
深圳大学
Shenzhen University

# 操作系统

第三讲 进程同步

谭舜泉
计算机与软件学院

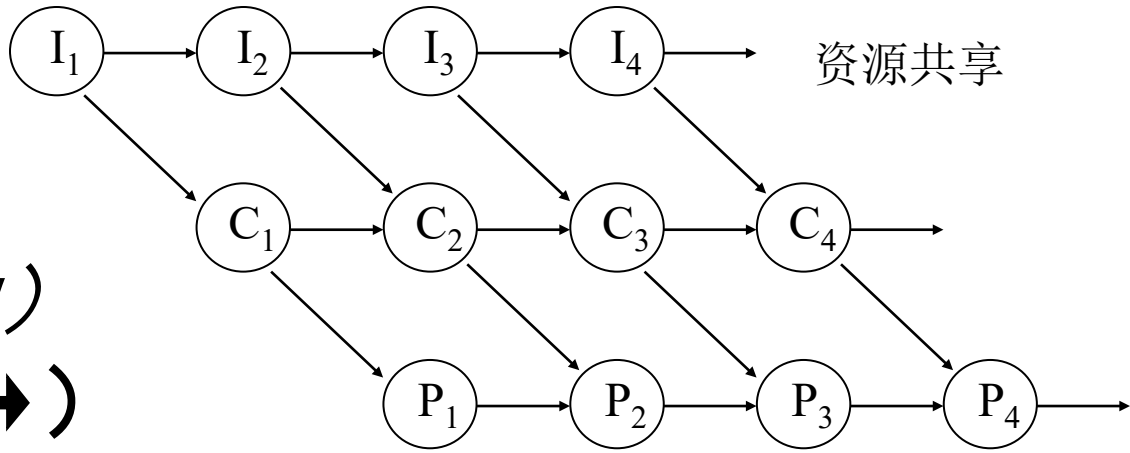
# 第3章　进程的同步与通信

## 第一节　进程同步的基本概念

一、进程之间的关系

- **相互合作关系**（*直接相互制约*）
  （I→C→P）

- **资源共享关系**
  （*间接相互制约*）
  （$I_1$→ $I_2$→ $I_3$→）




相互合作

*I：输入进程　C：计算进程　P：输出进程*

# 第3章　进程的同步与通信

## 第一节　进程同步的基本概念

### 二、临界资源

**1、举例：银行存款问题**（*count是共用变量，即共享资源，临界资源*）

P1：count += 300;

P2：count  -= 200;

*count = 初值+100*

*LD R, count*

*ADD R, 300*

*LD count, R*

P1：LD R1, count;

P2：LD R2, count;

P2：SUB R2, 200;

P2：LD count, R2;

P1：ADD R1, 300;

P1：LD count, R1;

*count = 初值 + 300*

P1：LD R1, count;

P2：LD R2, count;

P1：ADD R1, 300;

P1：LD count, R1;

P2：SUB R2, 200;

P2：LD count, R2;

*count = 初值 - 200*


深圳大学
Shenzhen University

# 第3章　进程的同步与通信

## 第一节　进程同步的基本概念

二、临界资源

### 2、临界资源

- 临界资源是一个时刻只能由一个进程使用的资源
- 硬件资源：许多都属于临界资源，如打印机，磁带机等
- 软件资源：如变量、表格、队列等

# 第3章　进程的同步与通信

## 第一节　进程同步的基本概念

二、临界资源

### 3、临界资源使用方法

- 对临界资源的使用采用互斥方式
- 互斥：
  一个进程使用完之后，另一个进程才能使用

# 第3章　进程的同步与通信

## 第一节　进程同步的基本概念

三、临界区

1、**临界区**

- 进程中访问临界资源的**代码段**

2、**同类临界区**

- 涉及同一临界资源的不同进程中的临界区

# 第3章　进程的同步与通信

## 第一节　进程同步的基本概念

三、临界区

3、**进入区**

- 临界区前检查临界资源使用情况的代码段

4、**退出区**

- 临界区后面恢复临界资源访问标志的代码段

# 第3章　进程的同步与通信

## 第一节　进程同步的基本概念

三、临界区

### 5、临界资源使用的同步准则

- **空闲让进**：*（提高效率）*
- **忙则等待**：*（解决互斥）*
- **有限等待**：等待进入临界区的要求应在一有限时间满足*（以免死等）*
- **让权等待**：*放弃占用CPU（以免忙等）*

# 第3章　进程的同步与通信

## 第一节　进程同步的基本概念

### 三、临界区

**6、临界资源的状态判断与设置**

**例1：flag=true代表有进程正在使用资源**

*flag初值为false*

```
Pi: repeat
    while flag do no_op
    flag := true;
    临界区代码
    flag := false;
        :
Until false;
```

```
Pj: repeat
    while flag do no_op
    flag := true;
    临界区代码
    flag := false;
        :
Until false;
```

*出现“同时进入”问题*

*不符合“忙则等待”准则*


深圳大学 Shenzhen University

# 第3章　进程的同步与通信

## 第一节　进程同步的基本概念

三、临界区

**6、临界资源的状态判断与设置**

- 对临界资源状态（是否正在使用）必须判断与设置同时进行
- 即判断与状态设置（状态改变）为原子操作，否则，会出现问题

# 第3章　进程的同步与通信

## 第二节　信号量机制

一、信号量

1、信号量（Semaphore）

- 信号量是一种特殊的变量（S）
- 除初始化外，对信号量变量（S）的操作只能由两个标准的原子操作（不可中断）实现
- wait(S):等待操作（也叫P操作）
- signal(S):发信号操作（也叫V操作）

# 第3章　进程的同步与通信

## 第二节　信号量机制

二、信号量机制

- 信号量变量是由一个整型数和一个阻塞等待进程链表构成的记录型数据结构

```
type semaphore = record
     value: integer;
     L:     list of process;
end;
```

- 所有对同一个信号量进行操作且进入等待状态的进程，都自动进入阻塞等待（让权等待）状态，并将其挂在该信号量阻塞等待队列L中

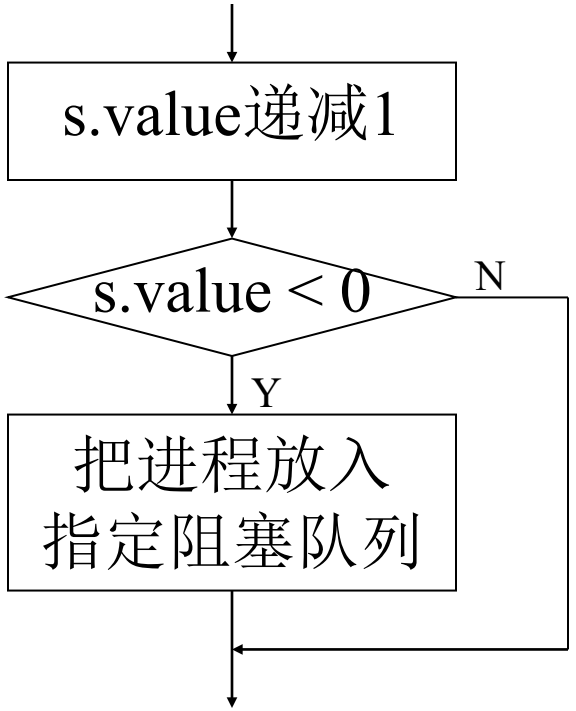
# 第3章　进程的同步与通信

## 第二节　信号量机制

### 二、信号量机制

### 1、P原子操作（wait）

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



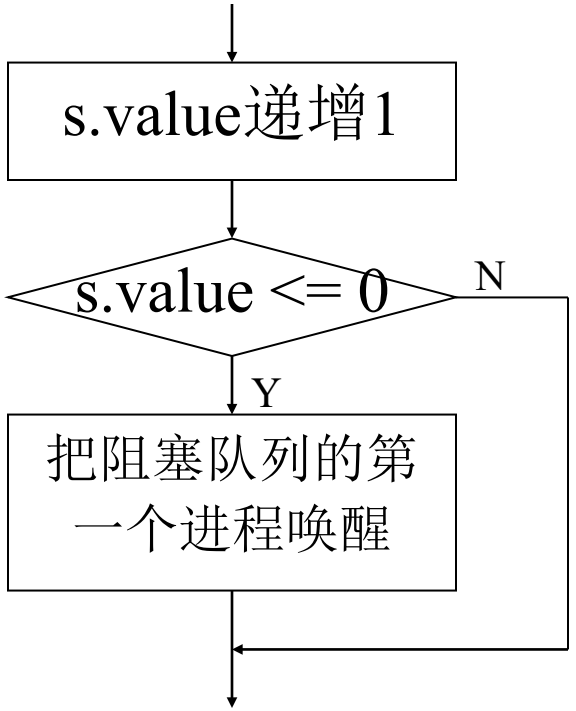
# 第3章　进程的同步与通信

## 第二节　信号量机制

### 二、信号量机制

### 2、V原子操作（signal）

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



# 第3章　进程的同步与通信

## 第二节　信号量机制

二、信号量机制

### 3、利用信号量实现进程互斥同步(p56)

- 银行存款问题

  count是共用变量(即共享资源)，采用mutex互斥信号量实现互斥访问

```
var mutex: semaphore := 1;                                  //定义变量
 procedure P1;        //增加存款进程    procedure P2;        //减少存款进程
 begin                                  begin
    wait(mutex);                           wait(mutex);
    count += 300;                          count -= 200;
    signal(mutex);                         signal(mutex);
 end;                                   end;
```

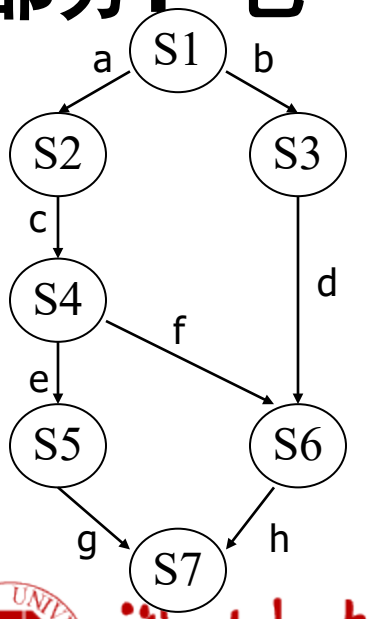
# 第3章　进程的同步与通信

## 第二节　信号量机制

### 二、信号量机制

### 4、利用信号量实现较复杂的进程同步(p56)

- S1… S7分别为进程P1…P7的执行部分，它们的执行顺序如右图所示




```
var a, b, c, d, e, f, g, h: semaphore := 0, 0, 0, 0, 0, 0, 0, 0;
P1: begin S1; signal(a); signal(b); end;
P2: begin wait(a); S2; signal(c); end;
P3: begin wait(b); S3; signal(d); end;
P4: begin wait(c); S4; signal(e); signal(f); end;
P5: begin wait(e); S5; signal(g); end;
P6: begin wait(f); wait(d); S6; signal(h);
P7: begin wait(g); wait(h); S7; end;
```

# 第3章　进程的同步与通信

## 第二节　信号量机制

二、信号量机制

### 5、信号量机制的特性

- wait(s)操作和signal(s)操作必须成对出现（可以不在同一进程中）
- 缺少wait(s)不能保证资源互斥使用
- 缺少signal(s)将可能使资源永远得不到释放
- 不存在“忙等”问题

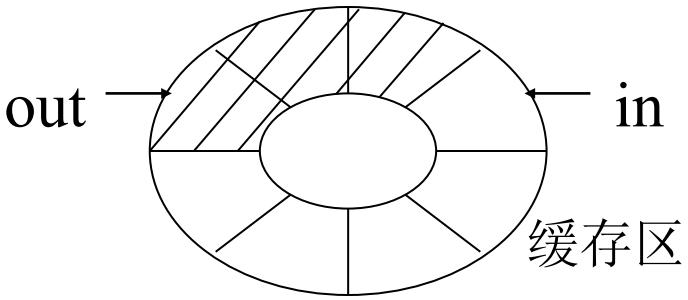
# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

### 一、生产者—消费者问题

**1、生产者—消费者问题描述**

- **有一群生产者进程在生产消息，并将此消息提供给消费者进程去消费**
- **生产者进程（多个），产品存放缓冲区（n个），消费者进程（多个）**




*缓冲区是临界资源*

# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

一、生产者—消费者问题

**2、生产者—消费者问题举例**

- **数据输入进程（生产者）与计算进程（消费者）**
- **数据计算进程（生产者）与数据输出（如打印）进程（消费者）**

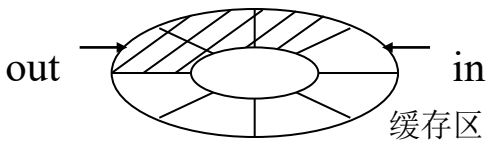
# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

### 一、生产者—消费者问题

### 3、生产者—消费者关系

- **公用信号量mutex：** 初值为1，实现临界资源（缓冲区）互斥使用。(如果缓冲区只有一个缓冲块，可以只用这一个信号量)
- **生产者私用信号量empty：** 初值为n，指示空缓冲块数目
- **消费者私用信号量full：** 初值为0，指示满缓冲块数目
- **整型量in，out：** 初值均为0，in指示首空缓冲块序号，out指示首满缓冲块序号



# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

### 一、生产者—消费者问题

### 4、采用信号量的生产者-消费者进程描述

```
var mutex, empty, full: semaphore := 1, n, 0
    buffer: array[0, …, n-1] of item;                    定义变量
    in, out: integer: =0, 0;
procedure producer;                  生产者进程
begin
    repeat
        生产一个产品nextp;
        wait(empty);    wait(mutex);
        buffer(in) : = nextp;
        in := (in + 1) mod n;
        signal(mutex);    signal(full);
    until false;
end;
```

```
procedure consumer;                  消费者进程
begin
    repeat
        wait(full);    wait(mutex);
        nextc := buffer(out);
        out := (out + 1) mod n;
        signal(mutex);    signal(empty);
        消费一个产品nextc;
    until false;
end;
```

# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

一、生产者—消费者问题

### 5、采用信号量应注意的问题

- 公用（互斥）信号量（mutex）和私用信号量（empty, full）都必须成对出现
- 在进入临界区之前，先对私用信号量（empty, full）进行wait操作，再对公用信号量进行wait操作（不能颠倒，否则易于造成“死锁”）

# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

二、阅览室管理问题

### 1、阅览室管理问题描述

- 阅览室有一批座位，可以坐下若干同学自习；有一张唯一的登记表，记录空座位数目
- 某一同学入室时，查看登记表；如果空座位数目＞0，则座位数减一，并进入阅览室坐下自习；否则，在阅览室门口等待
- 当从阅览室中出来时，使登记表中空座位数加1

# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

二、阅览室管理问题

### 2、阅览室管理问题举例

- 阅览室中有50个座位（即最多可坐下50位同学自习），只有一张登记表，欲进入阅览室的同学须先在登记表上登记，然后进入阅览室自习；出门时注销登记。


深圳大学
Shenzhen University

登记表是临界资源

# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

二、阅览室管理问题

### 3、进入与出门关系

- **互斥公用信号量mutex：** 初值为1，实现临界资源（登记表）互斥使用。

- **空座位信号量seat：** 初值为50，指示空座位数目

# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

### 二、阅览室管理问题

### 4、采用信号量的阅览室管理进程描述

```
var mutex, seat: semaphore := 1, 50                    //定义变量

procedure roomin;          //进门进程
begin
    wait(seat);
    wait(mutex);
    登记，同学进入阅览室……;
    signal(mutex);
        :
end;
```

```
procedure roomout;     //出门进程
begin
    wait(mutex);
    撤消登记，同学走出阅览室….;
    signal(mutex);
    signal(seat);
        :
end;
```

# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

三、读者－写者问题

### 1、读者－写者问题描述

- 指多个只读进程（reader）可以同时访问共享对象，而改写进程（writer）必须与其它进程（包括reader，writer）互斥地访问共享对象的同步问题

# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

三、读者－写者问题

### 2、读者－写者问题举例

- 数据库数据的读（读者）操作、写（写者）操作
- 民航订票：查询订票信息（读者）操作，售票人员售出机票（写者）操作

# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

三、读者－写者问题

### 3、读者－写者关系

- **互斥信号量wmutex：** 初值为1，实现临界资源（文件）互斥使用。
- **读者数目：** readcount，初值为0，表示正在读的进程数目

  如果readcount=0，读者进程必须执行wait(wmutex)操作

  读者操作完成之后，如果readcount-1=0，读者进程必须执行signal(wmutex)操作
- **读操作信号量rmutex：** 初值为1，实现对readcount的互斥访问

# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

### 三、读者－写者问题

### 4、采用信号量的读者–写者进程描述

```
var rmutex, wmutex: semaphore := 1, 1 ; 定义变量
    readcount: integer: =0;



procedure writer;                写者进程
begin
   repeat
      wait(wmutex);
      执行写操作;
      signal(wmutex);
   until false;
end;
```

```
procedure reader;                读者进程
begin
   repeat
      wait(rmutex);
      if readcount = 0 then wait(wmutex);
      readcount := readcount + 1;
      signal(rmutex);
      执行读操作
      wait(rmutex);
      readcount := readcount - 1;
      if readcount = 0 then signal(wmutex);
      signal(rmutex);
   until false;
end;
```

# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

三、读者—写者问题

### 5、采用信号量应注意的问题

- 写（互斥）信号量（wmutex）必须成对出现
- 在对readcount（临界资源）进行处理之前，必须先进行wait(rmutex)操作；处理完之后（+1或-1），再执行signal(rmutex)操作

# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

四、过独木桥问题

### 1、过独木桥问题描述

- 有一座南北向独木桥，只能行走一人。如果桥上有一个人，则同向的人可以上桥并通过，反向的人只能等待。
- 过独木桥问题实际上是两类（读者）进程同时访问共享对象的问题，同种类的（读者）进程可以同时操作（读），但不同种类的（读者）进程不能同时操作（读）

# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

四、过独木桥问题

### 2、过独木桥问题举例

- 三峡大坝航运问题
- 巴拿马运河航运问题

# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

四、过独木桥问题

### 3、过独木桥关系

- **互斥信号量mutex：** 初值为1，实现临界资源（独木桥）互斥使用。

# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

四、过独木桥问题

### 3、过独木桥关系

- **南桥头行人数目：** sCount，初值为0，表示正在从南向北过桥的行人数目，如果sCount=0(第一个由南桥头上桥的行人)，进程必须执行wait(mutex)操作；行人过完桥后，如果sCount-1=0(最后一个从南桥头过桥的行人)，进程必须执行signal(mutex)操作

- **南桥头私用信号量：** sMutex，初值为1，实现对sCount互斥访问功能

# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

四、过独木桥问题

### 3、过独木桥关系

- **北桥头行人数目：** nCount，初值为0，操作与南桥头相同
- **北桥头私用信号量：** nMutex，初值为1，实现对nCount互斥访问功能

# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

四、过独木桥问题

## 4、采用信号量的过独木桥进程描述

```
var mutex, smutex, nmutex: semaphore := 1, 1, 1 ; 定义变量
    sCount, nCount: integer: =0;
procedure SouthPassenger;          //南桥头行人进程
begin
    wait(smutex);
    if sCount = 0 then wait(mutex);    //得到过桥权
    sCount := sCount + 1;
    signal(smutex);
    向北，过桥……
    wait(smutex);
    sCount := sCount - 1;
    if sCount = 0 then signal(mutex);   //释放过桥权
    signal(smutex);
end;
```

```
procedure NorthPassenger;          //北桥头行人进程
begin
    wait(nmutex);
    if nCount = 0 then wait(mutex);    //得到过桥权
    nCount := nCount + 1;
    signal(nmutex);
    向南，过桥……
    wait(nmutex);
    nCount := nCount - 1;
    if nCount = 0 then signal(mutex);   //释放过桥权
    signal(nmutex);
end;
```

# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

四、过独木桥问题

### 5、采用信号量应注意的问题

- 公用互斥信号量（mutex）必须成对出现
- 在对sCount（临界资源）进行处理之前，必须先进行wait(sCount) 操作；处理完之后（+1或-1），再执行signal(sCount)操作
- 对nCount的操作与sCount相同

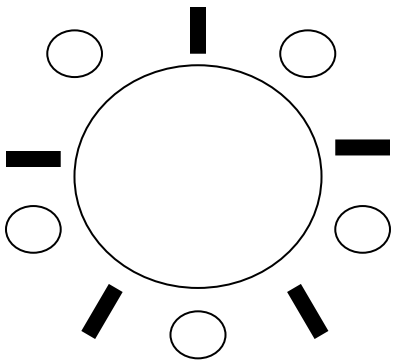
# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

五、哲学家进餐问题

### 1、哲学家进餐问题描述

- 五个哲学家，围着圆桌交替地进行思考和进餐；每次进餐时，必须同时拿到左右两边的两只筷子才能进餐；进餐后，再放下筷子继续思考。



# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

五、哲学家进餐问题

### 2、哲学家进餐问题所用信号量

- 这是一个典型的同时需要两个资源的例子
- 采用矩阵型信号量chopstick[i]（i=0...5）
- 当chopstick[i]和chopstick[i+1]两个临界资源都取得时，第i个进程（哲学家）可以执行（进餐）

# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

### 五、哲学家进餐问题

### 3、采用信号量的哲学家进餐进程描述

```
var chopstick: array[0,…,4]: semaphore := 1, 1, 1, 1, 1;        //定义变量
procedure OddProcess;  //奇数哲学家进程
begin
    wait(chopstick[i]);
    wait(chopstick[i+1]);
    吃饭……
    signal(chopstick[i]);
    signal(chopstick[i+1]);
end;
```

```
procedure EvenProcessr; //偶数哲学家进程
begin
    wait(chopstick[i+1]);
    wait(chopstick[i]);
    吃饭……
    signal(chopstick[i+1]);
    signal(chopstick[i]);
end;
```

# 第3章　进程的同步与通信

## 第三节　经典进程同步问题

五、哲学家进餐问题

### 3、采用信号量应注意的问题

- 相邻的哲学家最好不要同时拿同方位的筷子
- 至少有一个哲学家与其他哲学家第一次拿的筷子方位不同

# 第3章　进程的同步与通信

## 第四节　特殊信号量机制

一、AND型信号量集机制 (p54)

- 将进程运行过程中需要的所有临界资源，**一次性地**全部分配给进程（即要么全部分配，要么一个也不分配）
- 进程使用完后再一起释放

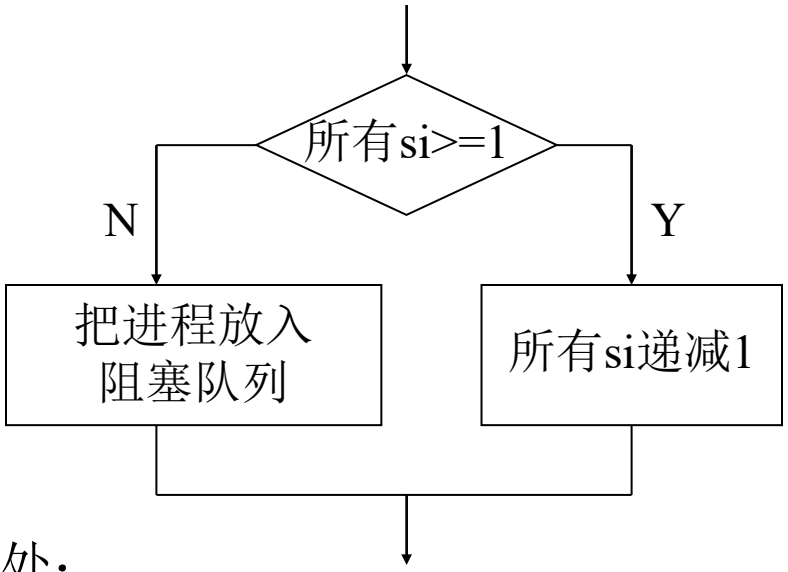
# 第3章　进程的同步与通信

## 第四节　特殊信号量机制

### 一、AND型信号量集机制

### 1、P原子操作（Swait）

```
procedure Swait(s1, s2, …, sn);
var s1, s2, …, sn: semaphore;
begin
    if  (s1 >= 1) and … and (sn >= 1)
    then {for i := 1 to n do  si := si – 1;
            return true;}
    else  {将程序计数器设置为本函数开始处;
            将进程插入阻塞队列;}
end;
```



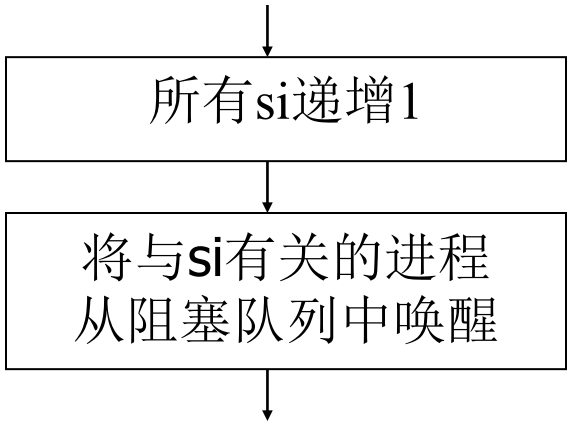
# 第3章　进程的同步与通信

## 第四节　特殊信号量机制

### 一、AND型信号量集机制

### 2、V原子操作（Ssignal）

```
procedure Ssignal(s1, s2, …, sn);
var s1, s2, …, sn: semaphore;
begin
    for i := 1 to n do  si := si + 1;
    将与si有关的进程从阻塞队列中唤醒
end;
```



# 第3章　进程的同步与通信

## 第四节　特殊信号量机制

### 二、采用AND信号量解决哲学家进餐问题

```
var chopstick: array[0, …, 4] of semaphore := 1, 1, 1, 1, 1;        //定义变量
 procedure i;
 begin
     repeat
         思考;
         Swait(chopstick[i], chopstick[(i+1) mod 5]);
         进餐;
         Ssignal(chopstick[i], chopstick[(i+1) mod 5]);
     until false;
 end;
```

# 第3章　进程的同步与通信

## 第四节　特殊信号量机制

三、一般信号量集机制 (p55)

- 进程运行需要多个临界资源
- 一次需要N个同类临界资源
- 大于t个同类临界资源才准予分配

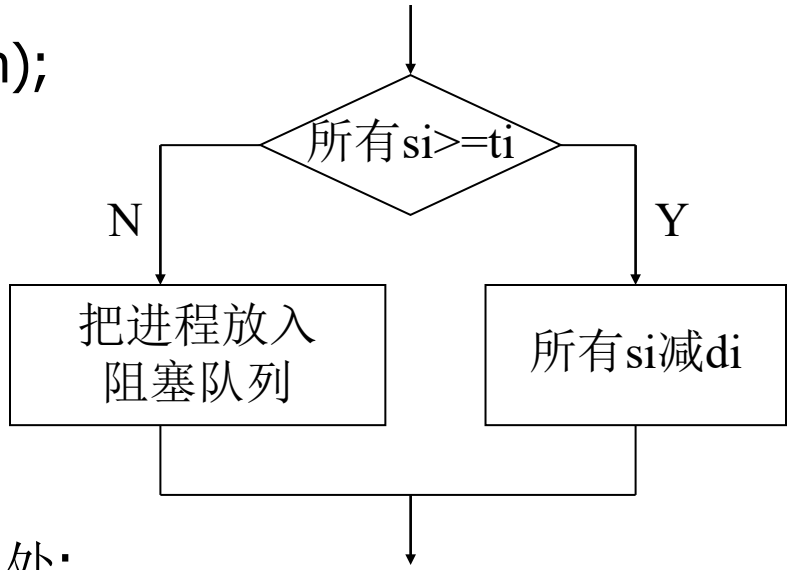
# 第3章　进程的同步与通信

## 第四节　特殊信号量机制

三、一般信号量集机制

### 1、P原子操作（Swait）

```
procedure Swait(s1, t1, d1; …; sn, tn, dn);
var s1, s2, …, sn: semaphore;
begin
    if  (s1 >= t1) and … and (sn >= tn)
    then  {for i := 1 to n do  si := si – di;
              return true; }
    else  {将程序计数器设置为本函数开始处;
              将进程插入阻塞队列; }
end;
```



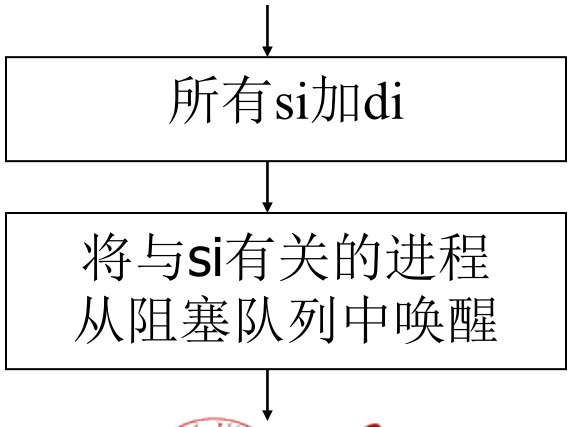
# 第3章　进程的同步与通信

## 第四节　特殊信号量机制

三、一般信号量集机制

### 2、V原子操作（Ssignal）

```
procedure Ssignal (s1, d1; …; sn, dn);
var s1, s2, …, sn: semaphore;
begin
    for i := 1 to n do  si := si + di;
    将与si有关的进程从阻塞队列中唤醒
end;
```



# 第3章　进程的同步与通信

## 第四节　特殊信号量机制

三、一般信号量集机制

### 3、一般信号量集特例

- Swait(s, d, d):

  一个信号量，每次同时分配d个同类资源

- Swait(s, 1, 1):

  等效于记录型信号量

- Swait(s, 1, 0):

  s=1，允许多个进程进入特定区；

  s=0，阻止任何进程进入特定区

# 第3章　进程的同步与通信

## 第四节　特殊信号量机制

### 四、采用一般信号量集解决读者－写者问题（最多只允许RN个读者同时写）

```
var rmutex, wmutex: semaphore := RN, 1 ; 定义变量

procedure writer;                          写者进程
begin
    repeat
        Swait(wmutex, 1, 1; rmutex, RN, 0);
        执行写操作;
        Ssignal(wmutex, 1);
    until false;
end;
```

```
procedure reader;                          读者进程
begin
    repeat
        Swait(rmutex, 1, 1);
        Swait(wmutex, 1, 0);
        执行读操作
        Ssignal(rmutex, 1);
    until false;
end;
```

*最多只能有RN个读者*

# 第3章　进程的同步与通信

## 第五节　管程机制

一、管程的定义（p57）

### 1、信号量机制的一些问题

- 临界区分散在各进程之中，不便于管理和控制
- 很多临界区的操作是相同的，重复编写，使进程结构不清晰
- 如果编程出现差错，不便于检查，且会带来严重后果

# 第3章　进程的同步与通信

## 第五节　管程机制

一、管程的定义

### 2、管程的定义

- 一个管程定义了一个**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**，这组操作能**同步进程**和**改变管程中的数据**
- **管程**实际上是一种能实现进程同步的特殊的**子程序（函数、过程）**的集合

# 第3章　进程的同步与通信

## 第五节　管程机制

一、管程的定义

### 3、管程的组成

- **名称**：该管程的标识
- **共享变量说明**：**局部**于管程的变量说明（包括特殊同步变量即条件变量）
- **一组过程**：对该数据结构进行操作的**程序段**（相当于临界区代码段）
- **初始化**：对局部于管程的数据设置初始值

# 第3章　进程的同步与通信

## 第五节　管程机制

### 一、管程的定义

### 4、管程的语法结构

```
Type 管程名称 = monitor
    共享变量说明语句
    procedure entry P1(…)
        begin    …    end;
    procedure entry P1(…)
        begin    …    end;
          :
    begin
        初始化语句
    end;
```

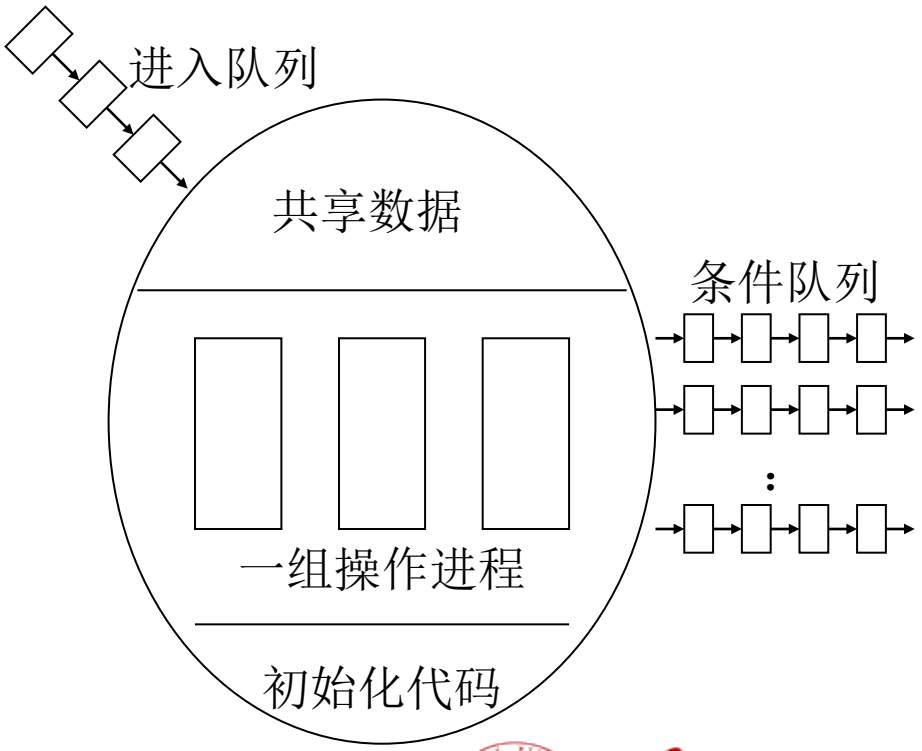
# 第3章　进程的同步与通信

## 第五节　管程机制

一、管程的定义

### 5、进程的互斥访问

- 进程访问临界资源，必须经过管程
- 管程每次只允许一个进程进入



# 第3章　进程的同步与通信

## 第五节　管程机制

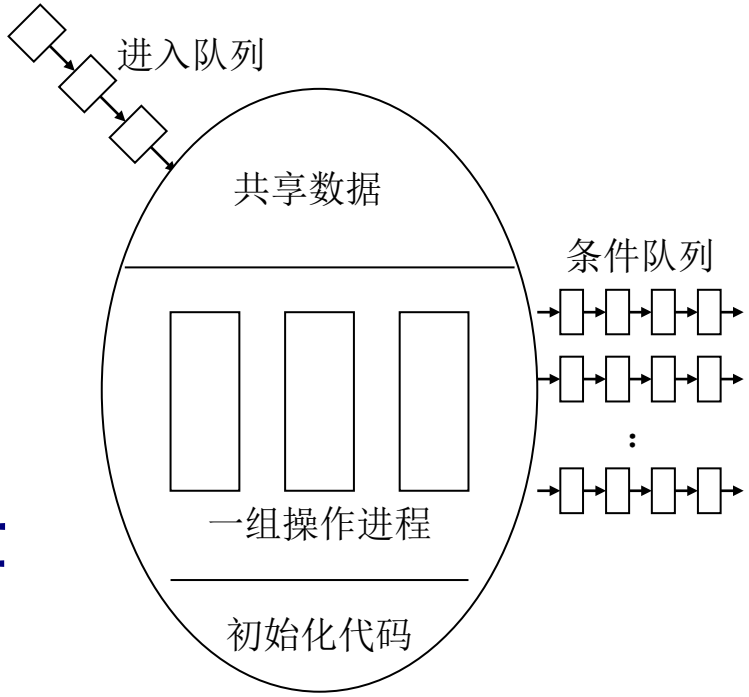
一、管程的定义

### 6、进程的同步

- 设置**条件变量**（*相当于互斥信号量*）

  ```
  var x: condition
  ```

- 当**临界资源**已被占用，执行x.wait，将进程挂在**x条件变量**的**阻塞等待队列**中
- 当临界资源已空闲，执行x.signal，从**x条件变量**的阻塞等待队列中，**唤醒**第一个进程



# 第3章　进程的同步与通信

## 第五节　管程机制

### 二、利用管程解决生产者—消费者问题

### 1、过程

- put(item)过程
  - 生产者将产品放入缓冲区
- get(item)过程
  - 消费者从缓冲区取得一个产品

# 第3章　进程的同步与通信

## 第五节　管程机制

二、利用管程解决生产者—消费者问题

### 2、变量

- 整型量in，out：初值均为0，in指示首空缓冲块序号，out指示首满缓冲块序号
- Buffer[0...n-1]：缓冲区
- 整形量count：满缓冲块数目
- 条件变量notfull，notempty：指示缓冲区已满、已空

# 第3章　进程的同步与通信

## 第五节　管程机制

### 二、利用管程解决生产者—消费者问题

#### 3、管程描述

```
Type PC = monitor
   var in, out, count: integer;
       buffer: array[0, …, n-1] of item;
       notfull, notempty: condition;
   procedure entry put(var nextp: item)
      begin
         if count >= n then notfull.wait
         buffer(in) := nextp;
         in := (in + 1) mod n;
         count := count + 1;
         if notempty.queue then
                notempty.signal
      end;
   procedure entry get(var nextc: item)
      begin
         if count <= 0 then
                notempty.wait
         nextc := buffer(in);
         out := (out + 1) mod n;
         count := count - 1;
         if notfull.queue then
                notfull.signal
      end;
begin
   in :=0; out := 0; count := 0;
end;
```

# 第3章　进程的同步与通信

## 第五节　管程机制

### 二、利用管程解决生产者—消费者问题

### 4、进程描述

```
procedure producer;        生产者进程
begin
    repeat
        生产一个产品nextp;
        PC.put(nextp);
    until false;
end;
```

```
procedure consumer;        消费者进程
begin
    repeat
        PC.get(nextc);
        消费一个产品nextc;
    until false;
end;
```

# 第3章　进程的同步与通信

## 第五节　管程机制

三、管程的优点

- 使用临界资源的进程进行调用时非常简单
- 进程结构清晰
- 易于查错

# 第3章　进程的同步与通信

## 第六节　进程通信（p67）

一、进程通信

- 进程通信是指进程之间的信息交换
- 采用信号量机制可以实现进程（低级）通信（如生产者—消费者），但其通信效率低，通信对用户不透明（用户必须自己编写访问临界资源的程序或管程，包括数据结构的设置、数据的传送、进程的互斥与同步等）——*是一种进程之间的低级通信*

# 第3章　进程的同步与通信

## 第六节　进程通信

一、进程通信

- **进程同步的三种方式：**

**1、发送进程阻塞，接收进程阻塞**

**2、发送进程不阻塞，接收进程阻塞**

**3、发送进程不阻塞，接收进程不阻塞**

# 第3章　进程的同步与通信

## 第六节　进程通信

一、进程通信

- **进程通信的四种类型：**

**1、共享存储器系统**（*无格式*）

**2、消息传递系统**（*有格式*）

**3、管道通信系统**（*相当于文件*）

**4、客户机-服务器系统**

# 第3章　进程的同步与通信

## 第六节　进程通信

二、共享存储器系统

- 进程间以**共享存储器**方式进行数据通信

1、基于**共享数据结构**的通信方式

**由用户程序**定义数据结构、申请内存，并**同步**各进程对该数据结构的访问

*如生产者—消费者问题*

# 第3章　进程的同步与通信

## 第六节　进程通信

二、共享存储器系统

**2、基于共享存储区的通信方式**

**系统设置一共享存储区，由系统同步各进程对该共享存储区的访问。**

**用户需要共享存储区时，到系统中去申请，系统分配一部分共享存储分区给用户并返回该分区的名字，相关进程按名共享该分区**

# 第3章　进程的同步与通信

## 第六节　进程通信

三、消息传递通信（p68）

- 进程间的数据交换以消息（message）为单位
- 操作系统直接提供一组命令（原语）实现通信

# 第3章　进程的同步与通信

## 第六节　进程通信

三、直接消息传递通信

1、特点：

- 发送进程直接将消息发送给接收进程，并将它挂在接收进程的消息缓冲队列上
- 接收进程从消息缓冲队列中取得消息

# 第3章　进程的同步与通信

## 第六节　进程通信

三、直接消息传递通信

### 2、消息通信命令

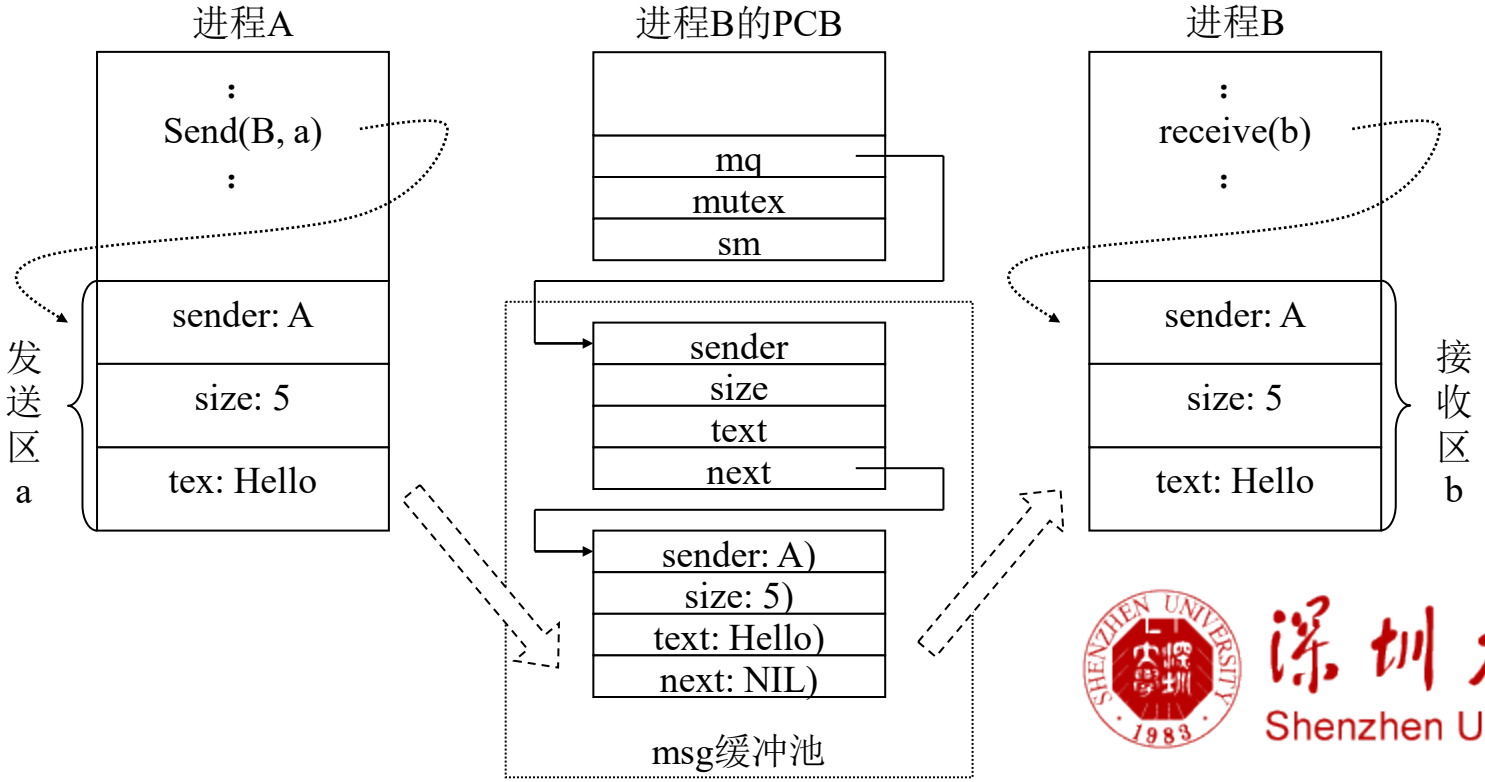
- 直接消息传递通信主要采用两种命令（原语）
- 发送消息原语Send(Receiver, message)
- 接收消息原语Receive(message)

# 第3章　进程的同步与通信

## 第六节　进程通信

### 三、直接消息传递通信

### 3、直接通信示意图



# 第3章　进程的同步与通信

## 第六节　进程通信

三、直接消息传递通信

### 4、直接通信机构

- mq：进程的消息队列首指针
- mutex：进程的消息队列互斥信号量，初值为1
- sm：同步信号量，用于消息队列中的消息计数，初值为0

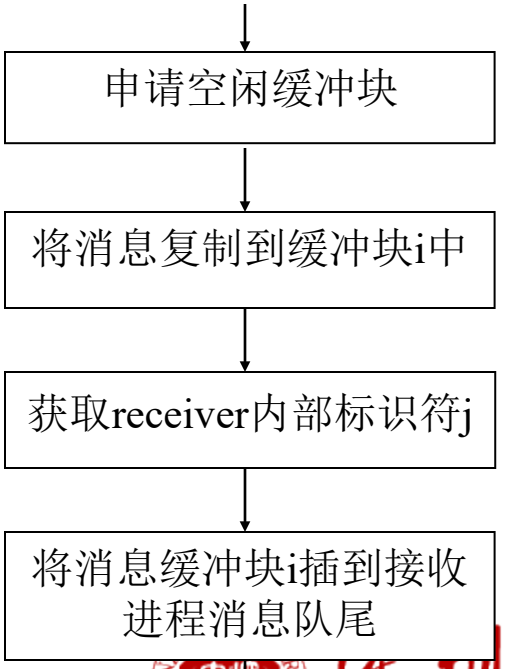
# 第3章　进程的同步与通信

## 第六节　进程通信

### 三、直接消息传递通信

### 5、直接通信Send原语

```
procedure Send(receiver, a)
begin
    getbuf(a.size, i);
    i.sender = a.sender;  i.size = a.size;
    i.text = a.text;          i.next = NIL;
    getid(PCB set, receiver, j);
    wait(j.mutex);
    insert(j.mq, i);
    signal(j.mutex);
    signal(j.sm);
end;
```



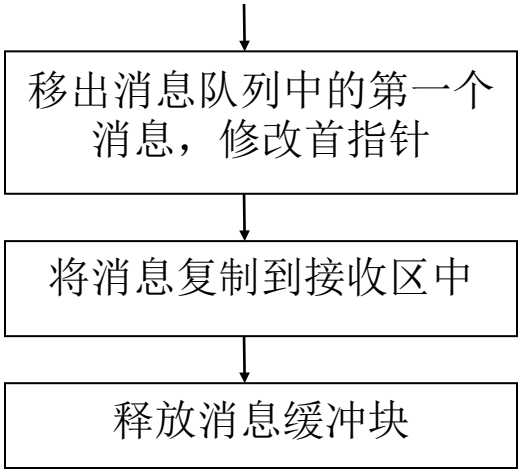
# 第3章　进程的同步与通信

## 第六节　进程通信

### 三、直接消息传递通信

### 6、直接通信Receive原语

```
procedure Receive(b)
begin
   j = internal name;
   wait(j.sm);  wait(j.mutex);
   remove(j.mq, i);
   signal(j.mutex);
   b.sender = i.sender ; b.size = i.size;
   b.text  = i.text;
   putbuf(i);
end;
```



# 第3章　进程的同步与通信

## 第六节　进程通信

三、直接消息传递通信

### 7、Window系统举例

SendMessage(hWnd, message, wParam, lParam)

*说明：*

HWND hWnd：进程名

unsigned message：消息

WORD wParam：参数

LONG lParam：参数

# 第3章　进程的同步与通信

## 第六节　进程通信

三、直接消息传递通信

### 7、Window系统举例

GetMessage(lpMsg,hWnd,*,*)

*说明:*

HWND hWnd：进程名

lpMsg ：消息结构

lpMsg = (hWnd,message,wParam,lParam,*,*)

# 第3章　进程的同步与通信

## 第六节　进程通信

四、间接消息传递通信

1、特点：

- 需要某种共享数据结构的实体作为中介
- 发送进程将发送给目标进程的消息暂存于该中介中
- 接收进程从中介中，取出发送给自己的消息
- 中介一般称为信箱(mailbox)，因此，间接消息传递通信也称信箱通信

# 第3章　进程的同步与通信

## 第六节　进程通信

四、间接消息传递通信

### 2、信箱通信命令

- 信箱创建命令Create(mailbox)
- 信箱撤消命令Delete(mailbox)
- 消息发送命令Send(mailbox, message)
- 消息接收命令Receive(mailbox, message)

# 第3章　进程的同步与通信

## 第六节　进程通信

四、间接消息传递通信

### 3、信箱种类

- **私用信箱**：由用户进程创建，其它进程只能向该信箱发送消息
- **公用信箱**：由操作系统创建，并提供给系统中的所有核准进程使用
- **共享信箱**：由某进程创建，并指明共享者名字；所有共享者和创建者进程都可以取走自己的消息

# 第3章　进程的同步与通信

## 第六节　进程通信

五 、管道通信（p68）

### 1、管道（pipe）

- 管道是一个打开的共享文件（pipe文件）
- 管道用于连接一批读进程和一批写进程，实现它们之间的通信。所有相关进程之间的同步都由系统控制
- 管道采用先进先出（FIFO）操作

# 第3章　进程的同步与通信

## 第六节　进程通信

五 、管道通信

### 3、管道通信特点

- 发送进程可以源源不断地从管道一端写入数据流；每次写入的信息长度可变；但一个时刻只能有一个发送进程写
- 接收进程可以从管道另一端读出数据，同样每次读出的信息长度可变；但一个时刻只能有一个接收进程读

# 第3章　进程的同步与通信

## 第六节　进程通信

六 、客户机-服务器系统 （p68）

- 其主要的实现方法分为三类：
  - 套接字（Socket）；
  - 远程过程调用和远程方法调用——是一个通信协议，允许运行于一台主机（本地）系统上的进程调用另一台主机（远程）系统上的进程，而对程序员表现为常规的过程调用，无需额外地为其编程。

# 第七节　线程 (p75)

一、线程的引入

- 在传统OS中，进程既是CPU调度和分配的基本单位，同时，进程又是拥有资源的独立实体

- 在处理（创建、撤消、调度）进程时，所费开销较大

# 第七节　线程

一、线程的引入

- 在OS中引入进程，主要是为了提高计算机系统的并发执行能力
- 将进程的两个属性分开，即让进程仅成为拥有资源的单位
- 而让线程成为调度和执行的基本单位，是为了进一步提高并发能力，并有利于多处理机系统的调度

# 第七节　线程

二、线程与进程的关系

- 线程是进程中的运行实体
- 一个进程可包含多个线程
- 一个进程中至少包含一个线程，称主线程
- 进程相当于线程的载体

# 第七节　线程

三、多线程OS中的进程

- 作为系统资源分配的单位
- 可包括多个线程
- 进程不是一个可执行的实体

# 第七节　线程

四、线程的属性

- **轻型进程：线程只拥有程序计数器、堆栈、TCB**
- **独立调度和分派的基本单位：线程切换开销少**
- **可并发执行：同一进程或不同进程的线程都可并发执行**
- **共享进程资源：内存空间、公用变量、信号量等**

# 第七节　线程

五、线程的状态

- **执行状态**：线程正获得CPU而运行
- **就绪状态**：一旦获得CPU就可运行
- **阻塞状态**：因等待某一事件而暂停

# 第七节　线程

六、线程的创建和终止

- **创建：新线程由正在执行的线程所创建**

  **线程创建一般通过创建函数（或系统调用）来实现，并提供相应的参数，如指向线程主程序的入口指针，堆栈的大小，优先级等**

- **终止：线程同样有生命期，当线程完成任务后，会自动终止**

  **线程终止还有可能由其它线程实现**

# 第七节　线程

七、线程的分类

- 内核支持线程：线程的处理均由内核实现，内核为线程保留TCB，并通过TCB感知线程

- 用户级线程：线程的处理不通过内核实现，线程的状态信息等全部存放在用户空间中，内核不能感知用户级线程

# 第七节　线程

九、线程控制

## 1、内核支持线程的实现

- 进程创建线程时，调用内核函数
- 内核支持的线程，其创建、调度都与仅有进程的创建与调度情况类似

# 第七节　线程

九、线程控制

## 2、用户级线程的实现

### (1)、运行时系统（p82）

用于管理和控制线程的函数（过程）的集合：

- 用于创建和撤消线程的函数
- 线程同步和通信的函数
- 实现线程调度的函数等

# 第七节　线程

九、线程控制

## 2、用户级线程的实现

### (2)、内核控制线程（p82）

- 内核控制线程又称为**轻型进程**（LWP），专门用于用户级线程与内核发生联系
- 每个用户级线程**只要连接到LWP上**，就具有了**内核支持线程**的所有属性

# 第七节　线程

九、线程控制

## 2、用户级线程的实现

## (2)、内核控制线程

设置LWP的优点：

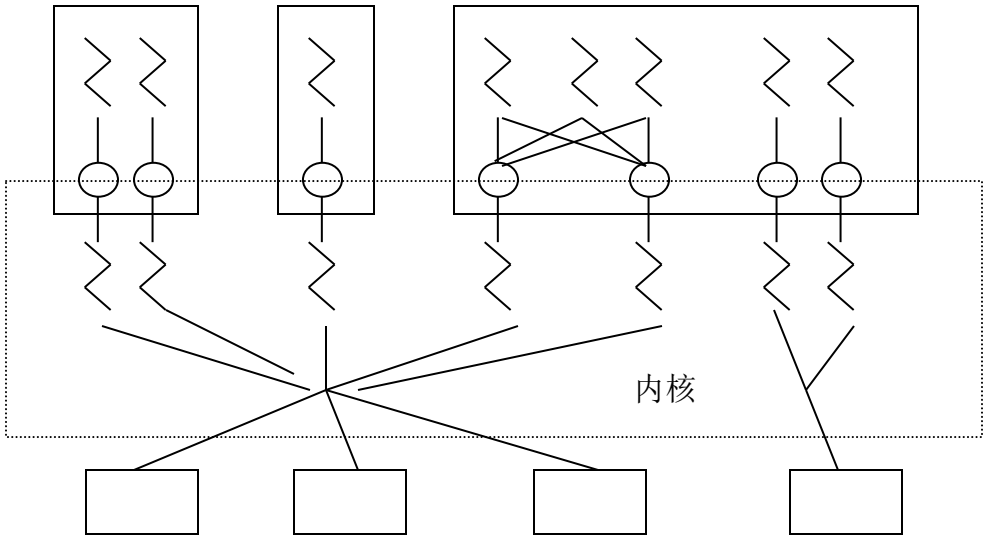
- 使系统保持尽量少的线程数目，节省系统开销
- 用户级线程数目不限

# 第七节　线程

## 九、线程控制

## 2、用户级线程的实现

## (2)、内核控制线程




用户级线程

轻型进程LWP

内核线程

CPU