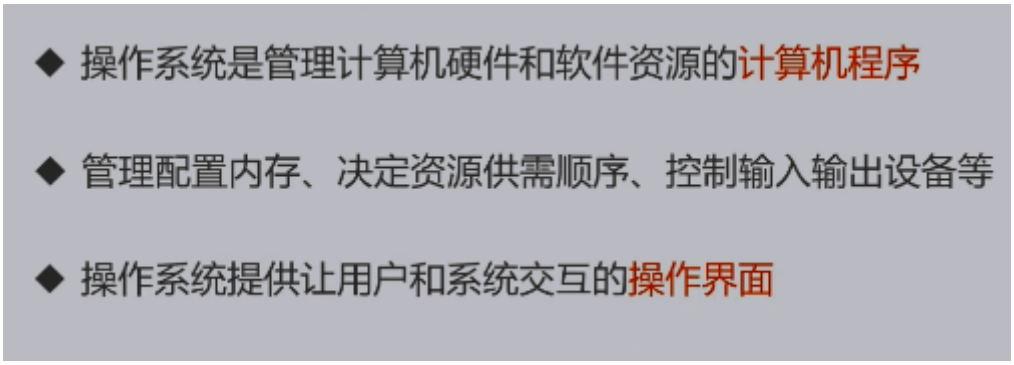
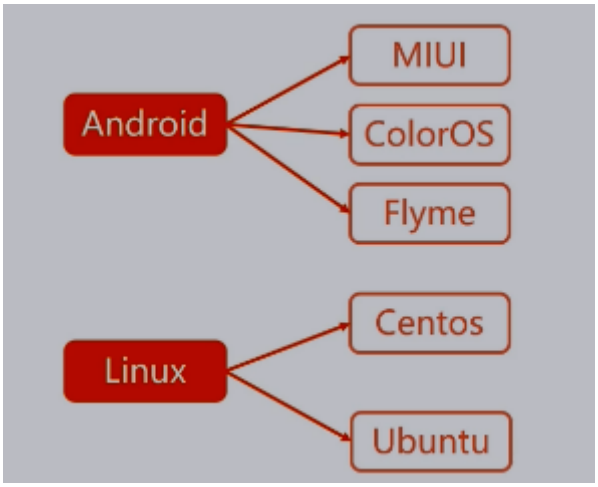
操作系统基础篇

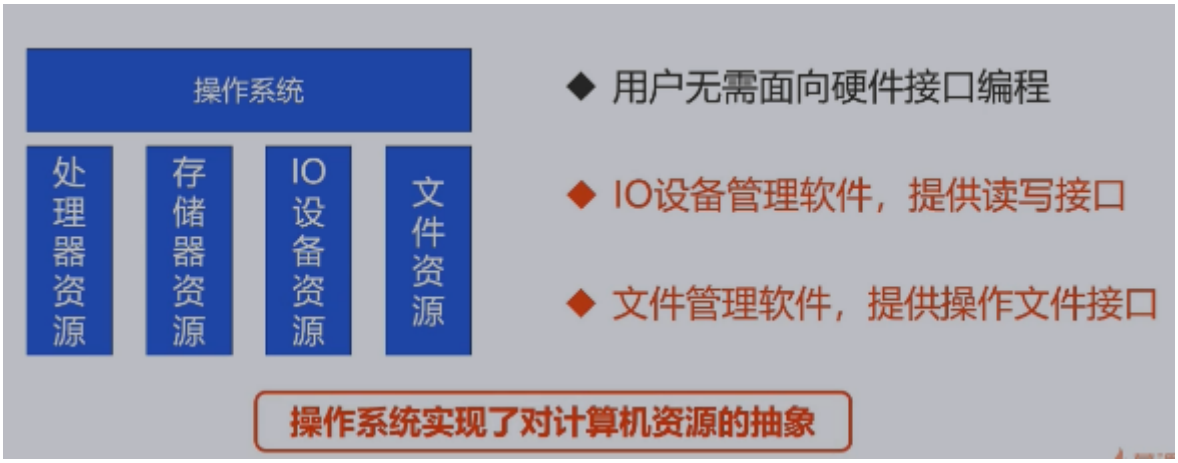
操作系统概览

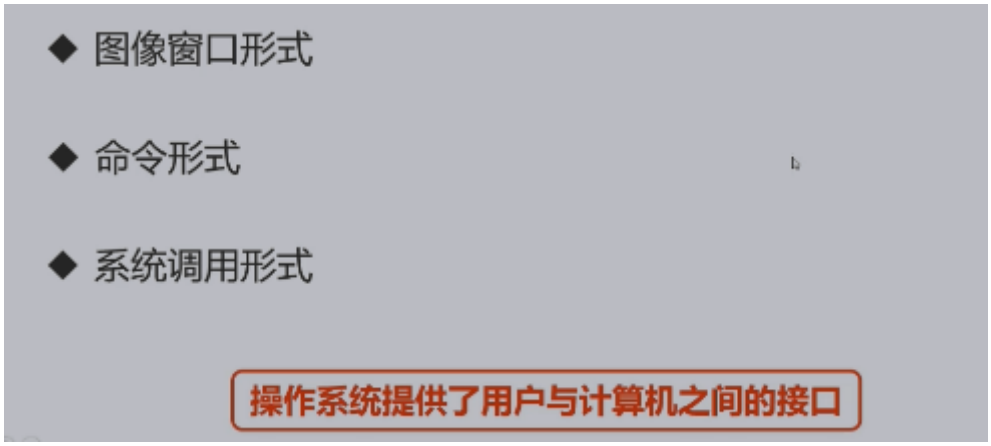
What&Why

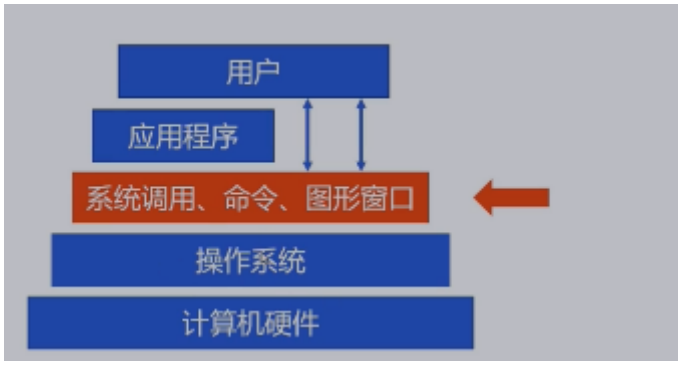




基本功能

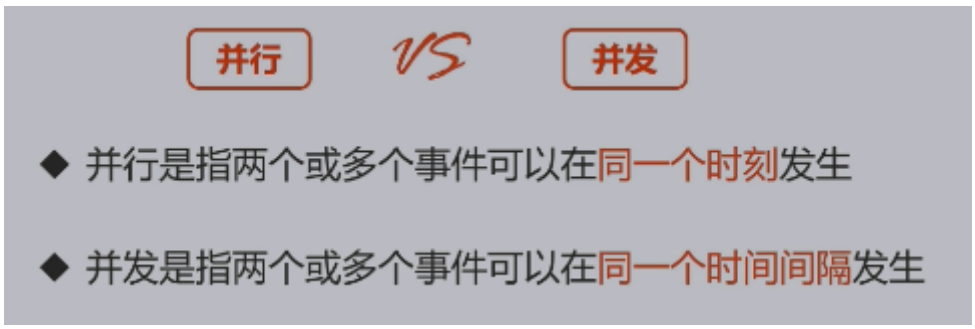

操作系统
处理器资源
存储器资源
IO设备资源
文件资源
◆ 用户无需面向硬件接口编程
◆ IO设备管理软件，提供读写接口
◆ 文件管理软件，提供操作文件接口
操作系统实现了对计算机资源的抽象


◆ 图像窗口形式
◆ 命令形式
◆ 系统调用形式
操作系统提供了用户与计算机之间的接口

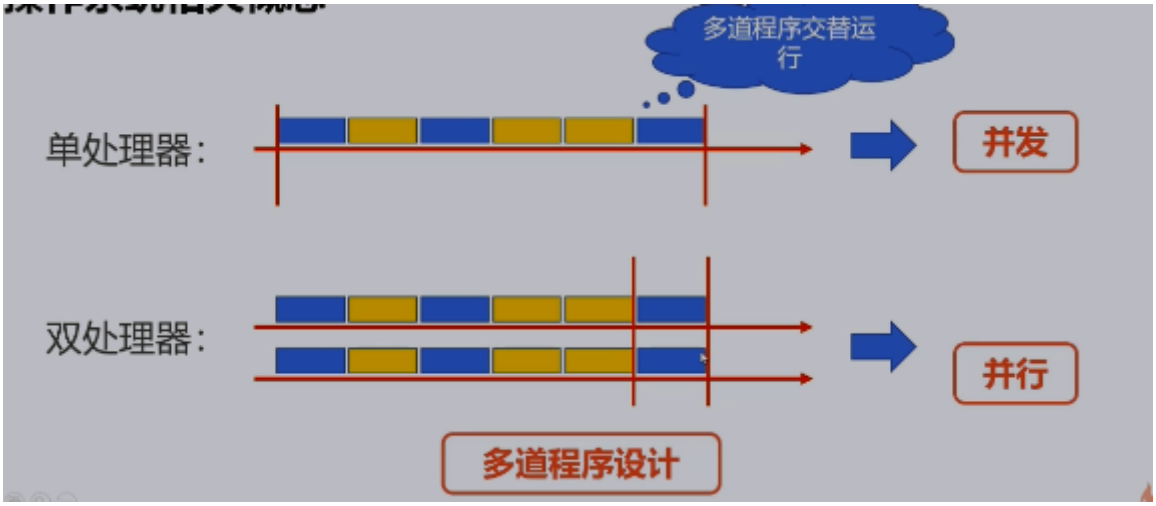

用户
应用程序
系统调用、命令、图形窗口
操作系统
计算机硬件

相关概念

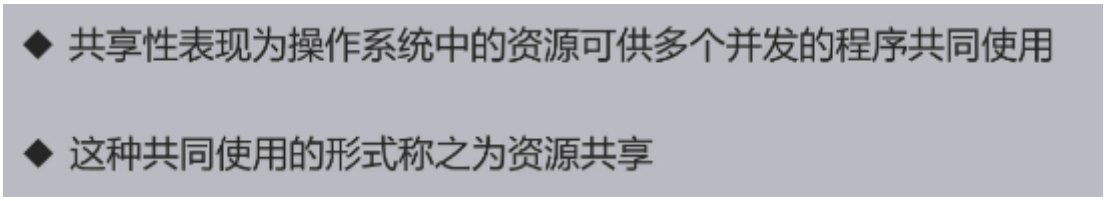
并发性

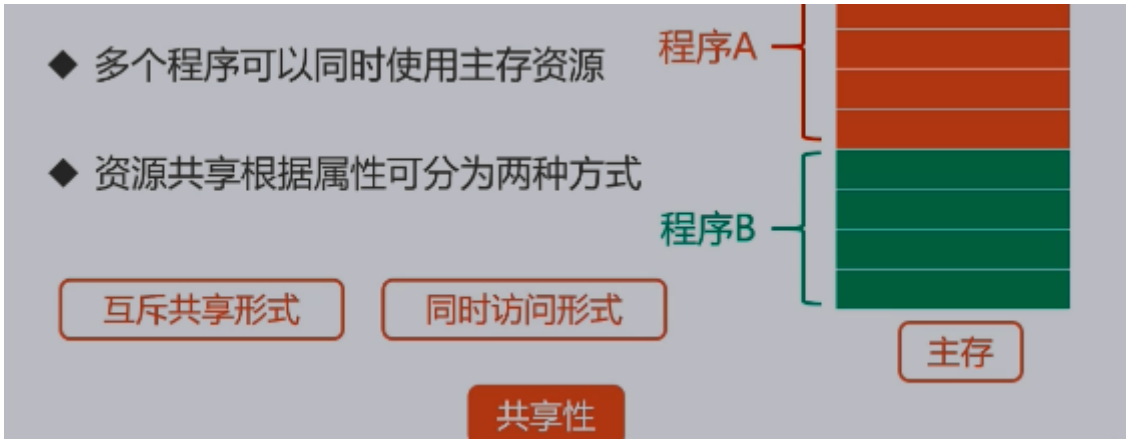

并行
VS
并发
◆ 并行是指两个或多个事件可以在同一个时刻发生
◆ 并发是指两个或多个事件可以在同一个时间间隔发生


◆ 8:00P.M. 咚咚呛老师在准备课程
◆ 8:00P.M. 同学们在认真学习
咚咚呛
同学们
8:00P.M.
并行
◆ 8:00P.M. 咚咚呛老师在准备课程
◆ 8:05P.M. 咚咚呛老师喝了口水
两个事件不是并行的
8:00P.M.
8:05P.M.
并发

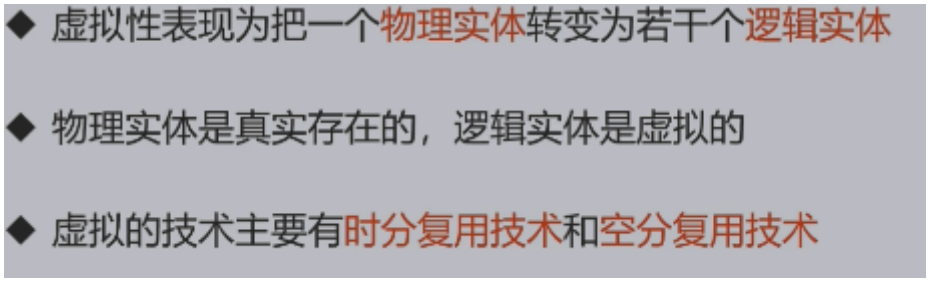

多道程序交替运行
单处理器:
并发
双处理器:
并行
多道程序设计

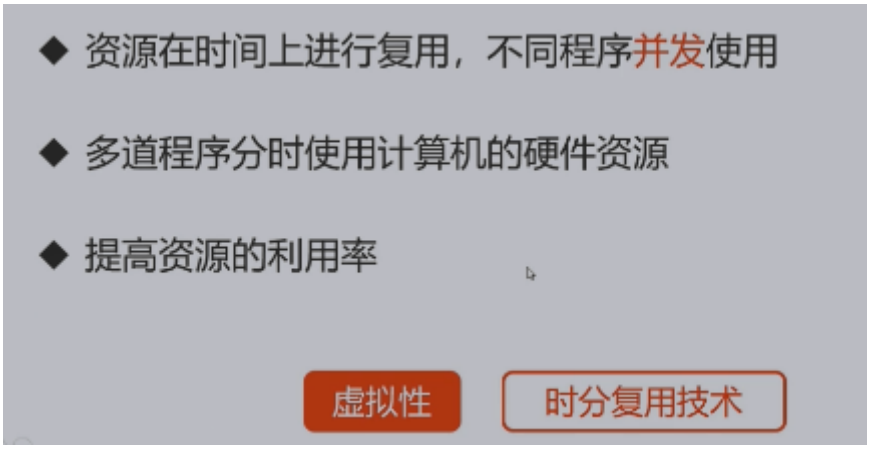
共享性

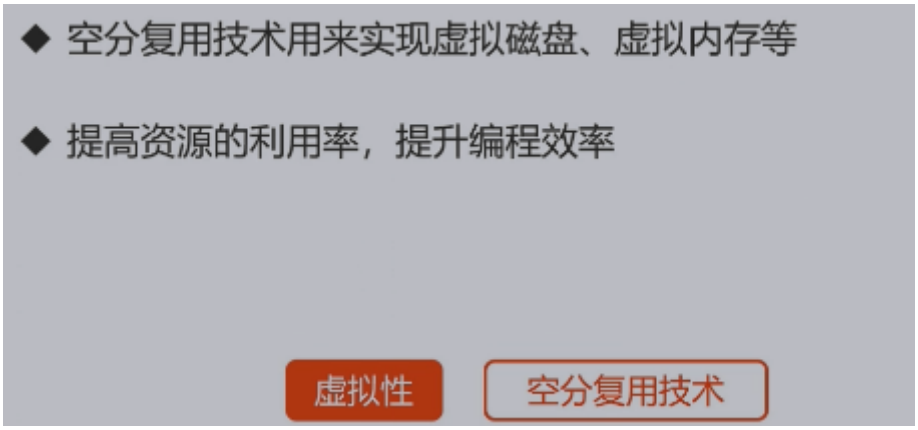

◆ 共享性表现为操作系统中的资源可供多个并发的程序共同使用
◆ 这种共同使用的形式称之为资源共享


◆ 多个程序可以同时使用主存资源
◆ 资源共享根据属性可分为两种方式
互斥共享形式
同时访问形式
共享性
程序A
程序B
主存

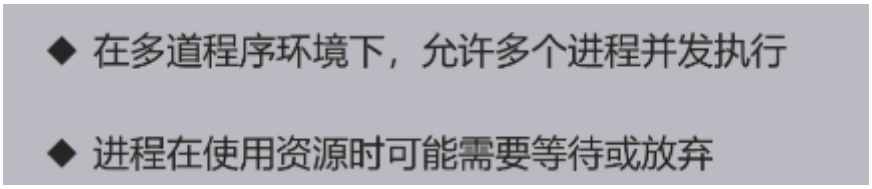
虚拟性


◆ 虚拟性表现为把一个物理实体转变为若干个逻辑实体
◆ 物理实体是真实存在的，逻辑实体是虚拟的
◆ 虚拟的技术主要有时分复用技术和空分复用技术



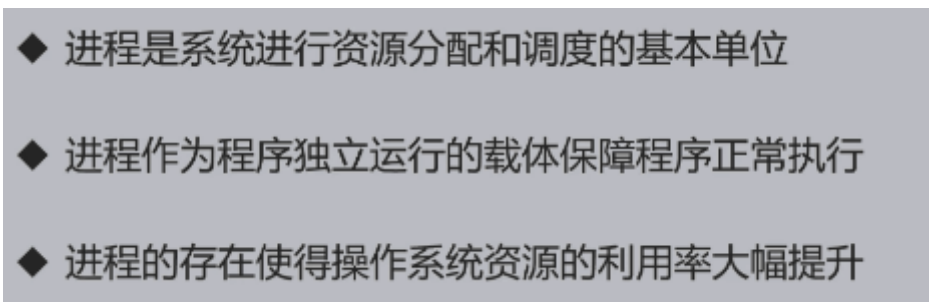


异步性



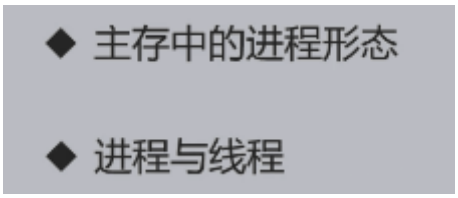
进程管理之进程实体

进程

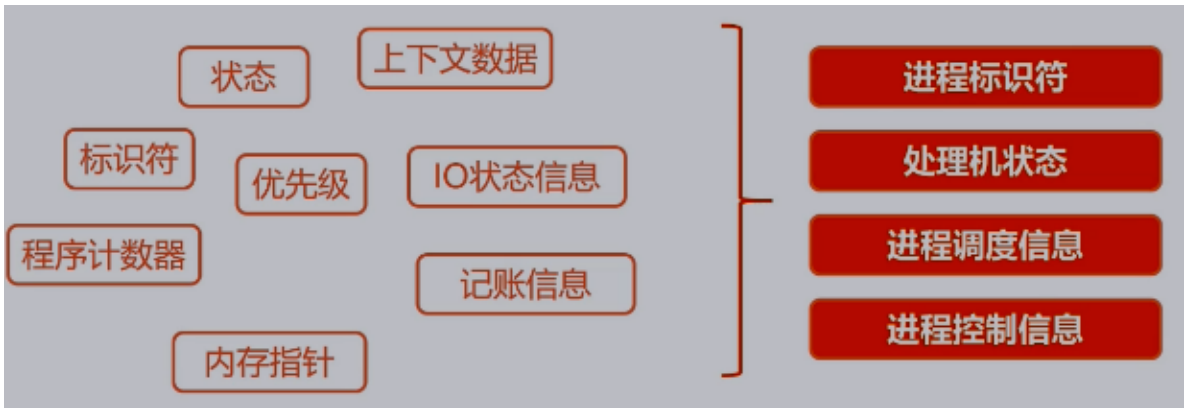


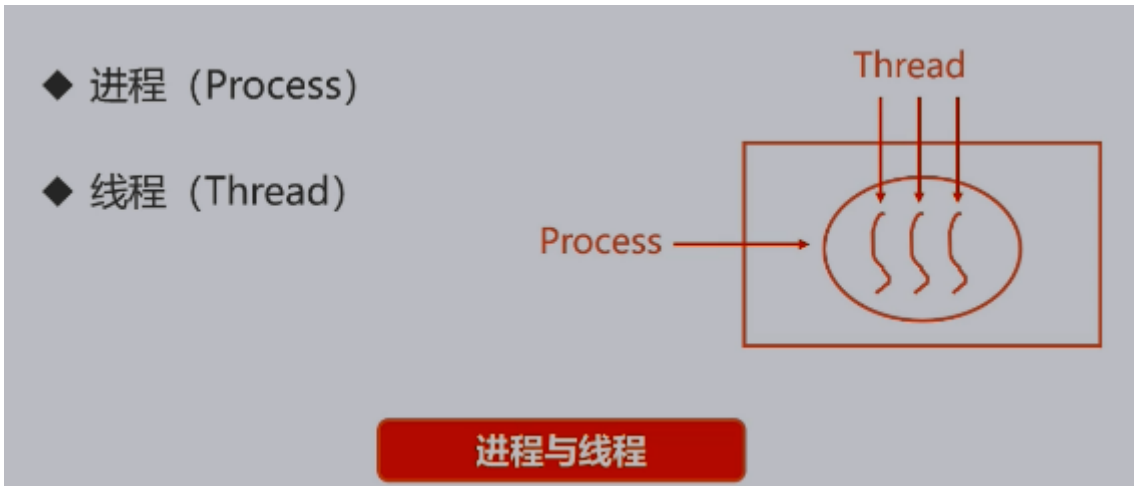
多道程序设计

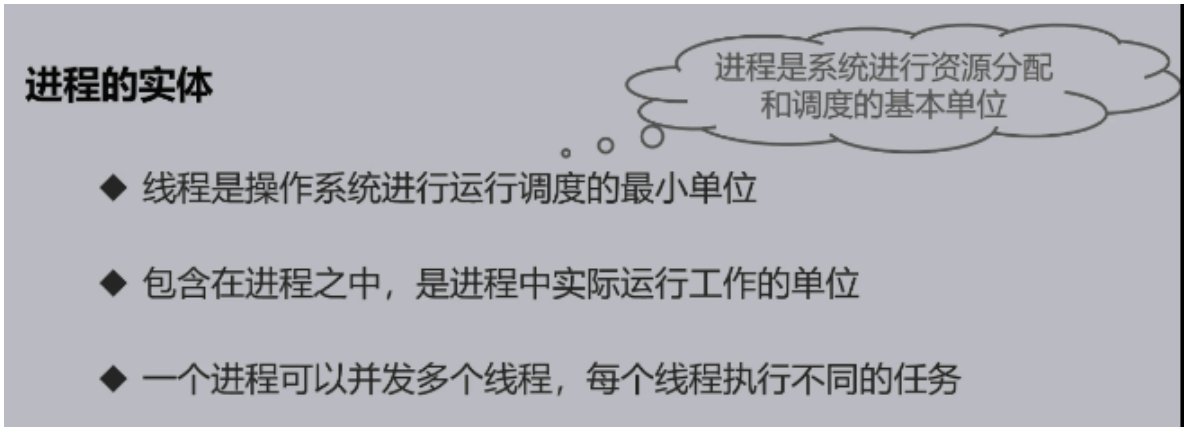
进程的实体


◆ 主存中的进程形态
◆ 进程与线程


状态
上下文数据
标识符
优先级
IO状态信息
程序计数器
记账信息
内存指针
主存中的进程形态
状态
优先级
程序计数器
内存指针
上下文数据
IO状态信息
记账信息
...
进程控制块

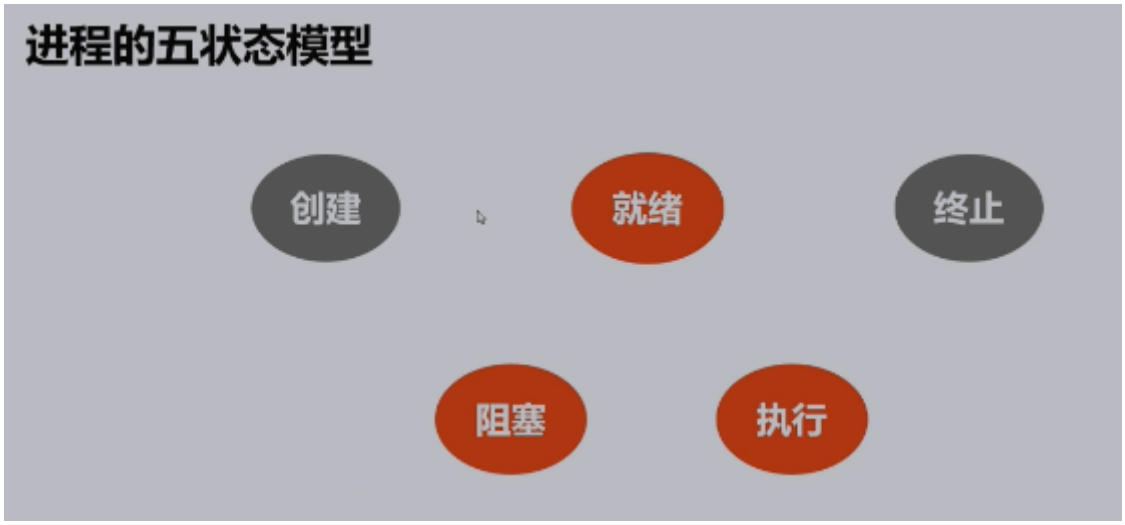

状态
上下文数据
标识符
优先级
IO状态信息
程序计数器
记账信息
内存指针
进程标识符
处理机状态
进程调度信息
进程控制信息


◆ 进程（Process）
◆ 线程（Thread）
Thread
Process
进程与线程

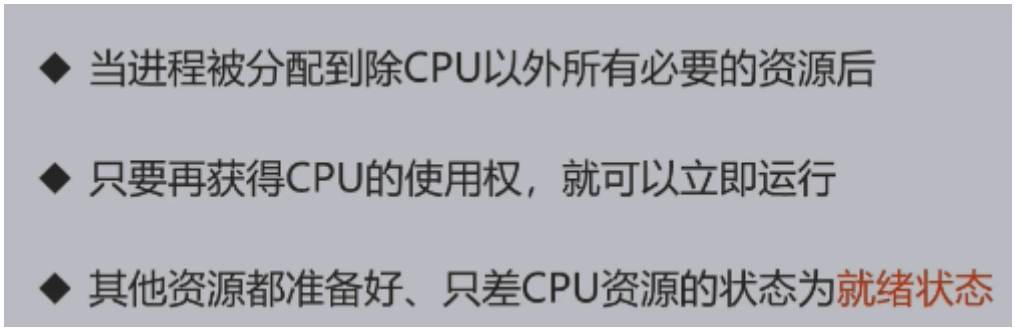


	进程	线程
资源	资源分配的基本单位	不拥有资源
调度	独立调度的基本单位	独立调度的最小单位
系统开销	进程系统开销大	线程系统开销小
通信	进程IPC	读写同一进程数据通信

进程管理之五状态模型



就绪



◆ 在一个系统中多个处于就绪状态的进程通常排成一个队列

执行

◆ 进程获得CPU，其程序正在执行称为执行状态

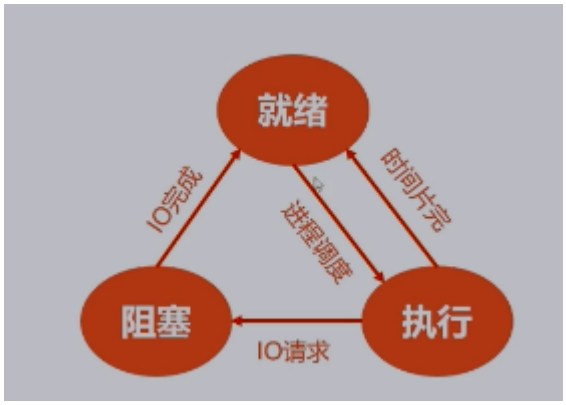
◆ 在单处理机中，在某个时刻只能有一个进程是处于执行状态

阻塞

◆ 进程因某种原因如：其他设备未就绪而无法继续执行

◆ 从而放弃CPU的状态称为阻塞状态

切换过程



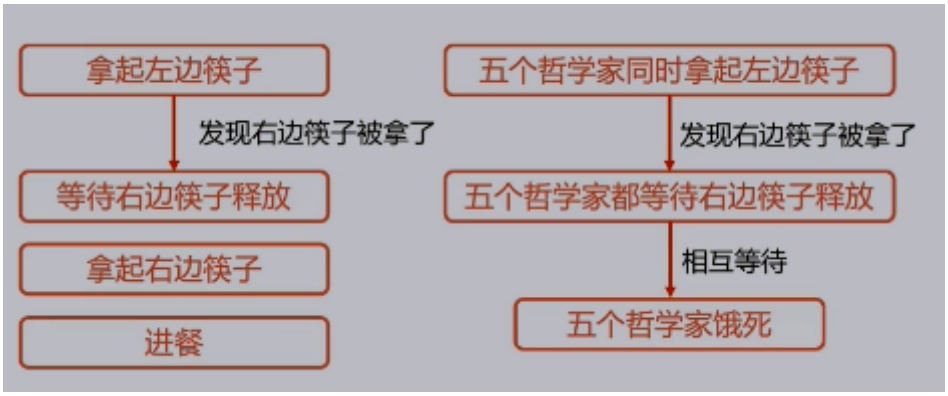
创建

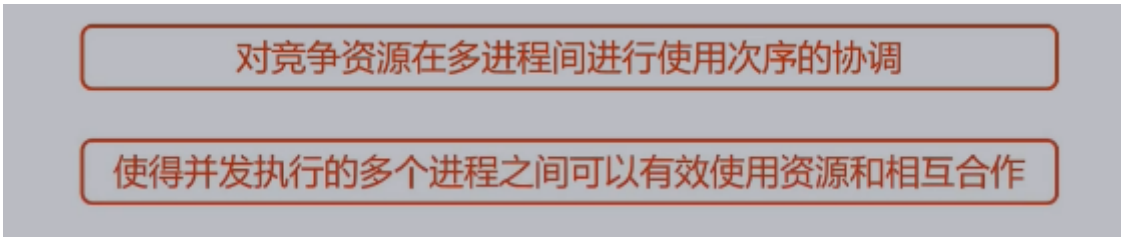
◆ 创建进程时拥有PCB但其他资源尚未就绪的状态称为创建状态

终止

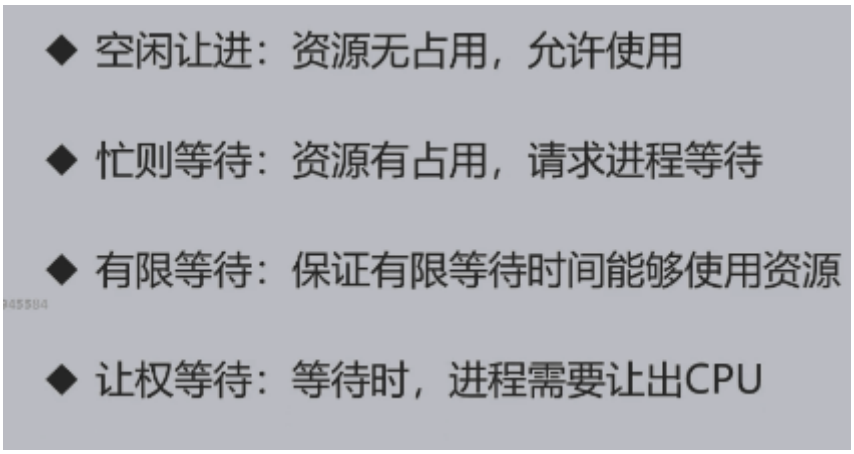
进程管理之进程同步

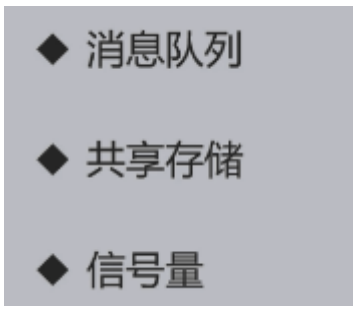
进程同步





原则

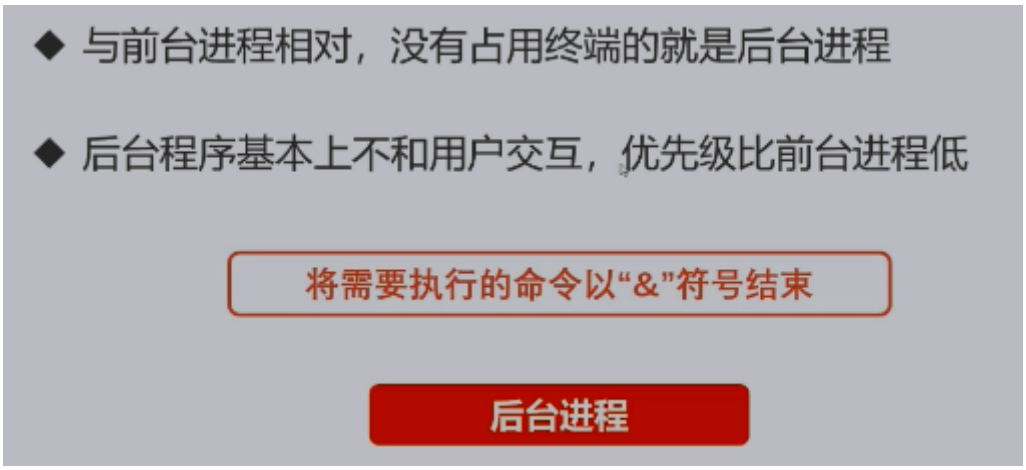


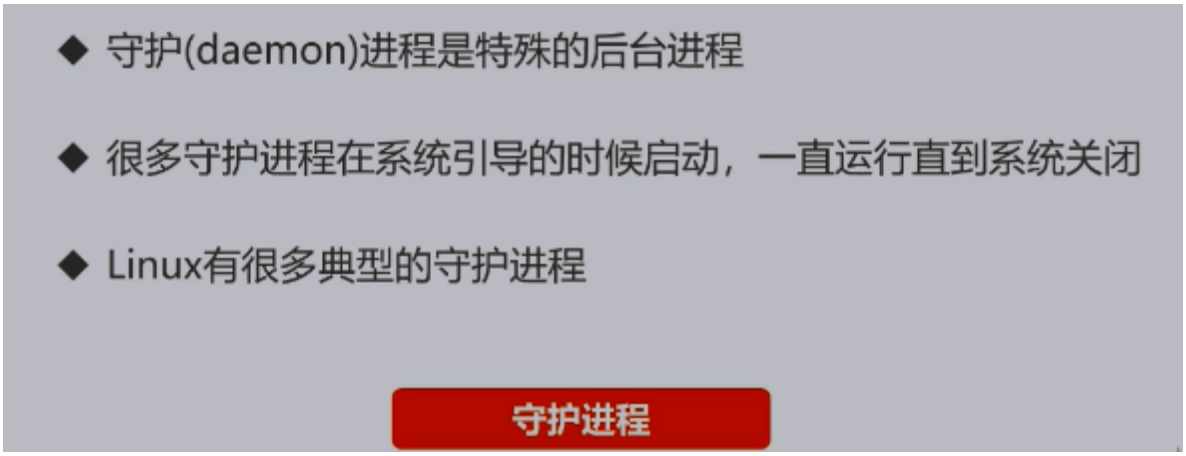


线程同步

Linux 进程管理

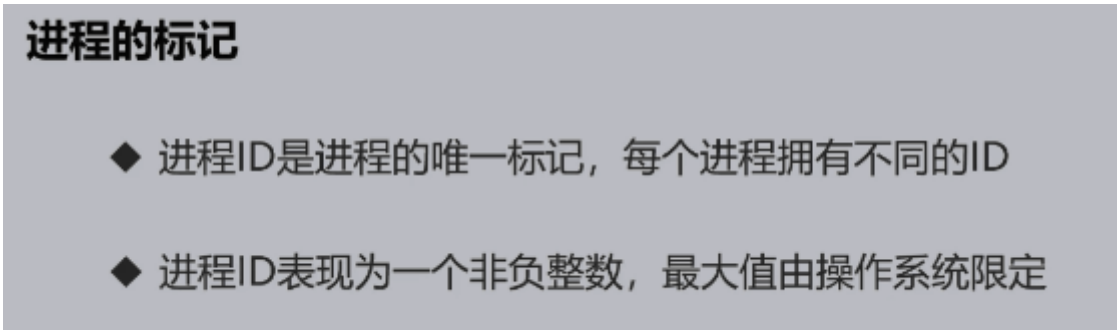
相关概念

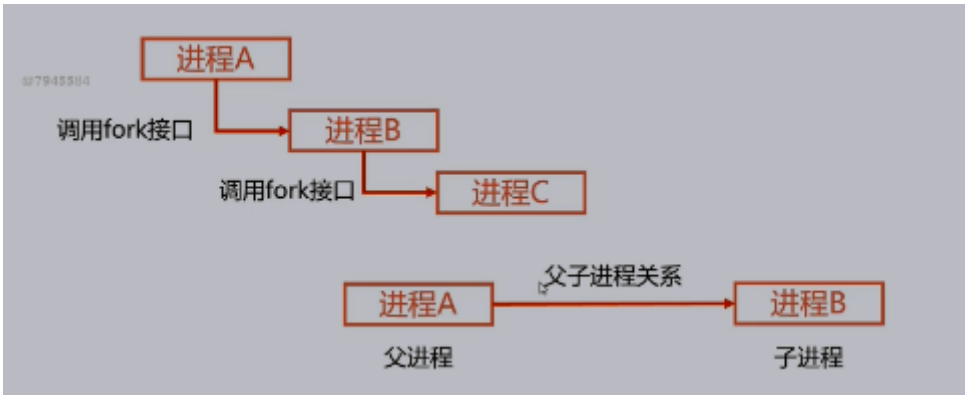


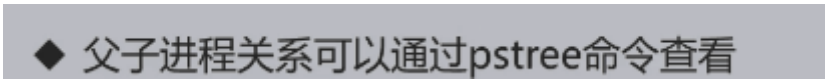


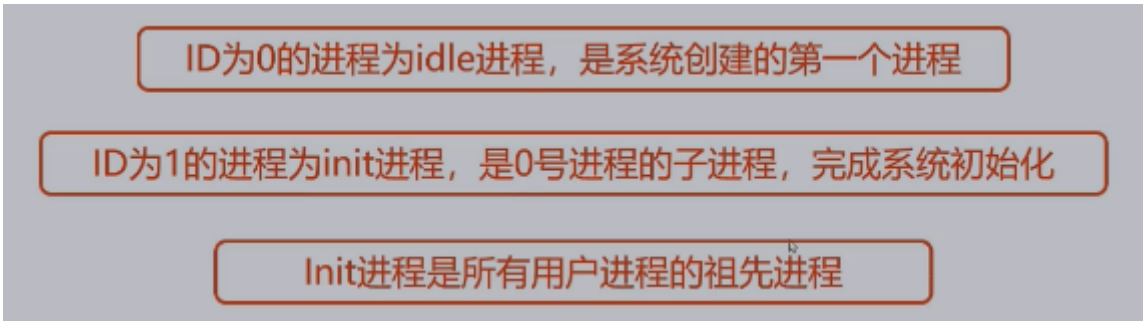

crond
sshd
httpd
mysqld
进程名字以“d”结尾的一般都是守护进程
守护进程

进程的标记


进程的标记
◆ 进程ID是进程的唯一标记，每个进程拥有不同的ID
◆ 进程ID表现为一个非负整数，最大值由操作系统限定


进程A
调用fork接口
进程B
调用fork接口
进程C
父子进程关系
进程A
进程B
父进程
子进程


◆ 父子进程关系可以通过pstree命令查看


ID为0的进程为idle进程，是系统创建的第一个进程
ID为1的进程为init进程，是0号进程的子进程，完成系统初始化
Init进程是所有用户进程的祖先进程

状态符号	状态说明
R	(TASK_RUNNING)，进程正处于运行状态
S	(TASK_INTERRUPTIBLE)，进程正处于睡眠状态
D	(TASK_UNINTERRUPTIBLE)，进程正在处于IO等待的睡眠状态
T	(TASK_STOPPED)，进程正处于暂停状态
Z	(TASK_DEAD or EXIT_ZOMBIE)，进程正处于退出状态，或僵尸进程

命令

- ps命令
- top命令
- kill命令

ps 列出命令

- ps命令常用于显示当前进程的状态
- ps命令常配合aux参数或ef参数和grep命令检索特定进程

TOP查看状态

KILL 给进程发信号

- kill命令发送指定信号给进程
- kill –l 可以查看操作系统支持的信号

只有(SIGKILL 9)信号可以无条件终止进程，其他信号进程有权忽略

作业管理之进程调度

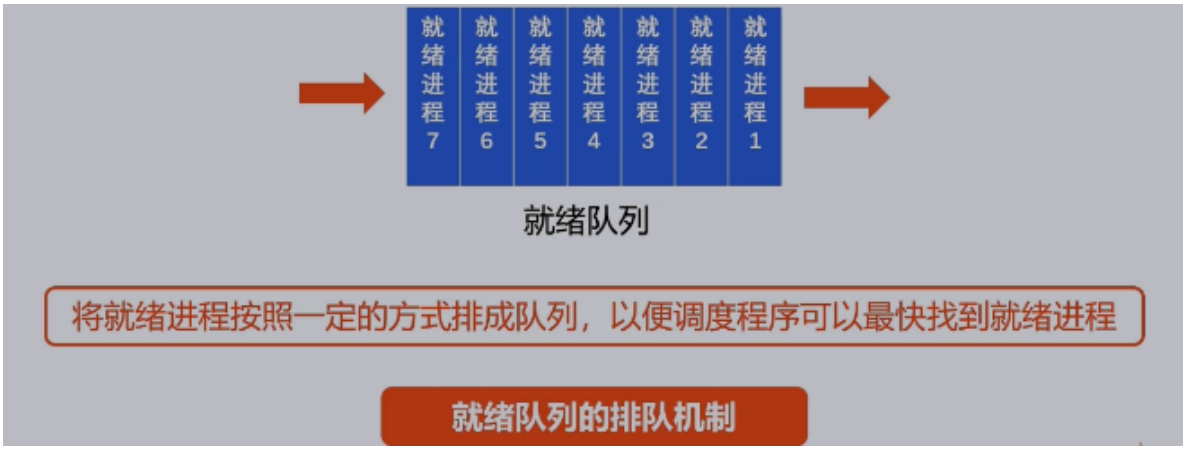
概述

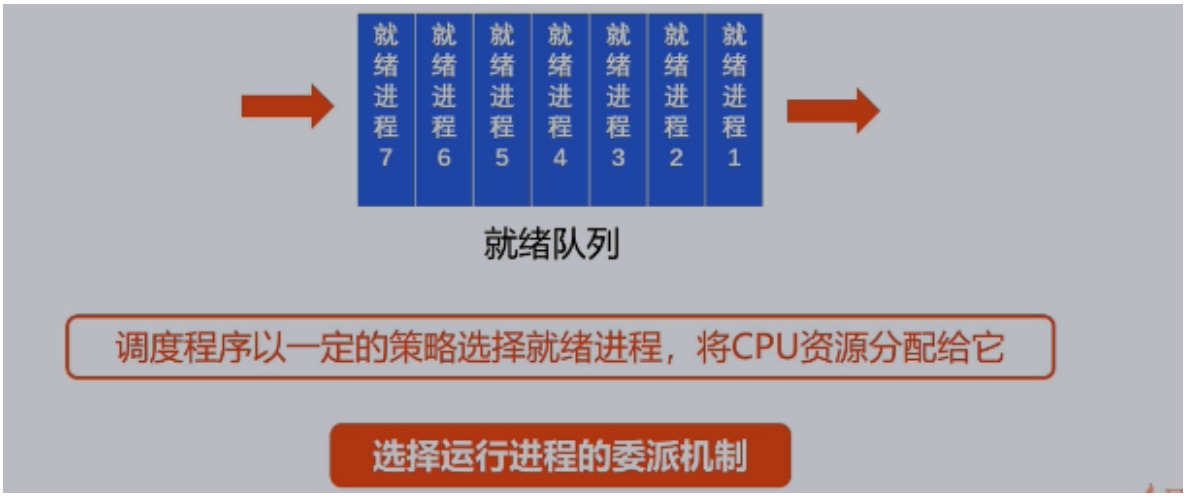
进程调度是指计算机通过决策决定哪个就绪进程可以获得CPU使用权

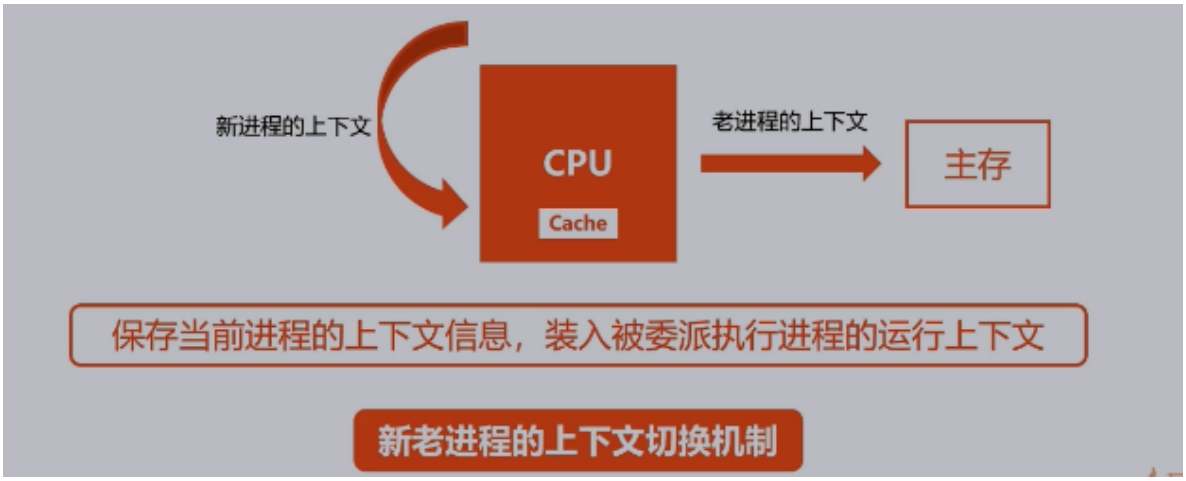
多道程序设计

- 保留旧进程的运行信息，请出旧进程（收拾包袱）

- 就绪队列的排队机制
- 选择运行进程的委派机制
- 新老进程的上下文切换机制







进程的调度

- 非抢占式的调度
- 抢占式的调度

进程的调度

- 处理器一旦分配给某个进程，就让该进程一直使用下去
- 调度程序不以任何原因抢占正在被使用的处理器
- 直到进程完成工作或因为IO阻塞才会让出处理器

非抢占式的调度

进程的调度

- 允许调度程序以一定的策略暂停当前运行的进程
- 保存好旧进程的上下文信息，分配处理器给新进程

抢占式的调度

	抢占式调度	非抢占式调度
系统开销	频繁切换，开销大	切换次数少，开销小
公平性	相对公平	不公平
应用	通用系统	专用系统

算法

- 先来先服务调度算法
- 短进程优先调度算法
- 高优先权优先调度算法
- 时间片轮转调度算法

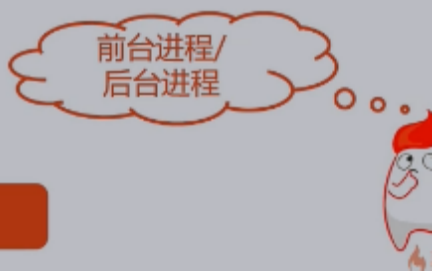
就绪队列

先来先服务调度算法

- 调度程序优先选择就绪队列中估计运行时间最短的进程
- 短进程优先调度算法不利于长作业进程的执行

短进程优先调度算法

- 进程附带优先权，调度程序优先选择权重高的进程
- 高优先权优先调度算法使得紧迫的任务可以优先处理



高优先权优先调度算法

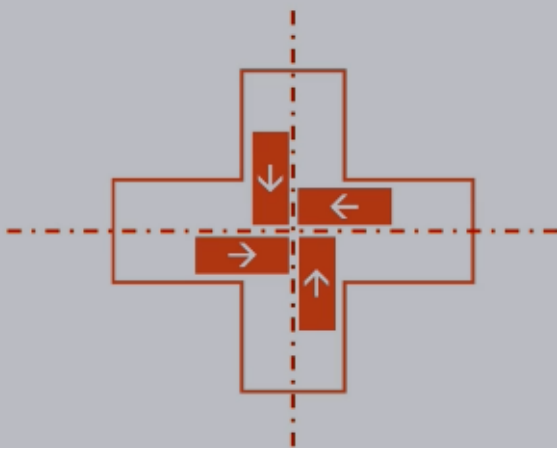
- 按先来先服务的原则排列就绪进程
- 每次从队列头部取出待执行进程，分配一个时间片执行
- 是相对公平的调度算法，但不能保证及时响应用户

时间片轮转调度算法

作业管理之死锁

死锁

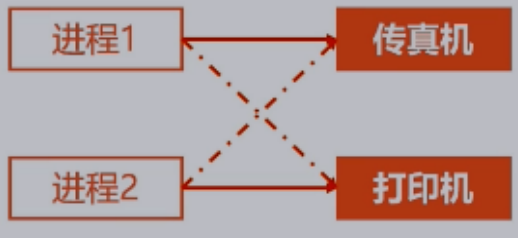
死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。



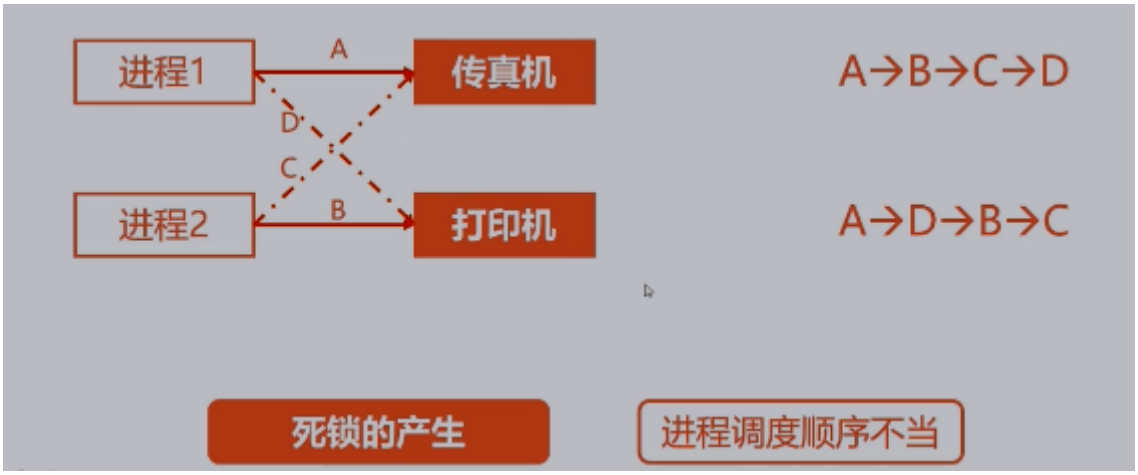
产生

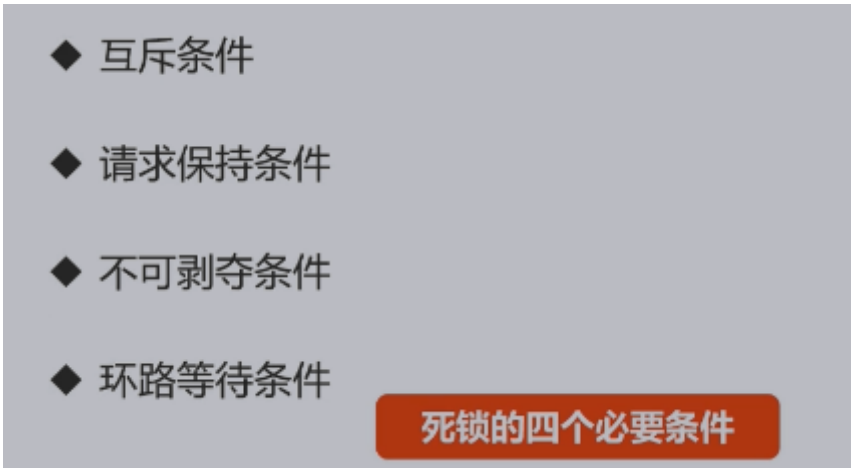
- 共享资源数量不满足各个进程需求
- 各个进程之间发生资源进程导致死锁

死锁的产生　竞争资源



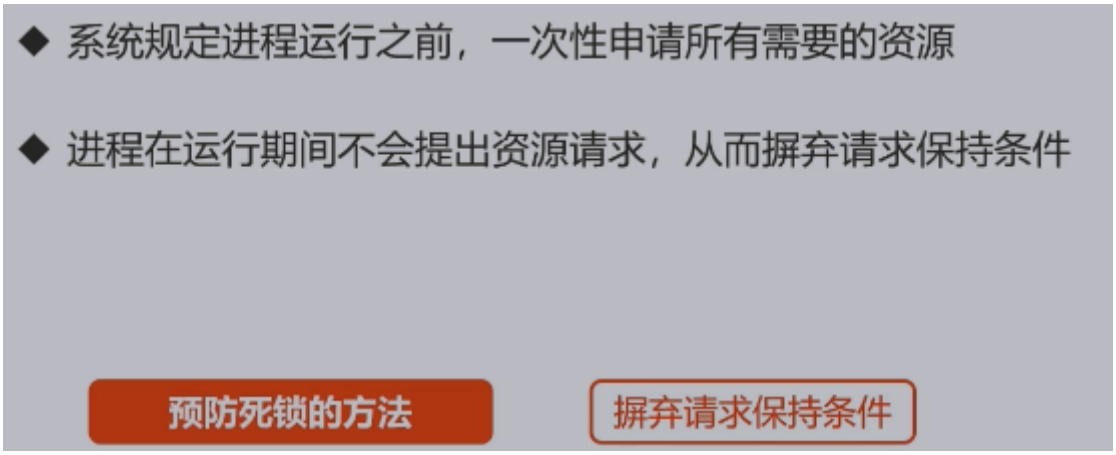
- 等待请求的资源被释放
- 自身占用资源不释放

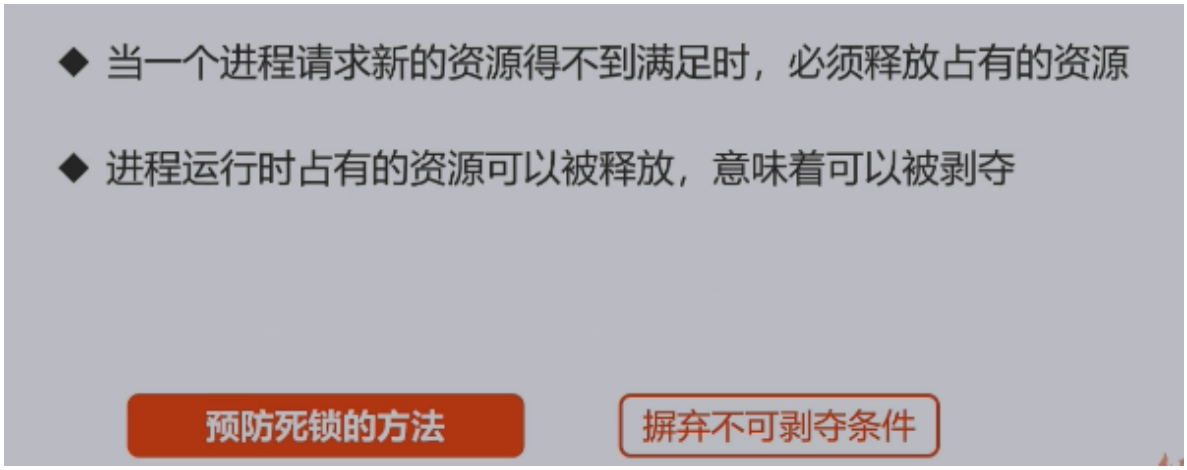

进程1
A
传真机
A→B→C→D
D
C
进程2
B
打印机
A→D→B→C
死锁的产生
进程调度顺序不当

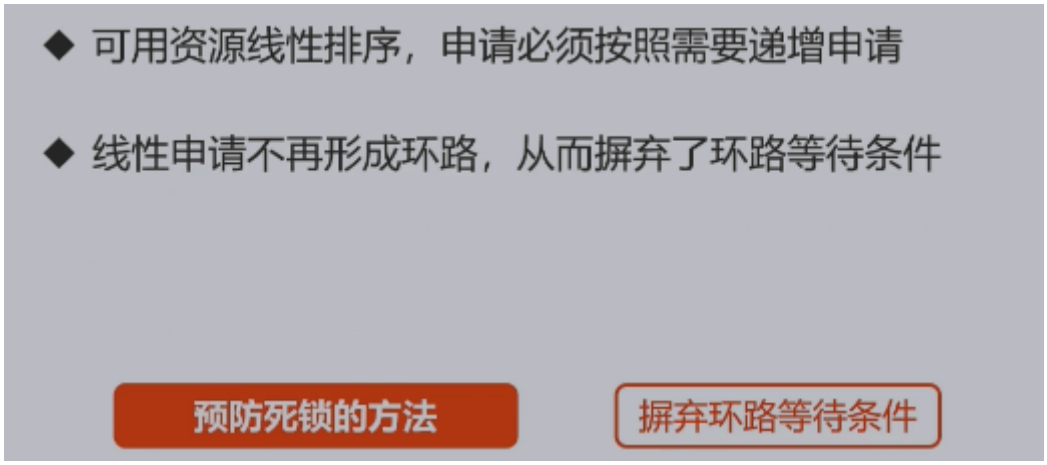

互斥条件
请求保持条件
不可剥夺条件
环路等待条件
死锁的四个必要条件

处理

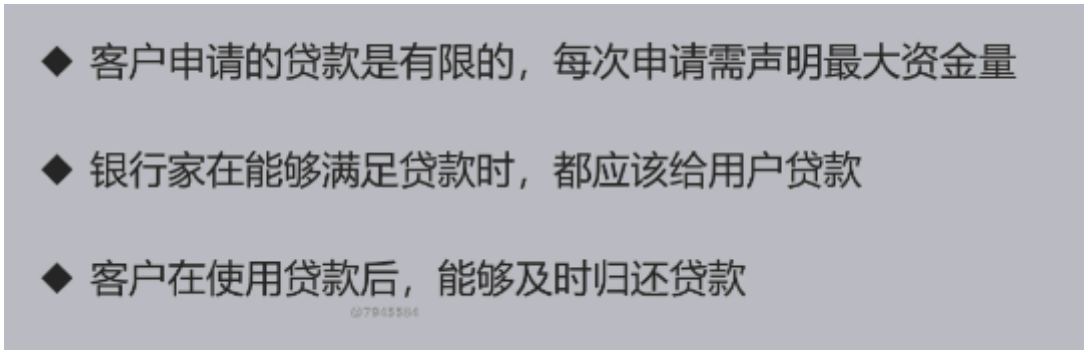
预防死锁的方法


系统规定进程运行之前，一次性申请所有需要的资源
进程在运行期间不会提出资源请求，从而摒弃请求保持条件
预防死锁的方法
摒弃请求保持条件

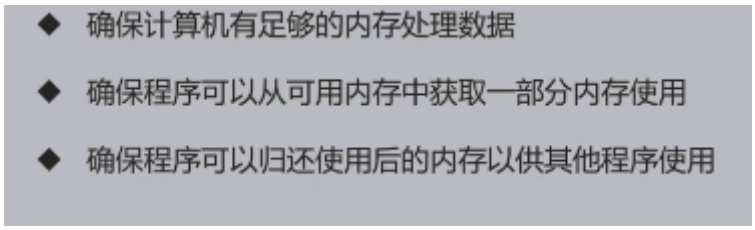

当一个进程请求新的资源得不到满足时，必须释放占有的资源
进程运行时占有的资源可以被释放，意味着可以被剥夺
预防死锁的方法
摒弃不可剥夺条件



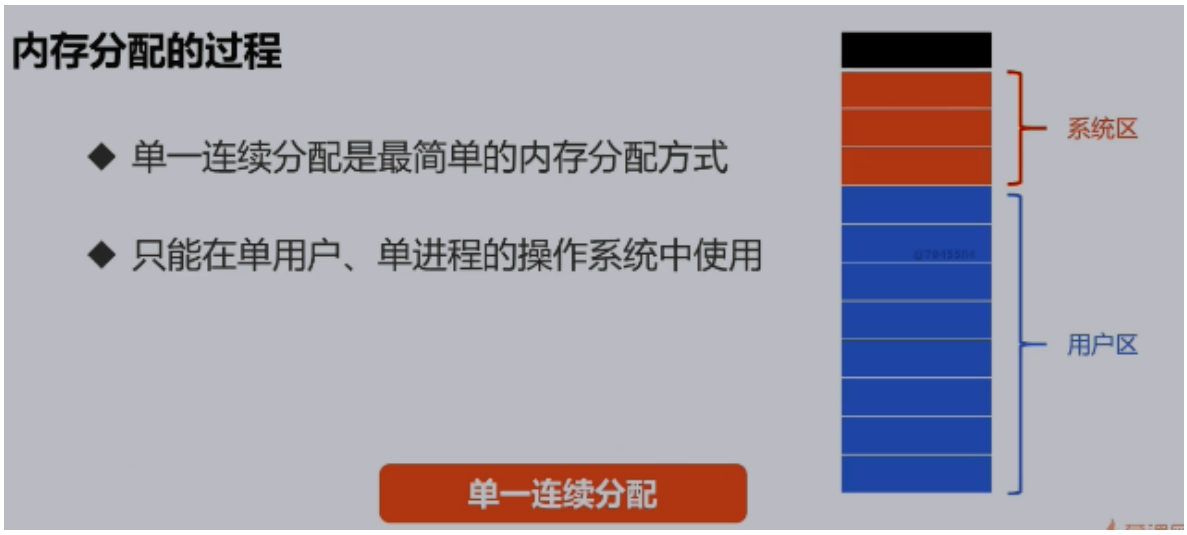
银行家算法



存储管理之内存分配与回收



分配的过程



内存分配的过程

- 固定分区分配是支持多道程序的最简单存储分配方式
- 内存空间被划分为若干固定大小的区域
- 每个分区只提供给一个程序使用，互不干扰

固定分区分配

内存分配的过程

分区	1	2	3	4	5	6	7	8	9	10	11
标记	0	1	0	0	1	1	0	0	1	1	0

动态分区分配 动态分区空闲表数据结构

内存分配的过程

- 节点需记录可存储的容量

动态分区分配 动态分区空闲链数据结构

内存分配的过程

- 首次适应算法(FF算法)
- 最佳适应算法(BF算法)
- 快速适应算法(QF算法)

动态分区分配　　动态分区分配算法

回收的过程

四种情况

- 不需要新建空闲链表节点
- 只需要把空闲区1的容量增大为空闲区即可

- 将回收区与空闲区合并
- 新的空闲区使用回收区的地址

◆ 将空闲区1、空闲区2和回收区合并

◆ 新的空闲区使用空闲区1的地址

◆ 为回收区创建新的空闲节点

◆ 插入到相应的空闲区链表中去