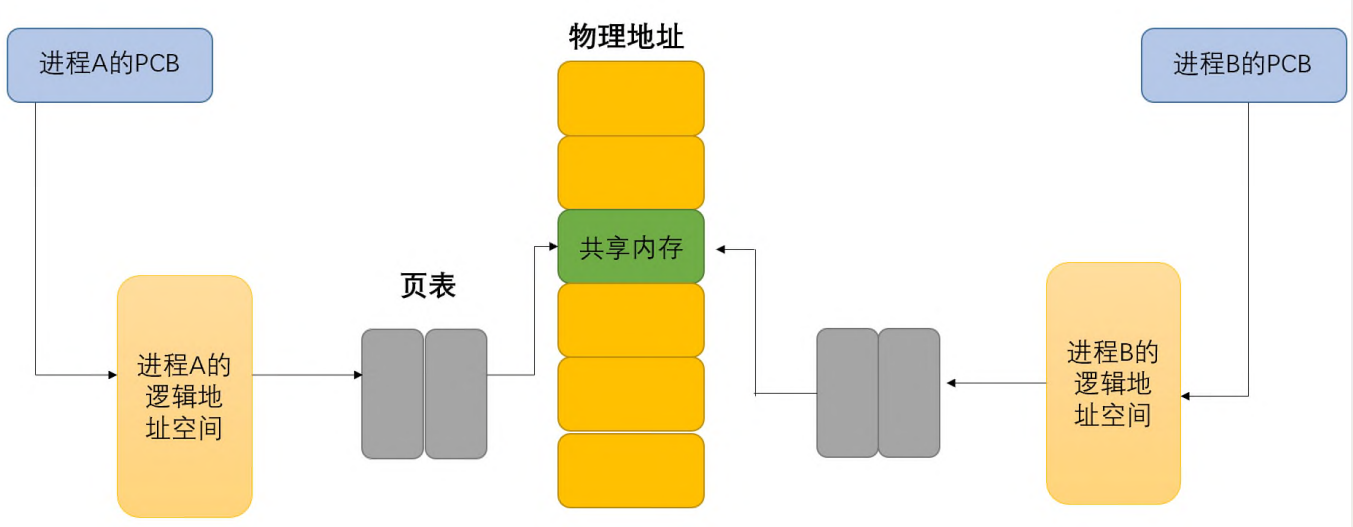
口 间接通信方式: 间接通信方式，以“信箱”作为中间实体进行消息传递。



考点六: 进程通信

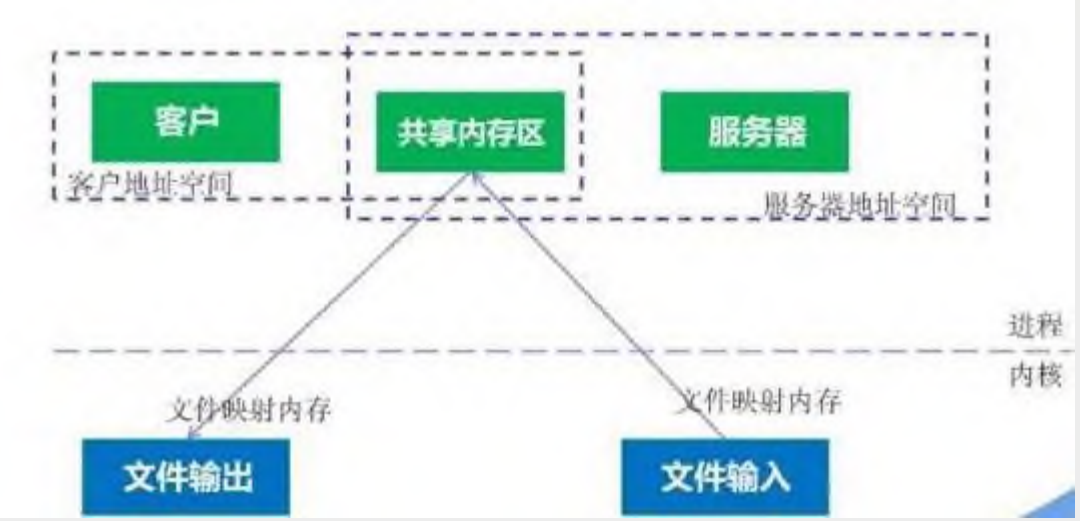
共享存储器系统

- 共享内存就是允许不相干的进程将同一段物理内存连接到它们各自的地址空间中，使得这些进程可以访问同一个物理内存，这个物理内存就成为共享内存。如果某个进程向共享内存写入数据，所做的改动将立即影响到可以访问同一段共享内存的任何其他进程。



考点六: 进程通信

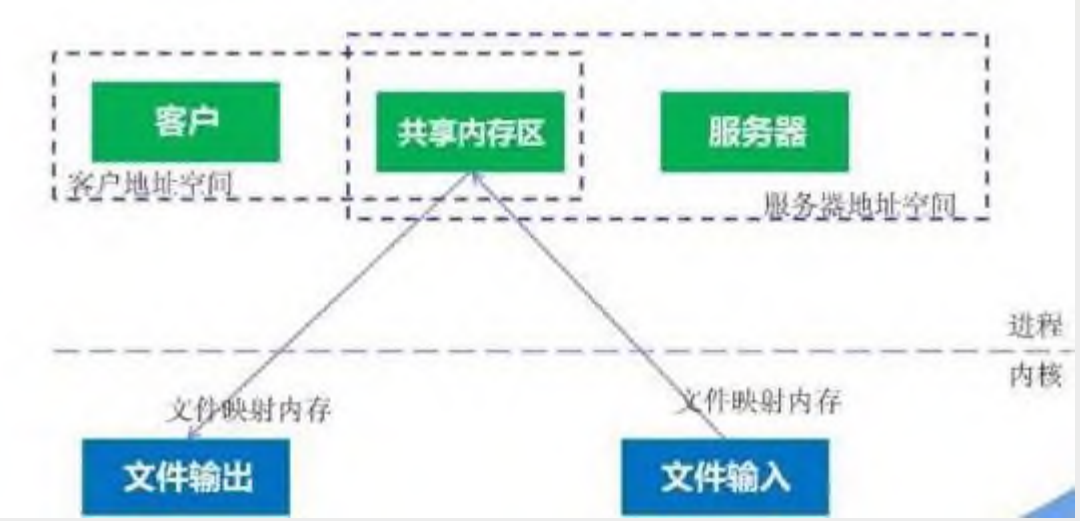
共享存储器系统



- 基于数据结构的共享：比如共享空间里只能放一个数组。这种共享方式速度慢、限制多，是一种低级通信方式

考点六: 进程通信

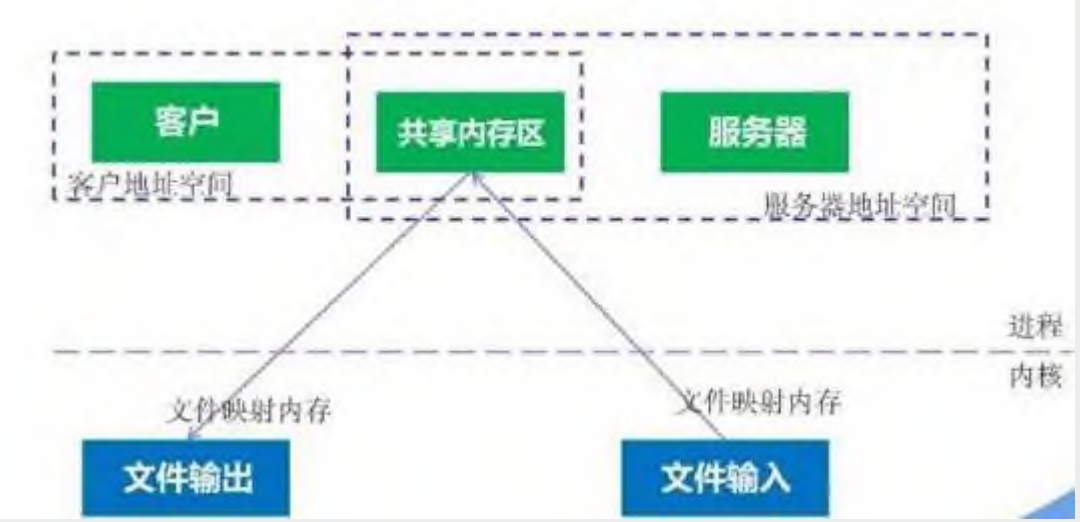
共享存储器系统



- 基于存储区的共享：操作系统在内存中划出一块共享存储区，数据的形式、存放位置都由通信进程控制，而不是操作系统。这种共享方式内存 速度很快，是一种高级通信方式

考点六: 进程通信

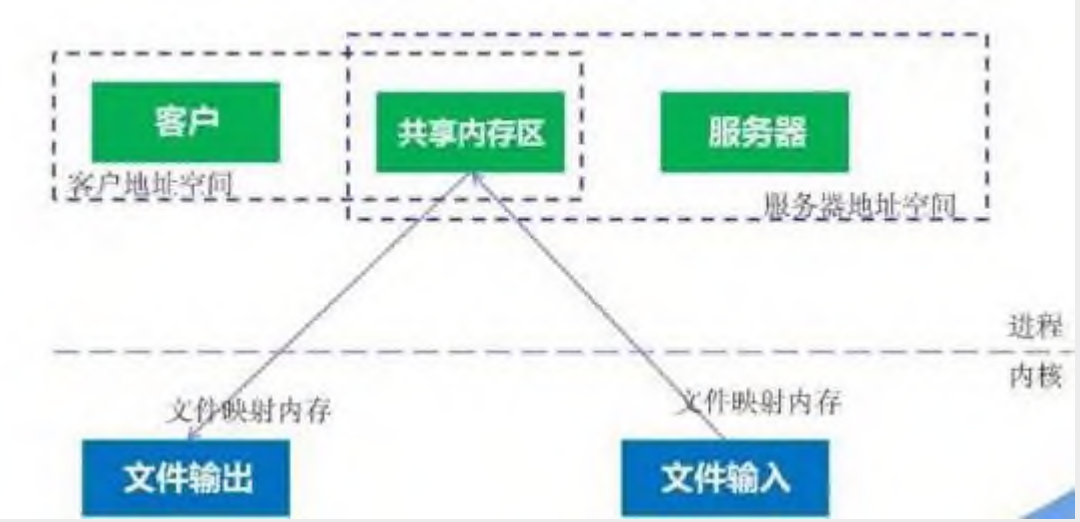
共享存储器系统



- 由于多个进程共享一段内存，因此需要依靠某种同步机制（如信号量）来达到进程间的同步及互斥。

考点六: 进程通信

共享存储器系统



- 不同于消息队列频繁的系统调用，对于共享内存机制来说，仅在建立共享内存区域时需要系统调用，一旦建立共享内存，所有的访问都可作为常规内存访问，无需借助内核。
- 数据就不需要在进程之间来回拷贝，所以这是最快的一种进程通信方式。

【牛刀小试】

1．进程之间交换数据不能通过( )途径进行。

A．共享文件　　B．消息传递

C．访问进程地址空间　　D．访问共享存储区

C【**解析**】进程代表运行中的程序，操作系统将资源分配给进程，进程是参加资源分配的主体。每个进程都包含独立的地址空间，只能执行自己地址空间中的程序，且只能访问自己地址空间中的数据，因此，进程之间不能直接交换数据，但可以利用操作系统提供的共享文件、消息传递、访问共享存储区等进行通信。因此选择C。

2．用信箱实现进程间互通信息的通信机制要有两个通信原语，它们是( )。

A．发送原语和执行原语　　B．就绪原语和执行原语

C．发送原语和接收原语　　D．就绪原语和接收原语

C【**解析**】用信箱实现进程间互通信息的通信机制要有两个通信原语，它们是发送原语和接收原语，因此选择C。

3．两个合作进程(Cooperating Processes)无法利用( )交换数据。

A．文件系统

B．共享内存

C．高级语言程序设计中的全局变量

D．消息传递系统

C【**解析**】不同的进程拥有不同的代码段和数据段，虽然是全局变量，但是在不同的进程中是不同的变量，没有任何联系，所以不能用于交换数据。因此选择C。

4．【中国科学院大学 2015】下面的叙述中，正确的是( )。

A．在一个进程中创建一个新线程比创建一个新进程所需的工作量多

B．同一进程中的线程间通信和不同进程中的线程间通信差不多

C．同一进程中的线程间切换由于许多上下文相同而简化

D．同一进程中的线程间通信需要调用内核

C【**解析**】线程的优点（与进程比较）：共享进程的代码、数据和资源；创建速度快，在一个已有进程中创建一个新线程比创建一个全新进程所需时间要少很多；终止所用的时间少，终止一个线程要比终止一个进程花费的时间少；切换时间少（保存和恢复工作量小），同一进程内线程间切换比进程间切换花费的时间少；提高了不同的执行程序间通信的效率，独立进程间的通信需要内核的介入，以提供保护和通信所需要的机制，但是，由于在同一个进程中的线程共享内存和文件，因此它们无需调用内核就可以互相通信。进程在执行过程中拥有独立的内存单元，所以多进程的程序更加健壮；多个线程共享内存，从而能极大地提高程序的运行效率。因此选择C。

5．下列关于线程和进程的叙述中，正确的是(　)。

Ⅰ．线程包含CPU现场，可以独立执行程序

Ⅱ．每个线程都有自己独立的地址空间

Ⅲ．线程之间的通信必须使用系统调用函数

Ⅳ．线程切换都需要内核的支持

Ⅴ.线程是资源分配的单位，进程是调度和分配的单位

Ⅵ．不管系统中是否有线程，进程都是拥有资源的独立单位

A．Ⅰ、Ⅱ、Ⅳ、Ⅴ　　B．Ⅰ、Ⅵ

C．Ⅱ、Ⅳ　　D．Ⅲ、Ⅵ

B【**解析**】线程包含CPU现场，可以独立执行程序；线程可以直接交换数据；用户级线程的切换不一定需要内核的支持；进程是资源分配的单位，线程是调度和分配的单位；不管系统中是否有线程，进程都是拥有资源的独立单位。因此选择B。

6．【华东师范大学 2014】以下对于线程的描述哪个是错误的？（ ）

A一个用户态线程的阻塞(block)会引起它所属的整个进程（包括其中的其他线程）的阻塞

B．同一个进程的不同线程可以共享地址空间中的堆

C．线程间通信必须通过内核态系统调用进行

D．同一个进程的不同线程必须维护各自的调用栈和CPU状态

C【**解析**】若操作系统不支持多线程机制，则会出现用户级线程阻塞而导致整个进程阻塞的情况。因为操作系统内核无法感知用户级线程的存在，系统内核管理的是核心线程，只能为核心线程分配CPU，若核心线程不支持多线程机制，核心线程仅能感知一个用户进程（而不是我们以为的用户级线程）的存在，内核无法将多个用户线程调度到多个CPU上运行，因而阻塞线程相当于阻塞进程，A正确。

6．【华东师范大学 2014】以下对于线程的描述哪个是错误的？（ ）

A一个用户态线程的阻塞(block)会引起它所属的整个进程（包括其中的其他线程）的阻塞

B．同一个进程的不同线程可以共享地址空间中的堆

C．线程间通信必须通过内核态系统调用进行

D．同一个进程的不同线程必须维护各自的调用栈和CPU状态

C【**解析**】线程共享的环境：进程代码段、进程的公有数据（利用这些共享的数据，线程很容易实现相互之间的通信）、进程打开的文件描述符、信号的处理机、进程的当前目录和进程用户ID与进程组ID，B正确。

6．【华东师范大学 2014】以下对于线程的描述哪个是错误的？（ ）

A一个用户态线程的阻塞(block)会引起它所属的整个进程（包括其中的其他线程）的阻塞

B．同一个进程的不同线程可以共享地址空间中的堆

C．线程间通信必须通过内核态系统调用进行

D．同一个进程的不同线程必须维护各自的调用栈和CPU状态

C【**解析**】线程间通信：由于多线程共享地址空间和数据空间，因此多个线程间的通信是一个线程的数据可以直接提供给其他线程使用，而不必通过操作系统（也就是内核的调度），C错误。

6．【华东师范大学 2014】以下对于线程的描述哪个是错误的？（ ）

A一个用户态线程的阻塞(block)会引起它所属的整个进程（包括其中的其他线程）的阻塞

B．同一个进程的不同线程可以共享地址空间中的堆

C．线程间通信必须通过内核态系统调用进行

D．同一个进程的不同线程必须维护各自的调用栈和CPU状态

C【**解析**】同一进程中的多个线程将共享该进程中的全部系统资源，如虚拟地址空间、文件描述符和信号处理，等等。但同一进程中的多个线程有各自的调用栈、自己的寄存器环境、自己的线程本地存储，D正确。因此选择C。

谢谢大家

考点七:

经典同步问题

考点框架

- 经典同步问题概述
- 生产者-消费者问题
- 哲学家进餐问题
- 读者-写者问题
- 营业员-顾客问题

考点七: 经典同步问题

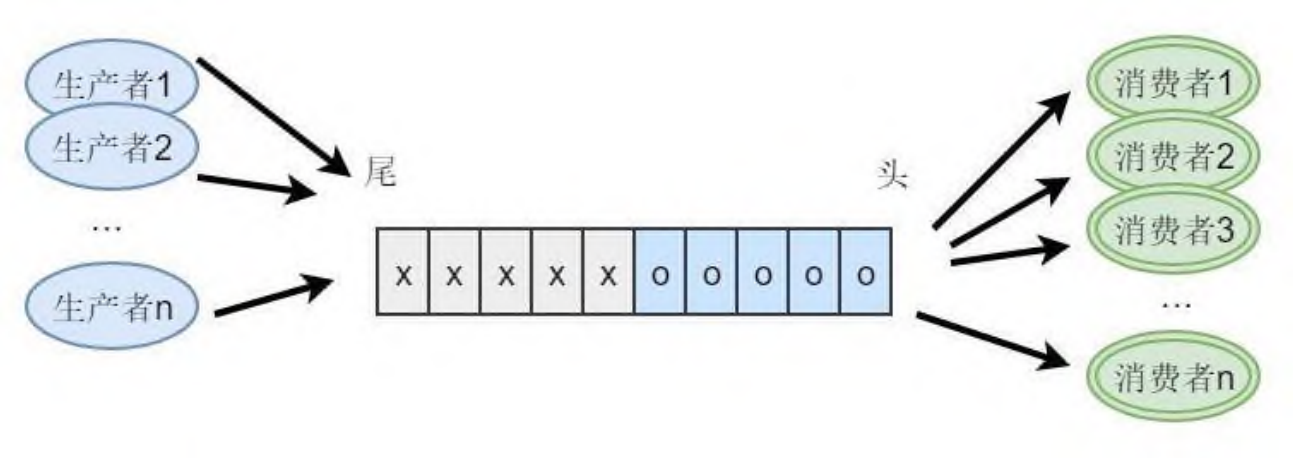
经典同步问题概述

	进程特征	互斥	合作
生产者-消费者问题	相同	√	√
哲学家进餐问题	相同	√	X
读者-写者问题	不同	√	X
营业员-顾客问题	不同	X	√

考点七: 经典同步问题

生产者-消费者问题

- 生产者—消费者问题(The producer-consumer problem)做了一些描述，但未考虑进程的互斥与同步问题，因而造成了数据Counter的不定性。由于生产者—消费者问题是相互合作的进程关系的一种抽象，例如， 在输入时，输入进程是生产者，计算进程是消费者；而在输出时，则计算进程是生产者，而打印进程是消费者。

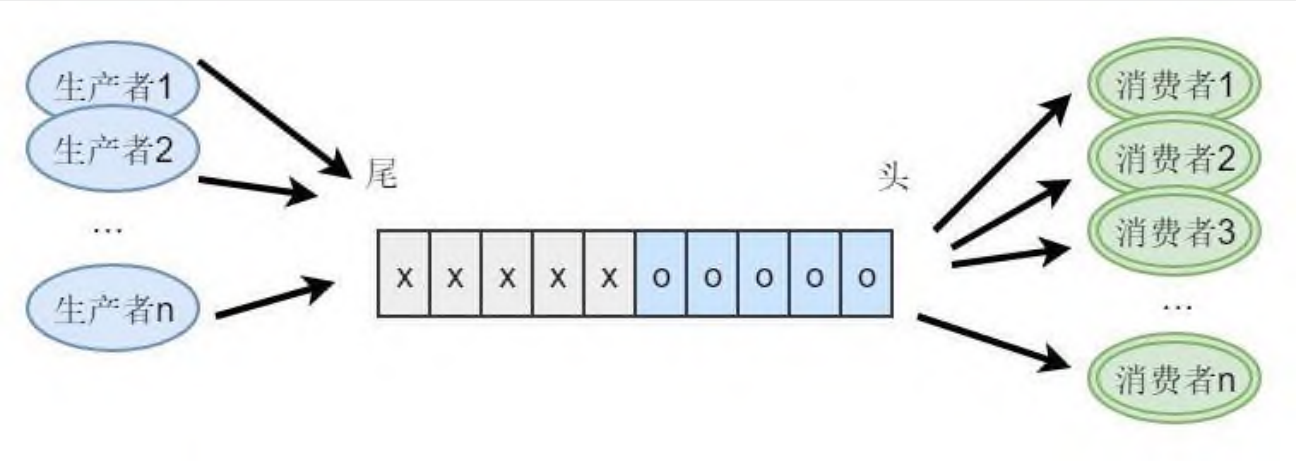


考点七: 经典同步问题

生产者-消费者问题

- 假定在生产者和消费者之间的公用缓冲池中，具有n个缓冲区

- 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。



考点七: 经典同步问题

生产者-消费者问题

- 假定在生产者和消费者之间的公用缓冲池中，具有n个缓冲区，这时可利用互斥信号量mutex实现诸进程对缓冲池的互斥使用(只有一个进程可以使用)；
- 利用信号量empty和full分别表示缓冲池中空缓冲区和满缓冲区的数量。又假定这些生产者和消费者相互等效，只要缓冲池未满，生产者便可将消息送入缓冲池；只要缓冲池未空，消费者便可从缓冲池中取走一个消息。

考点七: 经典同步问题

生产者-消费者问题

```
var mutex, empty, full: semaphore : =1,n,0;
producer:
    begin
        repeat
        producer an item nextp;
        P(empty);
        P(mutex);
        buffer(in) : = nextp;   //放物品
        in : = (in+1) mod n;    //指针+1
        V(mutex);
        V(full);
        until false;
    end
```

考点七: 经典同步问题

生产者-消费者问题

```
var mutex, empty, full: semaphore : =1,n,0;
producer:
    begin
            repeat
            producer an item nextp;
            P(empty);
            P(mutex);
            buffer(in) : = nextp;   //放物品
             in : = (in+1) mod n;    //指针+1
            V(mutex);
            V(full);
            until false;
    end
```

考点七: 经典同步问题

生产者-消费者问题

```
var mutex, empty, full: semaphore : =1,n,0;
consumer:
   begin
        repeat
         P(full);
         P(mutex);
         nextc :   =buffer(out);  //取出物品
         out :    =(out+1) mod n;    //指针+1
         V(mutex);
         V(empty);
         consumer the item in nextc;
        until false;
end
```

在每个程序中用于实现互斥的wait(mutex)和signal(mutex)必须成对地出现；

考点七: 经典同步问题

生产者-消费者问题

```
var mutex, empty, full: semaphore : =1,n,0;
consumer:
   begin
        repeat
         P(full);
         P(mutex);
         nextc :   =buffer(out);  //取出物品
         out :    =(out+1) mod n;    //指针+1
         V(mutex);
         V(empty);
         consumer the item in nextc;
        until false;
end
```

对资源信号量empty和full的wait和signal操作，同样需要成对地出现，但它们分别处于不同的程序中。

考点七: 经典同步问题

生产者-消费者问题

```
var mutex, empty, full: semaphore : =1,n,0;
consumer:
   begin
        repeat
         P(full);
         P(mutex);
         nextc :   =buffer(out);  //取出物品
         out :    =(out+1) mod n;    //指针+1
         V(mutex);
         V(empty);
         consumer the item in nextc;
        until false;
end
```

在每个程序中的多个wait操作顺序不能颠倒。应先执行对资源信号量的wait操作，然后再执行对互斥信号量的wait操作，否则可能引起进程死锁。

考点七: 经典同步问题

生产者-消费者问题

```
producer {
    while( true ) {
        // 等待缓冲区有空闲位置， 在使用PV操作时，条件变量需要在互斥锁之前
        P( empty );
        P( mutex );                    // 保证在product时不会有其他线程访问缓冲区
        // product
        buf.push( item, in );     // 将新资源放到buf[in]位置
        in = ( in + 1 ) % N;
        V(mutex );              // 唤醒的顺序可以不同
        V( full );              // 通知consumer缓冲区有资源可以取走
    }
}
```

考点七: 经典同步问题

生产者-消费者问题

```
consumer {
    while( true ) {
        P( full );                  // 等待缓冲区有资源可以使用
        P( mutex );                 // 保证在consume时不会有其他线程访问缓冲区
        // consume
        buf.pop( out );             // 将buf[out]位置的的资源取走
        out = ( out + 1 ) % N;
        V( mutex );                 // 唤醒的顺序可以不同
        V( empty );                 // 通知缓冲区有空闲位置
    }
}
```

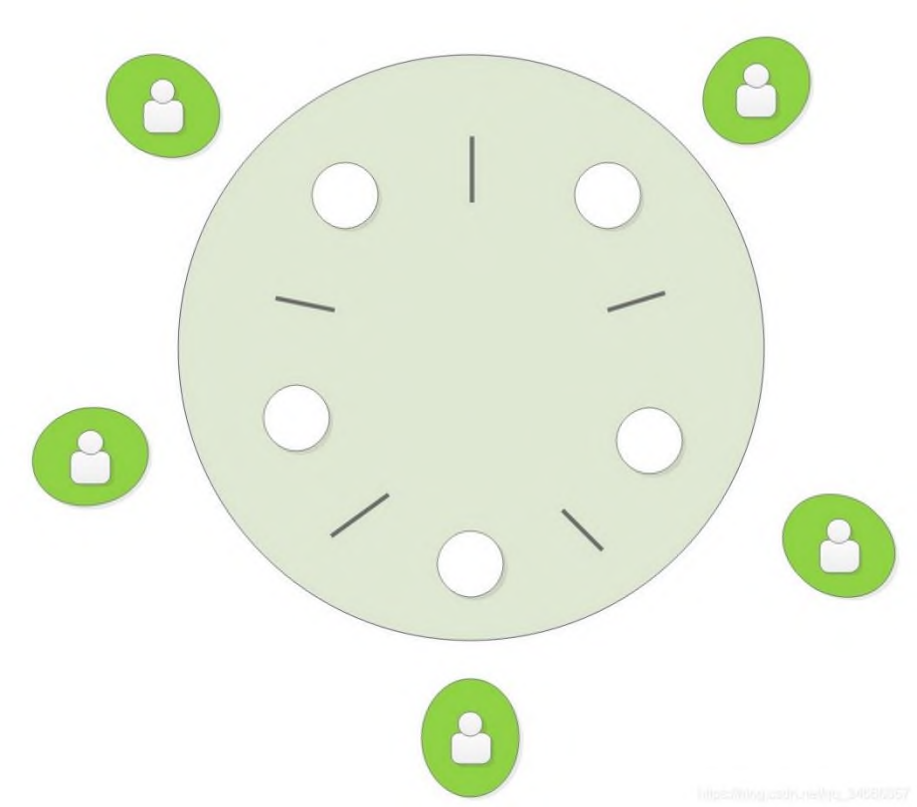
考点七: 经典同步问题

哲学家进餐问题

有五个哲学家围在一张圆桌，分别坐在周围的五张椅子上，在圆桌上有五个碗和五支筷子，他们的生活方式是交替的进行思考和进餐。平时，一个哲学家进行思考，饥饿时便试图取用其左右最靠近他的筷子，只有在他拿到两支筷子时才能进餐。进餐完毕后，放下筷子继续思考。

考点七: 经典同步问题

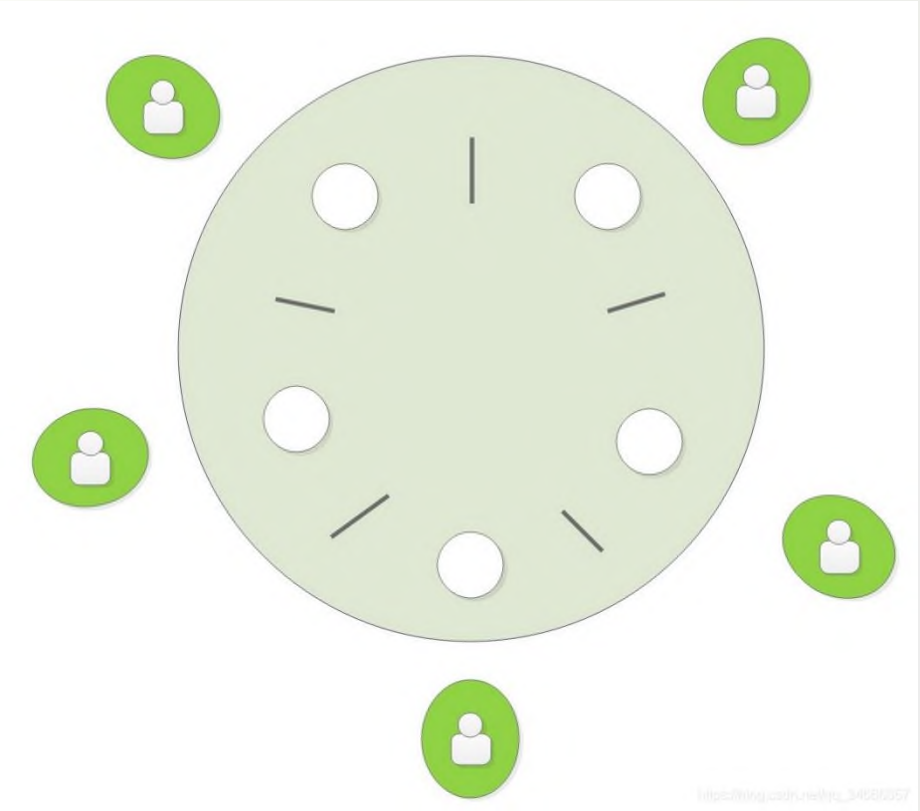
哲学家进餐问题



- 我们可以从上面的题目中得出，筷子是临界资源，同一根筷子同一时刻只能有一个哲学家可以拿到。

考点七: 经典同步问题

哲学家进餐问题



- 由问题描述我们可以知道，一共有五个哲学家，也就是五个进程；五支筷子，也就是五个临界资源；
- 哲学家想要进餐，必须要同时获得左边和右边的筷子，这就是要同时进入两个临界区（使用临界资源），才可以进餐。
- semaphore mutex[5] = {1,1,1,1,1}; //初始化信号量

考点七: 经典同步问题

哲学家进餐问题

```
semaphore mutex[5] = {1,1,1,1,1};                    //初始化信号量
void philosopher(int i){
  while(true){
     //thinking                    //思考
     P(mutex[i]);                  //判断缓冲池中是否仍有空闲的缓冲区
     P(mutex[(i+1)%5]);     //判断是否可以进入临界区（操作缓冲池）
     //eat                         //进餐
     V(mutex[i]);                 //退出临界区，允许别的进程操作缓冲池
     V(mutex[(i+1)%5]);   //缓冲池中非空的缓冲区数量加1，可以唤醒等待的消费者进程
  }
}
```

考点七: 经典同步问题

哲学家进餐问题

可采取以下几种解决方法:

(1) 至多只允许有四位哲学家同时去拿左边的筷子，最终能保证至少有一位哲学家能够进餐，并在用毕时能释放出他用过的两只筷子，从而使更多的哲学家能够进餐。

(2) 仅当哲学家的左、右两只筷子均可用时，才允许他拿起筷子进餐。

考点七: 经典同步问题

哲学家进餐问题

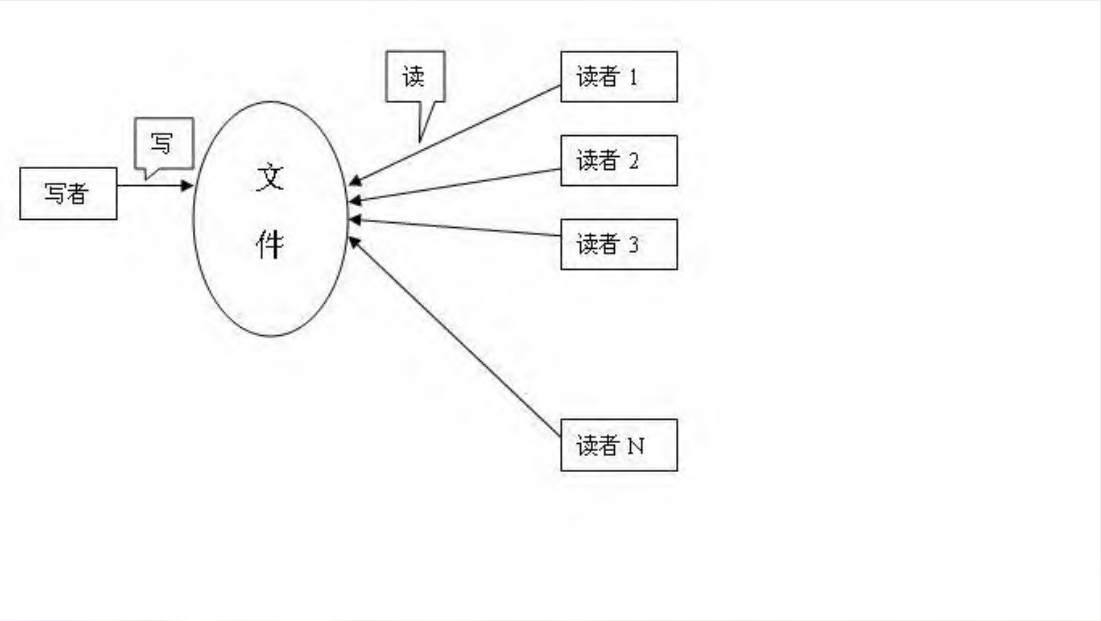
可采取以下几种解决方法：

(3) 规定奇数号哲学家先拿他左边的筷子，然后再去拿右边的筷子；而偶数号哲学家则相反。按此规定，将是1、 2号哲学家竞争1号筷子；3、4号哲学家竞争3号筷子。即五位哲学家都先竞争奇数号筷子，获得后，再去竞争偶数号筷子，最后总会有一位哲学家能获得两只筷子而进餐。

考点七: 经典同步问题

读者-写者问题

- 一个数据文件或记录可被多个进程共享。
- 只要求读文件的进程称为“Reader进程”，其它进程则称为“Writer进程”。
- 允许多个进程同时读



考点七: 经典同步问题

读者-写者问题

“读者--写者问题”是保证一个Writer进程必须与其他进程互斥地访问共享对象的同步问题。

	读者	写者
读者	共享	互斥
写者	互斥	互斥

考点七: 经典同步问题

读者-写者问题

- 互斥信号量wmutex: 实现Reader与Writer进程间在读或写时的互斥，
- 整型变量Readcount: 表示正在读的进程数目;
- 由于只要有一个Reader进程在读，便不允许Writer进程写。所以，仅当Readcount=0，即无Reader进程在读时，Reader才需要执行P(wmutex)操作。若Wait(wmutex)操作成功，Reader进程便可去读，相应地，做Readcount+1操作。
- 仅当Reader进程在执行了Readcount减1操作后其值为0时，才需执行V(wmutex)操作，以便让Write进程写
- 互斥信号量rmutex: Reader进程间互斥访问Readcount

考点七: 经典同步问题

读者-写者问题

```
semaphore  wmutex = 1;
semaphore  rmutex = 1;
int  Readcount = 0;
```

考点七: 经典同步问题

```
process Reader(){
    while(true){
        P(rmutex);
        if Readcount=0 then P(wmutex);
        Readcount = Readcount +1;
        V(rmutex);
            读;
        P(rmutex);
        Readcount = Readcount -1;
        if Readcount=0 then V(wmutex);
        V(rmutex);
    }
}
```

考点七: 经典同步问题

```
process Writer(){
    While(true){
        P(wmutex);
        写;
        V (wmutex);
    }
}
```

考点七: 经典同步问题

- 读者优先的设计思想是读进程只要看到有其它读进程正在读，就可以继续进行读；写进程必须等待所有读进程都不读时才能写，即使写进程可能比一些读进程更早提出申请。
- 该算法只要还有一个读者在活动，就允许后续的读者进来，该策略的结果是，如果有一个稳定的读者流存在，那么这些读者将在到达后被允许进入。
- 而写者就始终被挂起，直到没有读者为止。

考点七: 经典同步问题

营业员-顾客问题

假设有一个理发店只有一个理发师，一张理发时坐的椅子，若干张普通椅子顾客供等候时坐。

- 没有顾客时，理发师就坐在理发的椅子上睡觉。
- 顾客一到，他不是叫醒理发师，就是离开。
- 如果理发师没有睡觉，而在为别人理发，他就会坐下来等候。
- 如果所有的椅子都坐满了人，最后来的顾客就会离开。

考点七: 经典同步问题

营业员-顾客问题

（1）设置理发师的资源信号量为barber，初值为初始状态可用的资源数，故设barber初值为0（因为没有顾客的时候理发师在睡觉呀）。

（2）设置顾客的资源信号量为customers，初值为0（刚开始没有顾客来）。

（3）用互斥信号量mutex实现进程互斥。

（4）用变量waiting来记录等待的顾客数，判断有没有空闲椅子。

考点七: 经典同步问题

营业员-顾客问题

```
semaphore barber,customers,mutex;
barber = customers = 0,mutex = 1;
int waiting = 0;
process barber{
    while(true){
      P(customers);//barber在等顾客喊他，没有顾客就睡觉
      P(mutex);//只能被一个顾客叫醒
      waiting=waiting-1;   //有顾客将得到服务，有等待区的椅子空出来
      V(barber);//一个理发师资源被释放
       V(mutex);
      cut_hair();//理发师在工作
    }
}
```

考点七: 经典同步问题

营业员-顾客问题

```
process customers{
        P(mutex);     //一次只能有一个顾客进行以下操作，即访问椅子
        if(waiting<n){    //来了个顾客在门口往店里看了看，如果有空闲的椅子
            waiting=waiting+1;     //进店坐下了
            V(customers);            //顾客跟理发师打招呼说“我来了”
            V(mutex);//访问椅子结束
            P(barber);//等待理发师
            get_haircut();//得到服务
          } else {//看了看发现店里没有位置
            V(mutex);//走了
        }
}
```

考点七: 经典同步问题

营业员-顾客问题

假设一个系统中有三个抽烟者进程，每个抽烟者不断地卷烟并抽烟。抽烟者卷起并抽掉一颗烟需要有三种材料：烟草、纸和胶水。

- 一个抽烟者有烟草，
- 一个有纸，
- 另一个有胶水。

系统中还有两个供应者进程，它们无限地供应所有三种材料，但每次仅轮流提供三种材料中的两种。得到缺失的两种材料的抽烟者在卷起并抽掉一颗烟后会发信号通知供应者，让它继续提供另外的两种材料。这一过程重复进行。

考点七: 经典同步问题

营业员-顾客问题

```
int  random;
Semaphore  offer1 = 0;  定义烟草和纸的组合信号量
Semaphore offer2 = 0;  定义烟草和胶水的组合信号量
Semaphore  offer3 = 0;  定义胶水和纸的组合信号量
Semaphore finish = 0;  定义抽烟是否完成的信号量
Process P1(){
   while(1){
      random = random();     random = random % 3;
     if( random == 0 )
         v(offer1);
      else if(random == 1)
         v(offer2);
       else   v(offer3);
      p(finish)
  }
}
```

考点七: 经典同步问题

营业员-顾客问题

```
Process P2(){
   while(1){
      p(offer3);
      拿纸和胶水，卷成烟
      v(finish)
    }
}
```

```
Process P3(){
   while(1){
      p(offer2);
      拿烟草和胶水，卷成烟
      v(finish)
    }
}
```

```
Process P4(){
   while(1){
      p(offer1);
      拿纸和烟草，卷成烟
      v(finish)
    }
}
```

考点七: 经典同步问题

	进程特征	互斥	合作
生产者-消费者问题	相同	√	√
哲学家进餐问题	相同	√	X
读者-写者问题	不同	√	X
营业员-顾客问题	不同	X	√

谢谢大家

【真题实战】

1、下列选项中，降低进程优先级的合理时机是_____。

A. 进程的时间片用完

B. 进程刚完成 I/O，进入就绪列队

C. 进程长期处于就绪列队中

D. 进程从就绪态转为运行态

A【解析】进程刚完成 I/O，进入就绪列队、进程长期处于就绪列队中、进程从就绪态转为运行态等情形表示进程一直处于等待执行的状态，也就是其等待时间会增加。为了让这些进程尽快得到执行，应该增加器优先级。因此B、C错误。进程从就绪态转为运行态，如果此时降低进程的优先级，那么该进程有可能会被剥夺CPU，导致进程调度，带来系统开销，因此此时不适合降低进程的优先级，D错误。当进程的时间片用完时，说明该进程已经使用了足够的时间片来执行，系统应该调度其他优先级更高的进程来占用 CPU 时间，因此降低该进程的优先级是合理的,A选项正确。

2、进程P0和P1的共享变量定义及其初值为

boolean flag[2];　　int turn = 0;

flag[0] = FALSE;　　flag[1] = FALSE;

若进程 P0 和 P1 访问临界资源的类 C 伪代码实现如下：

```
void P0()   // 进程 P0
{
    while(TRUE)
    {
        flag[0]=TRUE; turn=1;
        while(flag[1]&&(turn==1))
            ;
        临界区;
        flag[0]=FALSE;
    }
}
```

```
void P1()   // 进程 P1
{
    while(TRUE)
    {
        flag[1]=TRUE; turn=0;
        while(flag[0]&&(turn==0))
            ;
        临界区;
        flag[1]=FALSE;
    }
}
```

则并发执行进程 P0 和 P1 时产生的情形是_____。

A. 不能保证进程互斥进入临界区，会出现“饥饿”现象

B. 不能保证进程互斥进入临界区，不会出现“饥饿”现象

C. 能保证进程互斥进入临界区，会出现“饥饿”现象

D. 能保证进程互斥进入临界区，不会出现“饥饿”现象

2、进程P0和P1的共享变量定义及其初值为

boolean flag[2];　　int turn = 0;

flag[0] = FALSE;　　flag[1] = FALSE;

若进程 P0 和 P1 访问临界资源的类 C 伪代码实现如下：

```
void P0()   // 进程 P0
{
    while(TRUE)
    {
        flag[0]=TRUE; turn=1;
        while(flag[1]&&(turn==1))
            ;
        临界区;
        flag[0]=FALSE;
    }
}
```

```
void P1()   // 进程 P1
{
    while(TRUE)
    {
        flag[1]=TRUE; turn=0;
        while(flag[0]&&(turn==0))
            ;
        临界区;
        flag[1]=FALSE;
    }
}
```

D【解析】根据给出的代码，进程 P0 和 P1 采用的是 Peterson 算法，其中 flag 数组用来表示各自的申请访问临界区的标志，turn 用来记录轮到哪个进程访问临界区。这种算法的核心思想是让每个进程都先申请访问临界区，如果另一个进程没有申请访问临界区，则直接让该进程进入临界区；如果另一个进程已经申请访问临界区，则通过轮流使用 turn 来控制进程的访问。但是，该算法仅仅是解决了互斥的问题，并能解决饥饿问题，因此，选项 D 正确。

3、有两个并发执行的进程P1和P2，共享初值为1的变量x。P1对x加1，P2对x减1。加1和减1 操作的指令序列分别如下所示。

```
// 加1操作                                    // 减1操作
①load R1，x    // 取x到寄存器R1中            ④load R2，x
②inc R1                                       ⑤dec R2
③store x，R1   // 将R1的内容存入x            ⑥store x，R2
```

两个操作完成后，x的值_____。

A．可能为-1或3　　B．只能为1

C．可能为0、1或2　　D．可能为-1、0、1或2

C【解析】（1）当按照①②③④⑤⑥的顺序执行，x=1

（2）当按照①④②③⑤⑥的顺序执行，x=0

（3）当按照①④⑤②⑥③的顺序执行，x = 2。

因此x的值可能是0、1和2，因此选择C。

4、一个进程的读磁盘操作完成后，操作系统针对该进程必做的是_____。

A．修改进程状态为就绪态

B．降低进程优先级

C．给进程分配用户内存空间

D．增加进程时间片大小

A【解析】进程申请读磁盘操作的时候，因为要等待磁盘I/O 操作完成，此时进程的状态时阻塞状态。当 I/O 操作完成后，进程得到了想要的资源，就会从阻塞态转换到就绪态，因此A正确。

因为进程处于阻塞状态时，进程的等待时间再增减，因此应该增加进程优先级。而进程从阻塞态转换到就绪态时，分配用户内存空间和增加进程的时间片大小都是无关的概念。

5、下列关于管道(Pipe)通信的叙述中，正确的是_____。

A．一个管道可实现双向数据传输

B．管道的容量仅受磁盘容量大小限制

C．进程对管道进行读操作和写操作都可能被阻塞

D．一个管道只能有一个读进程或一个写进程对其操作

C【解析】管道是一个文件，是指用于连接一个读进程和一个写进程以实现它们之间通信的一个共享文件，又名pipe文件，但它不是普通的文件，它不属于某种文件系统。因此B错误。

管道存在两个操作，即写操作和读操作，但是管道通信是一种半双工的通信方式，也就是在任意时刻只能存在一个操作，即写的时候不能读，读的时候不能写（分时操作时，可能存在多个读操作或者写操作），因此A错误，D错误。

C正确，这是因为如果存在多个写操作或者读操作时，当其中一个操作在使用管道时，其他的操作必须等待。

6、使用TL(Test and Set Lock)指令实现进程互斥的伪代码如下所示。

```
do{
        ……
    while(TSL(&lock));
    critical section;
    lock=FALSE;
        ……
}while(TRUE);
```

下列与该实现机制相关的叙述中，正确的是_____。

A. 退出临界区的进程负责唤醒阻塞态进程

B. 等待进入临界区的进程不会主动放弃CPU

C. 上述伪代码满足“让权等待”的同步准则

D. while(TSL(&lock)语句应在关中断状态下执行

B【解析】由于TSL中存在语句while(TSL(&lock));，lock为TRUE时，等待进入临界区的进程不会主动放弃CPU，处于忙等状态，此时进程并不处于阻塞状态。当其他进程退出临界区时，会将lock设置FALSE，可以允许处于忙等的进程进入临界区。因此退出临界区的进程并不负责唤醒阻塞态进程（事实上，处于忙等的进程并没有处于阻塞状态）。A错误。

由于TSL中存在语句while(TSL(&lock));，lock为TRUE时，将会执行空语句，因此上述伪代码不满足“让权等待”的同步准则，C错误，且等待进入临界区的进程不会主动放弃CPU，B正确。

若 while(TSL(&lock))在关中断状态下执行，当 TSL(&lock)一直为TRUE时，且不能被打断，系统会一直执行下去，但是系统并没有做任何有效运算，白白浪费资源，while(TSL(&lock)语句不可以在关中断状态下执行因此D 错误。

7、进程P1和P2均包含并发执行的线程,部分伪代码描述如下所示

```
//进程 P1
    int x=0;
    Thread1()
    {    int a;
    a=1;    x+=1;
    }
    Thread2()
    {    int a;
         a=2; x+=2;
    }
```

```
//进程 P2
    int x=0;
    Thread3()
    {    int a;
    a=x;    x+=3;
    }
    Thread4()
    {    int b;
         b=x; x+=4;
    }
```

下列选项中,需要互斥执行的操作是_____。

A. a=1与a=2　　B. a=x与b=x

C. x+=1与x+=2　　D. x+=1与x+=3

程序的执行粒度有进程和线程，那么当遇到共享变量（同名变量）时，根据变量的位置和同步/互斥关系，总结如下表，也就是位于进程外的变量，进程和同一进程中的线程均需要同步/互斥；进程内的变量进程之间不需要进程同步/互斥（进程的独立性），同一进程中的线程需要同步/互斥（同一进程中线程共享进程的资源）；线程内的变量，进程和线程均无需同步/互斥。事实上，当需要共享时，就需要同步/互斥。

变量位置	进程	同一进程中的线程
进程外	需要同步/互斥	需要同步/互斥
进程内、线程外	不需要同步/互斥	需要同步/互斥
线程内	不需要同步/互斥	不需要同步/互斥

7、进程P1和P2均包含并发执行的线程,部分伪代码描述如下所示

```
//进程 P1
    int x=0;
    Thread1()
    {    int a;
    a=1;    x+=1;
    }
    Thread2()
    {    int a;
         a=2; x+=2;
    }
```

```
//进程 P2
    int x=0;
    Thread3()
    {    int a;
    a=x;    x+=3;
    }
    Thread4()
    {    int b;
         b=x; x+=4;
    }
```

下列选项中,需要互斥执行的操作是_____。

A. a=1与a=2　　B. a=x与b=x

C. x+=1与x+=2　　D. x+=1与x+=3

C【解析】通过观察，这是两个进程，且变量均位于进程内，所以进程之间无需同步/互斥，那么只要分别分析进程P1和P2内的变量关系。P1中线程1的x+=1和线程2的x+=2，共享进程P1中的变量x，因此线程之间需要互斥。P2中线程1的x+=3和线程2的x+=4，共享进程P2中的变量x，因此线程之间需要互斥。其余的不需要互斥，因此选C。

8、下列关于管程的叙述中,错误的是_____。

A. 管程只能用于实现进程的互斥

B. 管程是由编程语言支持的进程同步机制

C. 任何时候只能有一个进程在管程中执行

D. 管程中定义的变量只能被管程内的过程访问

A【解析】管程定义了一个数据结构和能为并发进程所执行(在该数据结构上)的一组操作，代表共享资源的数据结构以及由对该共享数据结构实施操作的一组过程所组成的资源管理程序共同构成了一个操作系统的资源管理模块，管程可以被请求和释放资源的进程所调用。管程由四部分组成：(1)管程的名称;(2)局部于管程的共享数据结构说明;(3)对该数据结构进行操作的一组过程;(4)对局部于管程的共享数据设置初始值的语句。封装于管程内部的数据结构仅能被封装于管程内部的过程所访问，任何管程外的过程都不能访问它;反之，封装于管程内部的过程也仅能访问管程内的数据结构。从上述概念可以得出，B和D正确。

8、下列关于管程的叙述中,错误的是_____。

A. 管程只能用于实现进程的互斥

B. 管程是由编程语言支持的进程同步机制

C. 任何时候只能有一个进程在管程中执行

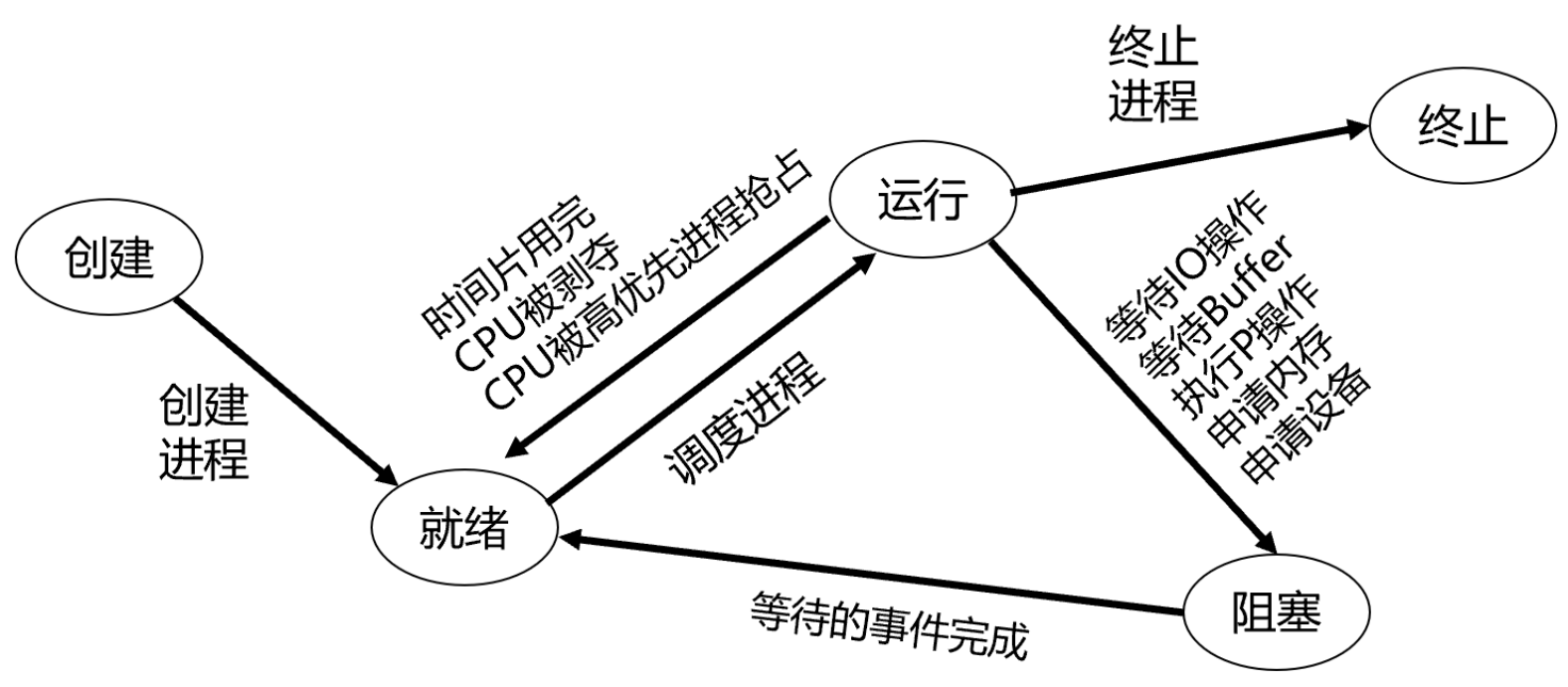
D. 管程中定义的变量只能被管程内的过程访问

A【解析】所有进程要访问临界资源时，都只能通过管程间接访问，而管程每次只准许一个进程进入管程，执行管程内的过程，从而实现了进程互斥，因此C正确。

- 管程中通过设置条件变量及等待/唤醒操作来解决同步问题，也即是让一个进程或者线程在条件变量上等待（此时，该进程应该释放管程的使用权）；可以通过唤醒操作将等待在变量上的进程或线程唤醒，因此管程既可以实现互斥，也可以实现进程的同步。

9、进程的基本状态( )可以由其他两种基本状态转变而来。

A．就绪状态　　B．执行状态　　C．阻塞状态　　D．新建状态



A【解析】进程共有三种基本状态,分别是就绪状态、执行状态、阻塞状态。只有就绪状态可以由其他两种基本状态转变而来。

10、下面的情况中,进程调度可能发生的时机有()。

Ⅰ．正在执行的进程时间片用完

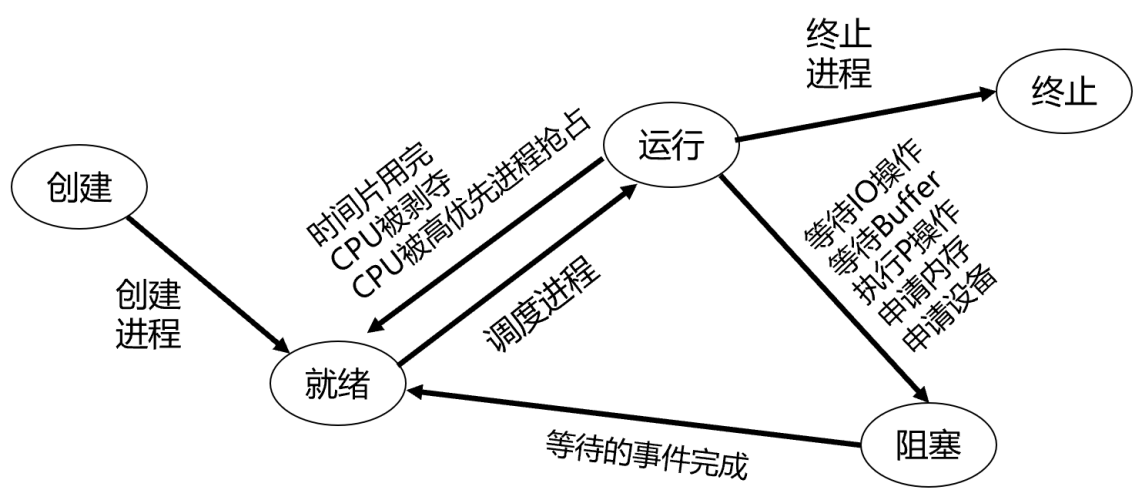
Ⅱ．正在执行的进程提出L/O请求后进入等待状态

Ⅲ．有新的用户登录进人系统

Ⅳ．等待硬盘读取数据的进程获得了所需的数据

A、Ⅰ　　B、Ⅰ、Ⅱ、Ⅲ、Ⅳ

C、Ⅰ、Ⅱ、Ⅳ　　D、Ⅰ、Ⅲ、Ⅴ



10、下面的情况中,进程调度可能发生的时机有()。

Ⅰ. 正在执行的进程时间片用完

Ⅱ. 正在执行的进程提出I/O请求后进入等待状态

Ⅲ. 有新的用户登录进入系统

Ⅳ. 等待硬盘读取数据的进程获得了所需的数据

A、Ⅰ　　B、Ⅰ、Ⅱ、Ⅲ、Ⅳ

C、Ⅰ、Ⅱ、Ⅳ　　D、Ⅰ、Ⅲ、Ⅴ

B【解析】正在执行的进程时间片用完后进入就绪状态,系统会调入一个新的进程分配处理机执行;

正在执行的进程提出IO请求后进入等待状态,系统同样会调入一个新的进程分配处理机执行;

有新的用户登录进入系统会创建新的进程,若处理机空闲,可进行进程调度;

等待硬盘读取数据的进程获得了所需的数据后,若处理机空闲,可进行进程调度。

11、设两个进程共用一个临界资源的互斥信号量mutex, 当mutex =1时表示()。

A.一个进程进入了临界区,另一个进程等待

B. 没有一个进程进入临界区

C. 两个进程都进入了临界

D. 两个进程都在等待

B【解析】当mutex=1时,表示没有进程进入临界区;当mutex=0时,表示有一个进程进入临界区运行,另一个进程必须等待,挂入阻塞队列;当mutex =-1时,表示有一个进程正在临界运行,另一个进程因等待而阻塞在信号量队列中,需要被当前已在临界区运行的进程退出时唤醒。

谢谢大家