# 进程

## 进程的引入

在OS中，进程是资源分配的基本单位，也是独立运行的基本单位。

## 程序的顺序执行

程序的顺序执行有如下特征：

(1) 顺序性

处理器的操作严格按照程序规定的顺序执行。

(2) 封闭性

程序一旦开始运行，其执行结果不受外界因素影响。

(3) 可再现性

只要程序执行的初始条件和执行环境相同，当程序重复执行时，将获得相同的结果。

## 程序的并发执行

程序的并发执行指若干个程序（或程序段）同时在系统中运行，这些程序（或程序段）的执行在时间上是重叠的，即一个程序（或程序段）的执行尚未结束，另一个程序（或程序段）的执行已经开始。

程序的并发执行与顺序执行相比有如下特征：

(1) 间断性

程序在并发执行时，由于它们共享资源或为完成同一项任务而相互合作，致使并发程序间形成了相互制约关系，这种相互制约关系导致并发程序具有"执行—暂停执行—执行"这种间断性的活动规律。

(2) 失去封闭性

程序在并发执行时，~~多个程序~~共享系统的各种资源，因而这些资源的状态将由多个程序来改变，致使程序的执行失去封闭性。

(3) 不可再现性

程序并发执行时，由于失去封闭性，也将导致失去其运行结果的可再现性。

在多道程序环境下，程序的并发执行破坏了程序的封闭性和可再现性，程序这种静态概念已经不能如实反映程序活动的特征，为此引入了进程的概念。

## 进程的定义及描述

### 1、定义

进程的定义有很多，各有侧重，但本质一样。

- ~~进程是程序在处理器上的一次执行过程。~~
- 进程是可以和别的进程并行执行的计算。
- 进程是程序在一个数据集合上的运行过程，是系统进行资源分配和调度的一个独立单位。

### 2、特征

进程具有以下几个基本特征：

① • 动态性

进程是程序在处理器上的一次执行过程，因而是动态的，会经历创建、调度、暂停、消亡。

② • 并发性

多个进程同时存在于内存中，能在一段时间内同时运行。

引入进程的目的是使程序能与其他程序并发执行，以提高资源利用率。

③ • 独立性

进程是一个能独立运行的基本单位，也是系统进行资源分配和调度的独立单位。

④ • 异步性

进程以各自独立的、不可预知的速度向前推进。

3、 • **结构特征**

为描述进程的执行过程，并使之正确运行，为每一个进程分配一个PCB（进程控制块，Process Control Block），PCB是进程存在的唯一标志。

从结构上看，每个进程都由程序段、数据段和一个PCB组成，三部分构成了进程映像，也称进程实体。

## 6 进程和程序的关系

- 进程是动态的，程序是静态的，进程是程序的执行。
- 进程是暂时的，程序是永久的。
- 进程和程序的组成不同。
- 一个程序可通过多次执行产生多个进程，一个进程可以通过调用执行多个程序，创建多个进程。
- 进程具有并行特性（独立性、异步性），而程序没有。

## 7. 进程的状态与转换

通常，一个运行中的进程至少可划分为5种基本状态。

① • 就绪状态 ready

已获得除处理器以外的所有资源，一旦获得处理器，即可立即执行。

② • 执行（运行）状态 running

进程获得必要的资源，正在CPU上运行。

③ • 阻塞（等待）状态 waiting

由于发生某事件而暂时无法执行下去（如等待I/O完成）。

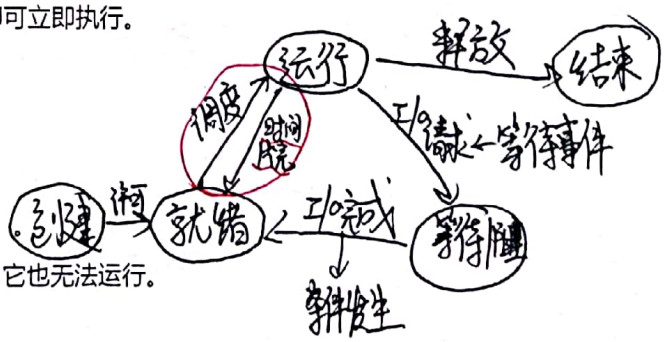
当进程处于阻塞状态时，即使把处理器分配给该进程，它也无法运行。

④ • 创建状态 new

进程正在被创建，尚未转到就绪状态。

⑤ • 结束状态 terminated

进程正在从系统中消失。




进程状态转换图

## 进程的控制

进程的控制负责进程的创建、进程的撤销、进程的状态转换等等。

16、PCB是什么？英文？ Control Process Block

## 8. 进程的阻塞与唤醒

阻塞原语（P原语/wait原语）的功能是将进程由执行状态变为阻塞状态。

唤醒原语（V原语/signal原语）的功能是将进程由阻塞状态变为就绪状态。

## 9. 进程通信

进程通信指进程之间的信息交换。

进程的互斥与同步就是一种进程间的通信方式，它们是两种低级进程通信方式。

高级进程通信方式可以分为3类：

1. 共享存储器系统
2. 消息传递系统
3. 管道系统

# 线程

## 10. 线程的引入

线程的引入提高了程序并发执行的程度，从而进一步提高了系统吞吐量。

为了让进程能并发执行，需要创建进程、进程切换和撤销进程，其中需要为进程分配资源、回收资源、保存现场信息等等，这些工作需要付出较多时空开销。

## 11. 线程的定义

- 线程是进程内的一个执行单元，比进程更小。
- 线程是进程内的一个可调度实体。
- 线程本身不能单独运行，只能包含在进程中，只能在进程中执行。

## 12. 进程和线程的比较

- 调度

  在传统的OS中，拥有资源和独立调度的基本单位都是进程。

  在引入线程的OS中，线程是独立调度的基本单位，进程是拥有资源的基本单位。

- 拥有资源

  不论是传统OS还是引入线程的OS，进程都是拥有资源的基本单位。

- 并发性

  在引入线程的OS中，不仅进程间可以并发执行，而且同一进程内的线程也可以并发执行。

- 系统开销

  线程的开销比进程的开销小。

# 同步与互斥

## 13. 两种形式的制约关系

区分互斥和同步只要记住，只要是同类进程即为互斥关系，不同类进程即为同步关系。

- 间接相互制约关系（互斥）

  这种制约关系源于多个同种进程需要互斥地共享某种系统资源。互斥设置在同种进程之间以达到互斥访问资源的目的。

- 直接相互制约关系（同步）

  这种制约关系主要源于进程间的合作。同步设置在不同进程之间以达到多种进程间的同步。

## 临界资源与临界区

- 临界资源

  同时仅允许一个进程使用的资源称为临界资源。

为了保证临界资源的正确使用，可以把临界资源的访问过程分成四部分：

- 进入区

  检查是否可以进入临界区；如果可以，通常设置标志以防止其它进程进入临界区。

- 临界区

  进程中用于访问临界资源的代码，又称临界段。

  每个进程的临界区代码可以不同。

- 退出区

  清除标志，允许其它进程进入临界区。

- 剩余区

  进程中除上述3部分以外的部分。

## 互斥的概念与要求

为了禁止两个进程同时进入临界区，软件算法与同步机构应遵循以下准则：

- 空闲让进

  没有进程处于临界区时，允许一个请求进入临界区的进程进入其临界区。

- 忙则等待

  当已有进程进入其临界区时，其它请求进入临界区的进程应等待。

- 有限等待

  访问临界资源的进程，应保证能在有限的时间内进入自己的临界区。

- 让权等待

  当一个进程因为某些原因不能进入自己的临界区时，应释放处理器给其它进程。

# 信号量机制

信号量机制的基本思想是在多个相互合作的进程之间使用简单的信号来同步。

## 信号量及同步原语

信号量可以是一个二元组$(s, q)$，OS利用信号量的状态对进程和资源进行管理。

- $s$是一个具有非负初值的整型变量，它表示系统中某类资源的数目，当其值大于0时，表示当前可用资源的数量；当其值小于0时，其绝对值表示系统中因请求该类资源而被阻塞的进程数量。
- $q$是一个开始状态为空的队列。

除初始化外，信号量的值仅能由wait操作（相当于申请资源）和signal操作（相当于释放资源）改变。

wait操作和signal操作在系统中一定是成对出现，但未必在一个进程中，可以分布在不同的进程中。

# 信号量的分类

整型信号量
记录型信号量
AND型信号量
信号量集

## ① 整型信号量

- 整型信号量是一个整型量s。
- 在进行wait操作时，若无可用资源，则进程持续对该信号量进行测试，存在"忙等"现象，未遵循"让权等待"准则。

其实可取值下限为0，因为wait中的判断

```
1  s=n;     // s初值为n，代表n个资源
2
3  wait(s)
4  {
5      // 没有资源
6      while(s<=0){
7          // do nothing
8      }
9      s--;
10 }
11
12 signal(s)
13 {
14     s++;
15 }
```

④ 信号量集：$s_i$：资源数，$t_i$：资源数下限，$d_i$：需求量

```
swait(s1,t1,d1,...){
    if(s1>=t1 and si>=ti){
        si = si-di;
    }
    else{
        阻塞至第一个si<ti的阻塞队列
    }
signal(si,di){
    for i=1 to n do
        si = si+di;
    唤醒置每个si的阻塞队列中阻塞进程
}
```

## ② 记录型信号量（资源信号量）

- 为解决整型信号量存在的"忙等"现象，添加了队列结构，用于链接所有等待该资源的进程。
- 该机制遵循"让权等待"准则
  - 当进程进行wait操作时，若此时无剩余资源可用，则进程自我阻塞，放弃处理器，并插入到等待队列中。
  - 当进程进行signal操作时，若队列中仍有等待该资源的进程，则唤醒队列中的第一个等待进程。

semaphore S. S.value=n; S.L 阻塞队列

```
1  s=n;     // s初值为n，代表n个资源
2
3  wait(s)
4  {
5      s--;
6      // 没有资源
7      if(s<0){
8          阻塞该进程，将其插入等待队列；block(S);
9      }
10 }
11
12 signal(s)
13 {
14     s++;
15     // 有进程在等待
16     if(s<=0){
17         唤醒队列q中第一个进程，将其插入就绪队列；wake(S);
```

AND型

```
③wait(S1,S2,S3,...,Sn){
    if(S1>=1 and S2>=1 and ... and Sn>=1)
    { S1-=1;
      ...
      Sn-=1;
    }
    else{
        将block该进程至第一个<1的Si.L中
    }
signal(S1,S2,...,Sn){
    S1+=1;
    S2+=1;
    唤醒置每个Si的所有阻塞进程
}
```

```
    }
}
```

进程同步 前趋
进程互斥
数量限制

## 18、信号量的应用

可以把S都理解为资源数量

### ① 实现进程同步

前趋

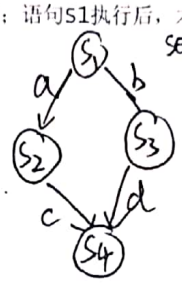
假设存在并发进程P1和P2，进程P1中有语句S1，进程P2中有语句S2，要求语句S1必须在语句S2之前执行。实现如下：

semaphore

同步：

```
semaphore s=0;    // s初值为0，代表刚开始是没有资源的；语句S1执行后，才有资源
                  
P1()
{
    // ...
    S1;
    signal(s);  // 释放资源
    // ...
}

p2()
{
    // ...
    wait(s);    // 申请资源
    S2;
    // ...
}
```




前趋：

semaphore a,b,c,d=0,0,0,0

P1:{ S1; signal(a); signal(b)}

P2:{ wait(a); S2; signal(c);}

P3:{ wait(b); S3; signal(d);}

P4:{ wait(c); wait(d); S4;}

### ② 实现进程互斥

假设存在进程P1和P2，两者有各自的临界区，但系统要求同时只能有一个进程进入临界区。实现如下：

比如生产者消费者

semaphore

```
semaphore s=1;    // s初值为1，表示资源数量为1

P1()
{
    // ...
    wait(s);    // 申请资源
    P1临界区
    signal(s);  // 释放资源
    // ...
}

P2()
{
    // ...
    wait(s);    // 申请资源
    P2临界区
    signal(s);  // 释放资源
    // ...
}
```

③实现数量限制 ← 互斥的实现版互斥是最多1个，数量限制是最多N个

# 19. 经典同步问题

## 生产者-消费者问题

生产者-消费者问题是著名的同步问题。它描述的情况是一组生产者向一组消费者提供产品，它们共享一个有界缓冲区，生产者向其中投入产品，消费者从其中取出产品。

这个问题是多个相互合作进程的抽象。比如，在输入时，输入进程是生产者，计算进程是消费者；在输出时，计算进程是生产者，打印进程是消费者。

为解决这一问题，应当设置两个同步信号量：empty表示空缓冲区数量，其初值为n；full表示满缓冲区数量（即产品数量），其初值为0。此外还需设置一个互斥信号量mutex，其初值为1，以保证多个生产者或多个消费者互斥访问缓冲池。

分析一下：

- 肯定是在有产品的情况下消费者才能消费，这个制约关系是一个同步，这个用full变量来实现。
- 缓冲区容量是有限的，即生产者不可以无限制生产，这个用empty变量来实现。
- 同类进程（生产者和另一个生产者，消费者和另一个消费者）访问缓冲区，这是个互斥关系，用mutex变量来实现。

  生产者和消费者之间也有互斥关系吧？比如在生产者进行运行至临界区代码时，消费者则不能运行其临界区代码。生产者和消费者都是读者写者问题中的写者，这是读者写者问题中写者间的互斥关系：同一时刻只能有一个写者写数据区。

```
1  Semaphore full=0;     // 刚开始满缓冲区的数量（即产品数量）为0
2  Semaphore empty=n;    // 刚开始空缓冲区的数量是n（即缓冲区的容量）
3  Semaphore mutex=1;    // 缓冲区在同一时刻只能被1个人使用，刚开始无人使用缓冲区
4
5  Pruducer()
6  {
7      wait(empty);      // 申请空缓冲区，自动更新空缓冲区数量
8      wait(mutex);      // 申请访问缓冲区权限
9      生产商品，放入缓冲池;
10     signal(mutex);    // 释放访问缓冲区权限
11     signal(full);     // 更新满缓冲区数量，增加一个满缓冲区
12 }
13
14 Consumer()
15 {
16     wait(full);       // 申请满缓冲区，自动更新满缓冲区数量
17     wait(mutex);      // 申请访问缓冲区权限
18     消费商品;
19     signal(mutex);    // 释放访问缓冲区权限
20     signal(empty);    // 更新空缓冲区数量，增加一个空缓冲区
21 }
```

请注意：

wait(empty)/wait(full)与wait(mutex)的顺序不能颠倒，必须先对资源信号量进程P操作，再对互斥信号量进行P操作，否则会导致死锁。（若某时刻缓冲区已满，而生产者先通过wait(mutex)得到缓冲池访问权限，再wait(empty)，由于此时缓冲区已满，empty=0，导致wait(empty)失败，生产者进程无法继续推进，始终掌握缓冲池访问权无法释放，即"忙等"，因而消费者进程无法取出产品，导致死锁。）而signal(full)/signal(empty)与signal(mutex)的顺序则没有要求，其顺序可以颠倒。

在生产者和消费者唯一的情况下，互斥信号量mutex可以去掉。

只要有多个同类进程，就一定需要互斥信号量（比如多个消费者进程都在使用empty信号量）；若同类进程只有一个，则记录型信号量即可以完成进程同步。换句话说，互斥信号量就是给同类进程准备的。

## 读者-写者问题

现有一个许多进程共享的数据区，有一些只读取这个数据区的进程（读者）和一些只往这个数据区写数据的进程（写者），它们需满足以下条件：

- 任意多个读者可以同时读这个数据区。
- 同时只能有一个写者可以写数据区。
- 如果一个写者正在写数据区，禁止任何读者和写者读或写该数据区，即一个写者进程和任何进程都互斥。

需要分情况实现该问题：读者优先、公平情况和写者优先。

## 读者优先

读者优先即：若有读者在读，新来的读者可立即开始读，新来的读者需要等待所有读者读完之后才能写。

```
1  int readerCount=0;  // 读者数量，大于0时写者不可以写
2  semaphore wmutex=1;  // 写者和其他进程互斥使用数据区
3  semaphore rmutex=1;  // 读者间互斥访问readerCount
4
5  Reader()
6  {
7      wait(rmutex);              // 申请readerCount使用权
8      // 第一个读者
9      if(readerCount==0){
10         wait(wmutex);          // 阻止写者写数据区
11     }
12     readerCount++;             // 更新读者数量
13     signal(rmutex);            // 释放readerCount使用权，允许其它读者使用
14
15     读;
16
17     wait(rmutex);              // 申请readerCount使用权
18     readerCount--;             // 更新读者数量
19     // 最后一个读者
20     if(readerCount==0){
21         signal(wmutex);        // 允许写者写数据区
22     }
23     signal(rmutex);            // 释放readerCount使用权，允许其它读者使用
24 }
25
26 Writer()
27 {
28     wait(wmutex);              // 申请数据区使用权，阻止其他进程使用
29     写;
30     signal(wmutex);            // 释放数据区使用权，允许其它进程使用
31 }
```

## 哲学家进餐问题

5个哲学家围绕一张圆桌而坐，桌子上放着5根筷子，每两个哲学家之间放1根筷子。哲学家只能思考或者进餐，进餐时需要同时拿起他左边和右边的两根筷子，思考时则将两根筷子放回原处。

可以使用信号量数组解决该问题，假设哲学家按照编号逆时针围桌而坐，0号哲学家左手边筷子为0号筷子，右手边筷子为1号筷子，以此类推。

以下这种方法会造成死锁。

```
1  fork[5]={1,1,1,1,1};     // 5根筷子是否可被使用
2  philosopher(int i)       // i=0,1,2,3,4,
3  {
4      wait(fork[i%5]);         // 拿起左手边筷子
5      wait(fork[(i+1)%5]);     // 拿起右手边筷子
6      进餐;
7      signal(fork[i%5]);       // 放下左手边筷子
8      signal(fork[(i+1)%5]);   // 放下右手边筷子
9  }
```

可由如下解决方法：规定奇数号哲学家先拿左手边筷子，偶数号哲学家先拿右手边筷子。实现如下：

```
1  fork[5]={1,1,1,1,1};     // 5根筷子是否可被使用
2  philosopher(int i)       // i=0,1,2,3,4,
3  {
4      if(i%2!=0){
5          wait(fork[i%5]);         // 拿起左手边筷子
6          wait(fork[(i+1)%5]);     // 拿起右手边筷子
7          进餐;
8          signal(fork[i%5]);       // 放下左手边筷子
9          signal(fork[(i+1)%5]);   // 放下右手边筷子
10     }
11     else{
12         wait(fork[(i+1)%5]);     // 拿起右手边筷子
13         wait(fork[i%5]);         // 拿起左手边筷子
14         进餐;
15         signal(fork[(i+1)%5]);   // 放下右手边筷子
16         signal(fork[i%5]);       // 放下左手边筷子
17     }
18 }
```

# 管程机制

用信号量机制可以实现进程间的同步和互斥，但由于信号量的控制分布在整个程序中，其正确性分析很难，使用不当还有可能导致进程死锁。

管程定义了一个数据结构和能为并发进程所执行的一组操作，这组操作能同步进程和改变管程中的数据。

由定义可知管程由三部分组成：

- 局部于管程的的共享数据结构说明
- 操作这些数据结构的一组过程
- 对局部于管程的数据设置初始值语句

理发师问题

一位理发师，一把理发椅，n个供顾客等候的凳子。

若无顾客，理发师在椅上睡觉。

一个顾客到，先叫醒理发师；若正在理发且有空凳子则坐下，无凳子则离开。

```
int waiting=0;          → 顾客数量，包括正在理发的，最大n+1   （如果没有该变量，无法实现人满时顾客离开）
semaphore mutex=1;      → 用于互斥操作waiting
semaphore bchair=1;     → 理发椅
semaphore wchair=n;     → 凳子                                  6个
semaphore ready=0;      → 顾客/理发师是否准备好，0:未准备好
semaphore finish=0;     → 理发师完成一次理发

barber(){
  while(true){
      wait(ready);      //等待顾客在理发椅上准备好
      理发
      signal(finish);   //理发结束，提醒顾客离开
  }
}
customer(){             → 内部：Java缩进：18行
    wait(mutex);        //申请waiting变量
    if(waiting<=n){
        waiting++;      //顾客人数加1
        signal(mutex)   //释放waiting变量
    }
    else{
        V(mutex)
        signal(mutex);  //释放
        离开;           //进程结束
    }
    wait(wchair);       //申请获得空凳子
    wait(bchair);       //找理发椅
    signal(wchair);     //释放凳子
    wait
    V(ready);           //告诉barber我准备好了
    P(finish);          //等待理好
    V(bchair);          //腾出理发椅
    P(mutex);
    waiting--;
    V(mutex);
}
```

直接

1. 找到凳子后找椅子
2. 找到椅子后释放两个：凳子和准备好
3. 理好后释放椅子和waiting.

离开

1. 进程的定义
2. 进程的基本特征(4个) 考 "并动异独并"
3. 进程的结构特征 考
4. 程序顺序执行的特点(3个)
5. 程序并发执行的特点(与顺序执行对比3个)
6. 进程与程序的关系 考
7. 进程的状态转换 考、英文、转换条件
8. 进程的阻塞与唤醒
9. 进程的通信(2低3高)
10. 线程引入的原因
11. 线程定义
12. 进程和线程的定义对比 考 "资调并开"
13. 同步与互斥的目的和判断 临界区
14. ~~互斥的~~ 临界资源是什么,临界资源访问过程的划分(4个)
15. 互斥的要求与概念
16. PCB是什么?英文(4个)
17. 信号量的分类(4种) 考
18. 信号量的应用(3种) 考
19. 经典问题(生消、读写、哲学)