# 第三章，进程管理

## 3.1 进程的概念

### 3.1.1 程序的并发执行

1.程序

程序描述计算机所要完成的具体**独立功能**的，并在时间上**按照严格次序**前后相继的计算机操作序列的**计算机操作序列集合，是一个静态的概念**。

2.程序执行的分类：

程序的执行可以分为两类：

1. 程序的**顺序执行**和**并发执行**

3.处理机与CPU的区别：

**处理机**

处理机是计算机系统中存储程序和数据，并按照程序规定的步骤执行指令的部件。程序是描述处理机完成某项任务的指令序列。指令则是处理机能直接解释、执行的信息单位。**处理机包括中央处理器（cpu），主存储器,输入-输出接口。**处理机加接外围设备就构成完整的计算机系统

**CPU**

中央处理器（CPU，Central Processing Unit）是一块超大规模的集成电路，是一台计算机的运算核心（Core）和控制核心（ Control Unit）。它的功能主要是解释计算机指令以及处理计算机软件中的数据。

4..顺序执行的特点

1.顺序性

2.封闭性（不受外界因素影响）

3.可再现性(顺序执行的最终结果可再现且与它的执行速度无关)

5.并发执行

1).什么是并发执行？

一组在逻辑上互相独立的程序或程序段在执行过程中，其执行时间在客观上互相重叠，即一个程序段的执行尚未结束，另一个程序段的执行已经开始的这种执行方式。

2).两个语句S1、S2并发执行的条件

将Si语句划分为两个变量的集合：R(Si),W(Si);

R(si)是进行读写的变量。

W(si)是进行修改和访问的变量。

如果满足：

(1).R(S1)交集W(S2)={空集}

(2).R(S2)交集W(S1)={空集}

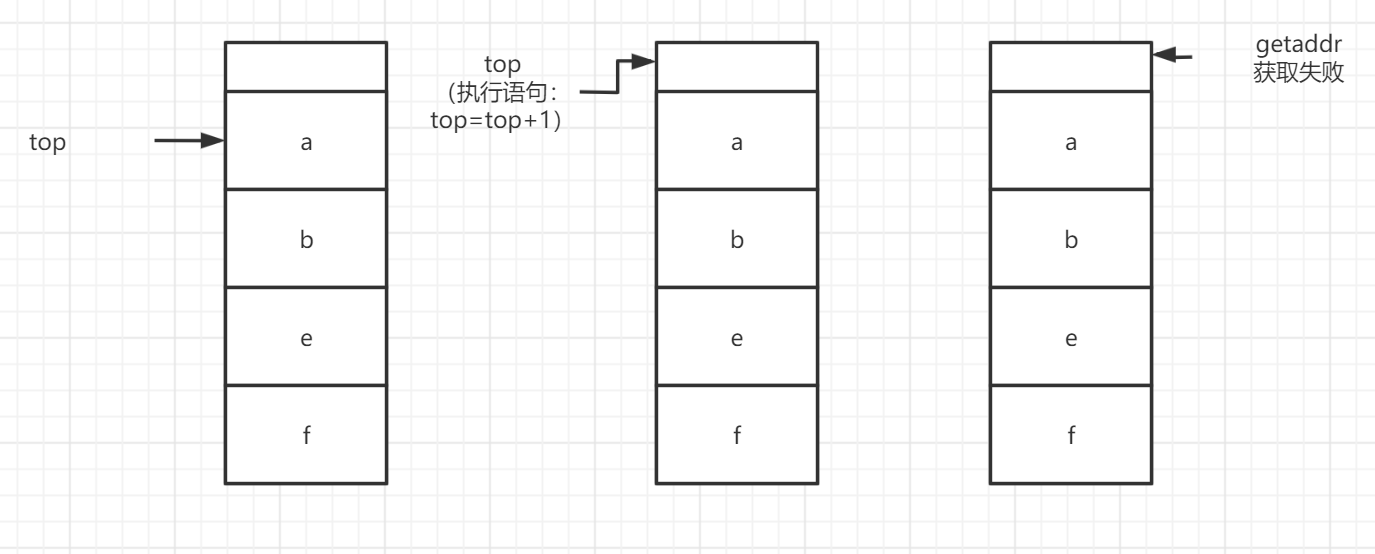
(3).W(S1)交集W(S2)={空集}

则语句S1、S2可并发执行。

3).并发执行会引起的影响

(1).充分利用系统资源，从而提高了系统处理能力，这个是并发执行的优点。

(2).如果并发执行不按照特定的规则和方法进行资源共享和竞争，则其的结果将失去封 闭性和可再现性。



堆栈的取数和存数的过程

上述出现错误的原因是两段程序共享资源堆栈。

6. 并发性与并行性区别：

并发性：两个或多个事件在同一时间间隔内发生

并行性：两个或多个事件在同一时刻发生（一定发生在多核处理器上）

### 3.1.2 进程的定义

1.进程的定义

能描述程序的执行过程且能用来共享资源的基本单位。简单的说，程序的一次运行。

2.进程和程序的区别

1.进程是动态的概念强调过程，程序是静态的概念。

2.进程具有并发特征，而程序没有

3.进程是竞争计算机系统资源的基本单位，从而其并发性受到系统自己的制约。这里的制约即是对进程独立性和异步性的限制。

4.不同的进程可以包含同一个程序，只要该程序所对应的数据集不同。一个进程在其生命周期内可以调用多个程序，一个程序可以创建多个进程

3.作业与进程的区别和关系：

(1) 作业是用户向计算机提交任务的**任务实体**。而进程则是完成用户任务的**执行实体**，是资源分配的基本单位。

(2) 一个作业可由一个或多个进程组成。

(3) 作业的概念主要用在批处理系统中，而进程的概念则用在几乎所有的多道系统中。

## 3.2 进程的描述

引言：

系统需要有描述进程存在和能够反映其变化的物理实体，即进程的静态描述。

静态描述：

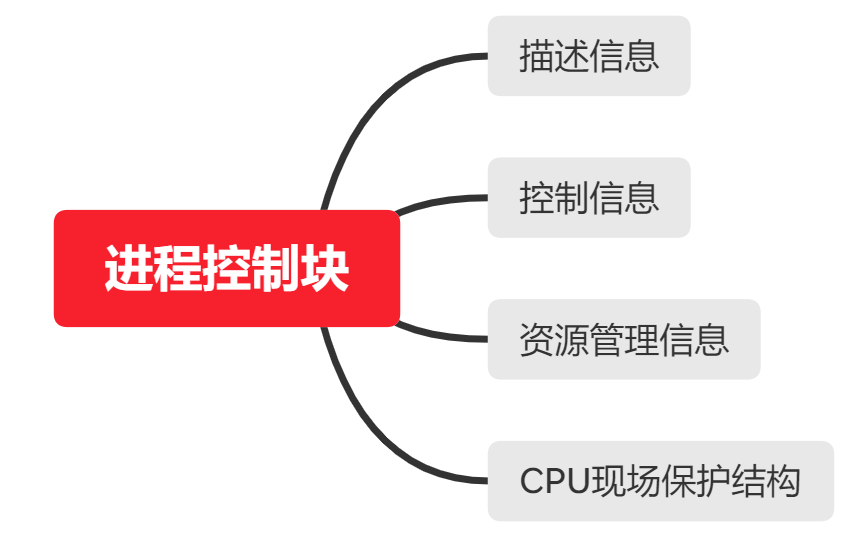
由三大部分组成:

进程控制块PCB：全部或部分常驻内存

程序段

数据结构集：外存中，执行时调入内存

### 3.2.1 进程控制块PCB



1.描述信息

1.进程名或进程标识说明

2.用户名或用户标识号

3.家族关系

2.控制信息

1.当前的状态

2.进程优先级

3.程序开始地址

4.各种计时信息

5.通信信息

3资源管理信息

1.占用内存大小及其管理用数据结构指针

2.对换和覆盖用的相关信息

…………

4.CPU现场保护

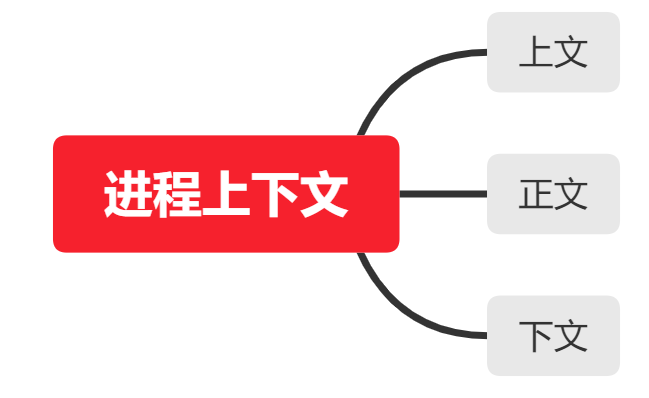
当前进程因等待某个事件而进入等待状态或因某种事件发生被终止在处理机上运行时，为了以后该进程能继续运行，需要保护当前进程的状态。

### 3.2.2 进程上下文

引言：

进程上下文实际上是一个与进程切换和处理机状态发生交换有关的概念。因此，进程上下文包括了每个进程的指令和数据，指令寄存器、堆栈和状态寄存器中的数据。

1.从上下文结构组成的角度来说分为：



上文：已执行过的进程指令和数据在相关寄存器与堆栈中的内容称为上文。

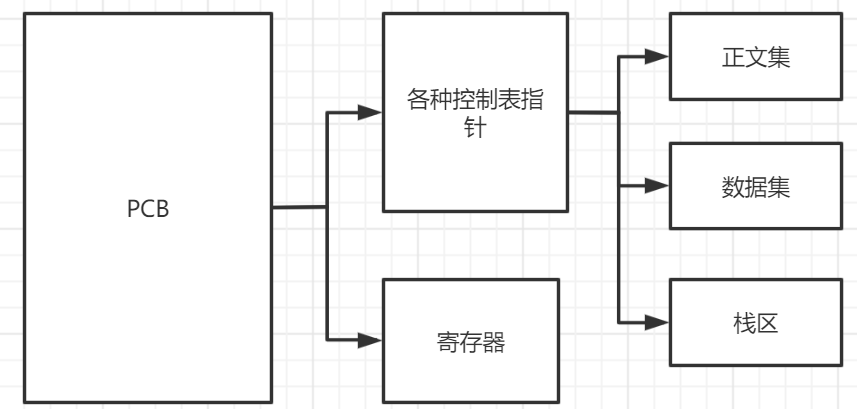
正文：正在执行的进程指令和数据在相关寄存器与堆栈中的内容称为正文。

下文：待执行的进程指令和数据在相关寄存器与堆栈中的内容称为下文。

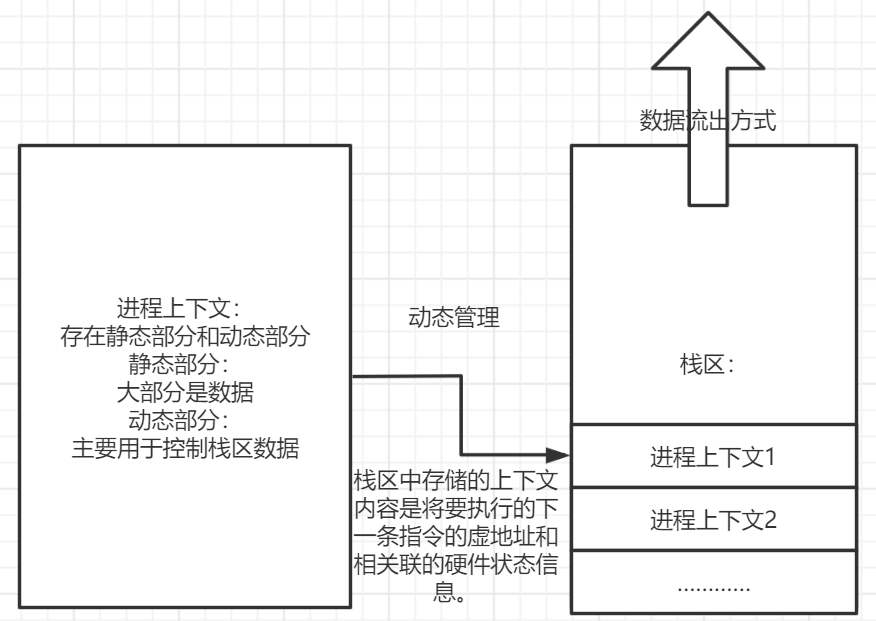
2.相关的寄存器和栈区很重要

例如，没有程序计数器pc和程序状态字寄存器PSW，CPU将无法知道下一条执行命令的地址和控制有关的操作。

3.进程上下文结构分析



进程上下文结构



进程上下文的组成

4.进程上下文切换

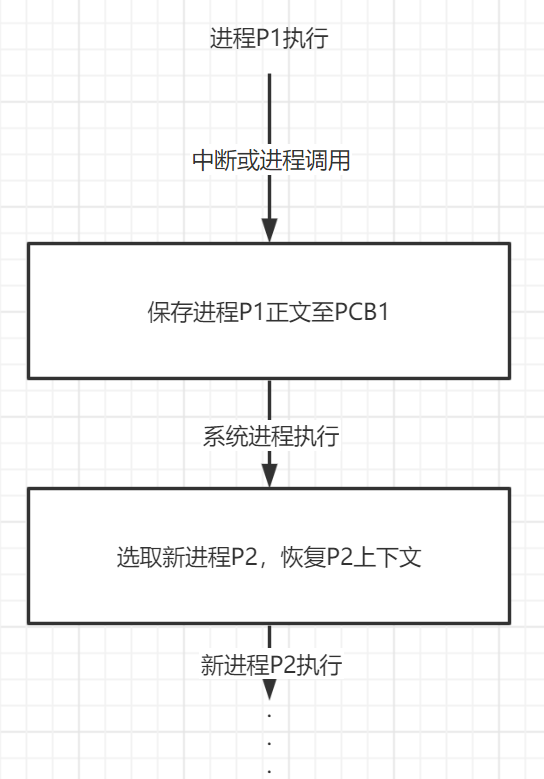
1).进程上下文切换是在不同的进程之间的而不是在同一个进程内。

2).进程上下文切换过程三大步：

(1).第一部分为保存被切换进程的正文部分至相关存储区，例如PCB中。

(2).操作系统进程中有关调度和资源分配程序执行，并选取新的进程。

(3).将被选中进程的原来被保存的正文部分从相关储存区中取出，送至相关寄存器与堆栈中激活和执行。

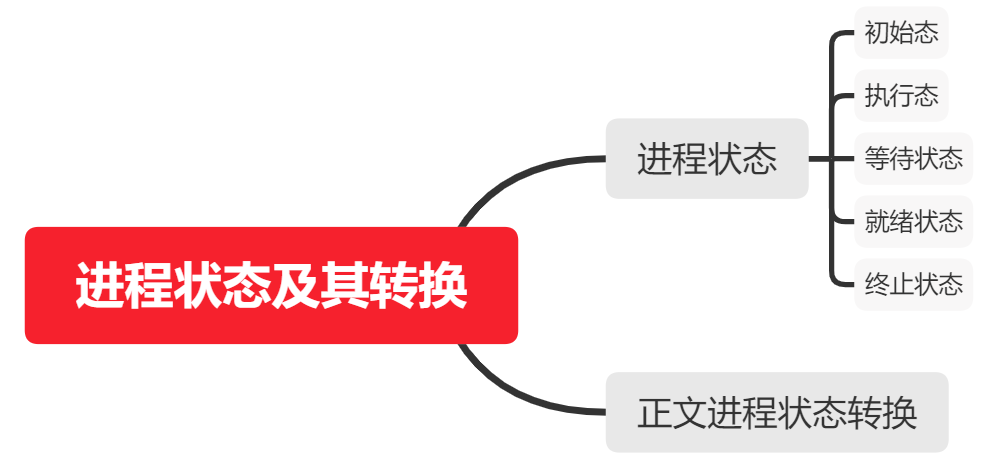


进程上下文的切换过程

1.进程空间

任何一个进程都有一个地址空间，该空间称为进程称为进程空间或者虚空间。进程空间还被划分为用户空间和系统空间。

## 3.3 进程状态及其转换



### 3.3.1 进程状态

1.进程状态分类：

1.初始态

2.执行态

3.等待状态

4.就绪状态

5.终止状态

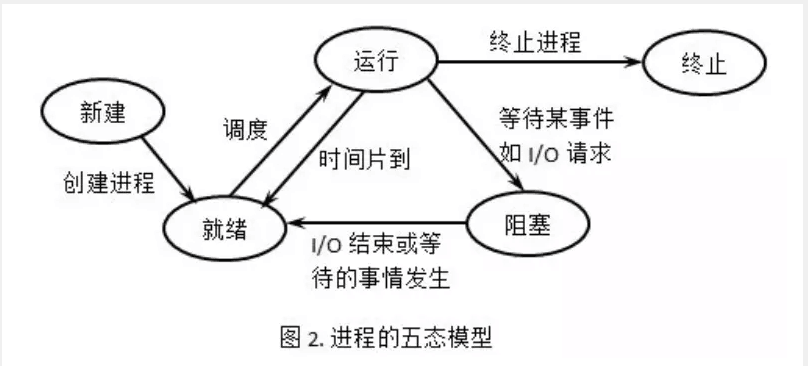
2.补充

1.在有些系统中就绪状态分为**内存就绪状态**和**外存就绪状态**。内存就绪状态可以直接执行，外存就绪状态下的进程要想执行必须先转化为内存就绪状态才行。

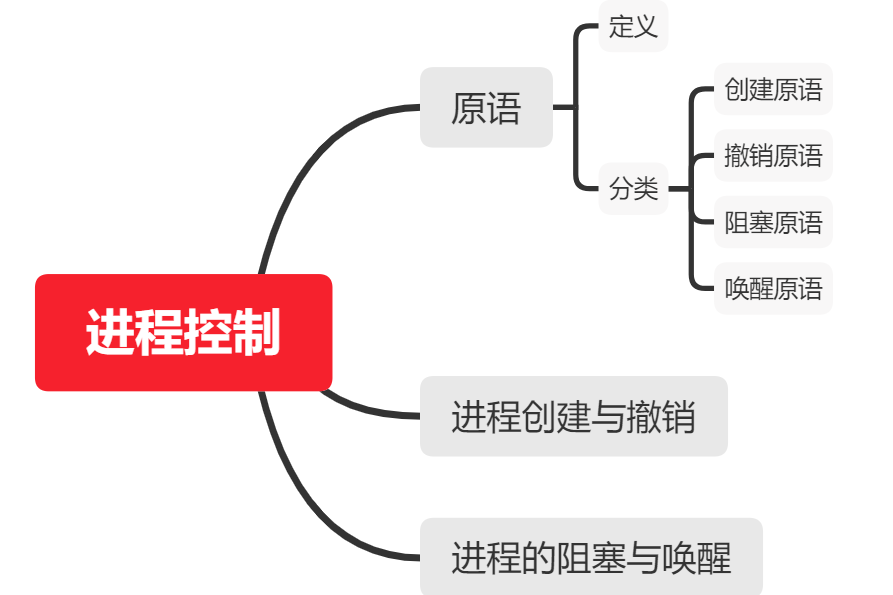
2.在单CPU系统中，处于执行状态的进程只有一个。

3.在某些操作系统中，执行状态还可以分为：**用户执行状态**和**系统执行状态**。之所以要划分执行状态，是因为要把用户执行程序和系统程序区分开来。

### 3.3.2 进程状态转化



## 3.4 进程的控制



1.原语定义

在操作系统中他是一个不可分割的基本单位。把系统下执行的某些具有特定功能的程序段称为原语。

2.原语的分类

1.创建原语

2.撤销原语

3.阻塞原语

4.唤醒原语

### 3.4.1 进程创建与撤销

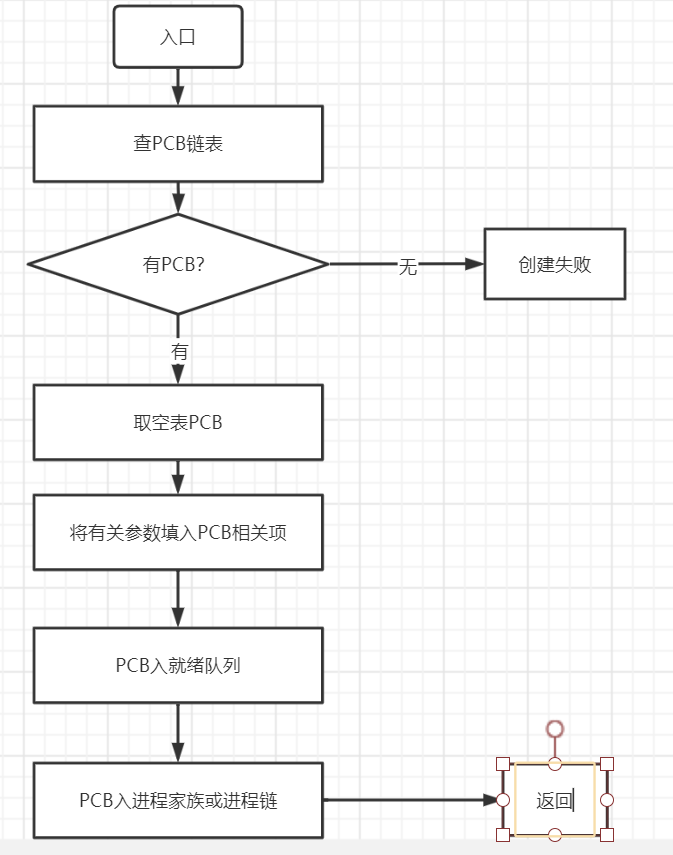
1.进程创建方式

1.由系统模块统一建立。例如在批处理系统中，由操作系统的作业调度程序为用户作业创建相应的进程以完成用户作业所要求的功能。

2.由父进程创建子进程。子进程继承父进程的所有资源。

2.创建原语

无论是以何种方式创建进程都需要调用创建原语来实现。



创建原语流图

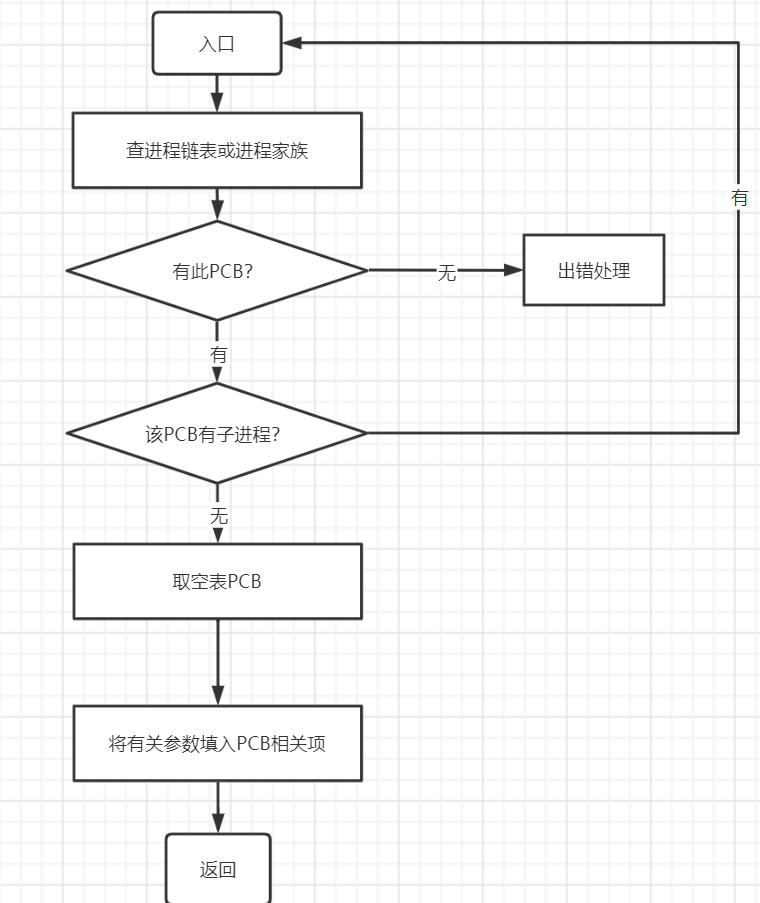
3.进程撤销原因

1.该进程已完成所要求的功能而正常终止。

2.由于某种错误导致非正常终止。

3.祖先进程要求撤销某个子进程。

4.撤销原语

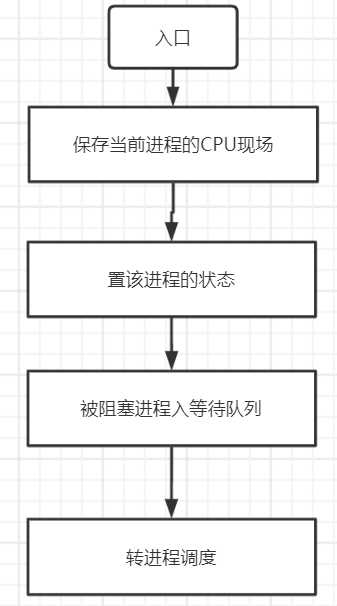


撤销原语流图

### 3.4.2 进程的阻塞与唤醒

1.阻塞原语

阻塞原语在一个进程期待某一事件发生，但发生条件还不具备时，该进程自己调用阻塞原语用来阻塞自己。阻塞原语在阻塞进程时，会先中断处理机和保存该进程的CPU现场。



阻塞原语

2.唤醒原语

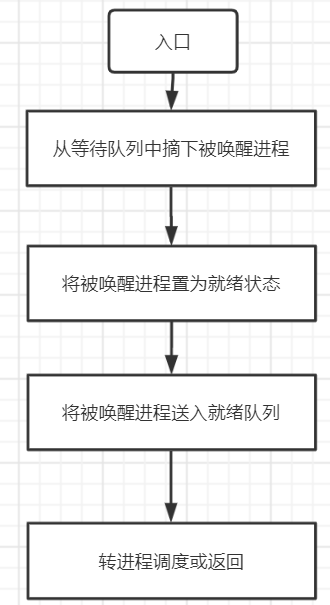
1.唤醒一个进程的方式：

（1）系统进程唤醒

（2）事件发生唤醒

因此唤醒原语可被系统进程调用，也可被事件发生进程调用。

2.调用唤醒原语的进程又称为唤醒进程。



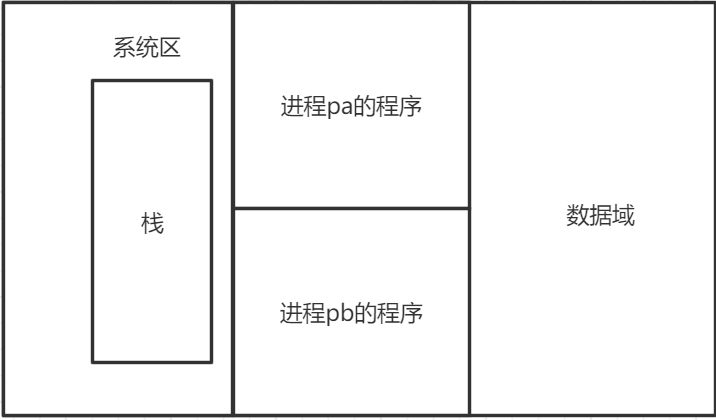
唤醒原语实现框图

## 3.5 进程互斥

### 3.5.1 资源共享所引起的制约

1.临界区：

我们把**不允许多个并发程序交叉执行的一段程序**称为临界区。



Pa和pb进程共享内存栈示例

2.间接制约

间接制约是一种现象：由于共享某一公有资源而引起的在临界区内不允许并发程序交叉执行的现象。

系统根据不同的公有数据划分为不同的集合，这些集合被称为类。

3.互斥

互斥是一种行为。由于间接制约导致计算机不允许两个以上的共享公有资源的并发进程进入临界区。

4.互斥执行的准则

1.各个并发进程享有平等地，独立地竞争公有资源的权利。

2.并发进程中某个进程没有进入临界区时，他不会阻止其他进程进入。

3.临界区只允许一个进程进入。

4.进程申请进入临界区中，如果获得权限，应该在有限的时间内进入，这个准则是为了防止死锁

### 3.5.2 互斥的加锁实现

实现并发进程互斥的方式1：加锁

实现代码：

Lock(key[S])

<临界区>

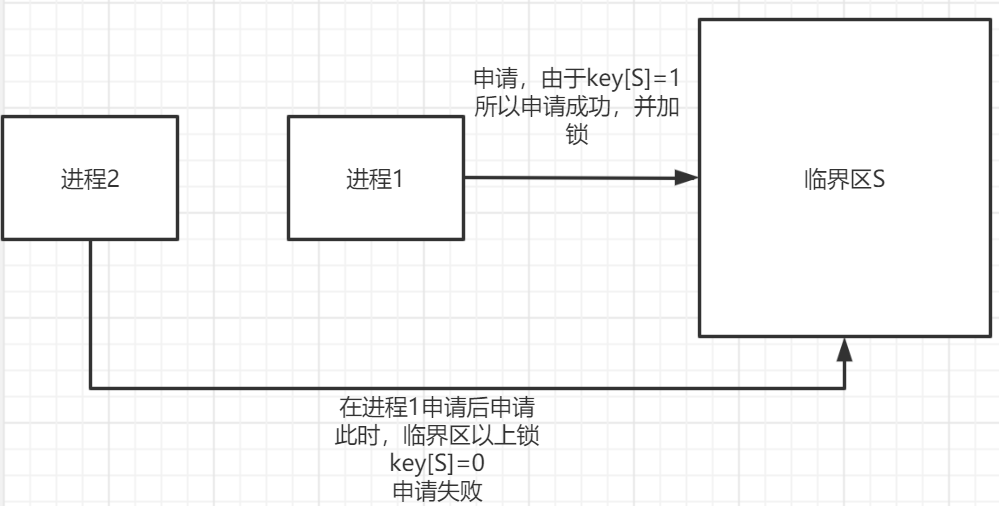
Unlock(key[S])

解析代码：

Lock是加锁的关键词，key[S]表示类名为S的临界区的类名。

Key[S]=1表示此临界区可用，key[S]=0表示此临界区不可用。

Unlock是解锁的操作。



加锁原理流程图

加锁实现互斥的方式存在一些缺陷：

1.如果某一组并发进程中的进程数较多，占用临界区的时间就偏多，其他组并发进程需要请求多次对临界区的使用，这样会加大开销

2.如果某进程代码如下：

A：Lock(key[S])

<临界区>

Unlock(key[S])

Goto A

由于并发进程请求临界区的方式是定时请求，而在A进程后有一句goto A,就会导致出现A进程一直调用临界区的情况。

### 3.5.3 信号量和P,V原语

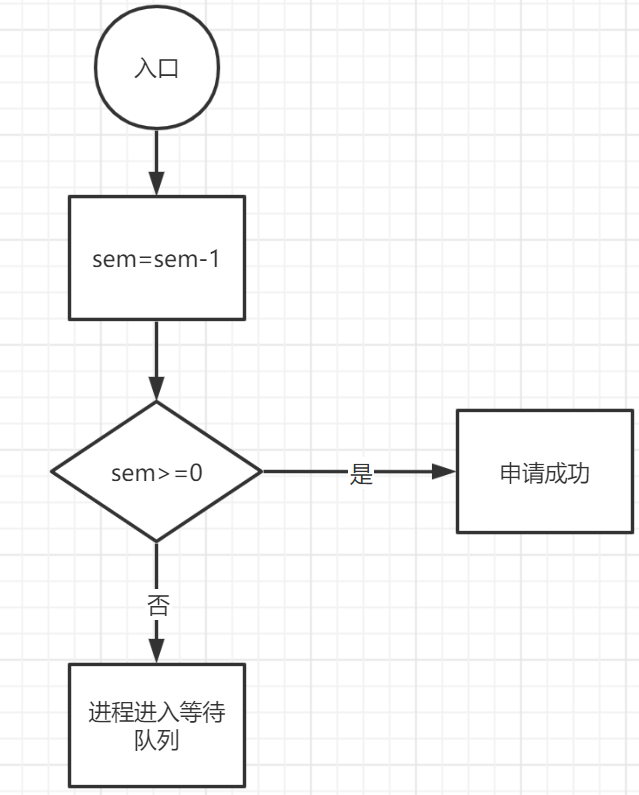
1.信号量（sem）(这里介绍的是公有信号量，在3.6.2会进一步介绍私有信号量)

信号量管理相应临界区的公有资源，他代表可用资源的实体。

Sem>0时，sem表示临界区中剩余的资源数量；sem<0时，sem代表等待队列中进程的数量。

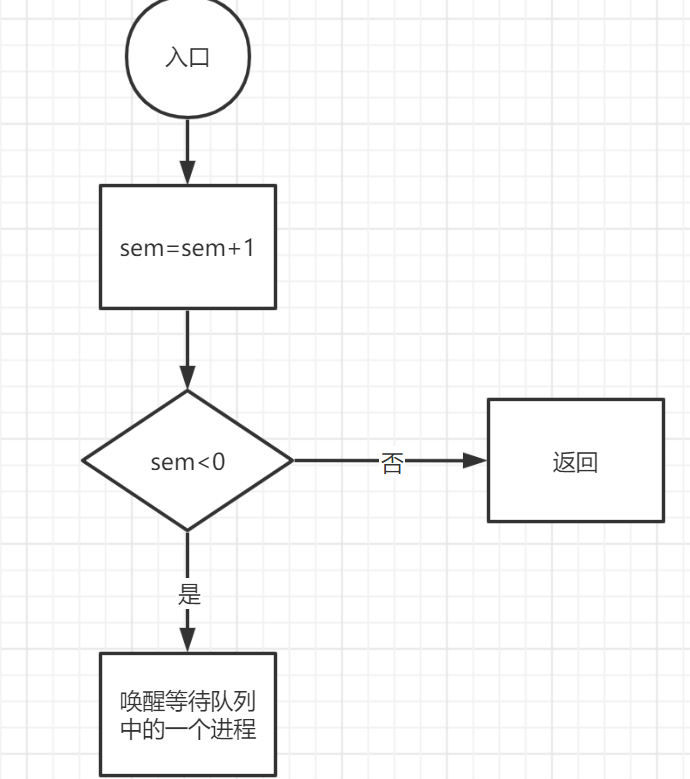
P、V原语

P原语：P原语相当于一次请求临界区，如果临界区没有空位就就将此并发进程拉入等待队列中，等待唤醒。



P原语操作功能图

V原语：释放占用临界区的并发进程资源，给信号量（sem）+1，如果小于0，则唤醒一个等待队列中的进程。



V原语操作功能

### 3.5.4 用P,V原语实现进程互斥

实现三部曲：

1.声明公有信号量

2.给公有信号量赋初值

3.实现过程描述

进程pa、pb

Pa:

P(sem)

<S>

V(sem)

.

.

.

Pb:

P(sem)

<S>

V(sem)

.

.

.

## 3.6 进程同步

### 3.6.1 同步的概念

1.制约

制约分为两类：直接制约和间接制约。

直接制约：在异步环境下的并发进程，各自的执行结果都是对方的执行条件，从而限制各进程执行速度的过程。

间接制约：由共享公有资源而造成的在临界区内不允许并发程序交叉执行引起。

2.异步环境

执行起始时间的随机性和执行速度的独立性。简单的说就是进程开始的时间不同执行时的速度也不同。

3.同步

把异步环境下的一组并发进程因**直接制约**而相互**发送消息**和**互相合作、互相等待**，使得各进程按一定速度执行的过程称为进程的同步。

4.消息、事件

同步进程互相发送的消息称为消息或事件

过程wait(消息名）:

表示进程等待合作进程发来的消息

过程signal（消息名）:

表示向合作进程发送消息

5.合作进程

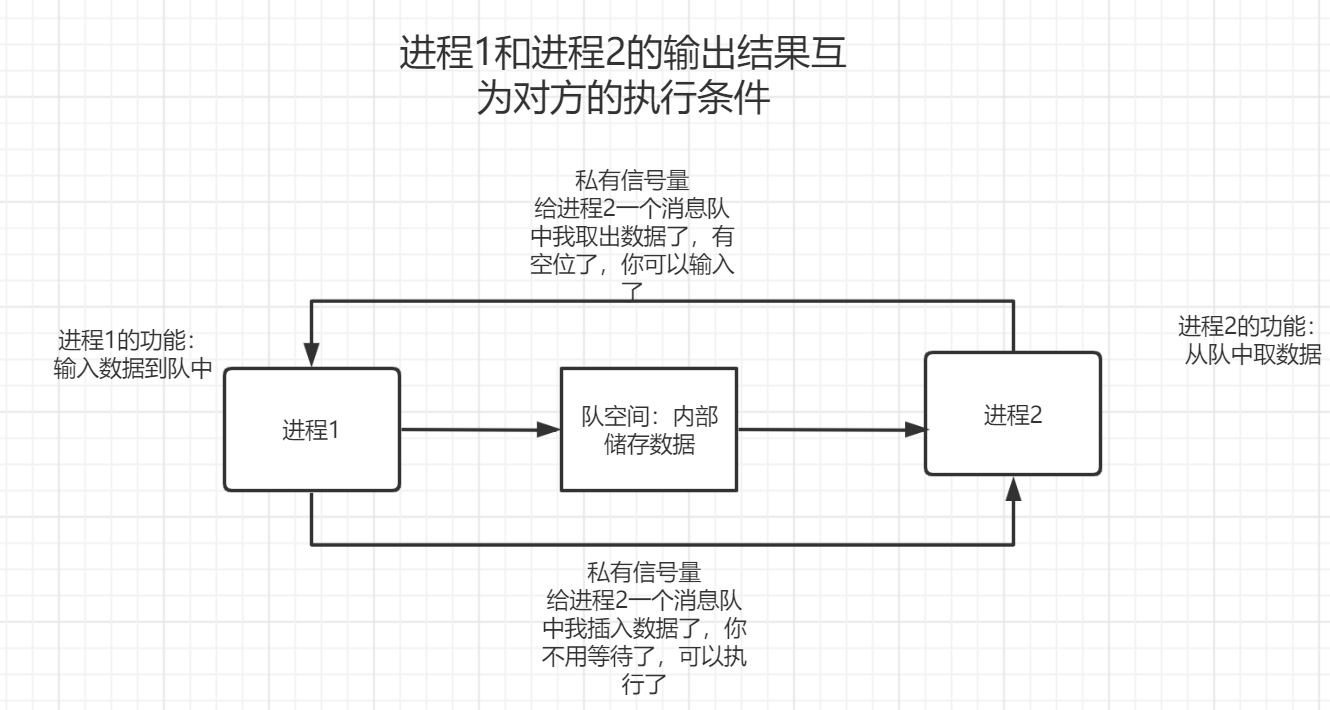
具有同步关系的一组并发进程称为合作进程。

### 3.6.2 私用信号量

1.私有信号量和公有信号量

私有信号量：和制约进程和被制约进程有关，私有信号量发送的是**执行条件所需要的信息**。

公有信号量：进程互斥时使用的信号量。



私有信号量图

### 3.6.3 用P,V原语操作实现同步

实现步骤：

1.为各并发进程设置私有信号量

2.为私有信号量赋初值

3.通过P、V原语和私有信号量规定各进程的执行顺序。

P原语+私有信号量：

语法格式：P(私有信号量)

作用：用于接收合作进程发送来的信息。

V原语+私有信号量：

语法格式：V(私有信号量)

作用：用于向合作进程发送的信息。

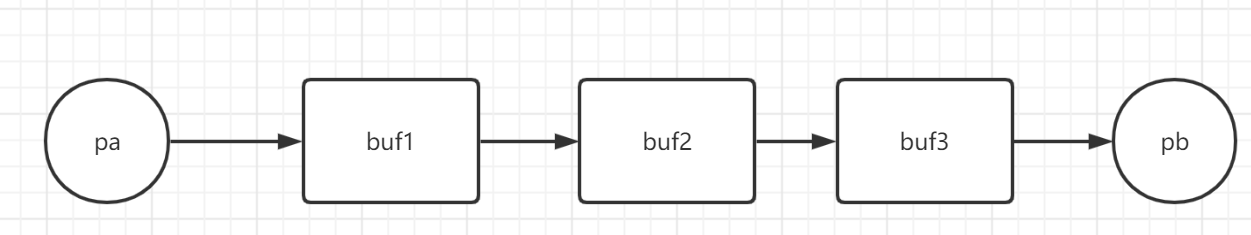
例题：

设Pa和Pb进程通过缓冲区队列传递数据。Pa为发送进程，pb为接收进程。

数据的发送和数据的接收需要满足一下条件：

1.在pa至少输入一个数据入缓冲区前，pb不能从缓冲区取出数据。

2.在pa进程往缓冲区传入数据时，缓冲区必须要有一个空位。



缓冲区队列

解：

1.定义私有信号量

2.给私有信号量赋初值

3.实现过程描述：

Pa:deposit(data):

Begin local x

P(私有信号量1);

…………

V(私有信号量2);

End

Pb:remove(data):

Begin local x

P(私有信号量2);

………………

V(私有信号量1);

End

### 3.6.4 生产者-消费者问题（进程的同步和互斥问题一般化）

1.消费者

使用系统中某一类资源的的进程称为消费者。

2.生产者

把释放同类资源的进程称为该资源的生产者。

例题：



生产者-消费者问题

通过上图分析：

生产者-消费者问题是一个同步问题，所以需要满足以下条件：

1.消费者想要接受数据需要有界缓冲区中至少有一个单元是满的。

2.生产者想要发送数据时，有界缓冲区至少要有一个单元是空的。

由于缓冲区是有界的，所以生产者进程和消费者进程需要互斥。

答：

1.声明一个公有信号量和各大进程的私有信号量。

2.给各个信号量赋初值。

3.描述过程

Deposit(data):

Begin

P(生产者私有信号量);

P(公有信号量);

送数据进入缓冲区

V(消费者信号量)

V(公有信号量)

End

Remove(data):

Begin

P(消费者私有信号量);

P(公有信号量);

从缓冲区取出数据

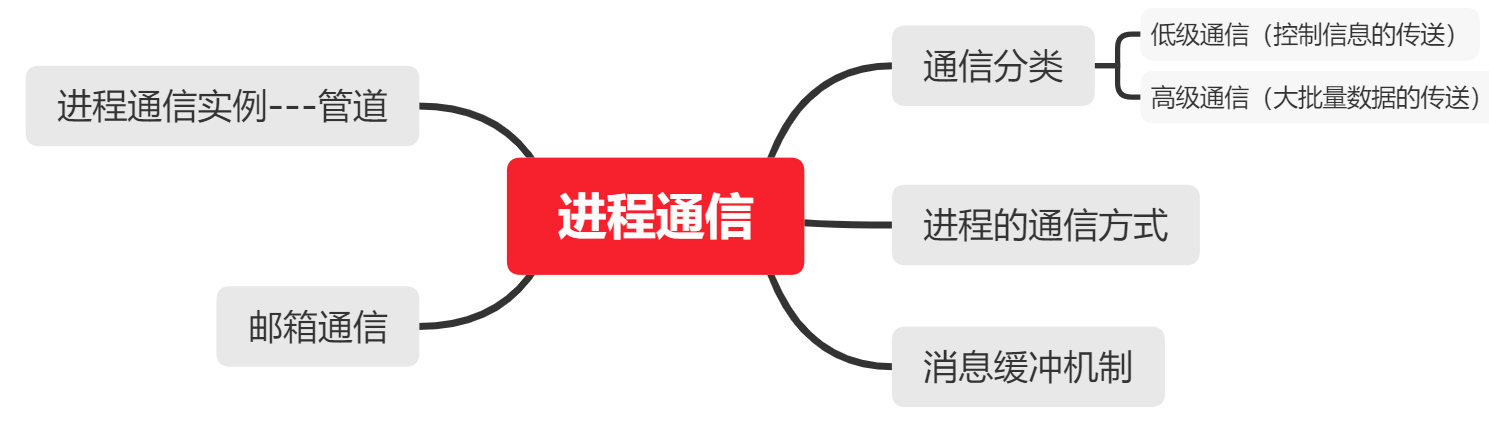
V(生产者私有信号量);

V(公有信号量);

End

注意：P、V原语操作顺序很有讲究，在执行P原语操作时，私有信号量一定要在公有信号量之前，否则会出现进程之间的死锁。

## 3.7 进程通信



### 3.7.1 进程的通信方式

通信形式：

1.主从式

主要特点：

（1）主进程可自由地使用使用从进程的资源或数据。

（2）从进程的动作受主进程的控制

（3）主进程和从进程的关系是固定的，例子：终端控制进程和终端进程

（思考：什么是终端控制进程和终端进程）

2.会话式

主要特点：

 通信的双方采用**请求和回应请求的方式**进行通信。通信双方分别叫做**使用进程**和 **服务进程**。主要有以下特点：

(1) 通信时，使用进程要得到服务进程的允许，方能使用服务进程；

(2) 服务进程是根据使用进程的请求服务的，而且在通信中，服务进程始终为控制进程；

(3) 进程在确定会话式方式时要建立固定的逻辑关联关系；

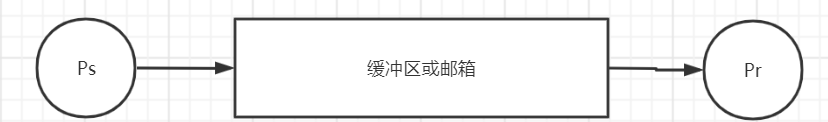
3.消息或邮箱机制

通信的双方地位对等，特点如下：

(1) 发送进程能否发送信息，只与**消息的缓冲区**或邮箱中是否有足够大的空闲空间来满足这次通信。与将消息发送到的**目的进程的状态无关**。

(2)发送进程与接收进程间**不要建立直接**的逻辑关系；

(3)消息的接受和发送要通过**消息缓冲**或者**邮箱**来完成。



消息或邮箱机制

4.共享储存方式

特点：

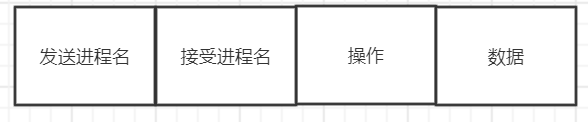
(1) 通信时，通信的数据或信息不发生存储移动；

(2) 需要交互时，通信双方通过一个共享存储区完成通信

(3) 共享存储区中的数据可以作为需要交互进程的一部分存储在进程体中；

5.消息

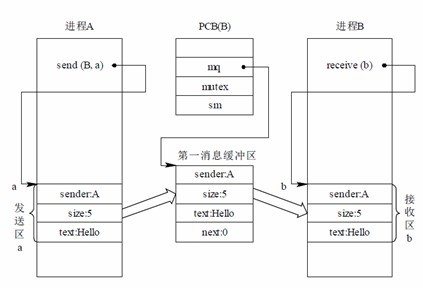
消息一般形式由四个部分组成：发送进程名、接受进程名、数据和有关数据的操作。



消息的组成

### 3.7.2 消息缓冲机制

Hansen在1973年首先提出了用消息缓冲作为进程通信的一种基本方式。发送进程在发送信息之前在自己的内存空间开一个发送区，然后把信息传入其中，然后在通过发送过程发送，接受进程也在自己的内存开辟一个接收区，然后用接受过程接受信息到接收区。



由于这个发送区和接受区是一个公用的机制，所以想要实现效果需要满足两大条件：

（1）在发送进程把消息写入缓冲区和把缓冲区挂入信息队列时，其他进程不能进入。

（2）当缓冲区没有消息时，接受进程不能接受到任何信息，发送进程是否可以发送消息由是否申请到缓冲区决定。

### 3.7.3 邮箱通信

工作过程

发送进程把消息送往邮箱，接收进程从邮箱中取出消息，从而完成进程间信息交换。

邮箱由邮箱头和邮箱体组成。其中邮箱头描述邮箱名称、邮箱大小、邮箱方向以及拥有该邮箱的进程名等。邮箱体主要用来存放消息。

### 3.7.5 进程通信的实例-----管道

1.管道：

管道可用于具有**亲缘关系进程**间的通信，允许一个进程和另一个与它有共同祖先的进程之间进行通信。

2.管道的分类：

 1)普通管道PIPE, 通常有种限制,一是半双工,只能单向传输;二是只能在父子进程间使用.

 2)流管道s\_pipe: 去除了第一种限制,可以双向传输.

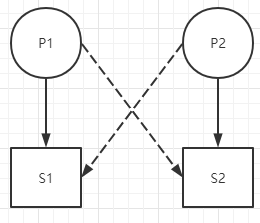
 3)命名管道:name\_pipe, 去除了第二种限制,可以在许多并不相关的进程之间进行通讯.

## 3.8 死锁问题

### 3.8.1 死锁的概念

**1.死锁的定义**

定义：各并发进程互相等待双方所拥有的资源，且这些并发进程在得到对方资源之前不会释放自己所拥有的资源。从而造成大家都想得到资源而又都得不到资源，各并发进程不能继续向前推进的状态。

虚线代表需求；实线代表拥有

**2.死锁的起因**

死锁的起因是并发进程的资源竞争，根本原因在于系统提供的资源个数少于并发进程所要求的该类资源数。

**3.产生死锁的必要条件**

（1）互斥条件。

（2）不剥夺条件。进程所获资源在未使用完毕之前，不能被其他资源强行剥夺，只能自己释放，

（3）部分分配。每次申请部分资源，在等待新资源的同时，继续占用已分配的资源

（4）环路条件。存在一种进程循环链，链中每一个进程已获得的资源同时被下一个进程所请求。

### 3.8.2 死锁的消除方法

解决死锁的方法一般可分为预防，避免，检测与恢复。

1. **死锁预防**

方法1：打破资源的互斥和不可剥夺这两个必要条件，允许进程同时访问某些资源， 做不到；

方法2：打破资源的部分分配这个必要条件。预先分配进程所需要的全部资源。但缺点很多。

方法3：打破死锁的环路条件。即把资源分类按顺序排列，使进程在申请、保持资源时不形成环路。缺点：限制了进程对资源的请求，而且对资源的分类编序耗费系统开销。

1. **死锁避免**

死锁避免算法：银行家算法。

银行家算法对于进程发出的每一个系统可以满足的资源请求命令加以检测，如果发现分配资源后系统进入不安全状态，则不予分配；若分配资源后系统仍处于安全状态，则实施分配

1. **死锁检测**

死锁检测方法对资源的分配不予限制，即允许死锁产生。但系统定时地运行一个死锁检测程序，判断系统是否发生死锁，若检测到有死锁，则设法加以解除。

1. **死锁解除**

死锁解除通常采用如下方法：

1. 资源剥夺法。从一些进程那里强行剥夺足够数量地资源分配给死锁进程
2. 撤销进程法。根据某种策略逐个撤销死锁进程，直至解除死锁为止。

## 3.9 线程的概念

### 3.9.1 为什么引入线程

当一个进程需要另一个实体执行某件事情时，该进程可以派生(fork)一个子进程，让子进程进行处理。

但fork有它自身的一些问题：

(1) fork子进程是父进程的一个拷贝，在有大量fork子进程时，会耗费大量的系统资源。

(2) fork子进程后，父子进程就相对独立了，相互间需要借助IPC来传递信息。并且子进程之间涉及复杂的上下文切换，花费大量时间。

### 3.9.2 线程的基本概念

引入线程主要是为了提高系统的执行效率，减少处理机的空转时间和调度切换（保护现场信息）的时间，以及便于系统管理。

一个进程内的基本调度单位称为线程或轻权进程 (thread or light weight process)。

### 3.9.3 线程和进程的区别

(1) 线程属于某一个进程，并与进程内的其他线程一起共享进程的资源。而不同的进程拥有不同的虚拟地址空间。

(2) 线程控制表TCB所含状态信息比PCB少得多。

(3) 发生进程切换与发生线程切换时相比较

### 3.9.4 线程的适用范围

使用线程的最大好处是在有多个任务需要处理机处理时，减少处理机的切换时间；而且，线程的创建和结束所需要的系统开销也比进程的创建和结束要小得多。由此，可以推出最适合使用线程的系统是多处理机系统。在多处理机系统中，同一用户程序可以根据不同的功能划分为不同的线程，放在不同的处理机上执行。

几种典型的应用是：

(1) 服务器中的文件管理或通信控制。由于服务器同时可能接受许多个文件访问要求，则系统可以同时生成多个线程来进行处理。

(2) 前后台处理。线程可被用来减少处理机切换时间和提高执行速度。

(3) 异步处理。程序中的两部分如果在执行上没有顺序规定，则这两部分程序可用线程执行。