进程管理

## 什么是进程？

进程是一个具有一定独立功能的程序关于某个数据集合的一次运行活动，是系统进行资源分配和调度的一个独立单位。

* 进程是程序的一次执行
* 进程是一个程序及其数据在处理机顺序执行时所发生的活动
* 进程是程序在一个数据集合上运行的过程
* 进程是系统进行资源分配和调度的一个独立单位（基本单位）

### 进程的结构

* 控制块（PCB）——进程唯一标识
* 数据段——存放原始数据、中间数据
* 程序段——存放在文本区域，可被多个进程共享

### 进程的特征

程序实体，也叫进程映像。

* 动态性——由创建而生，由撤销而亡
* 并发性——多个进程同时运行
* 独立性——独立资源分配
* 异步性——相互独立、互不干扰

### 什么是线程

* 进程的轻型实体，也叫“轻量级进程”，是一系列活动按事先设定好的顺序依次执行的过程，是一系列指令的集合
* 是一条执行路径，不能单独存在，必须包含在进程中
* 线程是OS中运算调度的最小单位

### 为什么引入线程

* 提高OS的并发性

### 线程的属性

* 轻型实体
* 独立调度和分派的基本单位
* 可并发执行
* 共享进程资源

### 进程与线程比较

* 调度
* 拥有资源
* 并发性
* 系统开销
* 地址空间和其他资源
* 通信

线程相对于进程，大大降低了创建、撤销和切换可执行实体的成本和难度。

### 线程的实现方式

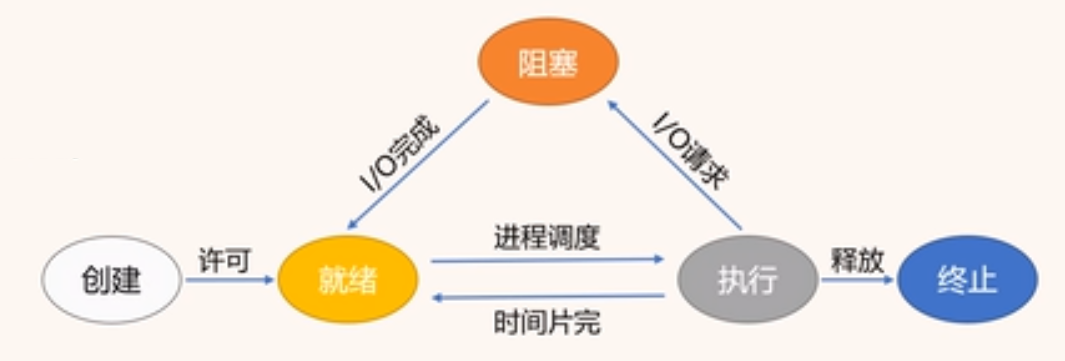
PCB在用户空间创建还是在内核空间创建。

* 用户级线程（ULT）
* 内核级线程（KLT）

## 进程是怎样运行的？

### 进程的状态

* 就绪（Ready）——可运行，但未运行，就差cpu
* 执行（Running）
* 阻塞（Blocked）
* 创建（New）
* 终止（Terminated）



### 进程控制

即OS对进程实现有效的管理，包括创建新进程、撤销已有进程、挂起、阻塞和唤醒、进程切换等多种操作。OS通过原语操作实现进程控制。

#### 原语的概念

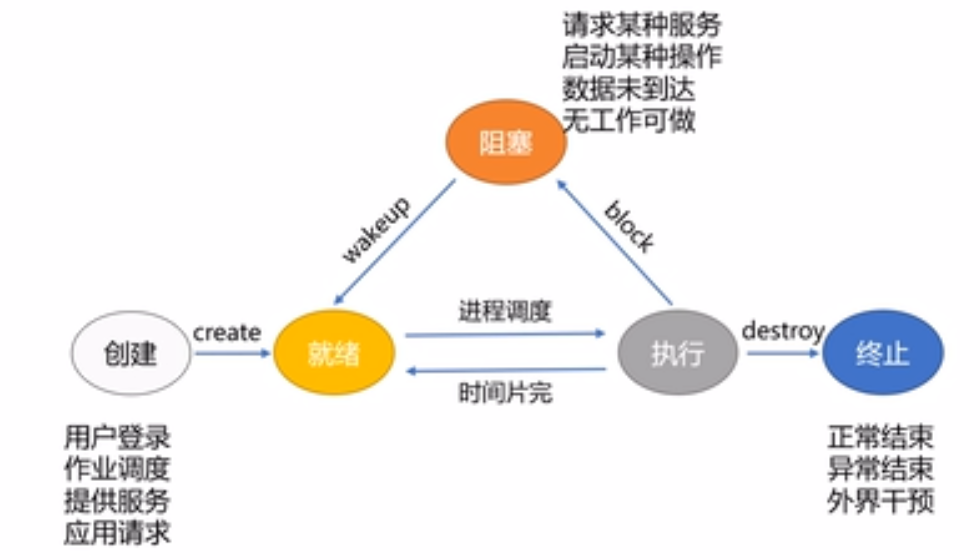
由若干条指令组成，完成指定的功能，是一种原子操作。

#### 原子的特点

* 原子操作，要么全做，要么全不做，执行过程不会被中断
* 在管态/系统态/内核态下执行，常驻内存
* 是内核三大支撑功能（中断处理/时钟管理/原语操作）之一

#### 相关原语

* 创建原语：create
* 阻塞原语：block
* 唤醒原语：wakeup
* 撤销原语：destory



#### 挂起与激活（是一种操作）

为了系统和用户观察和分析进程

* 挂起原语：suspend
  + 静止就绪：放外存，不调度
  + 静止阻塞：等待事件
* 激活原语：active
  + 活动就绪：等待调度
  + 活动阻塞：等待唤醒



#### 处理机调度

根据一定的算法和原则将处理机资源进行重新分配的过程

* 前提：作业/进程数远远大于处理机数
* 目的：提高资源利用率，减少处理机空闲时间
* 调度程序：一方面要满足特定系统用户的需求（快速响应），另一方面要考虑系统整体效率（系统平均周转时间）和调度算法本身的开销

#### 处理机调度层次

* 高级调度/作业调度
  + 把后备作业调入内存
  + 只调入一次，调出一次
* 中级调度/内存调度
  + 把进程调至外存，条件合适再调入内存
  + 在内、外存对换区进行进程兑换
* 低级调度/进程调度
  + 从就绪队列选取进程分配给处理机
  + 最基本的调度，频率非常高（相当于一个时间片完成）



#### 处理机调度方式

* 剥夺式/抢占式调度
  + 立即暂停当前进程
  + 分配处理机给另一个进程
  + 原则：优先权/短进程优先/时间片原则
* 非剥夺/非抢夺式调度
  + 若有进程请求执行
  + 等待直到当前进程完成或阻塞
  + 缺点：适用于批处理系统，不适用分时/实时系统

#### 处理机调度时机

* 进程运行完毕
* 进程时间片用完
* 进程要求I/O操作
* 执行某种原语操作
* 高优先级进程申请运行（剥夺式调度）

#### 处理机调度过程

* 保存镜像：记录进程现场信息
* 调度算法：确定分配处理机的原则
* 进程切换：分配处理机给其它进程
* 处理机回收：从进程收回处理机

#### 进程调度算法指标

* **CPU利用率** 忙碌时间/总时间（越多越好）
* **系统吞吐量** 完成作业數/总时间（越多越好）
* **周转时间** 作业完成时间-提交时间（越少越好）
  + 带权周转时间：周转时间/实际运行时间
* **等待时间** 作业等待处理机调度时间（越少越好）
  + 关注平均值
* **响应时间** 提交请求到首次响应间隔（越少越好）

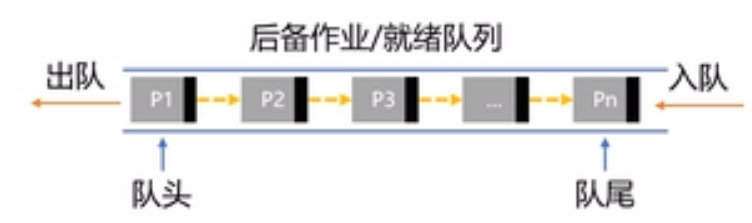
#### 处理机调度算法

* 先来先服务（FCFS）
* 短作业优先（SJF）
* 高响应比优先调度（HRRN）
* 优先级调度（PSA）
* 时间片轮转调度（RR）
* 多级反馈队列调度（MFQ）

#### 调度算法：先来先服务（FCFS）

* 算法内容：调度作业/就绪队列中最先入队者，等待操作完成或阻塞
* 算法原则：按作业/进程到达顺序服务（执行）
* 调度方式：非抢占式调度
* 适用场景：作业/进程调度
* 优缺点：有利于CPU繁忙型作业，充分利用CPU资源

不利于I/O繁忙型作业，操作费时，其他饥饿



#### 调度算法：短作业优先（SJF）

* 算法内容：所需服务时间最短的作业/进程优先服务（执行）
* 算法原则：追求最少得平均（带权）周转时间
* 调度方式：SJF/SPF非抢占式
* 适用场景：作业/进程调度
* 优缺点：平均等待/周转时间最少

长时间周转作业会增加或饥饿

估计时间不准确，不能保证紧迫任务及时处理



#### 调度算法：高响应比优先调度（HRRN）

* 算法内容：结合FCFS和SJF，综合考虑等待时间和服务时间计算响应比，高的优先调度
* 算法原则：综合考虑作业/进程的等待时间和服务时间
* 调度方式：非抢占式
* 适用场景：作业/进程调度
* 优缺点：响应比=（等待时间+服务时间）/服务时间，≥1

只有当前进程放弃执行权（完成/阻塞）时，重新计算所有进程响应比

长时间等待越久响应比越高，更容易获得处理机

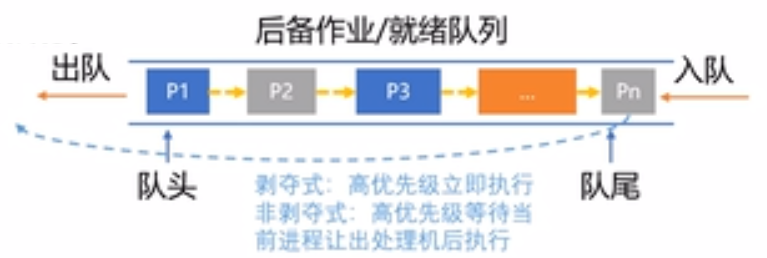


#### 调度算法：优先级调度（PSA）

* 算法内容：又叫优先权调度，按作业/进程的优先级（紧迫程度）进行调度
* 算法原则：优先级最高（最紧迫）的作业/进程先调度
* 调度方式：抢占/非抢占式（并不能获得及时执行）
* 适用场景：作业/进程调度
* 优缺点：静态/动态优先级

系统>用户；交互性>非交互性；I/O型>计算型

低优先级进程可能会产生“饥饿”

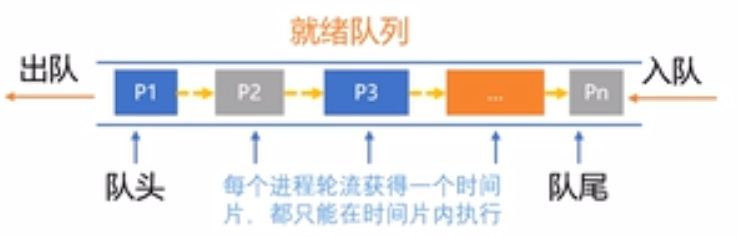


#### 调度算法：时间片轮转调度（RR）

* 算法内容：按进程到达就绪队列的顺序，轮流分配一个时间片去执行，时间用完则剥夺
* 算法原则：公平、轮流为每个进程服务，进程在一定时间内都能得到响应
* 调度方式：抢占式，由时钟中断确定时间到
* 适用场景：进程调度
* 优缺点：公平，响应快，适用于分时系统

时间片决定因素：系统响应时间、就绪队列进程数量、系统处理能力

时间片太大，相当于FCFS；太小，处理机切换频繁，开销增大



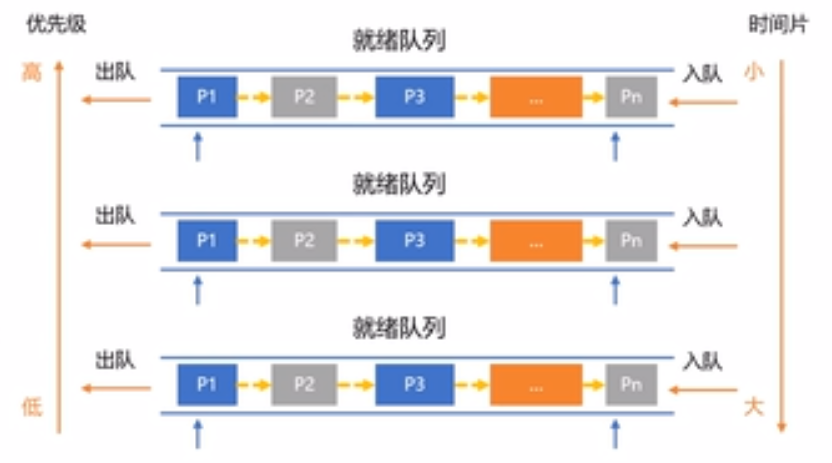
#### 调度算法：多级反馈队列调度（MFQ）

* 算法内容：
  + 设置多个按优先级排序的就绪队列
  + 优先级从高到低，时间片从小到大
  + 新进程采用队列降级法
    - 进入第一级队列，按FCFS分时间片
    - 没有执行完，移到第二级，第三级
  + 前面队列不为空，不执行后续队列进程
* 算法原则：集前几种算法优点，相当于PSA+RR
* 调度方式：抢占式
* 适用场景：进程调度
* 优缺点：对各类型相对公平；快速响应

终端型作业用户：短作业优先

批处理作业用户：周转时间短

长批处理作业用户：在前几个队列部分执行



## 进程之间是怎么协作的

### 进程通信

#### 概念：进程通信即进程间的信息交换

* 进程是资源分配的基本单位，各进程内存空间彼此独立
* 一个进程不能随意访问其他进程的地址空间

#### 特点

* 共享存储
* 消息传递
* 管道通信

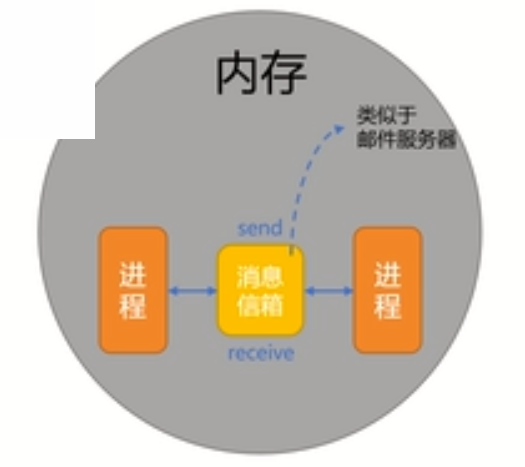
#### 进程通信：共享存储

* 基于共享数据结构的通信方式
  + 多个进程共用某个数据结构（OS提供并控制）
  + 由用户（程序员）负责同步处理
  + 低级通信：可以传递少量数据，效率低
* 基于共享存储区的通信方式
  + 多个进程共用内存中的一块存储区域
  + 由进程控制数据的形式和方式方式
  + 高级通信：可以传递大量数据，效率高



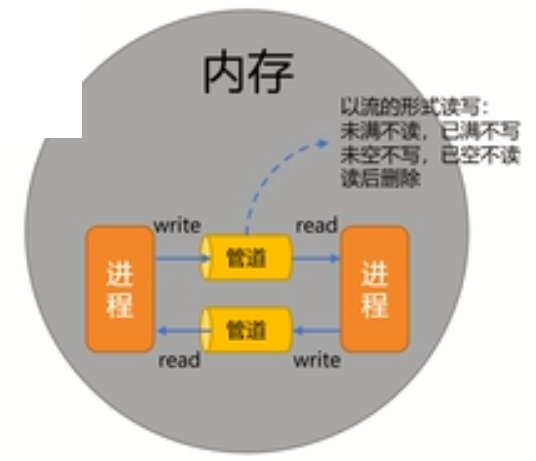
#### 进程通信：消息传递

* 直接通信：点到点发送
  + 发送和接收时指明双方进程的ID
  + 每个进程维护一个消息缓冲队列
* 间接通信：广播信箱
  + 以信箱为媒介，作为中间实体
  + 发进程将消息发送至信箱，收进程从信箱读取
  + 可以广播，容易建立双向通信链

#### 进程通信：管道通信

* 管道
  + 用于连接读/写进程的共享文件，pipe文件
  + 本质是内存中固定大小的缓冲区
* 半双工通信
  + 同一时段只能单向通信，双工通信需要两个管道
  + 以先进先出（FIFO）方式组织数据传输
  + 通过系统调用read()/write()函数进行读写操作



### 进程同步

协调进程间的相互制约关系，使它们按照预期的方式执行的过程

#### 前提

* 进程是并发执行的，进程间存在着相互制约关系
* 并发的进程对系统共享资源进行竞争
* 进程通信，过程中相互发送的信号称为消息或事件

#### 两种相互制约形式

* 间接相互制约关系（互斥）：进程排他性地访问共享资源
* 直接相互制约关系（同步）：进程间的合作，比如管道通信

#### 互斥的访问临界资源-访问过程

* 进入区：尝试进入临界区，成功则加锁（lock）
* 临界区：访问共享资源
* 退出区：解锁（unlock），唤醒其他阻塞进程
* 剩余区：其他代码

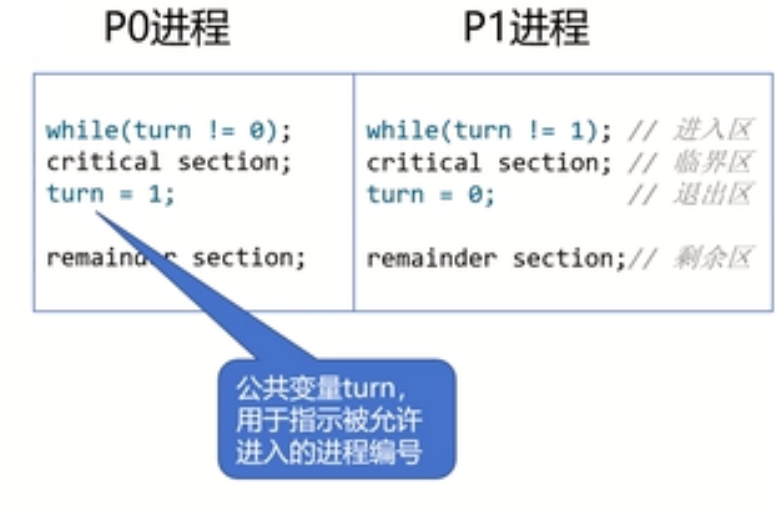


#### 互斥的访问临界资源-访问原则

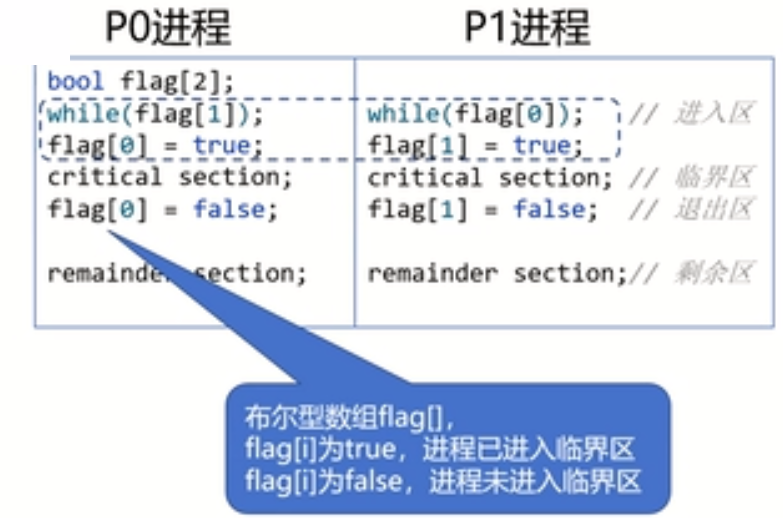
* 空闲让进：临界区空闲，允许一个进程进入
* 忙则等待：临界区已有进程，其它进程等待（阻塞状态）
* 有限等待：处于等待的进程，等待时间有限
* 让权等待：等待时应让出CPU执行权，防止“忙等待”

#### 互斥的访问临界资源-软件实现方法

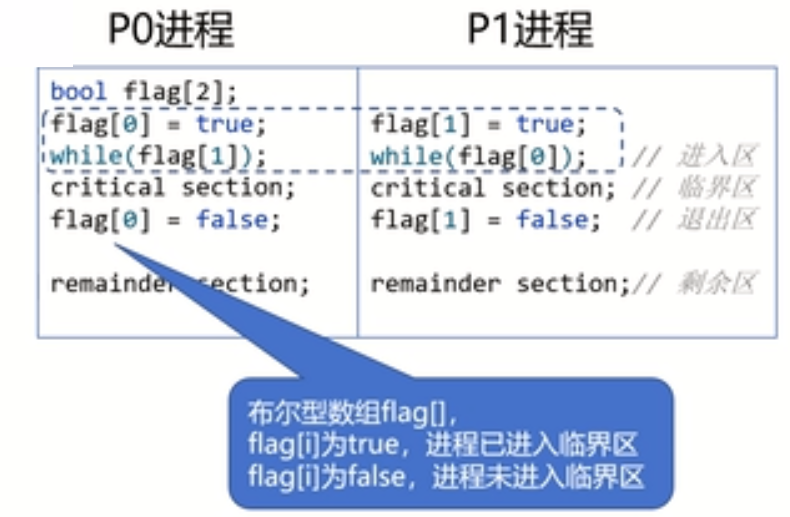
* 单标志法：违背“空闲让进”



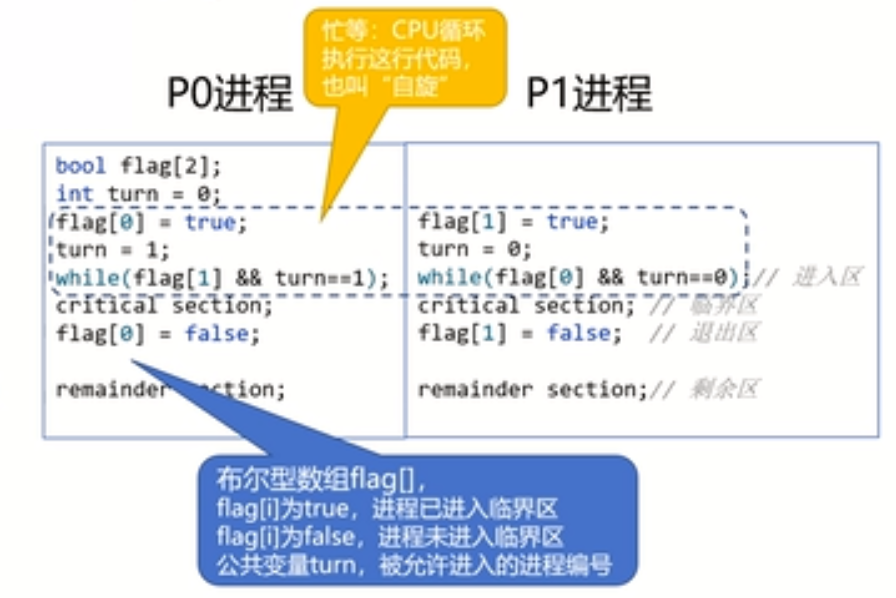
* 双标志法先检查：违背“忙则等待”



* 双标志法后检查：违背“空闲让进”、“有限等待”

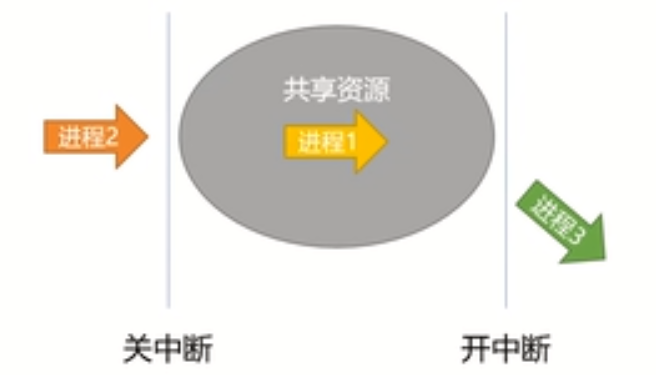


* 皮特森算法（Peterson's Algorithm）：违背“让权等待”，会发生“忙等”

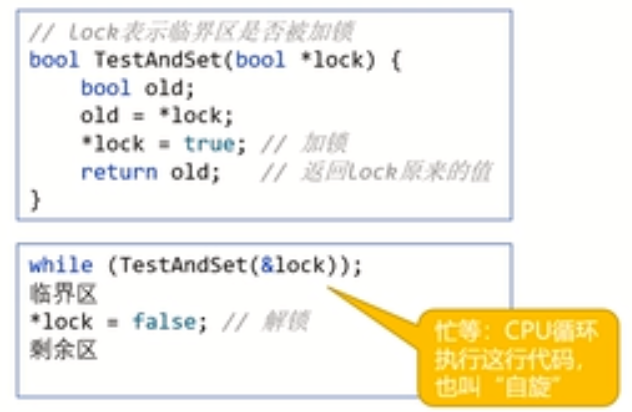


#### 互斥的访问临界资源-硬件实现方法

* 中断屏蔽方法：关中断/开中断
  + 禁止一切中断，CPU执行完临界区之前不会切换
  + 关中断可能会被滥用
  + 关中断时间长影响效率
  + 不适用于多处理机，无法防止其他处理机调度其它进程访问临界区
  + 只适用于内核进程 （该指令运行在内核态）



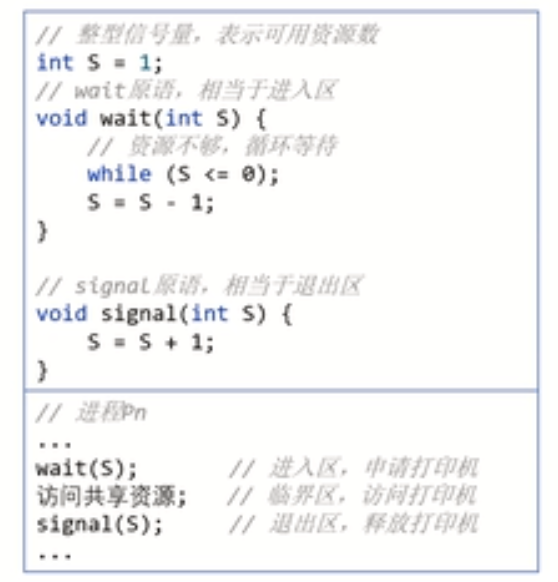
* Test-And-Set（TS指令/TSL指令）
  + 读出标志并设置为true，返回旧值，原子操作
  + 也被称作TSL指令（Test-And-Set-Lock）
  + 违背“让权等待”，会发生忙等

* Swap指令（EXCHANGE，XCHG指令）
  + 交换两个变量的值，原子操作
  + 违背“让权等待”

#### 信号量机制

* PV操作
  + P操作：wait原语，进程等待
  + V操作：signal原语，唤醒等待进程
* 整型信号量：违背“让权等待”，会发生忙等

* 记录型信号量：进程进入阻塞状态，不会忙等

#### 管程（监视器）

即“管理进程”，用于实现进程同步的工具。是由代表共享资源的数据结构和一组过程（进行PV操作的函数）组成的管理程序（封装）。

#### 管程的组成

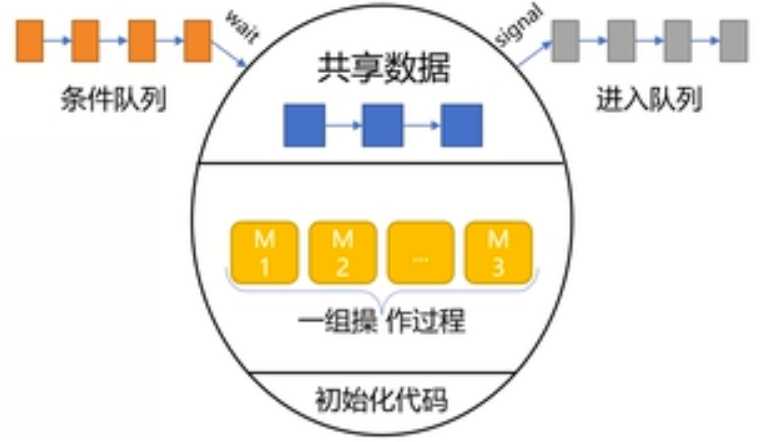
* 管程名称
* 局部于管程内部的共享数据结构
* 对该数据结构操作的一组过程（函数）
* 管程内共享数据的初始化语句

#### 管程的基本特性

* 是一个模块化的基本程序单位，可以单独编译
* 是一种抽象数据类型，包括数据和操作
* 信息掩蔽，共享数据只能被管程内的过程访问
* 管程内共享数据的初始化语句

#### 条件变量/条件对象

* 进入管程的进程可能由于天剑不满足而阻塞
* 此时进程应释放管程以便其他进程调用管程
* 进程被阻塞的条件（原因）有多个，移入不同的条件队列
* 进程被移入条件队列后，应释放管程



## 如何处理死锁问题？

### 死锁的概念

* 死锁定义

多个进程由于竞争资源而造成阻塞现象，若无外力作用，这些进程将无法继续推进。

* 相似概念：饥饿

等待时间过长以至于给进程推进和响应带来明显影响，"饿而不死"

* 死锁产生的原因
  + 系统资源的竞争
  + 进程推进顺序非法
* 死锁产生的必要条件
  + 互斥条件：共享资源的排他性访问
  + 不剥夺条件：访问时该共享资源不会被剥夺
  + 请求并保持条件：保持当前资源时请求另一个资源
  + 循环等待条件:：存在共享资源的循环等待链

### 死锁的处理策略

#### 死锁预防

* 死锁预防：破坏互斥条件
  + 将只能互斥访问的资源改为同时共享访问
  + 将独占锁改为共享锁
  + 不是所有资源都能改成可共享的
* 死锁预防：破坏不剥夺/不可抢占条件
  + 请求新资源无法满足时必须释放已有资源
  + 由OS协助强制剥夺某进程持有的资源
  + 实现复杂，代价高
  + 此操作过多导致原进程任务无法推进
* 死锁预防：破坏请求并保持条件
  + 进程开始运行时一次性申请所需资源
    - 资源浪费、进程饥饿
  + 阶段性请求和释放资源
* 死锁预防：破坏循环等待条件
  + 对所有资源现行排序，按序号请求资源
    - 请求时先低再高
    - 释放时先高再低
  + 对资源的编号应相对稳定，限制了新设备增加
  + 进程使用资源的顺序可能与系统编号顺序不同
  + 限制了用户编程

#### 死锁避免：安全性算法

* 系统安全状态
  + 安全状态一定不会出现死锁
  + 不安全状态可能出现死锁
* 银行家算法
  + 系统预判进程请求是否导致不安全状态
  + 是则拒绝请求，否则答应请求

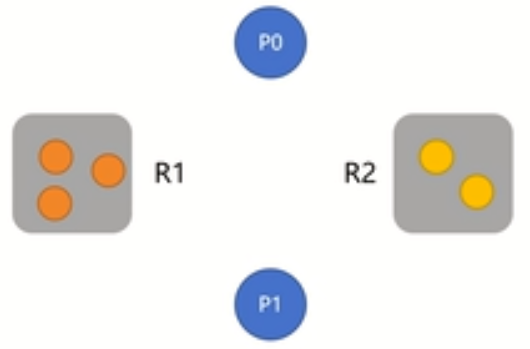
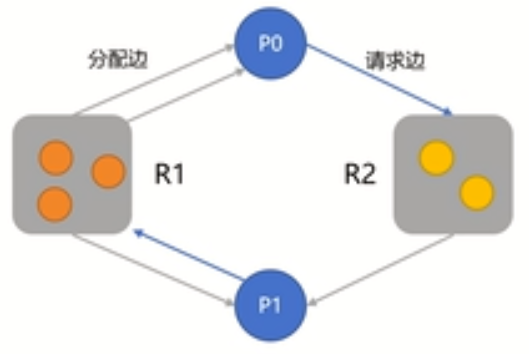


#### 死锁的检测

需要一种数据结构，保存有关资源的请求和分配信息

提供一种算法，利用这些信息检测是否形成了死锁

* 资源分配图（G=（N,E））
  + 两种资源
  + 两种节点
* 死锁定理（死锁状态的充分条件）：
  + 当且仅当此状态下资源分配图是不可完全简化的
  + 简化过程类似于“拓扑排序”算法（注意数据结构考察）



#### 死锁的解除

* 资源剥夺
  + 挂起死锁进程
  + 剥夺其他资源
  + 将资源分配给其他（死锁）进程
* 撤销进程
* 进程回退
  + 回退到足以避免死锁的地步
  + 需要记录进程历史信息，设置还原点