華中農業大學
HUAZHONG AGRICULTURAL UNIVERSITY

计算机操作系统

倪福川
fcni_cn@mail.hzau.edu.cn

华中农业大学 信息学院

第二章　进程的描述与控制

2.4 进 程 同 步

引入进程，多道程序并发执行，有效地改善资源利用率，提高系统的吞吐量;

多个进程对系统资源争夺使每次处理结果显现出其不可再现性。

2.4.1 进程同步的基本概念

1. 两种形式的制约关系

1) 间接相互制约关系　　共享资源

2) 直接相互制约关系　　相互合作

进程的两大基本关系

- 互斥
- 同步

互斥

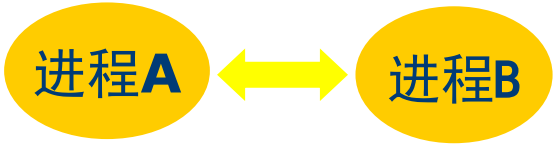
并发执行的多个进程由于竞争同一资源而产生的相互排斥的关系

同步

进程间共同完成一项任务时直接发生相互作用的关系

—同步进程间具有合作关系

—在执行时间上必须按一定的顺序协调进行



2. 临界资源(Critical Resouce)

一次仅允许一个进程使用的共享资源

如：打印机、磁带机、表格

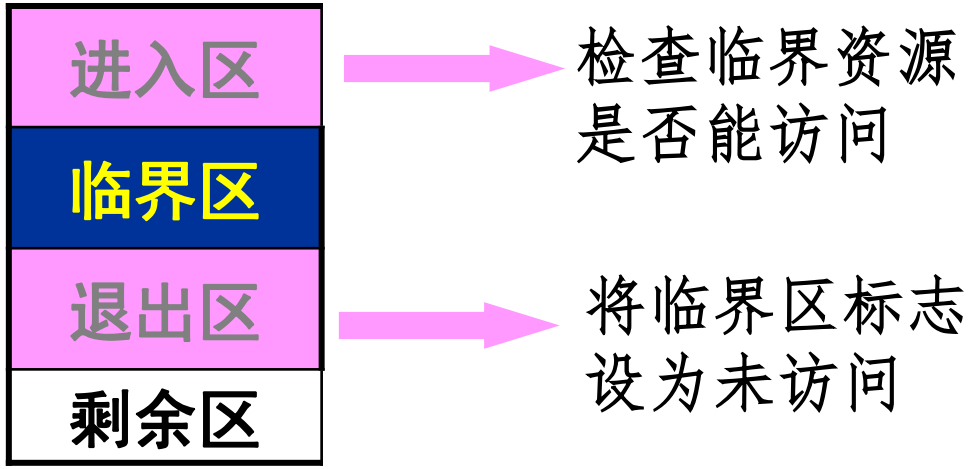
3. 临界区(critical section)　P50

在每个进程中访问临界资源的那段代码

进程必须互斥进入临界区

循环访问临界区的进程描述

While (True){



}

4. 同步机制应遵循的规则

P51

所有同步机制遵循的准则：

(1) 空闲让进。

(2) 忙则等待。　保证互斥

(3) 有限等待。　避免死等

(4) 让权等待。　避免忙等

2.4.2 硬件同步机制

1. 关中断
2. 利用Test-and-Set指令实现互斥
3. 利用Swap指令实现进程互斥

2.4.3 信号量机制

Dijkstra

1. 整型信号量

定义为一个用于表示资源数目的整型量S，除初始化外，仅能原子操作来访问。

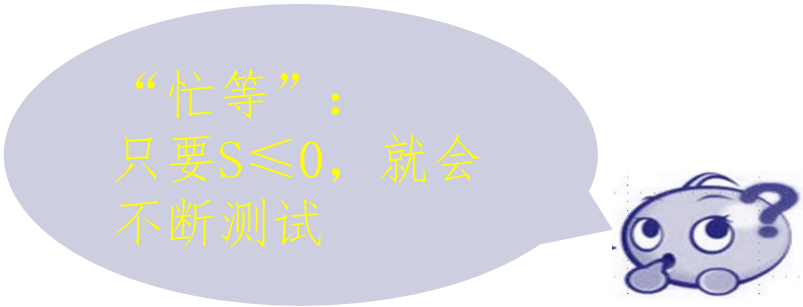
wait(S)　　P操作

signal(S)　　V操作

P V操作

Wait(s)和signal(s) 原子操作.

```
Wait（S）{
    while ( S≤0 );
      S=S-1;
}
Signal（S）{
      S=S+1;
}
```



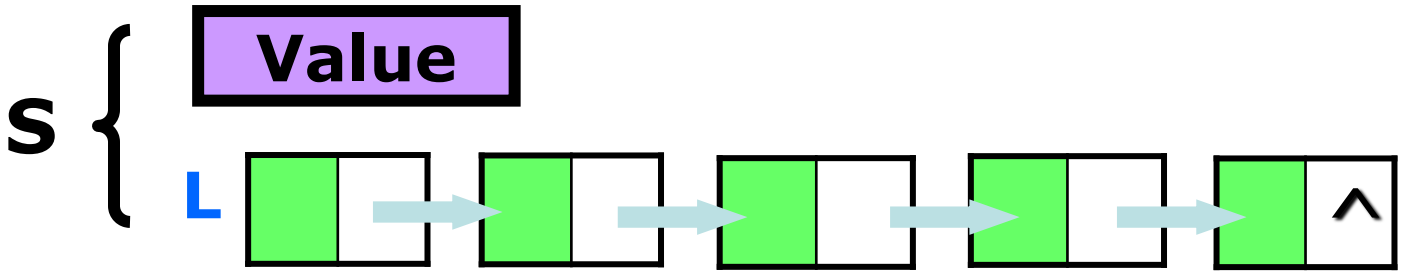
2. 记录型信号量

会出现多个进程等待访问同一临界资源的情况

采取“让权等待”的策略：

代表资源数目的整型变量value;

增加等待进程链表，用于链接所有等待进程。



适用：多个并发进程仅共享一个临界资源。

```
typedef struct {
    int value;
    struct process_control_block *list;
}
```

```
Wait（S）{
    S->value --;
    if(S->value <0 )
      block(S->list);
}
```

```
Signal（S）{
    S->value ++;
    if(S->value <=0 )
      wakeup(S->list);
}
```


```
Wait（S）{
    S->value --;
    if(S->value <0 )
      block(S->list);
}
```

S.value≥0：表示系统中可用的资源数量

S.value<0：其绝对值表示已阻塞的进程数量

S.Value初值为1时：只允许一个进程访问临界资源，是互斥信号量

3. AND型信号量

在有些应用场合，是一个进程往往需要获得两个或更多的共享资源后方能执行其任务。

可能 死锁

在wait中加入AND条件，又称AND同步或同时wait操作

将进程在整个运行中需要的所有资源，一次性全部分配给进程，待进程使用完后一起释放。

```
Swait(S1, S2, …Sn){
  while(ture){
    If (S1≥1  and  …  Sn≥1){
      for (i=1;i<=n;n++) Si--;
      break;
    }
    else{
          当发现第一个Si<1就把该进程
        放入等待队列并将其程序计数器
        置于Swait操作的开始位置
    }
  }
}
```


Ssignal(S_1, S_2, …S_n){

for (i=1;i<=n;n++){

S_i+ +;

将所有等待S_i的 进程由等待队列
取出放入到就绪队列

};

}

4. 信号量集

P55

仅对信号量施以加1或减1，低效，增加死锁的概率。

为确保系统安全性，当所申请的资源数量低于某一下限值时，还必须进行管制，不予以分配。

在每次分配之前，都必须测试资源的数量，判断是否大于可分配的下限值，决定是否予以分配。

Swait(L, d, d)

Swait(L, 1, 1)　　Swait(L, 1, 0)

2.4.4 信号量的应用

1. 利用信号量实现进程互斥

为使多个进程能互斥地访问某临界资源，只需为该资源设置一互斥信号量mutex，并设其初始值为1，然后将各进程访问该资源的临界区CS置于wait(mutex)和signal(mutex)操作之间即可。

用信号量实现互斥

必须成对地出现

```
semaphore mutex=1;

PA(){
while(1) {
    wait(mutex);
    临界区;
    signal(mutex);
    剩余区;
  }
}
```

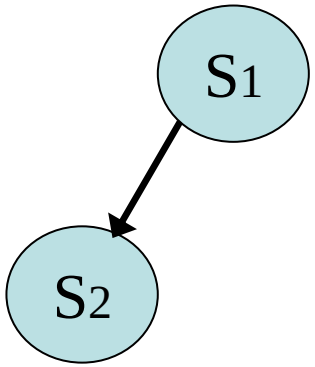
```
PB(){
  while(1) {
     wait(mutex);
     临界区;
     signal(mutex);
     剩余区;
  }
}
```


2. 利用信号量实现前趋关系

还可利用信号量来描述程序或语句之间的前趋关系。

P_1 (){S_1； signal(S)；}

P_2 (){wait(S)； S_2；}



若P_2先执行必定阻塞，只有在进程P_1执行完S_1；signal(S)；操作后使S增为1时，P_2进程方能成功执行语句S_2

```
p1() { S1; signal(a); signal(b);}
p2() { wait(a); S2; signal(c); signal(d);}
p3() { wait(b); S3; signal(e);}
p4() { wait(c); S4; signal(f);}
p5() { wait(d); S5; signal(g);}
p6() { wait(e); wait(f); wait(g); S6;}
main() {
        semaphore a, b, c, d, e, f, g;
        a.value=b.value=c.value=0;
        d.value=e.value=0;
        f.value=g.value=0;
        cobegin
                p1(); p2(); p3(); p4(); p5(); p6();
        coend
}
```

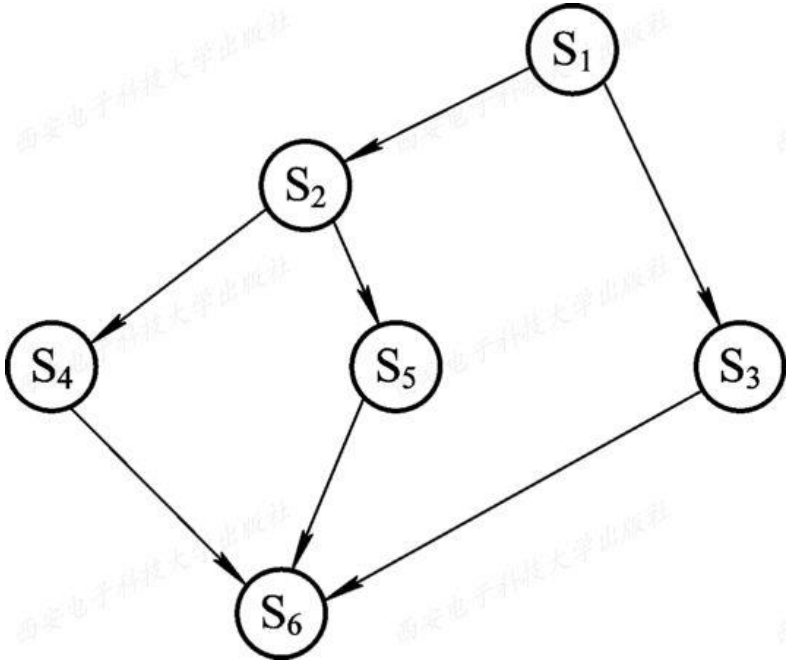


图2-14 前趋图举例

2.4.5 管程机制

进程自备同步操作，P(S)和V(S)操作大量分散在各个进程中，不易管理，易发生死锁。

1．管程的定义

代表共享资源的数据结构和对该数据结构实施的操作的一组过程所组成的资源管理程序，共同组成的操作系统的资源管理模块。

管程的语法描述：

```
Monitor monitor_name   {   /*管程名*/
   share variable declarations;        /*共享变量说明*/
   cond declarations;                  /*条件变量说明*/
   public:                             /*能被进程调用的过程*/
   void P1(……)                        /*对数据结构操作的过程*/
   {……}
   void P2(……)
   {……}
……
   void (……)
   {……}
   ……
   {                                   /*管程主体*/
      initialization code;             /*初始化代码*/
      ……
   }
}
```

局部于管程的共享数据结构说明

对该数据结构进行操作的一组过程

对局部于管程的共享数据设置初始值

①入口(等待)队列；

互斥进入

②资源等待队列；

等待资源的进程加入资源等待队列；由条件变量维护

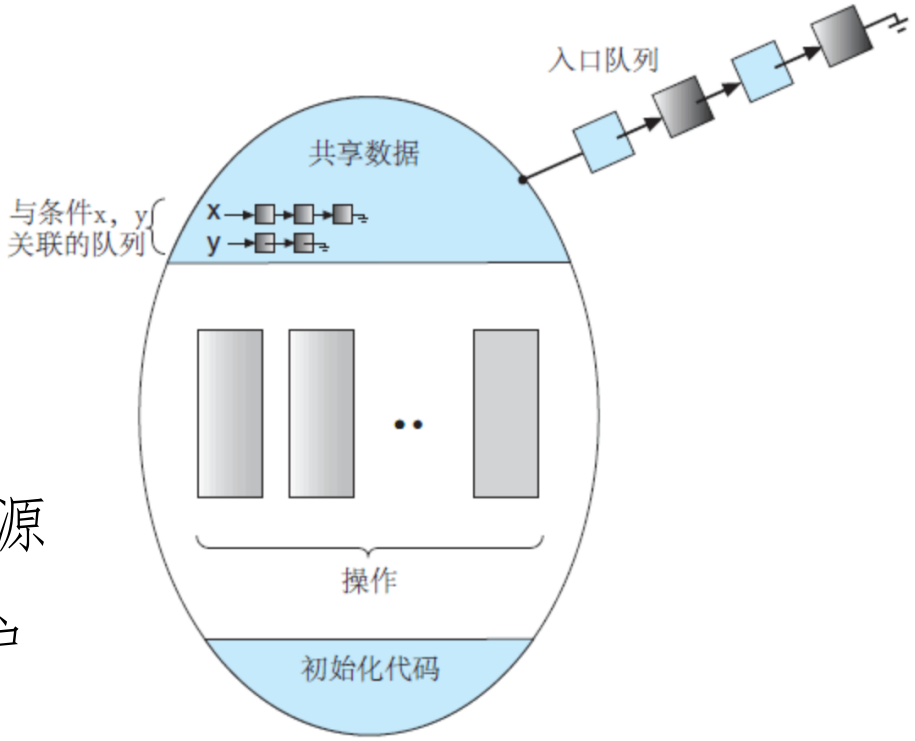


图 2 具有条件变量的管程

x.wait( )-调用进程阻塞并移入与条件变量x相关的队列中，并释放管程，直到另一个进程执行x. signal( )唤醒等待进程并将其移出条件变量x队列。

2.5 经典进程的同步问题

“生产者—消费者”问题、

“读者—写者问题”、

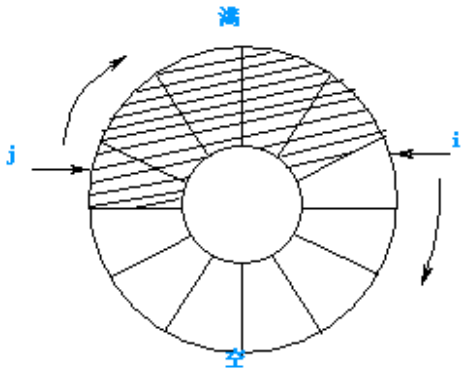
“哲学家进餐问题”

P49 P60

生产者—消费者问题

一组生产者进程生产产品给一组消费者进程消费。为使他们并发执行，设一个有n个缓冲区的缓冲池，生产者将其生产的产品放入一个缓冲区中；消费者可从一个缓冲区中取走产品去消费。

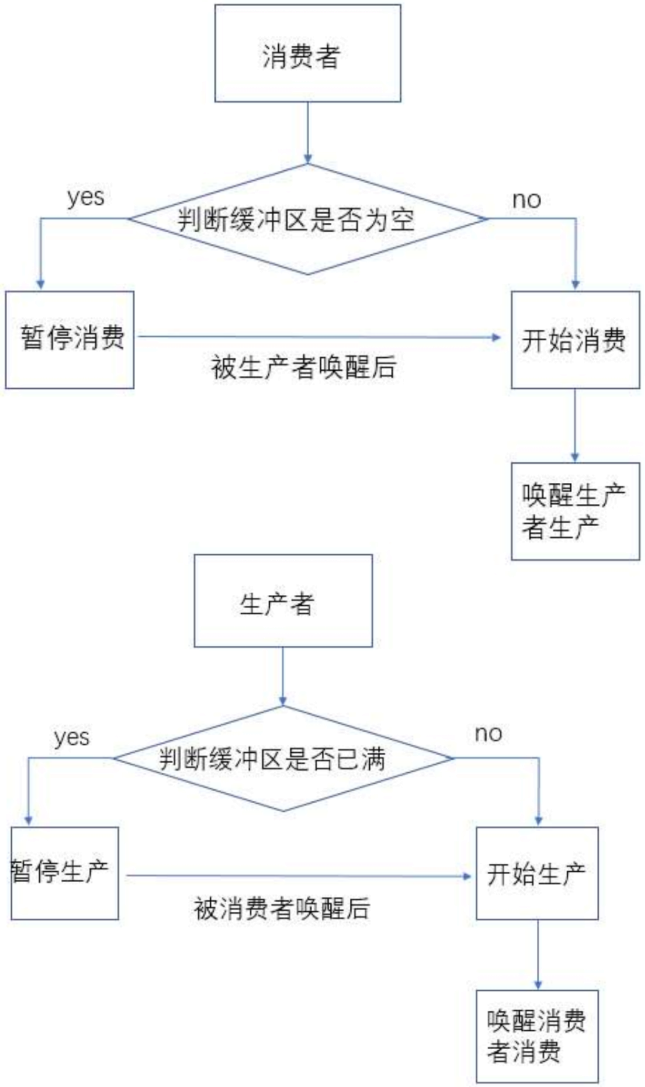
相互合作进程的抽象



制约关系

不允许消费者进程到一个空缓冲区中取产品

不允许生产者进程到一个已满且还没被取走的缓冲区中投放产品



2.5.1 生产者-消费者问题

1. 利用记录型信号量解决生产者-消费者问题

互斥信号量mutex：实现对缓冲池的互斥使用；

资源信号量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。

生产者-消费者问题描述

```
int in=0, out=0;
item buffer[n];
semaphore mutex=1, empty=n, full=0;

 void proceducer() {
    do {
         producer an item nextp;
         …
         wait(empty);
         wait(mutex);
         buffer[in] =nextp;
         in :=(in+1) % n;
         signal(mutex);
         signal(full);
    }while(TRUE);
 }
```

```
void consumer() {
    do {
         wait(full);
         wait(mutex);
         nextc= buffer[out];
         out =(out+1) % n;
         signal(mutex);
         signal(empty);
         consumer the item in nextc;
         …
    }while(TRUE);
 }

void main()    {
    cobegin
        proceducer();    consumer();
    coend
}
```


注意:

- 每个程序中互斥的wait(mutex)和signal(mutex）必须成对出现。
- 对资源信号量empty和full的P、V操作成对出现，但它们分别处于不同的程序中
- 每个程序中的P操作顺序不能颠倒。

应先执行对资源信号量的P操作，再执行对互斥信号量的P操作，否则可能引起进程死锁.

2. 利用AND信号量解决生产者-消费者问题

Swait(empty，mutex)代替wait(empty)和wait(mutex)；

Ssignal(mutex，full)代替signal(mutex)和signal(full)；

Swait(full，mutex)代替wait(full)和wait(mutex);

Ssignal(mutex，empty)代替Signal(mutex)和Signal(empty)。

```
int in=0, out=0;
item buffer[n];
semaphore mutex=1, empty=n, full=0;
void proceducer() {
   do {
      producer an item nextp;
      …
      Swait(empty，mutex);
      buffer[in] =nextp;
      in :=(in+1) % n;
      Ssignal(mutex，full);
   }while(TRUE);
}
```

```
void consumer() {
   do {
      Swait(full，mutex);
      nextc= buffer[out];
      out =(out+1) % n;
      Ssignal(mutex，empty);
      consumer the item in nextc;
      …
   }while(TRUE);
}
```


3. 利用管程解决生产者-消费者问题

建立命名为procducerconsumer管程(简称PC),其中包括两个过程：

(1) put(x)过程。生产的产品放到缓冲区 count 缓冲区中已有的产品数；count>=N 生产者等待

(2) get(x)过程。 消费者从缓冲区取出产品 count<=0 消费者等待

条件变量notfull和notempty操作：

(1) cwait(condition)过程：当管程被一个进程占用时，其他进程调用该过程时阻塞，并挂在条件condition的队列上。

(2) csignal(condition)过程：唤醒在cwait执行后阻塞在条件condition队列上的进程，如果这样的进程不止一个，则选择其中一个实施唤醒操作；如果队列为空，则无操作而返回。

```
Monitor procducerconsumer {
    item buffer[N];
    int in, out;
    condition notfull, notempty;
    int count;
    public:
    void put(item x) {
        if (count>=N) cwait(notfull);
        buffer[in] = x;
        in = (in+1) % N;
        count++;
        csignal(notempty);
    }
    void get(item x) {
        if (count<=0) cwait(notempty);
        x = buffer[out];
        out = (out+1) % N;
        count--;
        csignal(notfull);
    }
    {   in=0;out=0;count=0; }
}PC;
```


```
void producer() {
    item x;
    while(TRUE) {
        …
        produce an item in nextp;
        PC.put(x);
    }
}
```

```
void consumer() {
    item x;
    while(TRUE) {
        PC.get(x);
        consume the item in nextc;
        …
    }
}
```

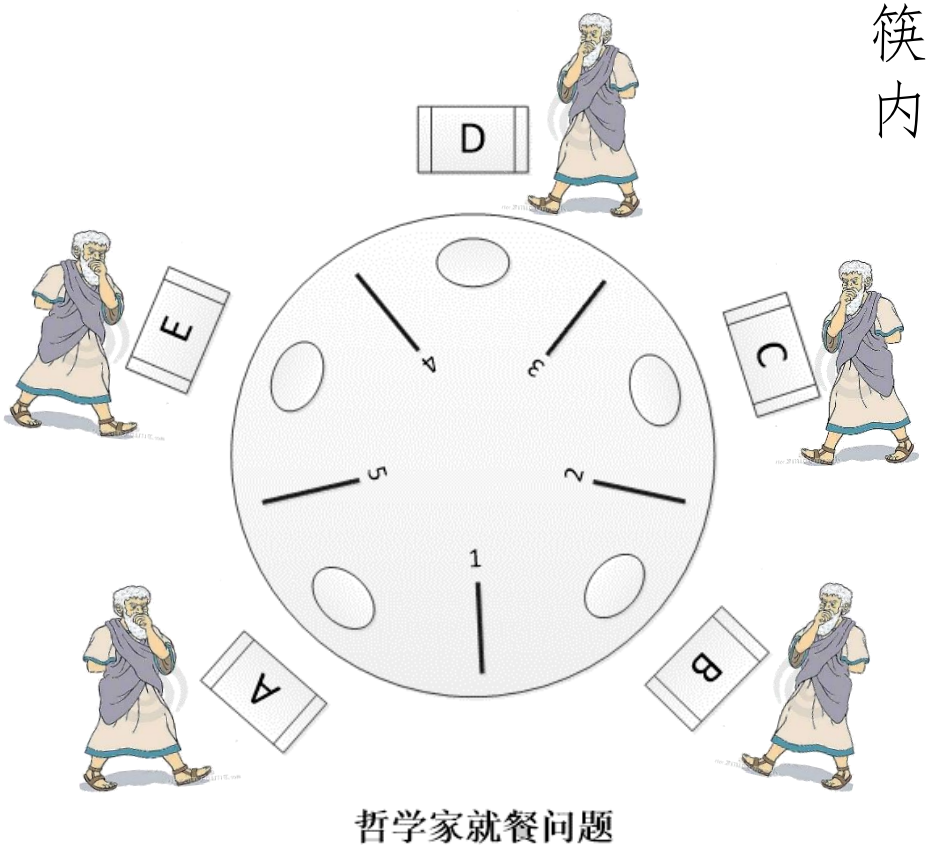

2.5.2 哲学家进餐问题

问题描述： P63

5个位哲学家共用一张圆桌，分别坐在周围的五张椅子上，在圆桌上有5个碗和5只筷子，他们的生活方式是交替的进行思考和进餐。

平时哲学家进行思考，饥饿时便试图取用其左右最靠近他的筷子，只有在他拿到两只筷子时才能进餐。进餐毕，放下筷子继续思考。

1. 利用记录型信号量解决



哲学家就餐问题

筷子：临界资源，一段时间内只允许一位哲学家使用

- 为实现对筷子互斥使用，可用一个信号量表示一只筷子；
- 五个信号量构成信号量数组

Chopstick[i]

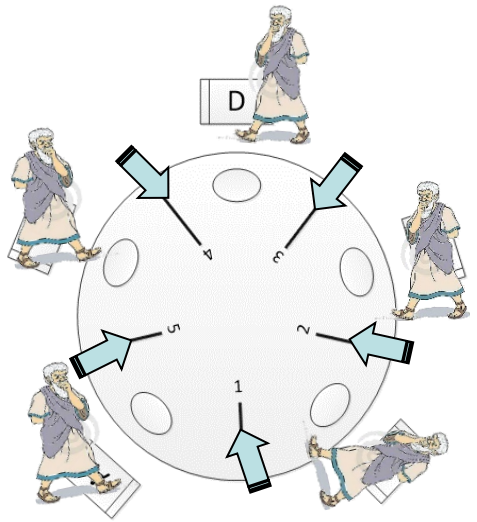
Chopstick[(i+1)%5]

第i位哲学家的活动：

```
do {
    wait(chopstick[i]);
    wait(chopstick[(i+1)%5]);
    …
    //eat
    …
    signal(chopstick[i]);
    signal(chopstick[(i+1)%5]);
    …
    //think
    …
}while[TRUE];
```

死锁

每位哲学家都拿起其左边的筷子，然后每位哲学家又都尝试去拿起其右边的筷子



哲学家就餐问题

每位哲学家都不能拿起其右边的筷子，只能等待筷子被其他哲学家释放

2.5.2 哲学家进餐问题---解决方法

(1)至多四个人同时拿左边的筷子，保证至少有一个人可以进餐，最终释放筷子使更多的人进餐。

(2)仅当哲学家的左右两支筷子均可用时，才允许他拿起筷子进餐。

(3)规定奇数号哲学家先拿起其左边的筷子，再拿右边的，偶数号哲学家则相反。总会有某一人进餐。

至多四个人同时拿左边筷子，保证至少有一个人可进餐

```
int chopstick[4]={1,1,1,1};
int room=4, i;

Void philosopher(int i){
    think;
    wait(room);
    wait(chopstick[i]);
    wait(chopstick[(i+1)mod 5]);
    eat;
    signal(chopstick[(i+1)mod 5]);
    signal(chopstick[i]);
    signal (room);
}
```

```
void main()
{
  philosopher(0);
  philosopher(1);
  philosopher(2);
  philosopher(3);
  philosopher(4);
}
```

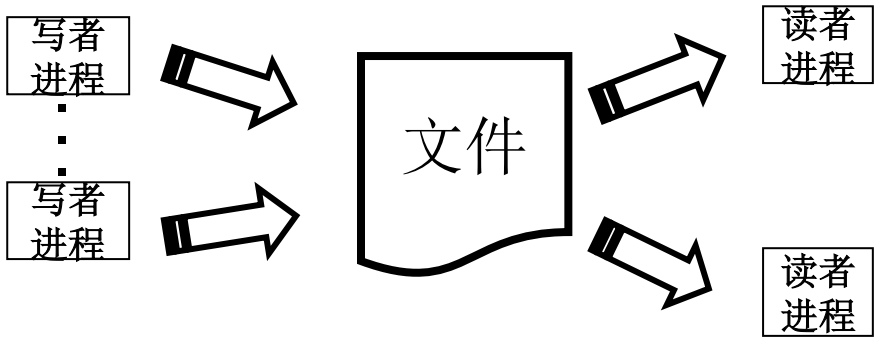

2. 利用AND信号量机制解决哲学家进餐问题

每个哲学家获得两个临界资源(筷子)后方能进餐，AND信号量同步，最简洁

```
semaphore chopstick chopstick[5]={1, 1, 1, 1, 1};
do {
   …
   //think
   …
   Sswait(chopstick[(i+1)%5], chopstick[i]);
         …
      //eat
         …
      Ssignal(chopstick[(i+1)%5], chopstick[i]);
}while[TRUE];
```


2.5.3 读者-写者问题

数据文件或记录可被多个进程共享。只要求读的进程称为“Reader进程”，要求行写或修改进程称为“Writer进程”。



- 允许多个Reader进程同时读一个共享对象；
- 不允许一个Writer进程和其它Reader进程或Writer进程同时访问共享对象。

保证Writer进程必须与其它进程互斥地访问共享对象的同步问题。

1. 利用记录型信号量解决读者-写者问题

Reader与Writer进程间需互斥访问文件(临界资源)；互斥信号量rmutex。

整型变量readcount表示正在读的进程数目；多个Reader进程修改 readcount (临界资源)，互斥信号量 rmutex

只要有一个Reader进程在读，便不允许Writer进程去写。

仅当Readcount=0，表示尚无Reader进程在读时，Reader进程才需要执行Wait(Wmutex)操作。

读者-写者问题描述

```
semaphore rmutex=1， wmutex=1;
int readcount=0;

void reader() {
   do {
      wait(rmutex);
      if (readcount==0) wait(wmutex);
      readcount++;
      signal(rmutex);
      …
      perform read operation;
      …
      wait(rmutex);
      readcount--;
      if (readcount==0) signal(wmutex);
      signal(rmutex);
   }while(TRUE);
}
```

若有Reader在读时，不需要Wait(Wmutex)操作

读者优先级高

```
void writer()   {
   do {
      wait(wmutex);
      perform write operation;
      signal(wmutex);
   }while(TRUE);
}
void main()   {
   cobegin
      reader();   writer();
   coend
}
```


2. 利用信号量集机制解决读者-写者问题

增加限制，最多只允许RN个读者同时读

引入信号量L，初值为RN，通过Swait(L, 1, 1)操作来控制读者的数目

每当有读者进入时，就要先执行Swait(L，1，1)操作，使L的值减1；

当有RN个读者进入读后，L便减为0；

第RN + 1个读者要进入，必然因Swait(L，1，1)操作失败而阻塞。

P55 P67

```
int RN;
semaphore L=RN, mx=1;
void reader() {
   do {
      Swait(L, 1, 1);
      Swait(mx, 1, 0);
      …
      perform read operation;
      …
      Ssignal(L, 1);
   }while(TRUE);
}
```

```
void   writer() {
   do {
      Swait(mx, 1, 1;  L, RN, 0);
      perform write operation;
      Ssignal(mx, 1);
      }while(TRUE);
}
void main()   {
   cobegin
      reader();   writer();
   coend
}
```

读者一写者同步（写者优先）

修改读者一写者的同步算法，使它对写者优先，即一旦有写者到达，后续的读者必须等待，而无论是否有读者在读文件。

用信号量和P 、V操作实现；

为了提高写者的优先级，增加信号量S，用于在写进程到达后封锁后续的读者。

用信号量和P、V操作实现

```
semaphore rmutex=1;
semaphore wmutex=1;
semaphore s=1;
int Count:=0;

  Writer(){
    While(1)
    { P(s);
      P(wmutex);
      写文件;
      V(wmutex);
     V(s);
    }
  }
```

```
Reader(){
  While(1)
  { P(s);
    P(rmutex);
    If(count==0) P(wmutex);
     Count++;
    V(rmutex);
    V(s);
    读文件;
    P(rmutex);
    Count--;
    If (count==0) v(wmutex);
     V(rmutex);
   }
}
```

```
Main()
{ cobegin
    Reader();
     Writer();
   Coend
}
```


课堂练习*

某商店有两种食品A和B, 最大数量各为m个. 该商店将A,B两种食品搭配出售, 每次各取一个。 为避免食品变质, 遵循先到食品先出售的原则, 有两个食品公司分别不断地供应A,B两种食品(每次一个)。为保证正常销售, 当某种食品的数量比另一种的数量超过k(k<m)个时, 暂停对数量大的食品进货, 补充数量少的食品。

(1) 问共需设置几个进程?

(2) 试用P,V操作解决上述问题中的同步和互斥关系.

共需要3个进程，商店、食品公司A、食品公司B

```
semaphore  S_BuffNum_A;    //A的缓冲区个数, 初值m
semaphore  S_Num_A;        // A的个数，初值为0
semaphore  S_BuffNum_B;    //B的缓冲区个数, 初值m
semaphore  S_Num_B;        // B的个数，初值为0
void  Shop( )
{
      while(1){
          P(S_Num_A);
          P(S_Num_B);
          分别取出A、B食品各一个;
          V(S_BuffNum_A);
          V(S_BuffNum_B);
           搭配地销售这一对食品;
       }
}
```


```
// “A食品加1，而B食品不变”允许出现的次数(许可证的
//数量)，其值等于k-(A-B)，初值为k
semaphore  S_A_B;
//  “B食品加1，而A食品不变”  允许出现的次数(许可证
//的数量)，其值等于k-(B-A)，初值为k
semaphore  S_B_A;

void  Producer_A ( )
{
  while(1){
     生产一个A食品;
     P(S_BuffNum_A);
     P(S_A_B);
     向商店提供一个A食品;
     V(S_Num_A);
     V(S_B_A);
  }
}

void  Producer_B ( )
{
   while(1){
      生产一个B食品;
      P(S_BuffNum_B);
      P(S_B_A);
       向商店提供一个B食品;
      V(S_Num_B);
      V(S_A_B);
   }
}
```


主观题 10分

设置

HUAZHONG AGRICULTURAL UNIVERSITY

在本课堂上，你想学习那些内容？
你有那些建议？

正常使用主观题需2.0以上版本雨课堂

作答