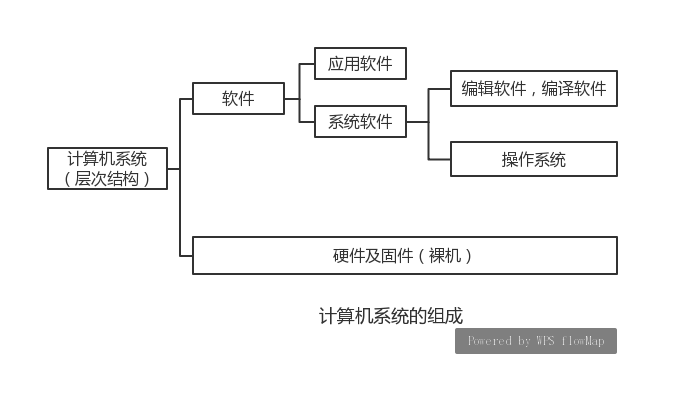
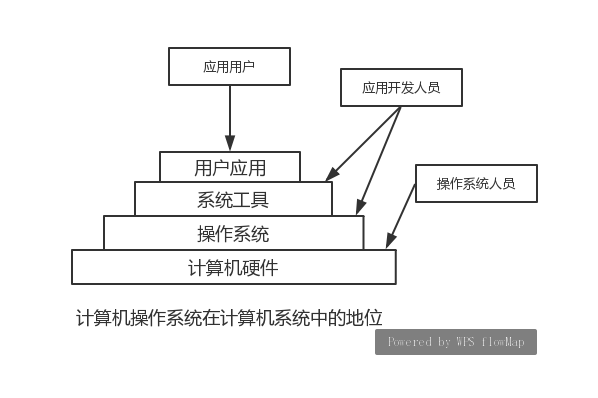
# 操作系统概论

## 什么是操作系统

### 操作系统的地位和目标





操作系统在计算机系统中的地位：操作系统是计算机硬件的首次扩展。

### 操作系统的概念

系统观点：操作系统是对计算机资源进行有效的管理；这些资源包括硬件和软件。操作系统包括硬件和软件。操作系统向用户提供了高级而调用简单的服务。

用户观点：操作系统为用户使用计算机提供良好的界面。操作系统是用户提供良好的接口，一般可以分为三种：命令方式、系统调用、图形界面。

软件观点：操作系统是程序和数据结构的集合。

定 义：操作系统是控制和管理计算机硬件和软件资源、合理地组织计算机工作流程。

### 操作系统的作用和组成

1、操作系统的作用：几种观点

1. OS是计算机硬件、软件资源的管理者。管理对象包括：CPU、存储器、外部设备、数据；  
   管理的内容：资源的当前状态、资源的分配、回收和访问操作，相应管理策略。
2. OS是用户使用系统硬件、软件的接口。  
   系统命令；系统调用；
3. OS是虚拟机（virtual machine）。

2、操作系统的组成

1. 管理模块：针对不同管理对象的程序模块（通常称为操作系统核心）
2. 用户接口：如外壳（shell）、窗口系统在shell中，通过运行其他程序来完成各种功能。

## 操作系统的发展历史

### 早期的人工操作方式

1946~50年代（电子管），集中计算（计算中心），计算机资源昂贵；

工作方式：手工；

用户：用户既是程序员，又是操作员；用户是计算机专业人员；

编程语言：为机器语言；

输入输出：纸带或卡片；

计算机的工作特点

1. 用户独占全机：不出现资源被其他用户占用，资源利用率低；
2. CPU等待用户：计算前，手工装入纸带或卡片；CPU利用率低；

主要矛盾

1. 用计算机处理能力的提高，手工操作的低效率；
2. 用户独占全机的所有资源；

### 单道批处理系统（Simple Batch Processing）

批处理中的作业的组成：包括用户程序、数据和作业说明书（作业控制语言）  
 “批”：供一次加载的磁带，通常由若干个作业组装成，在处理中使用一组相同的系统软件（系统带）。

脱机输入/输出

为了解决输入/输出设备与CPU的速度不匹配问题，引入了外围计算机——卫星机，让它来处理系统的输入/输出，把CPU从繁重的I/O处理中解放出来。

两种批处理方式

1. 联机批处理  
   慢速的输入输出处理人直接由主机CPU来完成；  
   用户提交作业：以纸带或卡片为介质；  
   问题：输入输出时，CPU出于等待状态。
2. 脱机批处理  
   利用卫星机完成输入输出功能。主机与卫星机可并行工作。  
   卫星机：完成面向用户的输入输出（纸带或卡片），中间结果暂存在磁带或磁盘上。

单道批处理的主要问题：

由于是单道处理，CPU和I/O设备使用忙闲不均（取决于当前作业的特征）。对计算为主的作业，外设空闲；对I/0为主的作业，CPU空闲。

### 中断和通道技术

中断：是指CPU在收到中断信号后，停止原来工作，转去处理该中断事件，完毕后回到原来断点继续工作。

通道：也就是I/O通道技术，他是控制一台或者多台外部设备的硬件机构，能够使得I/O操作与主机CPU并行工作，相当于一个I/O处理器。要比DMA(Director Memory Access,直接存储访问)技术早。

通道用于控制I/O设备与内存间的数据传输。后可独立于CPU运行，实现CPU与I/O的并行。

### 多道批处理系统（Multiprogramming System）

在系统中同时被执行的程序不止一个，在任意一个时刻，它们都处于开始点和终止点之间。

多道批处理的运行特点：

多道：内存中同时存放几个作业；

宏观上并行运行：均处于运行状态，但均未运行完；

微观上串行运行：各作业交替使用CPU；

## 现代操作系统类型

### 分时系统（time-sharing system）

分时的定义：把计算机的系统资源（尤其是CPU时间）进行时间上的分割，每个时间段称为一个小时间片，每个用户一次轮流使用时间片。

1. 多个用户分时
2. 前台和后台程序（foreground & background）分时

后台程序不占用终端输入输出，不予用户交互，而终端图形用户界面（GUI），其他程序均作为前台。

抢占式和非抢占式（Preemptive & non-preemptive）

出让CPU是OS强迫或程序主动；

抢占式：OS强迫程序让CPU；

非抢占式：程序主动出让CPU；

分时系统的特征：

1. 多路性：一台主机可以连接多台终端，多个终端用户可以同时使用计算机，共享硬软件资源。
2. 独立性：各个用户的操作互不干扰，每一个用户都认为整个计算机系统被他所独占，为他服务。
3. 交互性：用户能与系统进行对话。用户能通过键盘等设备输入数据或命令，系统获得用户的输入后作出响应。
4. 及时性：系统一般能在一秒钟内接受和响应用户的输入命令或数据，在数秒内显示命令的执行结果。

关键问题：

及时接受输入：多个I/O端口，设立多路缓冲区。

及时响应：提高对换速度（快速外存）、限制用户数目、缩短时间片（可能引起对换次数增多，开销增大，程序总运行时间增大）

减少对换信息量：可重入代码（re-entrant code）；强求也是存储管理--只对换部分程序

### 实时系统（real Time Operating System）

实时操作系统：主要用于过程控制、事务处理等有实时要求的领域，其主要特征是实时性和可靠性。

实时系统的概念：在限定的时间内对输入进行快速处理并作出响应的计算机处理系统，分为：硬实时系统，软实时系统；

用于工业过程控制、军事实时控制、金融等领域，包括实时控制、实时信息处理；

要求：响应时间短，在一定范围之内；系统可靠性高。

实时系统的特征：

1. 实时时钟管理：提供系统日期和时间、定时和延时等时钟管理功能；
2. 过载保护：缓冲区排队，丢弃某些任务，动态调整任务周期（降低某些周期性任务的频率）；  
   过载是指进入系统的任务数目超出系统处理能力
3. 高度可靠和安全性--容错能力（如故障自动复位）和冗余备份（双机，关机部件）；

### 嵌入式操作系统（Embeded Operating System）

嵌入式计算机系统通用型计算机系统相比具有以下特点：

1. 嵌入式系统通常是面向特定应用的
2. 嵌入式系统是将先进的计算机、半导体技术和电子技术与各个行业的具体应用相结合后的产物。这一点就决定了它必然是一个技术密集、资金密集、高度分散、不断创新的知识集成系统。
3. 嵌入式系统的硬件和软件都必须高效率地设计，量体裁衣、去除冗余，力争在同样的硅片面积上实现更高的性能，这样才能在具体应用中对处理器的选择更具有竞争力。
4. 为了提高执行速度和系统可靠性，嵌入式系统中的软件一般都固化在存储器芯片或单片机本身中，而不是存贮于磁盘等载体中。

嵌入式操作系统：

（1)Windows CE

Microsoft Windows CE是从整体上为有限资源的平台设计的多线程、完整优先权、多任务的操作系统。它的模块化设计允许它对于从掌上电脑到专用的工业控制器的用户电子设备进行定制。操作系统的基本内核需要至少200K的ROM。

(2）VxWorks

VxWorks是目前嵌入式系统领域中使用较广泛、市场占有率较高的系统。它支持多种处理器，如×86、i960、Sun Sparc、Motorola MC68xxx、MIPSRX000、POWERPC等等。大多数的VxWorks API是专有的。采用GNU的编译和调试器。

### 多处理操作系统（Multi-processor Operating System）

多处理器操作系统的出现是为了提高计算机系统性能和可靠性。1975年前后，出现多处理机系统（Multi-processor）。

多处理机系统的优点：

增加系统的吞吐量：N个处理器加速比达不到N倍（额外的调度开销，算法的并行化）

提高系统可靠性：故障时系统降级运行

### 网络操作系统（NOS，Network Operating System）

网络操作系统的功能

通常操作系统功能：处理机管理、存储器管理、设备管理、文件管理等；

网络通信功能：通过网络协议进行高效、可靠的数据传输；

资源管理：协调各用户使用；

网络服务：文件和设备共享，信息发布；

网络管理：安全管理、故障管理、性能管理等；

互操作：直接控制对方比较换数据更为困难；

### 分布式操作系统（Distaibuted Operating System）

分布式系统：处理和控制的分散（相对于集中式系统）

分布式系统以计算机和网络为基础，他的基本特征是处理上的分布，即功能和任务的分布。

分布式操作系统的所有系统任务可在系统中任何处理机运行，自动实现全系统范围内的任务分配并自动调度各处理机的工作负载。

分布式操作系统与网络操作系统的比较：

1. 耦合程度：
   1. 分布式系统是紧密耦合系统：分布式OS是在各机上统一建立的“OS同质”，直接管理CPU，存储器和外设：统一进行全系统的管理；
   2. 网络系统通常容许异种OS互连，各机上各种服务程序需按不同网络协议“协议同质”。
2. 并行性：
   1. 分布式OS可以将一个进程分散在各机器并行执行，“进程迁移”；
   2. 网络操作系统则各个机器上的进程独立。
3. 透明性：用户是否知道或指定资源在那个机器上（如CPU、内存或外设）。
   1. 分布式系统的网络资源调度对用户透明，用户不了解所占资源的位置；
   2. 网络操作系统中对网路资源的使用要由用户明确指定；
4. 健壮性：分布式系统要求更强的容错能力；

### 人计算机操作系统（Personal Computer OperatingSystem)

针对单用户使用的个人计算机进行优化的操作系统。个人计算机操作系统的特征

◆系统要求：使用方便、支持多种硬件和外部设备（多媒体设备、网络、远程通信）、效率不必很高。

常用的个人计算机操作系统

单用户单任务：MSDOS

单用户多任务：0S/2,Windows95,

多用户多任务：UNIX(SCO UNIX Solaris x86,Linux,FreeBSD)

## 操作系统的特征

### 并发（Concurrency）

* 多个事件在同一时间段内发生。
* 操作系统是一个并发系统，各进程间的并发、系统与应用间的并发。操作系统要完成并发过程的管理。
* 并行（parallel）是指多个事件在同一时刻发生。
* 在多道程序处理时，宏观上并发，微观上交替在单处理器情况下：  
  程序的静态实体是可执行文件，而动态实体 是进程（或称作任务），并发指的是进程。

### 共享（sharing）

多个进程共享有限的计算机系统资源。操作系统要对系统资源进行合理分配和使用，资源在一个时间段内交替被多个进程所用。共享有两种方式：

* 互斥共享（如音频设备）：资源分配后到释放前，不能被其他进程所用。
* 同时访问（如可重入代码，磁盘文件）资源分配难以达到最优化。

### 虚拟（Virtual）

一个物理实体映射为若干个对应的逻辑实体——分时或分空间。虚拟是操作系统管理系统资源的重要手段，可提高资源利用率。如：CPU-一每个用户（进程）的“虚处理机”存储器一一每个进程都占有的地址空间（指令+数据+堆栈）显示设备一-多窗口或虚拟终端（Virtual Termina）

### 异步性（Asynchronism，也称不确定性）

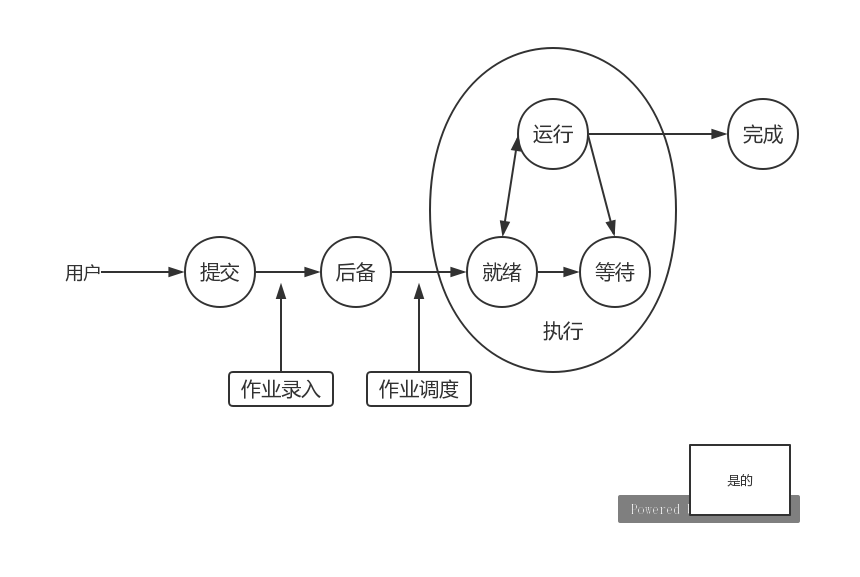
指进程的执行顺序和执行时间的不确定性。

# 作业管理与用户接口

## 作业的组织和管理

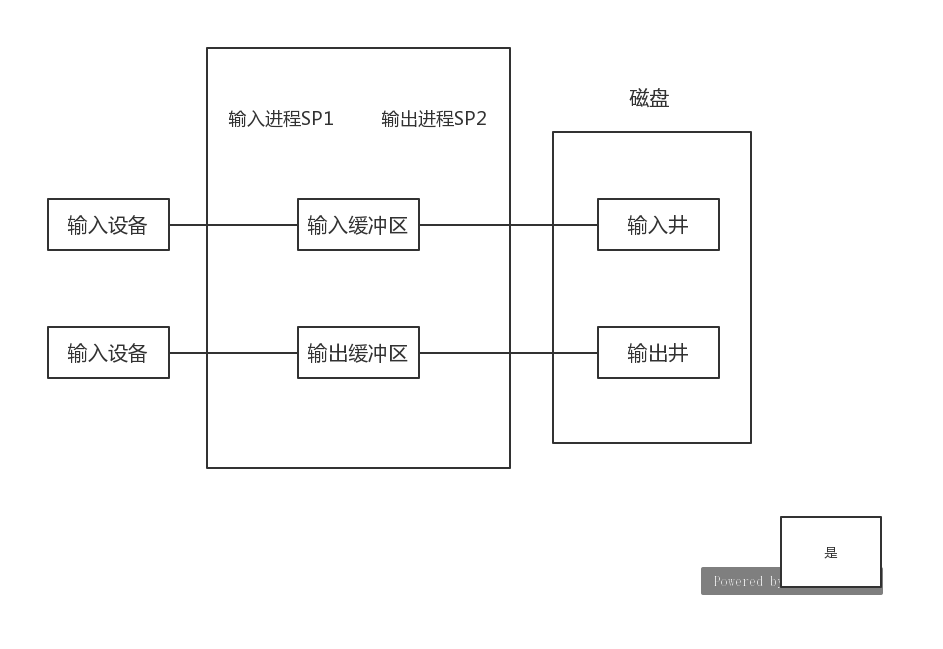
### 作业和作业处理过程

* 基本概念
  + 作业：就是用户一次请求计算机系统为其完成任务所进行的工作总和。
  + 一般来说，为了完成其任务，可以将作业细分成若干个作业步。作业步就是处理作业的各个独立的子任务，系统可以创建若干进程完成各作业步的计算，所以说一个作业是由若干作业步组成的。
  + 如：
    - 编辑：edlin user.asm
    - 汇编：masm user
    - 链接：mink user
    - 执行：user  
      最后获得执行user.exe文件所得的结果。
* 作业的类型：根据计算机系统作业处理方式的不同，可以把作业分为两大类：
  + 脱机作业：不直接交互，通常用于批处理系统
  + 联机作业：直接交互，通常用于分时系统和目前的微机系统作业控制方式：用户向操作系统提供
* 作业加工步骤的方式
  + 脱机作业控制方式
  + 联机作业控制方式
* 作业的组成：包括程序、数据、作业控制信息（如作业说明书）三部分
  + 作业说明书体现用户对作业控制的意图，包括：
    - 作业基本情况：如用户名、作业名
    - 作业控制描述：如作业控制方式、出错处理
    - 作业资源要求描述：如处理时间、优先级、内存空间
* 作业的处理过程
  + 输入：用户将自己的程序和数据提交给系统的后援存储器
  + 后备：建立作业控制块，加入后备作业队列，等待调度
  + 执行：被成功调度，分配资源，建立一组相应的进程，可进一步分为就绪、运行、阻塞状态
  + 完成：正常结束或因错误终止，退出系统



### 作业的输入/输入方式

* 联机输入/输出方式
  + 由主机CPU直接控制，由于CPU与外设的速度相差悬殊，从而降低了CPU的利用率
* 脱机输入/输出方式（人工干预）
  + 由外围处理机控制，人工干预，但手工操作效率低
* SPOOLing系统一Simultaneous Peripheral Operations On Line,外围设备同时联机操作
  + SPOOLing系统的核心思想：利用一台可共享的、高速大容量的块设备（磁盘）来模拟独占设备的操作，使一台独占设备变成多台可并行使用的虚拟设备。
  + SPOOLing系统由专门负责I/O的常驻内存的进程和输入井、输出井组成。
  + 优点：提高了I/O速度；将独占设备改造为共享设备；实现了虚拟设备功能。



### 作业控制块（Job Control Block, JCB）

* 作业控制块是作业存在的唯一标志是系统为管理作业所设置的一个数据结构。包括该作业的标识信息、状态信息、调度参数、资源需求和其他控制信息。
* 作业后备队列就是按照某种原则将后备作业的JCB排成的一个或多个序列，以便作业调度。  
  作业控制块（JCB，Job Control Block)
  + 作业名
  + 用户名
  + 语言程序类型（需调用的系统程序）
  + 内存需求量
  + 估计执行时间
  + 优先数（用于调度）
  + 作业类型
  + 作业说明书文件名
  + 资源要求：（静态，或中间可以随作业步变化一一效率不高；动态分配）
  + 作业状态：提交、后备、执行、就绪、等待、完成；

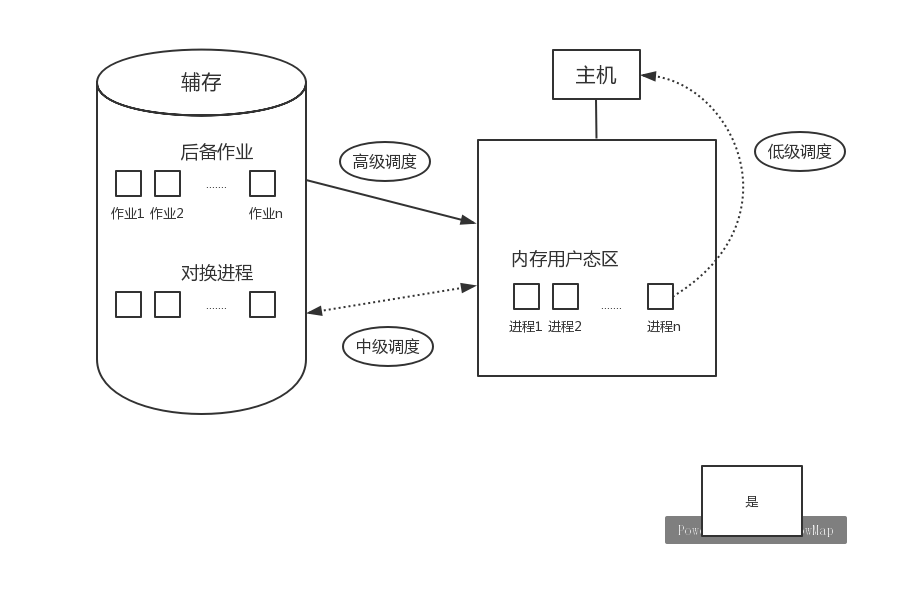
### 作业调度

在一些操作系统中，一个作业从提交到完成需要经过：

高级调度：即作业调度，选择后备作业，为其建立进程，并使其进入主机。

中级调度：即对换调度，决定进程在内存和辅存盘交换区间的对换。

低级调度：即进程级调度，决定哪个进程可以占用CPU，进入运行状态。



* 作业调度算法的评价因素
  + CPU利用率：越高越好
  + 吞吐量：单位时间内CPU完成作业的数量
  + 周转时间：通常与周转系数一起作为评价批处理系统的性能指标，定义如下：  
    其中，作业Ji的提交时刻为tsi，执行时刻为tri，完成时间为toi。  
    
  + n个作业的平均周转时间T和平均周转系数W分别为  
     
  + 对于每个用户来说，总是希望作业提交后立即执行，这样周转时间等于执行时间，周转系数为1；
  + 对于一个计算机系统来说，不可能满足每个用户的这种要求，只能使系统的平均周转时间最短。
* 常见的作业调度算法
  + 对于单道批处理系统，常用以下三种算法：
    - 先来先服务（FCFS）：按作业到达先后进行调度，即启动等待时间最长的作业。
      * 这种算法忽视了吞吐量和平均周转时间，有利于长作业，不利于短作业有利于CPU繁忙的作业，不利于I/O繁忙的作业。
    - 短作业优先调度算法（SJF）：以要求运行时间长短进行调度，即启动要求运行时间最短的作业。
      * 这种算法可以有效降低作业的平均等待时间，提高系统的吞吐量；但对长作业不利，容易致使长作业出现“饥饿”现象。
    - 最高响应比优先调度算法（HRP）：优先调度响应比高的作业。
      * 
  + 对于多道批处理系统，常用以下两种算法：
    - 优先级调度算法：由用户指定作业优先级，优先级高的作业先启动。指定原则：
      * 照顾时间要求紧迫的作业；
      * 照顾“I/0繁忙”的作业，以充分发挥外设的效率；
      * 在一个兼顾分时操作和批量处理的系统中，照顾终端会话型作业，以便获得合理的响应时间。
    - 均衡调度算法：这种算法的基本思想是根据系统的运行情况和作业本身的特性对作业进行分类。作业调度程序轮流地从这些不同类别的作业中挑选作业执行。这种算法力求均衡地使用系统的各种资源。如把作业分成A、B、C三个队列：
      * A队：短作业，其计算时间小于一定值，无特殊外设要求；
      * B队：要用到磁带的作业；
      * C队：长作业，其计算时间超过一定值。
* 单道程序环境下作业调度性能的分析
  + 设有四个作业，其提交时刻、运行时间如下表所示：

|  |  |  |
| --- | --- | --- |
| 作业号 | 提交时刻 | 运行时间 |
| 1 | 8.00 | 2.00 |
| 2 | 8.50 | 0.50 |
| 3 | 9.00 | 0.10 |
| 4 | 9.50 | 0.20 |

这里采用十进制计数，只是为了计算方便

①先来先服务调度算法：顺序为1 2 3 4，计算平均周转时间T和平均周转系数W,如下表所示。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 作业 | 提交时间 | 执行时间 | 开始时间 | 完成时间 | 周转时间 | 周转系数 |
| 1 | 8.00 | 2.00 | 8.00 | 10.00 | 2.00 | 1.00 |
| 2 | 8.50 | 0.50 | 10.00 | 10.50 | 2.00 | 4.00 |
| 3 | 9.00 | 0.10 | 10.50 | 10:60 | 1.60 | 16.00 |
| 4 | 9.50 | 0.20 | 10:60 | 10:80 | 1.30 | 6.50 |
| 平均周转时间T=1.725小时，平均周转系数W=6.875 | | | | | 6.90 | 27.50 |

②最短作业优先调度算法：由于在8.00开始执行作业，当时仅有1，而作业2，3，4尚未到达，故作业1是最短作业。作业1执行完成后是10.00，此时作业2，3，4均已经到达，故选最短作业3，然后是4，2。所以顺序为1,3，4，2。平均周转时间和平均周转系数的计算结果如下表所示。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 作业 | 提交时间 | 执行时间 | 开始时间 | 完成时间 | 周转时间 | 周转系数 |
| 1 | 8.00 | 2.00 | 8.00 | 10.00 | 2.00 | 1.00 |
| 2 | 8.50 | 0.50 | 10.30 | 10.80 | 2.30 | 4.60 |
| 3 | 9.00 | 0.10 | 10.00 | 10:10 | 1.10 | 11.00 |
| 4 | 9.50 | 0.20 | 10:10 | 10:30 | 0.80 | 4.00 |
| 平均周转时间T=1.55小时，平均周转系数W=5.15 | | | | | 6.20 | 20.60 |

③响应比高者优先算法：在作业1执行完成，计算作业2，3，4的响应比分别为：4，11，3.5，因此作业1执行完成后选中作业3完成。执行顺序为1，3，2，4。按此算法求得的平均周转时间和平均周转系数如下表所示。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 作业 | 提交时间 | 执行时间 | 开始时间 | 完成时间 | 周转时间 | 周转系数 |
| 1 | 8.00 | 2.00 | 8.00 | 10.00 | 2.00 | 1.00 |
| 2 | 8.50 | 0.50 | 10.10 | 10.60 | 2.10 | 4.20 |
| 3 | 9.00 | 0.10 | 10.00 | 10:10 | 1.10 | 11.00 |
| 4 | 9.50 | 0.20 | 10:60 | 1:30 | 1.30 | 6.50 |
| 平均周转时间T=1.625小时，平均周转系数W=5.6755 | | | | | 6.50 | 23.70 |

总结：就其平均周转时间和平均周转系数来说，最短作业优先算法最小，先来先服务算法最大，最高响应比优先算法居中。

* 多道程序环境下作业调度性能的分析
  + 具有两道作业的批处理系统，作业调度采用短作业优先算法，作业进驻内存后，采用以优先数为基础的抢占式调度算法，作业运行时间越短，优先数越大。4个作业序列如下：

|  |  |  |
| --- | --- | --- |
| 作业号 | 提交时刻 | 运行时间 |
| 1 | 10.00 | 30 |
| 2 | 10.05 | 20 |
| 3 | 10.10 | 20 |
| 4 | 10.20 | 10 |

总的执行顺序为1 2 4 3 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 作业 | 提交时间 | 执行时间 | 开始时间 | 完成时间 | 周转时间 | 周转系数 |
| 1 | 10.00 | 30 | 10.00 | 11.20 | 80 | 2.667 |
| 2 | 10.05 | 20 | 10.05 | 10.25 | 20 | 1 |
| 3 | 10.10 | 20 | 10.35 | 10:55 | 45 | 2.25 |
| 4 | 10.20 | 10 | 10:25 | 10:35 | 15 | 1.5 |
| 平均周转时间T=1.40分钟，平均周转系数W=1.854 | | | | | 160 | 7.417 |

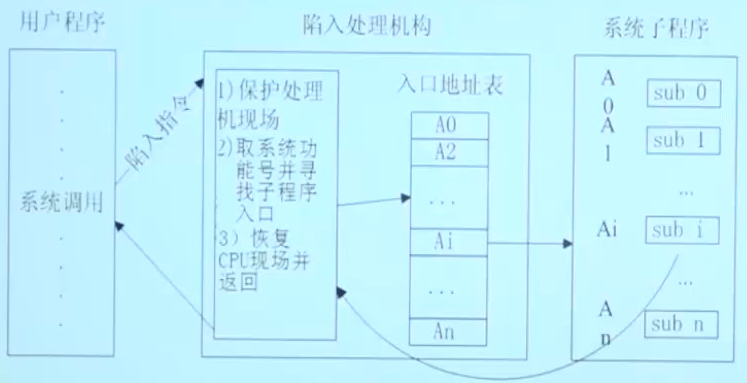
## 命令级用户接口

### MS-DOS的命令

* DOS命令处理程序：command.com
  + 命令类型：
    - 内部命令（dir，cd等）；
    - 外部命令（以com和exe为后缀的文件，如format等；
    - 批处理命令（以bat为后缀名，如autoexec.bat)
    - 输入/输出重定向（<，>，<<，>>）
    - 管道（|）
  + DOS批处理：由command.com执行
    - 有简单的变量替换，有条件转移和跳转、循环；
    - 注释语句rem
  + 如下面批处理将显示当前目录及其子目录所有
    - 文件名；
    - echo off  
      for /R%%f in (\*.\*）do echo %%f
* UNX的命令
  + Shell命令
    - 命令行：与DOS类似，分为内部命令（cd等）和外部命令(ls等)
    - 保留字：do，if，while等
    - Shell变量：PATH，PWD
    - 通配符：\*，？
  + Shell脚本文件
    - 类似DOS的批处理文件
  + 输入/输出重定向和管道
    - <,>,<<,>>,|

## 系统功能调用

* 程序级接口一系统调用是操作系统提供给软件开发人员的接口，用户在编写程序时使用操作系统提供的系统功能调用。OS核心中都有一组实现系统功能的过程（子程序），系统调用就是对上述过程的调用。
* 系统调用及实现
  + 程序的状态：计算机系统中的程序大体上分为系统程序和用户程序，前者是后者的管理者。为了便于管理，引入了
    - 管态（系统态）：操作系统程序运行的状态
    - 算态（目态）：用户程序运行的状态
* 特权指令：特权指令是一类只能在管态下执行而不能在算态下执行的特殊的指令。这些指令在不同的机器中有不同的规定，通常与硬件有很大的关系，常见的特权指令有如下几类：
  + 传送程序状态字的指令
  + 启动、测试和控制外设的指令
  + 存取特殊寄存器的指令
* 系统功能调用
  + 访管指令：本身不是特权指令，基本功能是“自愿进管”，能引起访管中断。
  + 系统功能调用：就是用户在程序中用访管指令调用由操作系统提供的子功能集合。有时把其中的每一个子功能称为一条广义指令。



* 系统调用的功能
  + 设备管理：设备的读写与控制
  + 文件管理：文件读写、文件控制和文件保护
  + 进程控制：创建、终止、暂停等控制
  + 进程通信：消息队列、共享存储区、socket等通信渠道的建立、使用和删除
  + 存储管理：内存的申请和释放
  + 系统管理：设置和读取时间、读取用户和主机标识等
* 文件管理：文件读写和文件控制

|  |  |
| --- | --- |
| Open | 文件打开。 |
| Close | 文件关闭 |
| Read | 读文件 |
| Write | 写文件 |
| Seek | 读写指针定位 |
| Stat | 读文件状态 |
| Mount | 安装文件系统 |
| Chmod | 修改问价属性 |

* 进程控制：创建、中止等控制；

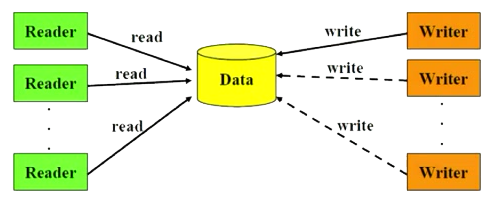
|  |  |
| --- | --- |
| Fork | 创建进程 |
| Exit | 进程自我终止 |
| Wait | 阻塞当前进程 |
| Sleep | 进程睡眠 |
| Getpid | 读父进程标识 |

* 系统调用与一般过程调用的区别
  + 运行在不同的系统状态：一般的过程调用，其调用和被调用的过程都运行在同一状态下，即管态或目态；而系统调用的调用过程是用户程序，运行在用户态，其被调用过程是系统过程，运行在系统态。
  + 进入方式不同：一般的过程调用可直接由调用过程转向被调用过程；而执行系统调用时只能通过软中断机制，先进入操作系统核心，才能转向相应的处理程序。
  + 返回问题：一般的过程调用，当被调用过程执行完后，将返回到调用过程继续执行。然而在采用抢先式调度的系统，在系统调用返回时，要进行重新调度的检查一是否有更高优先级的任务就绪。
  + 嵌套或递归调用：对系统调用，一般不允许在同一个进程中发生嵌套或递归（不同进程可以重入同一个系统调用）。

# 进程间相互作用

## 读者问题

问题描述：一个数据对象（文件、记录）可以为多个并发进程共享。其中有的进程只需要读其中的内容，我们称为“读者”；有的进程负责更新（读写）其中内容，我们称为“写者”。规定：“读者”可以同时读取共享数据对象；“写者”不能和其它任何进程同时访问共享数据对象。

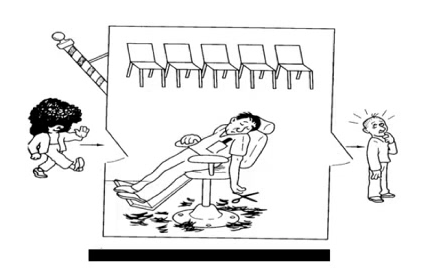


* 分析：
  + 进程行为：
  + 读进程的行为：系统中会有多个读进程同时访问共享数据。我们可以将他们分为三类：第一个进入的读进程（占有资源）、最后一个离开的读进程（释放资源）和其它读进程。因此需要设置一个计数器readnum来记录读进程的数目。写进程的行为：排他性的使用资源。
* 确定同步和互斥关系：
  + “读者-读者”：互斥访问readnum
  + “读者-写者”：互斥访问Data
  + “写者-写者”：互斥访问Data
* 确定临界资源：
  + Data：readnum

**算法：**

|  |  |
| --- | --- |
| 设置信号量：  Int readnum = 0;//计数，用于记录读者的数目  semaphore mutex = 1;//公用信号量，用于readnum互斥  Semaphore write = 1;//公用信号量，用于Data访问的互斥 | |
| 读者进程代码：  P(mutex); //对readnum互斥  Readnum++; //读者数目加1  If(readnum == 1) //第一个读进程  P(write); //申请使用data资源  V(mutex); //释放readnum  reading;  P(mutex); //对readnum互斥  readnum--;  if(readnum == 0) //最后一个读进程  V(write); //释放data资源  V(mutex); //释放readnum | 写者进程代码  P(write);//申请使用data资源  writing；  V(write);//释放data资源 |

## 理发师问题



问题描述（简化的）：理发店有一位理发师和一把理发椅。如果没有顾客，则理发师在理发椅上睡觉；当有颐客到达时，如理发师在睡觉则唤醒他理发，如果理发师正忙着理发，则坐在椅上等待。编写程序实现理发师和顾客行为的正确描述。

* 分祈
  + 理发师行为：睡觉或理发。没有顾客睡觉，有顾客理发。
  + 顾客行为：理发或等待。到达后如果理发师睡觉，则唤醒理发师理发；如果理发师在理发，则坐下等待。
* 相互作用
  + 理发师和顾客之间：同步
  + 顾客和顾客之间：无

**算法：**

|  |  |
| --- | --- |
| 设置信号量：  Semaphore customers = 0; //Customers表示等候理发的顾客数量  Semaphore barbers = 0; //barbers表示等候顾客的理发师数量 | |
| 理发师进程代码  {  While(1)  {  P(customers);//检查是否有顾客  V(barbers);  Cuthair();  }  } | 顾客进程代码 {  V(customers);  P(barbers);//检测是否有理发师  Get\_haircuter);  } |

增加条件：理发店有n把椅子，顾客到达时如果理发师空闲则理发，如果理发师忙，则看椅子上是否还有空位置，有空位置等待，没有空位置就离开。请写出相应的进程。

|  |  |
| --- | --- |
| 设置信号量：  Semaphore customers = 0; //customers表示等候理发师的顾客数量  Semaphote barbers = 0; //barbers表示等候顾客的理发师数量  int waiting = 0; //等候人数  Semaphore mutex = 1; //用于waiting的互斥 | |
| 理发师进程  {  P(customers);//检查是否有顾客  P(mutex);  Waiting--;  V(mutex);  V(barbers);  Cut hair();  } | 顾客进程  {  P(mutex);  if(waiting < n) then  {  waiting = waiting + 1;  V(mutex);  V(customers);  P(barbers);//检测是否有理发师  Get\_haircuter();  }  else  {  V(mutex);  }  } |

## 管程

* 管程（Monitor)
  + 是一种解决进程间相互作用的抽象数据类型。
  + 是由局部于自己的若干公共变量及其说明和所有访问这些公共变量的过程所组成的软件模块。
* 管程的特征
  + 模块化：管程是一个基本的软件模块，可以被单独编译。
  + 抽象数据类型：管程中封装了数据及对于数据的操作。
  + 信息隐藏：管程外的进程或其他软件模块只能通过管程对外的接口来访问管程提供的操作，管程内部的实现细节对外界是透明的。
  + 使用的互斥性：任何一个时刻，管程只能由一个进程使用。进入管程时的互斥由编译器负责完成。
* 管程的结构

|  |  |
| --- | --- |
| TYPE monitor name=MONITOR  {  局部变量说明;  条件变量说明;  初始化语句;  define管程内定义的在管程外可调用的过程或函数名表;  use管程外定义的在管程内将调用的过程或函数名表;  PROCEDURE过程名（形参表）{  <过程/函数体>;  }  PROCEDURE 过程名（形参表）{  <过程/函数体>;  }  } |  |

管程的工作原理

|  |  |
| --- | --- |
| 条件变量（c）：是管程内的一种数据结构，类似信号量。只在管程中才能被访问，通过两个原语wait,signal操作来控制它。  wait(c):如果条件不满足，在c上阻塞调用进程，直到另一个进程在该条件变量上执行signal（）。  signal(c):在c上执行signal操作，如果存在其他进程由于c而被阻塞，唤醒之。 |  |

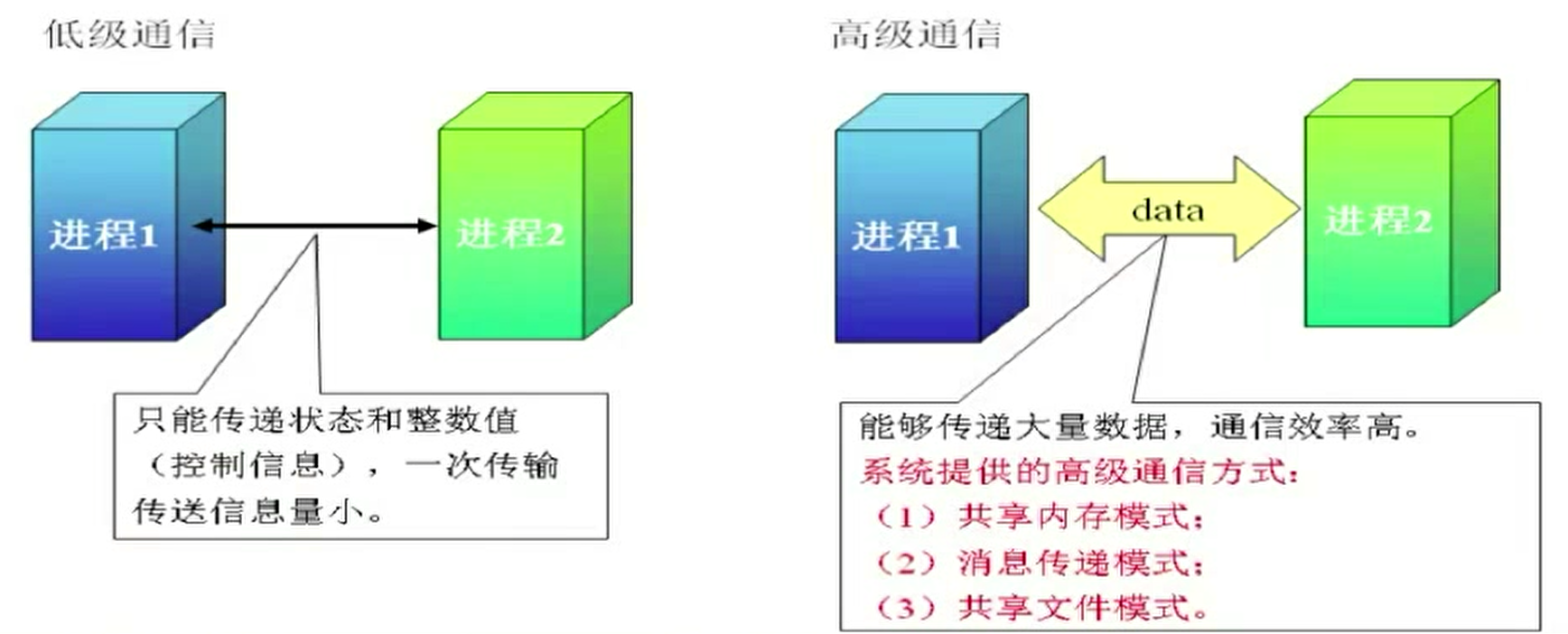
使用管程解决生产者消费者问题

|  |  |
| --- | --- |
| type ProducerConsumer=monitor{  condition notful1,notempty；//notfull缓冲区不满，notempty缓冲区不空  integer count；//count记录共有几件物品  define insert,remove;  use wait,signal；  procedure insert(){  if(count=n)wait(notfull)；  add to buffer；  count++；  signal(notempty)；  }  procedure removeO{  if(count=0)wait(notempty)；  get from buffer;  count--；  signal(notfull)；  ｝  count=0；  } | Producer(生产者进程）{  while(true){  produce；  ProducerConsumer.insertO；  } |
| Consumer(消费者进程）{  while(true){  ProducerConsumer.remove()；  consume；  } |

* 评价
  + 管程可以使得进程对资源的互斥由操作系统保证，同时集中存放的方法便于管理，从而减轻了程序员的负担，在一定程度上避免错误的发生。
  + 但是，管程不能保证程序执行其他逻辑的正确性，也就是说管程内函数的逻辑正确性仍然需要程序员保证，系统对此无能为力。
  + 故而大量进程/线程之间的逻辑正确性保证仍是一个困扰程序员的问题，时至今日仍未能很好的解决。

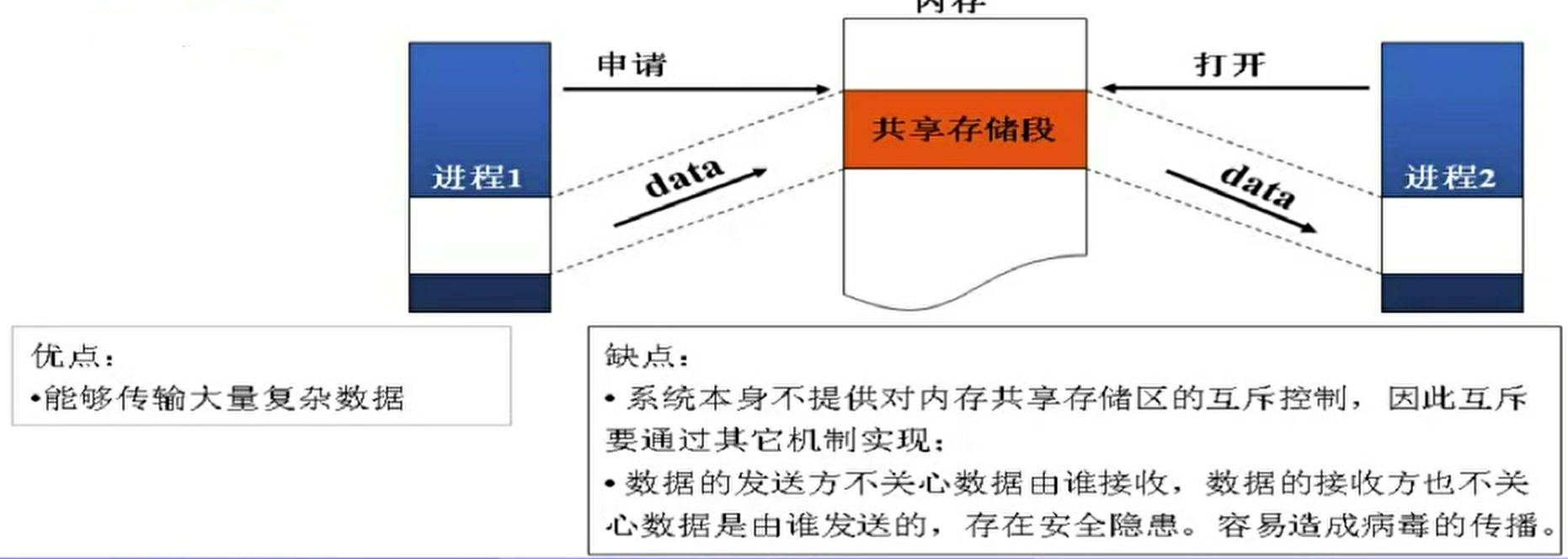
## 进程通信

* 进程通信：简单来说就是在进程间传输数据/交换信息。
* 根据交换信息量的多少和效率的高低，分为：

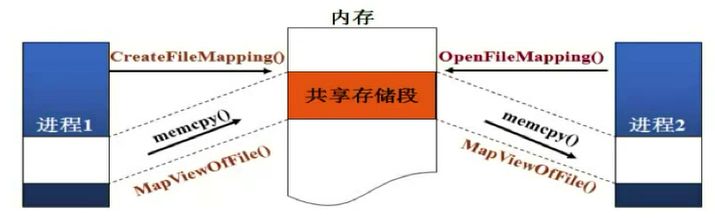


### 3.4.1.内存共享（sharing memory）

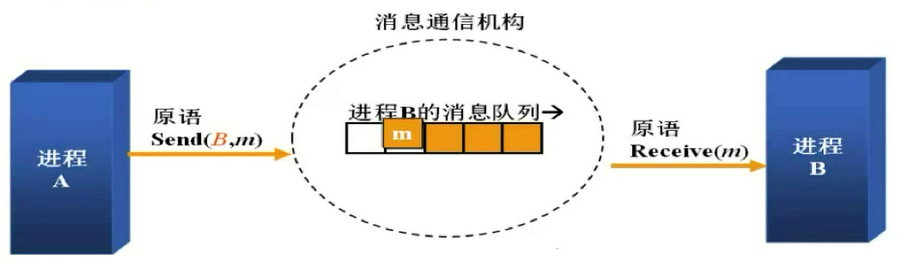
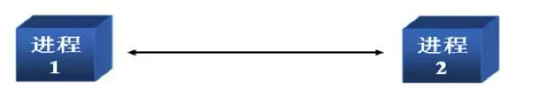
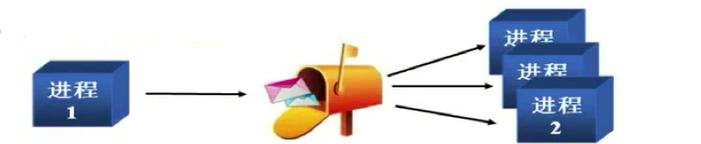
* 内存共享
  + 使用一段共享的内存区域交换数据。
  + 最为快捷有效的方式之一。
  + 原理



* Windows中共享内存模式的实现方式（API函数）
  + CreateFileMapping():函数创建一个内存映射文件对象
  + MapViewOfFile():将文件映射对象的视图映射进地址空间
  + memcpy()：数据复制到共享内存
  + OpenFileMapping():打开一个命名的共享文件对象



### 3.4.2.消息传递（message passing）

* 消息的概念：由发送方形成，通过一定的机制传递给接收方的一组信息，它的长度可以固定，也可以变化。
* 原理：  
  
* 消息传递的方式
  + 直接通信方式：点到点的发送。  
    Send(DestProcessName,Message):  
    Receive(SourceProcessName,Message):  
    
  + 间接通信方式：以信箱为媒介进行传递，可以广播。  
    Send(MailBox;Message):  
    Receive(MailBox,Message)；  
    

### 3.4.3.共享文件模式：管道（pipe)

* 共享文件模式：管道（pipe)
  + 管道是一种信息流缓冲机构（线性字节数组），使用文件读写方式进行访问。但管道并不是文件。
  + 以先进先出FIFO)方式的单向传送数据。
  + 匿名管道（Anonymous Pipes)
    - 通常用来在父进程和子进程之间传输数据。匿名管道总是本地的，不能在网络之间传递数据。
  + 命名管道（Named Pipes)
    - 命名管道是一种有名称的，可在同一台计算机的不同进程之间或跨越一个网络的计算机的进程之间传输数据。
  + 原理及实现（匿名管道）

|  |  |
| --- | --- |
|  | UNIX管道的实现：  创建管道：  int pipe(int fd[2])；  //fd[0]为管道里的读端  //fd[1]则为管道的写端  写管道：  int write(int handle,char \*buf,unsigned len)  读管道  int read(int handle,void \*buf,unsigned len) |

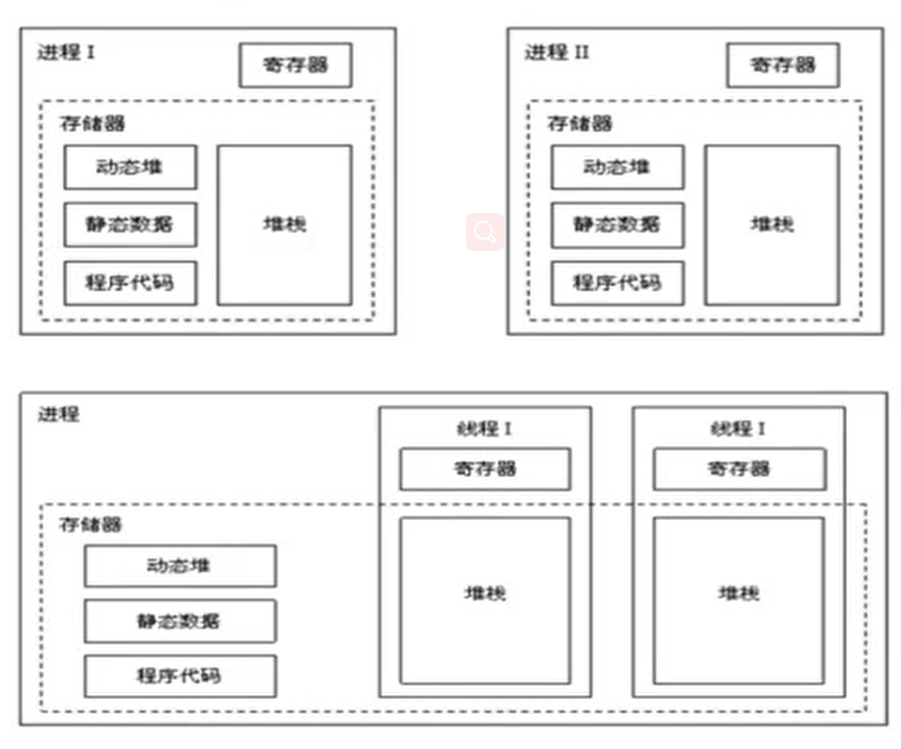
## 线程的基本概念

### 3.5.1.为何要有线程



**线程（thread）**：也叫轻量级进程，是进程中的一个可执行实体单元，是现代操作系统中处理机调度的基本单位。

### 3.5.2.线程和进程的关系



|  |  |
| --- | --- |
| 线程共享的资源：  地址空间  全局变量  打开的文件  子进程  信号量 | 线程独享的信息：  程序计数器  寄存器  栈  状态字  TCB |
| * 线程是进程的一个组成部分，一个进程中至少存在一个线程，线程还可以创建其它线程。 * 进程是资源分配和保护的基本单位，线程只能在进程的地址空间活动，线程只能使用其所在进程已经拥有的资源。 * 线程的创建和切换开销很小，拥有更高的并发度。线程的数量要比进程的多许多。 * 同一进程的线程之间拥有很高的通信效率。 * 每个线程拥有独立的上下文。 | |

### 3.5.3.线程的实现（管理）方式

* 线程的实现（管理）方式
  + 根据管理者身份的不同，分为：
    - 内核态线程：让操作系统来管理线程
    - 用户态线程：让进程来管理自己的线程
* 内核态线程

|  |  |
| --- | --- |
|  | |
| 优点：  用户编程简单 | 缺点：  效率低。每次调度均需要陷入内核态。  系统内核空间消耗巨大。 |

* 用户态线程

|  |  |
| --- | --- |
|  | |
| 优点：  灵活。可移植性好。  切换开销小。 | 缺点：  程序员需要考虑调度问题。  一个线程阻塞则整个进程受阻。 |

* 混合态线程

|  |
| --- |
|  |