# 第3章 进程管理

## 3.1进程的概念

### 3.1.0程序的顺序执行

程序：一个在时间上按严格次序前后相继的操作序列，是一个静态的概念。

顺序执行：具有独立功能的程序独占处理机直至最终结束的过程。

**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。

顺序执行特点：

(1) 顺序性

程序执行过程可看作一系列严格按程序规定的状态转移过程。

(2) 封闭性

程序执行得到的最终结果由给定的初始条件决定，不受外界因素的影响。

(3) 可再现性

只要输入的初始条件相同，则无论何时重复执行该程序都会得到相同的结果。

### 3.1.1 程序的并发执行

• 程序执行环境的重要特点：

(1) 程序的并发执行（独立性）

每道程序都是逻辑上独立的，它们之间不存在逻辑上的制约关系。

(2) 系统的用户随机地使用（随机性）

在多道程序环境下，特别是在多用户环境下，程序和数据的输入与执行开始时间都是随机的。

(3) 系统所拥有的资源被共享（资源共享）

资源共享将导致对进程执行速度的制约。

• 程序的并发执行

(1) 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。

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）

语句S1和S2**并发执行条件**：对于语句S1和S2，有

① R(S1)∩ W(S2)={∮} R：读

② W(S1)∩ R(S2)={∮} W：写

③ W(S1)∩ W(S2)={∮} 同时成立

(2) 程序的并发执行所带来的影响

并发执行充分地利用了系统资源，从而提高了系统的处理能力。

如果并发执行的程序段不按照特定的规则和方法进行资源共享和竞争，则其执行结果将不可避免地失去封闭性和可再现性。

### 3.1.2 进程的定义

进程的并发执行的程序在执行过程中分配和管理资源的基本单位。简单的说，程序的一次运行。

• 进程和程序区别和联系：

(1) 进程是一个动态概念，而程序则是一个静态概念。

(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，其并行性受到系统自己的制约。

(4) 不同的进程可以包含同一程序，只要该程序所对应的数据集不同。

• 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

进程的静态描述由三部分组成：

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

创建一个进程时，应首先创建其 PCB，然后才能根据PCB 中信息对进程实施有效的管理和控制。当一个进程完成其功能之后，系统则释放PCB，进程也随之消亡。

不同阶段，进程的 PCB所包含的内容有所不同。但下面内容是必需的：

(1) 描述信息

(2) 控制信息

① 进程当前状态

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

PCB 中包含最多的是资源管理信息，包括有关存储器的信息、使用输入输出设备的信息、有关文件系统的信息等。

(4) CPU 现场保护结构

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中止在处理机上的执行时，需要保护当前进程的 CPU现场（或称进程上下文）。

### 3.2.2 进程上下文

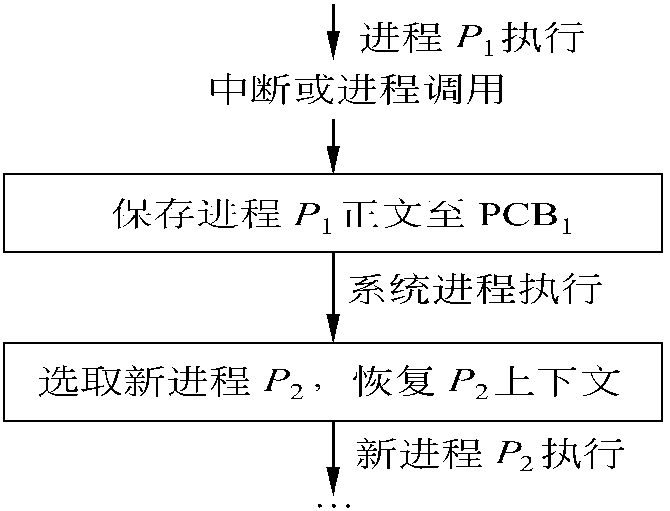
• 概念

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；

正文：**正在执行的**

下文：**待执行**

切换  
 进程上下文切换发生在不同的进程之间。



### 3.2.3 进程空间

进程空间或虚空间：进程的地址空间

在UNIX以及Linux等操作系统中，进程空间还被划分为用户空间（执行用户程序）和系统空间（执行操作系统内核程序）两大部分。

为了防止用户程序访问系统空间，造成访问出错，计算机系统还通过程序状态寄存器等设置不同的执行模式，即用户模式（用户态）和系统模式（系统态）来进行保护。

## 3.3 进程状态及其转换

### 3.3.1 进程状态

至少具有五种基本状态：初始状态、就绪状态、执行状态、等待状态和终止状态。

• 就绪状态

已经得到除 CPU之外的其他资源，只要由调度得到处理机，便可立即投入执行。

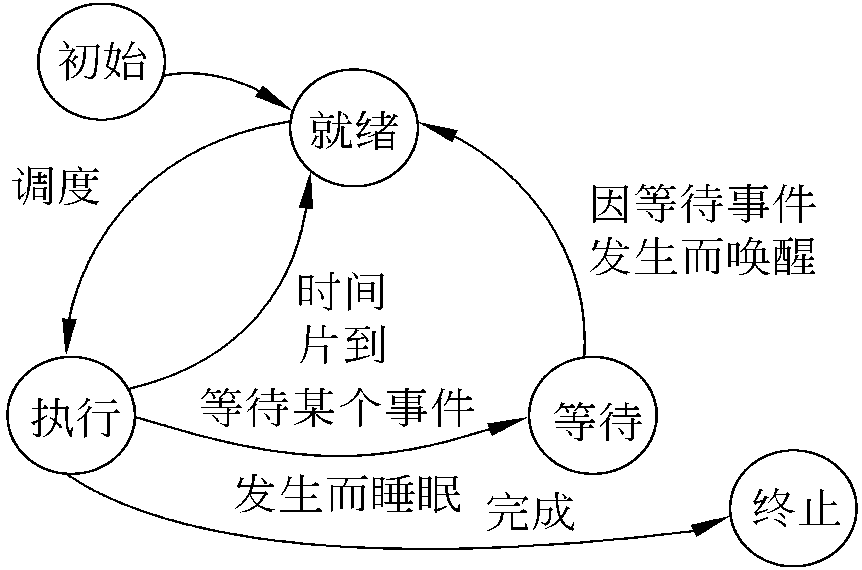
• 执行状态

任一时刻处于执行状态的进程只能有一个。只有处于就绪状态的进程经调度选中之后才可进入执行状态。

• 等待状态

进程因等待某个事件发生而放弃处理机进入等待状态。

### 3.3.2 进程状态转换



进程状态转换

## 3.4 进程控制

**原语**

系统态下执行的某些具有特定功能的程序段，被高层软件所调用。

为两类：

一类是机器指令级的，其特点是执行期间不允许中断；

另一类是功能级的，其特点是作为原语的程序段不允许并发执行。

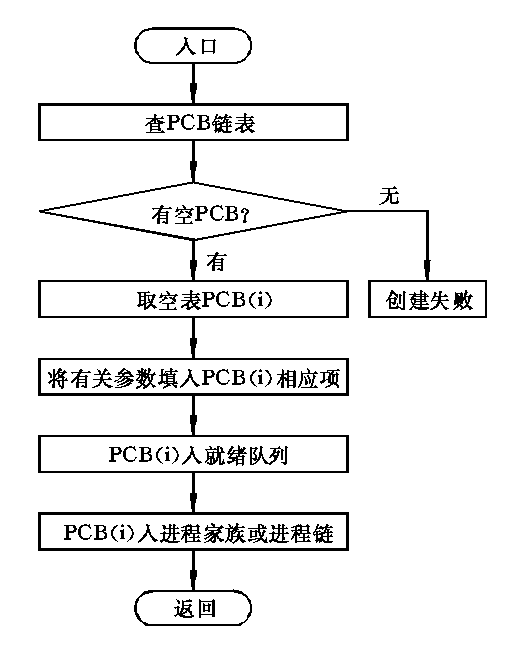
### 3.4.1 进程创建与撤消

1. 进程创建

创建方式：

(1) 由系统程序模块统一创建。

(2) 由父进程创建。



创建原语流图

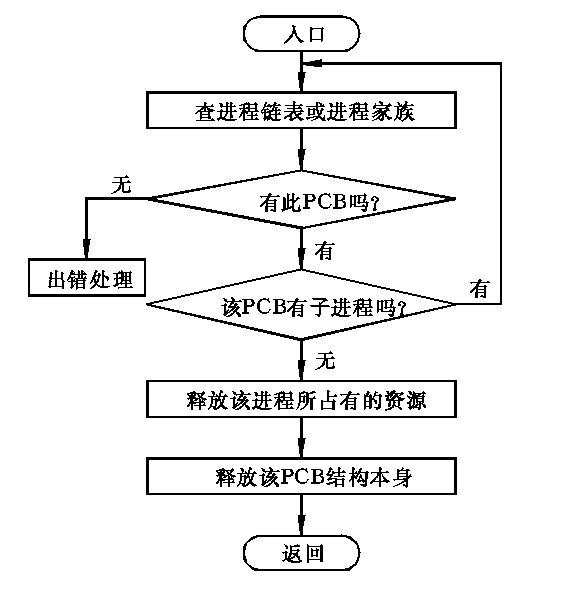
2. 进程撤消

以下几种情况导致进程被撤消：

(1) 该进程已完成所要求的功能而正常终止。

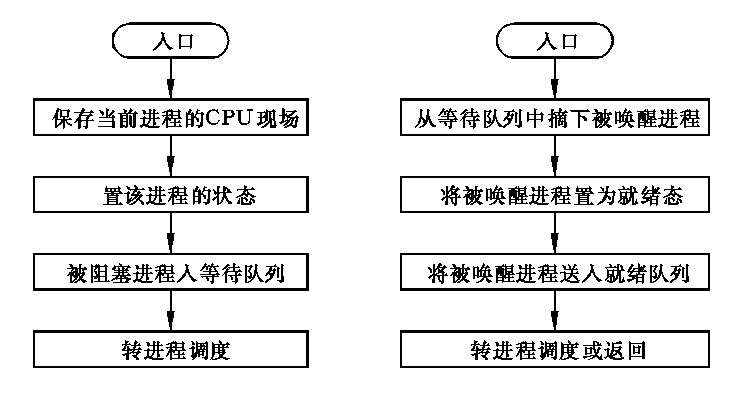
(2) 由于某种错误导致非正常终止。

(3) 祖先进程要求撤消某个子进程。



撤消原语流图

### 3.4.2 进程的阻塞与唤醒



阻塞原语图

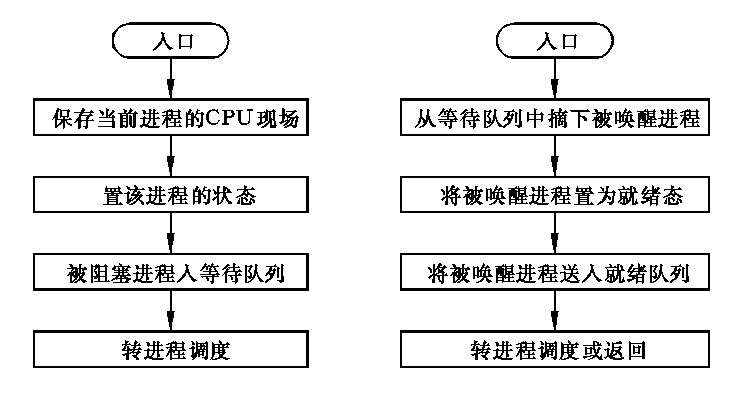


图3.10 唤醒原语

## 3.5 进程互斥

### 3.5.1 资源共享所引起的制约

进程的并发执行会导致对有限性、共享资源的竞争，同时对进程的执行过程进行制约。

1. 临界区

把不允许多个并发进程交叉执行的一段程序称为临界部或临界区，临界区也可以被称为访问公用数据的那段程序。

2. 间接制约

可以把那些不允许交叉执行的临界区按不同的公用数据划分为不同的集合。把这些集合称为类（class）。

可用下列标准形式来描述临界区： when〈类名〉do〈临界区〉od

“间接”二字主要是指各并发进程的速度受公有资源制约，而不是进程间直接制约的意思。

3. 什么是互斥

互斥定义：不允许两个以上的共享该资源的并发进程同时进入临界区称为互斥。

互斥执行时必须满足如下准则：

(1) 各并发进程享有平等的、独立的竞争共有资源的权利。（机会均等）

(2) 并发进程中的某个进程不在临界区时，它不阻止其他进程进入临界区。（有空则进）

(3) 并发进程中的若干个进程申请进入临界区时，只能允许一个进程进入。（两者择一）

(4) 并发进程中的某个进程申请进入临界区时开始，应在有限时间内得以进入临界区。(有限等待)

间接制约：竞争公有资源而引起的进程之间互斥；

直接制约：互相共享对方的私有资源所，要求各并发进程同步执行。

### 3.5.2 互斥的加锁实现

对临界区加锁以实现互斥

当某个进程进入临界区之后，它将锁上临界区，直到它退出临界区时为止。

加锁后的临界区程序描述如下：

key[S]表示该锁定位属于类名为Ｓ的临界区

lock(key ［Ｓ］)

〈临 界 区〉

unlock(key ［Ｓ］)

设key[S]=1时表示类名为Ｓ的临界区可用，

key[S]=0时表示类名为Ｓ的临界区不可用。

加锁方法缺点：

(1) 影响系统可靠性和执行效率的问题。循环

测试锁定位将损耗较多的 CPU计算时间。

(2) 在某些情况下出现不公平现象。

### 3.5.3 信号量和ＰＶ原语

1. 信号量（semaphore）

信号量管理相应临界区的公有资源，它代表可用资源实体。

信号量概念和ＰＶ原语。

在操作系统中，信号量sem是一整数。在sem大于等于零时代表可供并发进程使用的资源实体数，但sem小于零时则表示正在等待使用临界区的进程数。用于互斥的信号量sem的初值应该大于零。

2. Ｐ，Ｖ原语

Ｐ原语操作的主要动作是：

(1) sem减 1；

(2) 若sem减1后仍大于或等于零，则进程继续执行；

(3) 若sem减1后小于零，则该进程被阻塞后与该信号相对应的队列中，然后转进程调度。

Ｖ原语的操作主要动作是：

(1) sem加1；

(2) 若相加结果大于零，进程继续执行；

(3) 若相加结果小于或等于零，则从该信号的等待队列中唤醒一等待进程，然后再返回原进程继续执行或转进程调度。

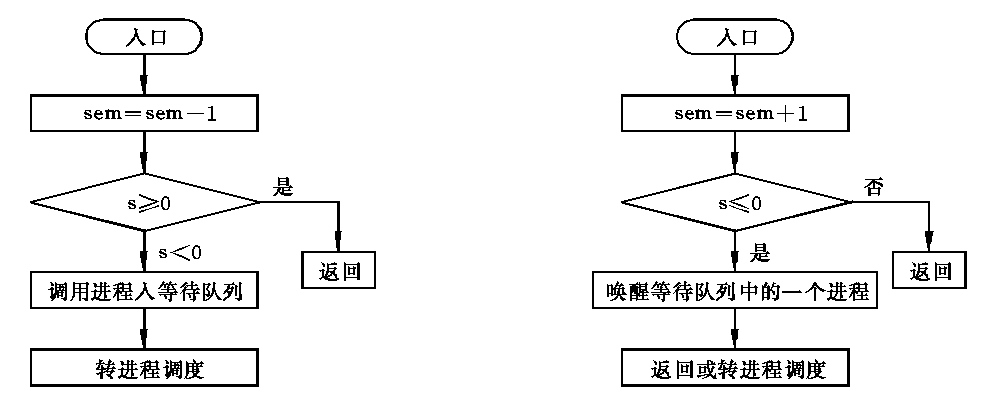


图3.12 Ｐ原语操作功能 图 3.13Ｖ原语操作功能

### 3.5.4 用Ｐ，Ｖ原语实现进程互斥

1) 设定信号量初始值

设 sem为互斥信号量，其取值范围为(1,0,-1)。

sem=1：表示进程PA和PB都未进入类名为Ｓ的临界区；

sem=0：表示进程PA或PB已进入类名为Ｓ的临界区；

sem=-1：表示进程PA和PB中，一个进程已进入临界区，而另一个进程等待进入临界区。

特别说明：

信号量初始值表示初始资源数目，不一定为1。

2) PV操作描述：

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

一组在异步环境下的并发进程，各自的执行结果互为对方的执行条件，从而限制各进程的执行速度的过程称为并发进程间的直接制约。

同步概念

把异步环境下的一组并发进程，因直接制约而互相发送消息而进行互相合作、互相等待，使得各进程按一定的速度执行的过程称为进程间的同步。

具有同步关系的一组并发进程称为合作进程，合作进程间互相发送的信号称为消息或事件。

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

### 3.6.2 私用信号量

把各进程之间发送的消息作为信号量看待。

与进程互斥时不同的是，这里的信号量只与制约进程及被制约进程有关而不是与整组(所有)并发进程有关。因此，称该信号量为私用信号量（Private Semaphvre），同步时使用。

与私用信号量相对应，称互斥时使用的信号量为公用信号量。

### 3.6.3 用Ｐ，Ｖ原语操作实现同步

步骤和方法

利用Ｐ，Ｖ原语实现进程同步的方法与利用wait和signal过程时相同，也是分为三步。

(0) 画出各进程的流程图，分析哪些步骤可能存在互斥同步的问题；

(1) 为各并发进程设置私用信号量；

(2) 为私用信号量赋初值；

(3) 利用Ｐ，Ｖ原语和私用信号量规定各进程的执行顺序。

## 3.7 进程通信

• 低级通信—进程间控制信息的交换。一般只传送一个或几个字节的信息，以达到控制进程执行速度的作用。

• 高级通信—进程间大批量数据的交换，目的交换信息。

### 3.7.1 进程的通信方式

在单机系统中，进程间通信可分为4种形式：

(1) 主从式；

(2) 会话式；

(3) 消息或邮箱机制；

(4) 共享存储区方式。

(1) 主从式通信系统

主要特点：

① 主进程可自由地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。

典型例子:终端控制进程和终端进程。

(2) 会话系统

通信进程双方可分别称为使用进程和服务进程。

其中，使用进程调用服务进程提供的服务。

特点：

① 使用进程在使用服务进程所提供的服务之前，必须得到服务进程的许可；

② 服务进程根据使用进程的要求提供服务，但对所提供服务的控制由服务进程自身完成。

③ 使用进程和服务进程在通信时有固定连接关系。

典型例子：用户进程与磁盘管理进程。

(3) 消息或邮箱机制(3.7.2、3.7.3细讲)

无论接收进程是否已准备好接收消息，发送进程都将把所要发送的消息送入缓冲区或邮箱.消息由4个部分组成：发送进程名、接收进程名、数据和有关数据的操作。

特点：

① 只要存在空缓冲区或邮箱，发送进程就可以发送消息。

② 与会话系统不同，发送进程和接收进程之间无直接连接关系，接收进程可能在收到某个发送进程发来的消息之后，又转去接收另一个发送进程发来的消息。

③ 发送进程和接收进程之间存在缓冲区或邮箱（图3.17）用来存放被传送消息。

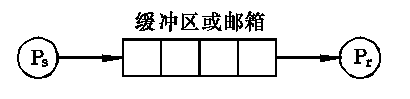


图3.17缓冲区或邮箱通信结构

(4) 共享存储区方式

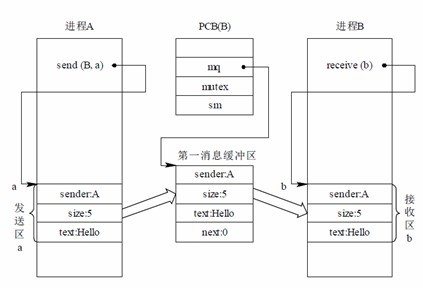
特点：

不要求数据移动。两个需要互相交换信息的进程通过对同一共享数据区（shared memory）的操作来达到互相通信的目的。这个共享数据区是每个互相通信进程的一个组成部分。

### 3.7.2 消息缓冲机制

工作过程（参见下页图）

发送进程在发送消息前，先在自己的内存空间设置一个发送区，把欲发送的消息填入其中，然后再用发送过程将其发送出去；接收进程则在接收消息之前，在自己的内存空间内设置相应的接收区，然后用接收过程接收消息。



由于使用的缓冲区(队列)为公用缓冲区，两通信进程必须满足如下条件：

① 在发送进程把消息写入缓冲区和把缓冲区挂入消息队列时，应禁止其他进程对该缓冲区消息队列的访问。当接收进程正从消息队列中取消息缓冲时，也应禁止其他进程对该队列的访问。

② 当缓冲区中无消息存在时，接收进程不能接收到任何消息。

发送进程能是发送消息，取决于是否申请到缓冲区

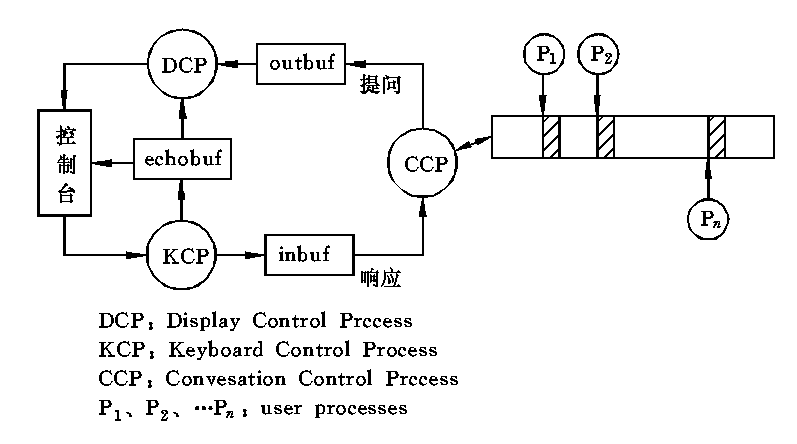
### 3.7.3 邮箱通信

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。

### 3.7.4 进程通信实例——用户终端和控制台的通信(了解)



### 3.7.5 进程通信的实例——管道

管道pipe

UNIX系统从SystemⅤ开始，提供有名管道和无名管道两种数据通信方式，这里介绍无名管道。

无名管道为建立管道的进程及其子孙提供一条以比特流方式传送消息的通信管道。该管道在逻辑上被看作管道文件，在物理上则由文件系统的高速缓冲区构成。

管道按FIFO（先进先出）方式传送消息，且只能单向传送消息（图3.21）。

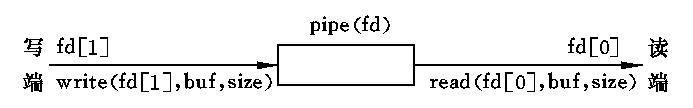


图3.21 管道通信

建立一条同步通信管道。其格式为：

pipe(fd)

int fd[2]； fd[1] 为写入端，fd[0]为读出端

## 3.8 死锁问题

### 3.8.1 死锁的概念

1. 死锁的定义

定义：各并发进程彼此互相等待对方所拥有的资源，且这些并发进程在得到对方的资源之前不会释放自己所拥有的资源。

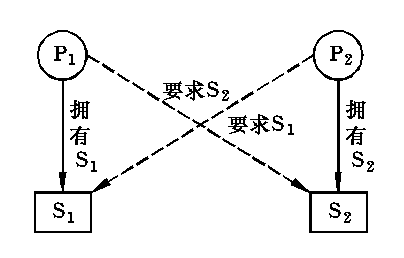


图3.24 死锁的概念

2. 死锁的起因

根本原因：系统提供的资源个数少于并发进程所要求的该类资源数。

根本解决办法：

无。因为资源总是有限的，不可能为所有要求资源的进程无限制地提供资源。

可以采用适当的资源分配算法，以达到消除死锁的目的。

3. 产生死锁的必要条件

(1) 互斥条件。资源不能同时被两个以上进程使用或操作，进程对它所需要的资源进行排他性控制。

(2) 不剥夺条件。进程所获得的资源在未使用完毕之前，不能被其他进程强行剥夺，而只能由获得该资源的进程自己释放。

(3) 部分分配。进程每次申请它所需要的一部分资源，在等待新资源的同时继续占用已分配到的资源。

(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。

只要使上述4个必要条件中的某一个不满足，则死锁就可以排除。

### 3.8.2 死锁的排除方法

三种：

预防(prevention)——静态、按序、剥夺式(分配资源)

避免(avoidance)——动态，实际分配资源前进行预分配

检测(detection)和恢复(recovery)——采用资源分配图RAG

在实际操作系统中大都使用检测与恢复法。

1. 死锁预防

• 基本思路：静态

打破四个条件中的某一个。

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。

2. 死锁避免

• 基本思路：动态预防

在实际分配资源之前进行预分配，预测出死锁发生的可能性并加以避免。

### 3. 死锁的检测和恢复

当进程进行资源请求时，死锁检测算法检查并发进程组是否构成资源的请求和保持环路。

RAG及等待图(RAG变型)、有限状态转移图和petriNet等技术都可用来有效地判断死锁发生。

死锁的恢复办法较多。最简单的办法是终止各锁住进程，或按一定的顺序中止进程序列，直至已释放到有足够的资源来完成剩下的进程时为止。另外，也可以从被锁住进程强迫剥夺资源以解除死锁。

## 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。