

操作系统原理

PRINCIPLES OF OPERATING SYSTEM

北京大学计算机科学技术系 陈向群

Department of computer science and Technology

Peking University

2015 春季

第4讲

处理器调度（CPU调度）

CPU调度

- ◎ CPU调度的相关概念
- ◎ 设计调度算法时要考虑的几个要点
- ◎ 批处理系统的调度算法
- ◎ 交互式系统的调度算法
- ◎ Windows操作系统的线程调度算法
- ◎ 其他

CPU 调度的相关概念

CPU调度

◎ CPU调度

—— 其任务是控制、协调进程对CPU的竞争

即按一定的调度算法从就绪队列中选择一个进程，
把CPU的使用权交给被选中的进程

如果没有就绪进程，系统会安排一个系统空闲进程或idle进程

◎ 系统场景

- N个进程就绪、等待上CPU运行
- M个CPU， $M \geq 1$
- 需要决策：给哪一个进程分配哪一个CPU？

CPU调度要解决的三个问题

WHAT: 按什么原则选择下一个要执行的进程

— 调度算法

WHEN: 何时选择

— 调度时机

HOW: 如何让被选中的进程上CPU运行

— 调度过程（进程的上下文切换）

CPU调度的时机(1/2)

事件发生 → 当前运行的进程暂停运行 → 硬件机制响应后 → 进入操作系统，处理相应的事件 → 结束处理后：

某些进程的状态会发生变化，也可能又创建了一些新的进程

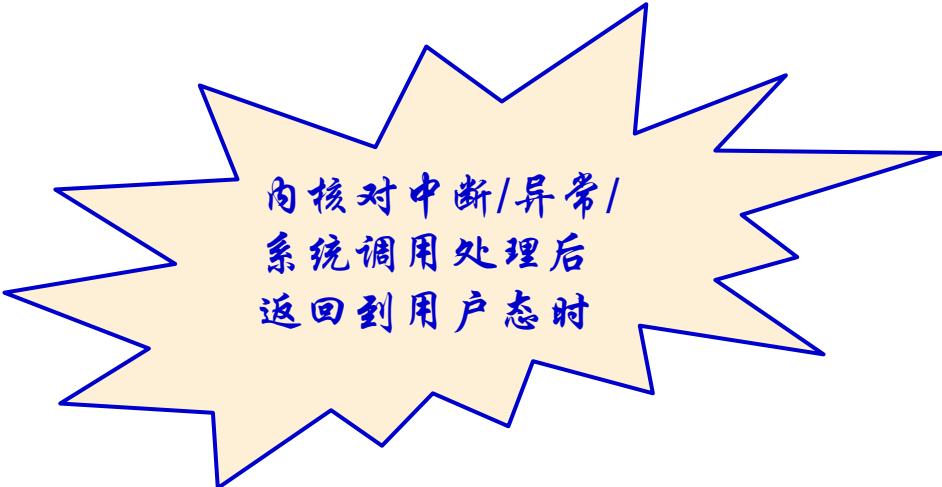
→ 就绪队列有调整 → 需要进程调度根据预设的调度算法从就绪队列选一个进程

典型的事件举例：

- 创建、唤醒、退出等进程控制操作
- 进程等待I/O、I/O中断
- 时钟中断，如：时间片用完、计时器到时
- 进程执行过程中出现abort异常

CPU调度的时机(2/2)

- ◉ 进程正常终止 或 由于某种错误而终止
- ◉ 新进程创建 或 一个等待进程变成就绪
- ◉ 当一个进程从运行态进入阻塞态
- ◉ 当一个进程从运行态变为就绪态



内核对中断/异常/
系统调用处理后
返回到用户态时

调度过程——进程切换(1/2)

- ◎ 进程调度程序从就绪队列选择了要运行的进程：
这个进程可以是刚刚被暂停执行的进程，也可能是另一个新的进程

→ 进程切换

- ◎ 进程切换：是指一个进程让出处理器，由另一个进程占用处理器的过程

调度过程——进程切换(2/2)

- ◎ 进程切换主要包括两部分工作：
 - 切换全局页目录以加载一个新的地址空间
 - 切换内核栈和硬件上下文，其中硬件上下文包括了内核执行新进程需要的全部信息，如CPU相关寄存器

切换过程包括了对原来运行进程各种状态的保存和对新的进程各种状态的恢复

上下文切换具体步骤

场景：进程A下CPU，进程B上CPU

- ④ 保存进程A的上下文环境（程序计数器、程序状态字、其他寄存器.....）
- ④ 用新状态和其他相关信息更新进程A的PCB
- ④ 把进程A移至合适的队列（就绪、阻塞.....）
- ④ 将进程B的状态设置为运行态
- ④ 从进程B的PCB中恢复上下文（程序计数器、程序状态字、其他寄存器.....）

上下文切换开销(COST)

什么是上下文切
换的开销?

- ◎ 直接开销：内核完成切换所用的**CPU**时间
 - 保存和恢复寄存器.....
 - 切换地址空间（相关指令比较昂贵）
- ◎ 间接开销
 - 高速缓存(Cache)、缓冲区缓存(Buffer Cache)和TLB(Translation Look-aside Buffer)失效

CPU调度算法的设计

- ◎ 什么情况下需要仔细斟酌调度算法？
 - 批处理系统 → 多道程序设计系统 → 批处理与分时的混合系统 → 个人计算机 → 网络服务器

	用户角度	系统角度
性能	周转时间 响应时间 最后期限	吞吐量 CPU利用率
其他	可预测性	公平性 强制优先级 平衡资源

调度算法衡量指标

- ◎ **吞吐量 Throughput** — 每单位时间完成的进程数目

- ◎ **周转时间 TT(Turnaround Time)**

每个进程从提出请求到运行完成的时间

- ◎ **响应时间 RT(Response Time)**

从提出请求到第一次回应的时间

- ◎ **其他**

- **CPU 利用率(CPU Utilization)**

CPU做有效工作的时间比例

- **等待时间(Waiting time)**

每个进程在就绪队列(ready queue)中等待的时间

设计调度算法前的 要点讨论

讨论几个要点

设计调度算法时要考虑以下几个问题：

- ◎ 进程控制块**PCB**中
需要记录哪些与**CPU**调度有关的信息
- ◎ 进程优先级及就绪队列的组织
- ◎ 抢占式调度与非抢占式调度
- ◎ **I/O**密集型与**CPU**密集型进程
- ◎ 时间片

1.进程优先级(数)

优先级 与 优先数 ? 静态 与 动态

静态优先级:

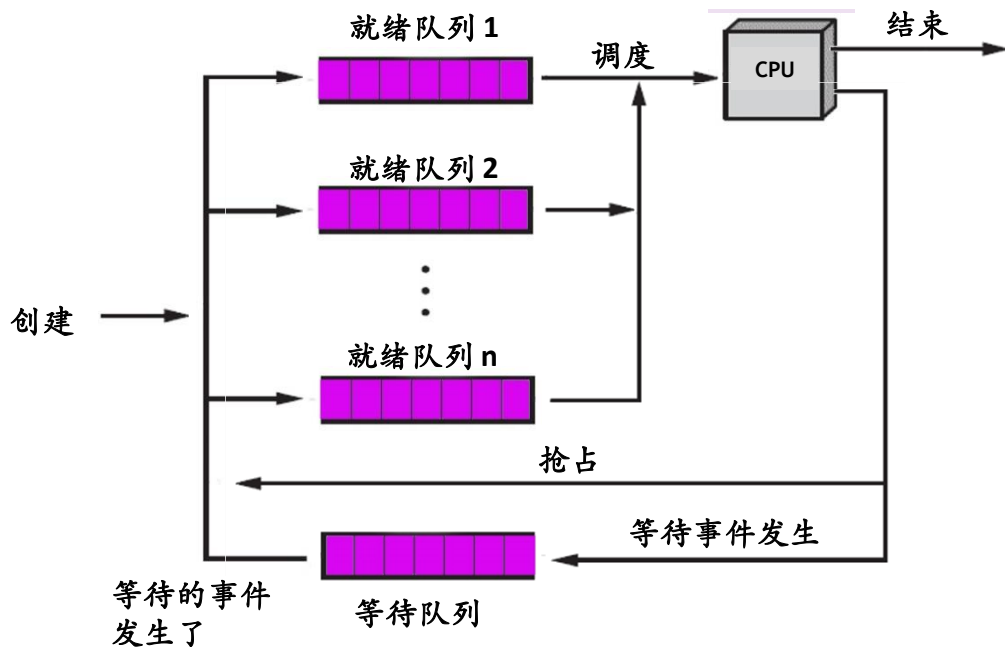
进程创建时指定，运行过程中不再改变

动态优先级:

进程创建时指定了一个优先级，运行过程中可以动态变化

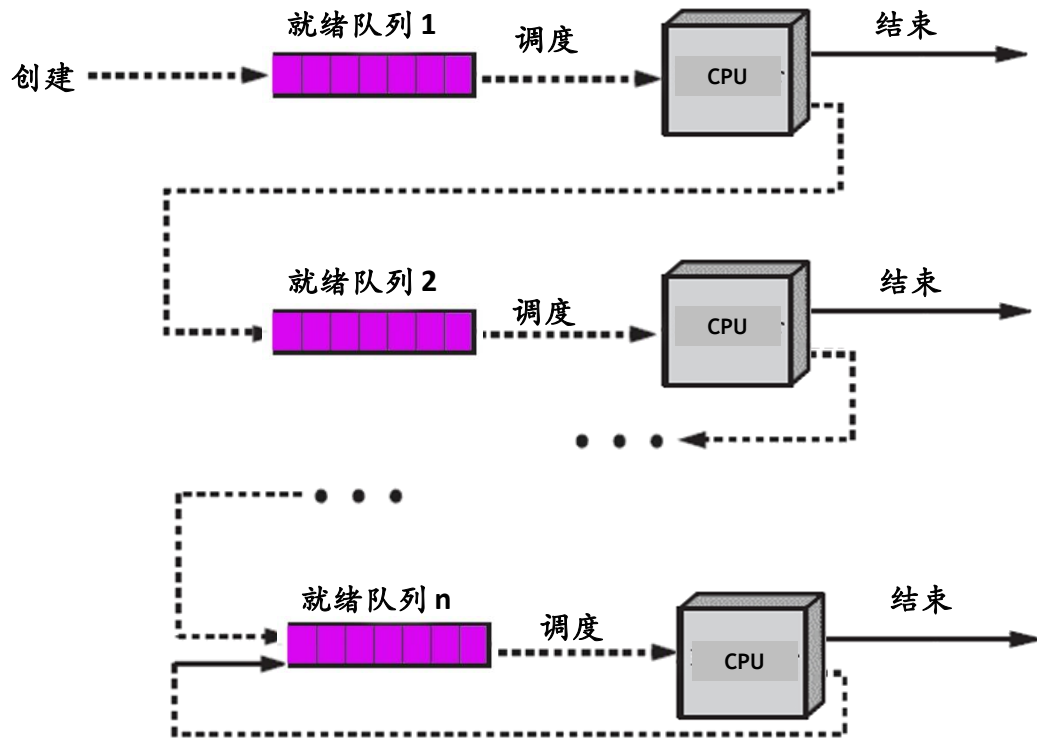
如：等待时间较长的进程可提升其优先级

2.进程就绪队列组织1



按优先级排队

2.进程就绪队列组织2



另一种排队方式

3. 抢占与非抢占

指占用CPU的方式:

- 可抢占式Preemptive（可剥夺式）

当有比正在运行的进程优先级更高的进程就绪时，系统可强行剥夺正在运行进程的CPU，提供给具有更高优先级的进程使用

- 不可抢占式Non-preemptive（不可剥夺式）

某一进程被调度运行后，除非由于它自身的原因不能运行，否则一直运行下去

4.I/O密集型与CPU密集型进程

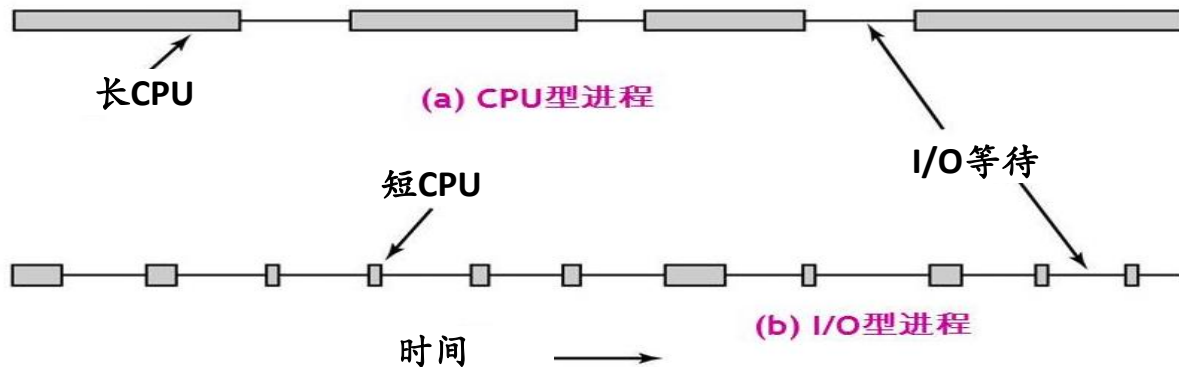
按进程执行过程中的行为划分：

- ◎ **I/O密集型或I/O型(I/O-bound)**

- 频繁的进行I/O，通常会花费很多时间等待I/O操作的完成

- ◎ **CPU密集型或CPU型或计算密集型(CPU-bound)**

- 需要大量的CPU时间进行计算



5.时间片

时间片长一点 or
短一点



固定与可变

- Time slice 或 quantum
- 一个时间段，分配给调度上CPU的进程，确定了允许该进程运行的时间长度
- 如何选择时间片呢？

考虑因素：

- ✓ 进程切换的开销
- ✓ 对响应时间的要求
- ✓ 就绪进程个数
- ✓ CPU能力
- ✓ 进程的行为

先来先服务、短作业优先、……

批处理系统的调度算法

批处理系统中采用的调度算法

- ◎ 先来先服务（FCFS-First Come First Serve）
- ◎ 最短作业优先（SJF-Shortest Job First）
- ◎ 最短剩余时间优先
（SRTN-Shortest Remaining Time Next）
- ◎ 最高相应比优先
（HRRN-Highest Response Ratio Next）



先来先服务(FCFS)

- ◎ **First Come First Serve**
- ◎ 先进先出 **First In First Out (FIFO)**
- ◎ 按照进程就绪的先后顺序使用CPU
- ◎ 非抢占

- ◎ 优缺点
 - 公平
 - 实现简单
 - 长进程后面的短进程需要等很长时间，不利于用户体验

FCFS调度算法举例

例子:

- 三个进程按顺序就绪: P1、P2、P3
进程P1 执行需要24s, P2和P3各需要3s
- 采用FCFS调度算法:



吞吐量: $3 \text{ jobs} / 30\text{s} = 0.1 \text{ jobs/s}$

周转时间TT: P1:24; P2:27; P3:30

平均周转时间: 27s

讨论

例子:

- 三个进程按顺序就绪: P1、P2、P3
进程P1 执行需要24s, P2和P3各需要3s
- 改变调度顺序: P2、P3、P1



吞吐量: $3 \text{ jobs} / 30\text{s} = 0.1 \text{ jobs/s}$

周转时间TT: P1:30; P2:3; P3:6;

平均周转时间: 13s

短作业优先SJF (1/3)

- ◉ **Shortest Job First**
- ◉ 具有最短完成时间的进程优先执行
- ◉ 非抢占式

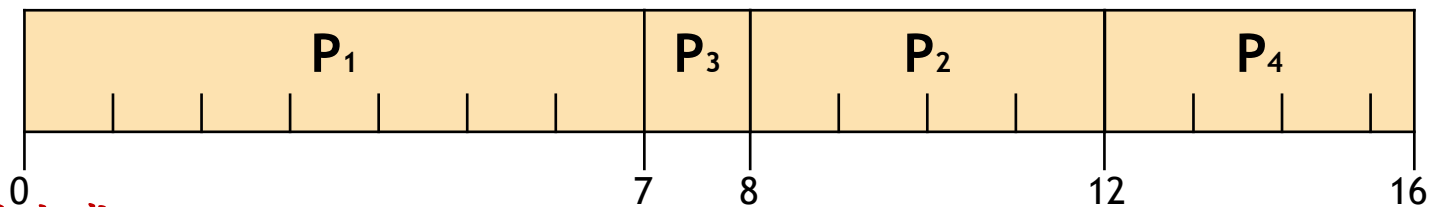
- ◉ 最短剩余时间优先
- ◉ **Shortest Remaining Time Next(SRTN)**
- ◉ SJF抢占式版本，即当一个新就绪的进程比当前运行进程具有更短的完成时间时，系统抢占当前进程，选择新就绪的进程执行

思路：先完成短的作业
改善短作业的周转时间

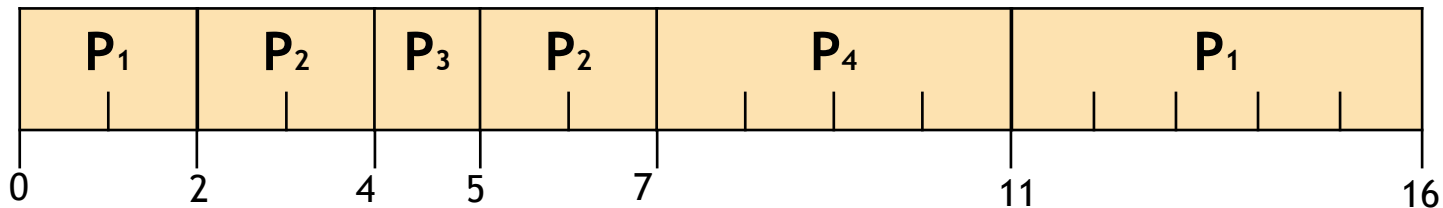
短作业优先SJF (2/3)

进程	到达时刻	运行时间
P1	0	7
P2	2	4
P3	4	1
P4	5	4

非抢占式



抢占式



短作业优先调度算法(3/3)

◎ 优缺点

■ 最短的平均周转时间

在所有进程同时可运行时，
采用SJF调度算法可以得到
最短的平均周转时间

■ 不公平

源源不断的短任务到来，可能使长的任务长时间得不到运行 → 产生“饥饿”现象 (starvation)

最高相应比优先HRRN

- ◎ Highest Response Ratio Next
- ◎ 是一个综合的算法
- ◎ 调度时，首先计算每个进程的响应比R；之后，总是选择 R 最高的进程执行



$$\begin{aligned}\text{响应比} R &= \text{周转时间} / \text{处理时间} \\ &= (\text{处理时间} + \text{等待时间}) / \text{处理时间} \\ &= 1 + (\text{等待时间} / \text{处理时间})\end{aligned}$$

时间片轮转、最高优先级、……

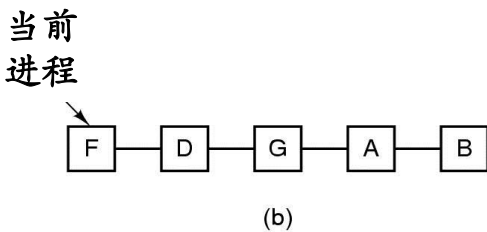
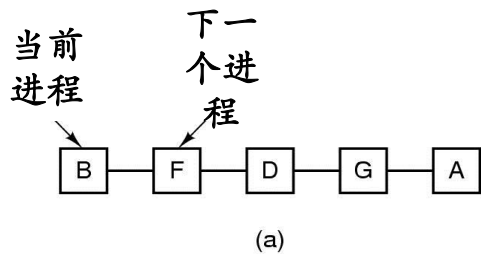
交互式系统的调度算法

交互式系统中采用的调度算法

- ◎ 轮转调度（RR-Round Robin）
- ◎ 最高优先级调度（HPF—Highest Priority First）
- ◎ 多级反馈队列（Multiple feedback queue）
- ◎ 最短进程优先（Shortest Process Next）



时间片轮转调度算法(1/4)



进程B用完自己的时间片后

- 目标
 - 为短任务改善平均响应时间
- 解决问题的思路
 - 周期性切换
 - 每个进程分配一个时间片
 - 时钟中断 → 轮换

时间片轮转调度算法(2/4)

◎ 如何合适的时间片?

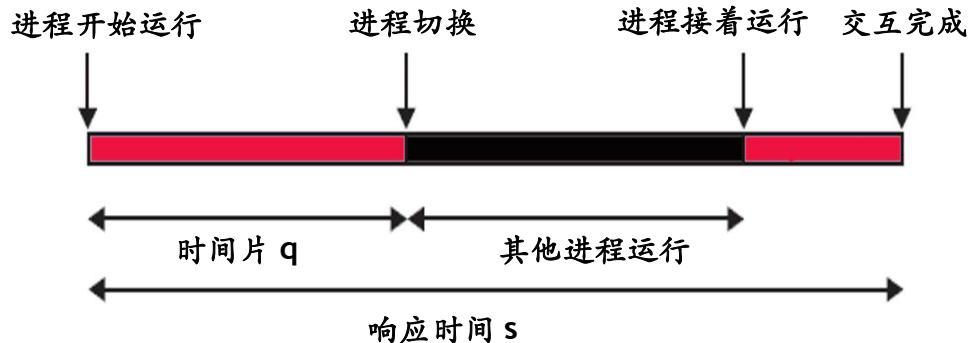
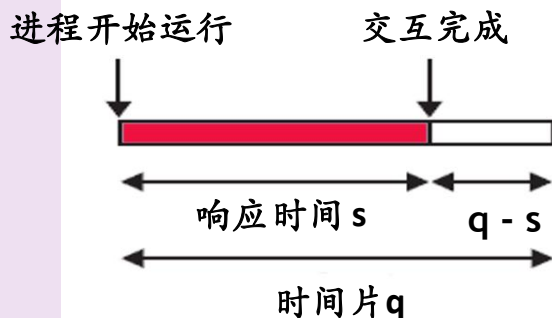
太长 -- 大于典型的交互时间

- 降级为先来先服务算法

- 延长短进程的响应时间

- 太短 -- 小于典型的交互时间

- 进程切换浪费CPU时间



时间片轮转调度算法(3/4)

◎ 优缺点

- 公平
- 有利于交互式计算，响应时间快
- 由于进程切换，时间片轮转算法要花费较高的开销

假设时间片 10ms，如果进程切换花费0.1ms，
CPU开销约占1%

进程运行时间	时间片	上下文切换次数
10	12	0
10	6	1
10	1	9

时间片轮转调度算法(4/4)

◎ 优缺点（续）

- RR对不同大小的进程是有利的
但是对于相同大小的进程呢？

- 两个进程A、B，运行时间均为100ms
- 时间片大小为1ms
- 上下文切换不耗时

假设

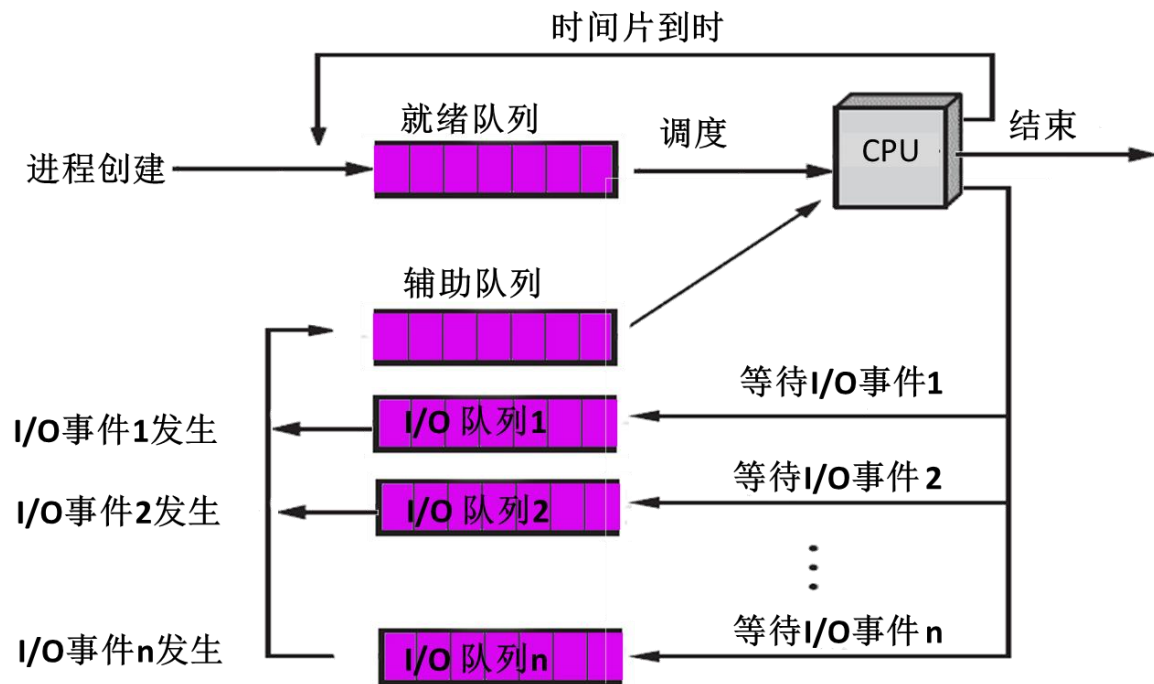
- 使用时间片轮转（RR）算法的平均完成时间？

199.5ms

ABABABAB..... A(199)B(200)

- 使用先来先服务（FCFS）算法呢？ **150ms**

虚拟轮转法(VIRTUAL RR)



虚拟轮转调度的队列图

最高优先级调度算法

- ◉ 选择优先级最高的进程投入运行
- ◉ 通常：系统进程优先级 高于 用户进程
前台进程优先级 高于 后台进程
操作系统更偏好 I/O型进程
- ◉ 优先级可以是静态不变的，也可以动态调整
 - 优先数可以决定优先级
- ◉ 就绪队列可以按照优先级组织
- ◉ 实现简单；不公平



饥饿

优先级反转问题(1/2)

基于优先
级的
抢占式

- Priority Inversion

- 又称：优先级反置、翻转、倒挂

- 现象

一个低优先级进程持有一个高优先级进程所需要的资源，使得高优先级进程等待低优先级进程运行

设H是高优先级进程，L是低优先级进程，M是中优先级进程（CPU型）

场景：L进入临界区执行，之后被抢占；

H也要进入临界区，失败，被阻塞；

M上CPU执行，L无法执行所以H也无法执行

优先级反转问题(2/2)

- ◎ 影响

- 系统错误
- 高优先级进程停滞不前，导致系统性能降低

- ◎ 解决方案

- 设置优先级上限
- 优先级继承
- 使用中断禁止

多级反馈队列调度算法
各种调度算法比较
多处理器调度算法

多级反馈队列调度算法(1/3)

- **Multilevel Feedback**
- 是UNIX的一个分支BSD（加州大学伯克利分校开发和发布的）5.3版所采用的调度算法
- 是一个综合调度算法



多级反馈队列调度算法(2/3)

- 设置**多个**就绪队列，第一级队列优先级最高
- 给不同就绪队列中的进程分配长度不同的时间片，第一级队列时间片最小；随着队列优先级别的降低，时间片增大
- 当第一级队列为空时，在第二级队列调度，以此类推
- 各级队列按照**时间片轮转方式**进行调度
- 当一个新创建进程就绪后，进入第一级队列
- 进程用完时间片而放弃CPU，进入下一级就绪队列
- 由于阻塞而放弃CPU的进程进入相应的等待队列，一旦等待的事件发生，该进程回到原来一级就绪队列 (?)

队首 or 队尾?

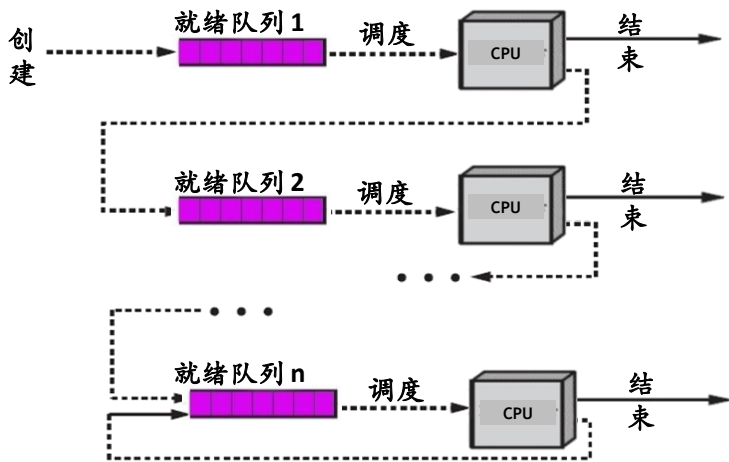
再次被调度上CPU时
对时间片的处理?

非抢
占式

多级反馈队列调度算法(3/3)

若允许抢占

- 当有一个优先级更高的进程就绪时，可以抢占CPU
- 被抢占的进程回到原来一级就绪队列末尾 (或者?)



队列模型示意

小结：各种调度算法的比较

调度算法	占用CPU方式	吞吐量	响应时间	开销	对进程的影响	饥饿问题
FCFS	非抢占	不强调	可能很慢，特别是当进程的 执行时间差别很大时	最小	对短进程不利；对I/O型的进程不利	无
Round Robin	抢占 (时间片用完时)	若时间片小， 吞吐量会很低	为短进程提供好的响应时间	最小	公平对待	无
SJF	非抢占	高	为短进程提供好的响应时间	可能较大	对长进程不利	可能
SRTN	抢占 (到达时)	高	提供好的响应时间	可能较大	对长进程不利	可能
HRRN	非抢占	高	提供好的响应时间	可能较大	很好的平衡	无
Feedback	抢占 (时间片用完时)	不强调	不强调	可能较大	对I/O型进程有利	可能

多处理器调度算法 需要考虑的问题

多处理器调度算法设计

- 不仅要决定选择哪一个进程执行
 - 还需要决定在哪一个CPU上执行
- 要考虑进程在多个CPU之间迁移时的开销
 - 高速缓存失效、TLB失效
 - 尽可能使进程总是在同一个CPU上执行
 - 如果每个进程可以调度到所有CPU上，假如进程上次在CPU1上执行，本次被调度到CPU2，则会增加高速缓存失效、TLB失效；如果每个进程尽量调度到指定的CPU上，各种失效就会减少
- 考虑负载均衡问题

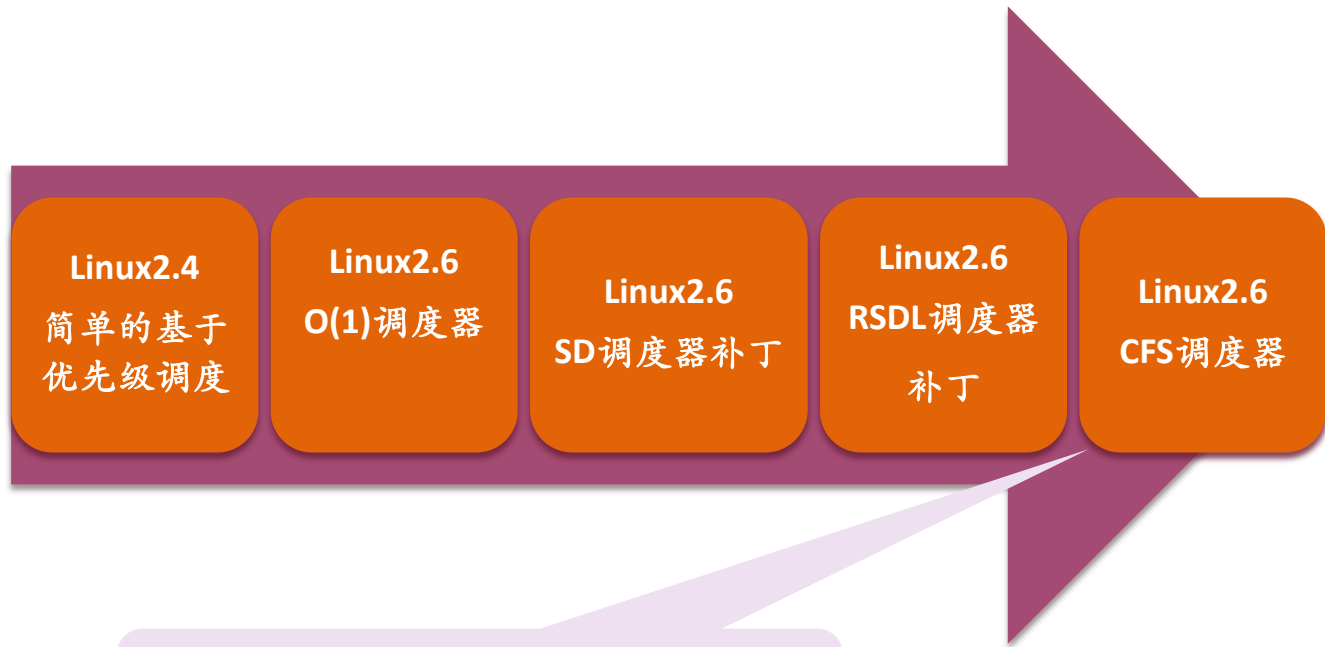
Windows 的线程调度算法

典型操作系统的
调度算法

典型系统所采用的调度算法

- ◎ **UNIX** 动态优先数法
- ◎ **5.3BSD** 多级反馈队列法
- ◎ **Linux** 抢占式调度
- ◎ **Windows** 基于优先级的抢占式多任务调度
- ◎ **Solaris** 综合调度算法

LINUX调度算法的发展历史



CFS：完全公平调度算法

调度算法实例介绍

WINDOWS线程调度

WINDOWS 线程调度

- 调度单位是线程
- 采用基于动态优先级的、抢占式调度，结合时间配额的调整

- ◎ 就绪线程按优先级进入相应队列
- ◎ 系统总是选择优先级最高的就绪线程运行
- ◎ 同一优先级的各线程按时间片轮转进行调度
- ◎ 多CPU系统中允许多个线程并行运行

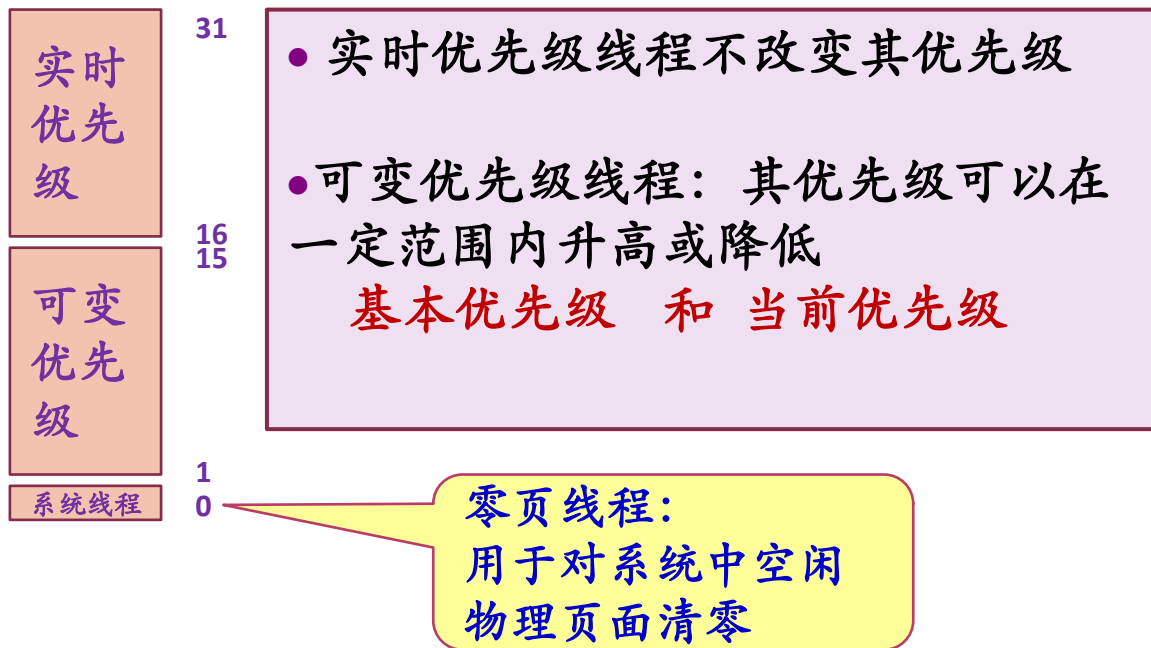
WINDOWS 线程调度

引发线程调度的条件:

- ⊙ 一个线程的优先级改变了
 - ⊙ 一个线程改变了它的亲和(Affinity)处理机集合
-
- ⊙ 线程正常终止 或 由于某种错误而终止
 - ⊙ 新线程创建 或 一个等待线程变成就绪
 - ⊙ 当一个线程从运行态进入阻塞态
 - ⊙ 当一个线程从运行态变为就绪态

线程优先级

- Windows使用32个线程优先级，分成三类



线程的时间配额

- 时间配额不是一个时间长度值，而是一个称为**配额单位 (quantum unit)**的整数
- 一个线程用完了自己的时间配额时，如果没有其他相同优先级的线程，**Windows**将重新给该线程**分配一个新的时间配额**，让它继续运行

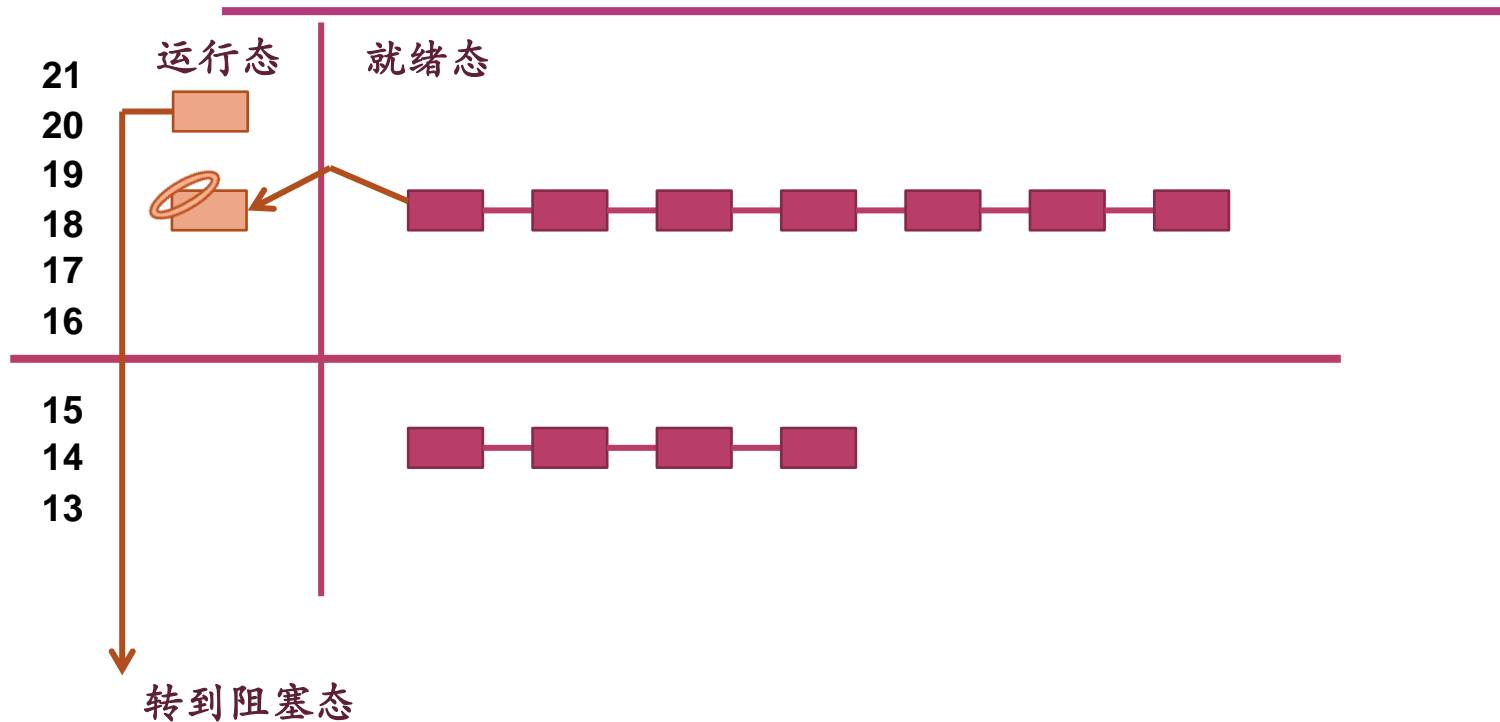
时间配额的一种特殊作用

- 假设用户首先启动了一个运行时间很长的电子表格计算程序，然后切换到一个游戏程序(需要复杂图形计算并显示，**CPU型**)
- 如果前台的游戏进程提高它的优先级，则后台的电子表格计算进程就几乎得不到**CPU**时间了
- 但增加游戏进程的时间配额，则不会停止执行电子表格计算，只是给游戏进程的**CPU**时间多一些而已

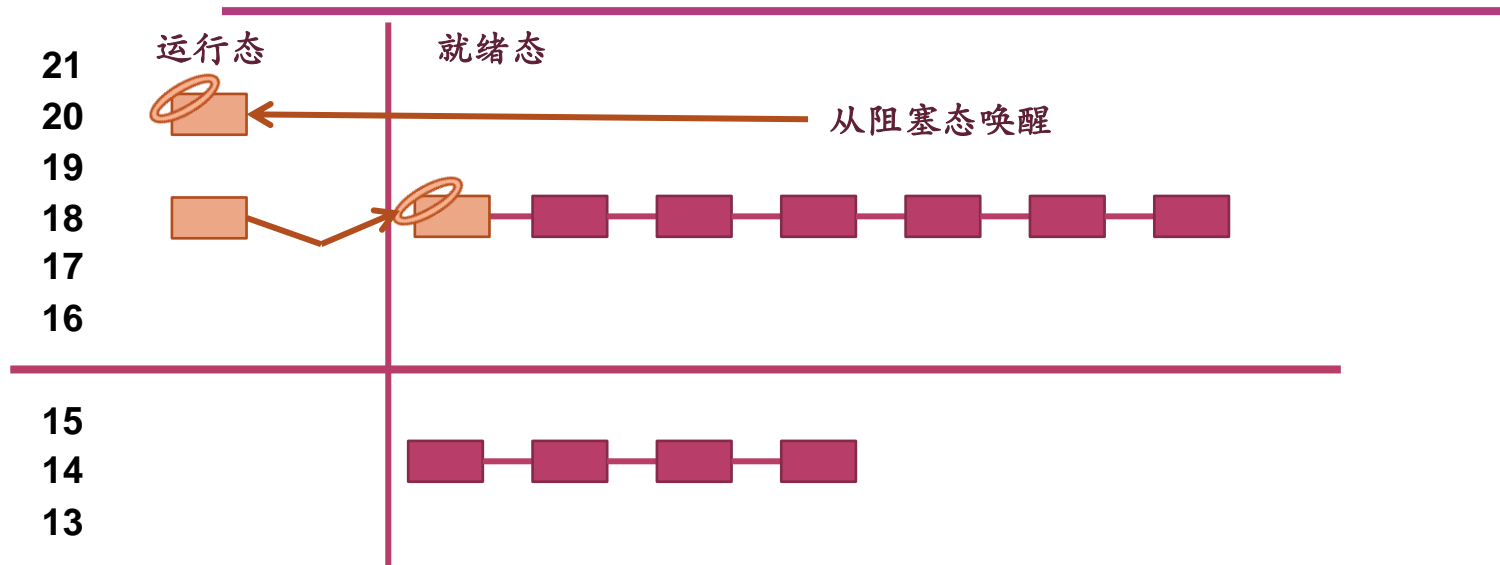
调度策略

- ◎ 主动切换
- ◎ 抢占
- ◎ 时间配额用完

(1)主动切换



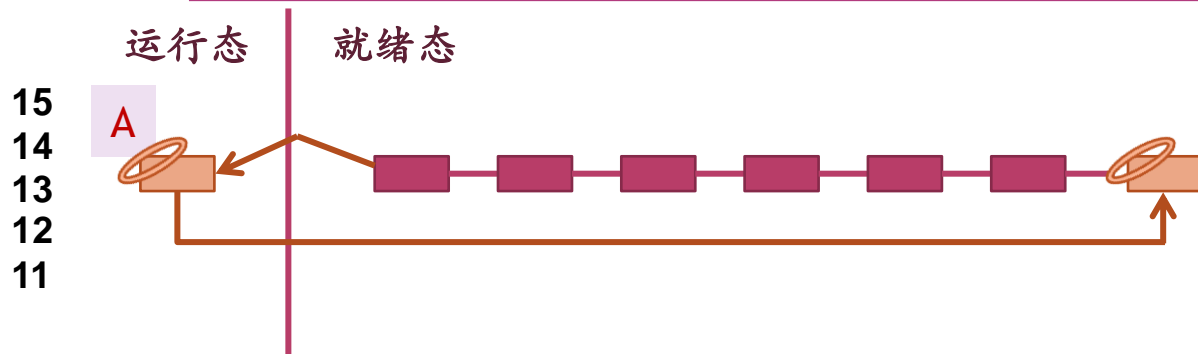
(2) 抢占



当线程被抢占时，它被放回相应优先级的就绪队列的队首

- 处于实时优先级的线程在被抢占时，时间配额被重置为一个完整的时间配额
- 处于可变优先级的线程在被抢占时，时间配额不变，重新得到CPU后将运行剩余的时间配额

(3)时间配额用完



假设线程A的时间配额用完

- ◎ **A的优先级没有降低**
 - ✓ 如果队列中有其他就绪线程，选择下一个线程执行，A回到原来就绪队列末尾
 - ✓ 如果队列中没有其他就绪线程，系统给线程A分配一个新的时间配额，让它继续运行
- ◎ **A的优先级降低了，Windows 将选择一个更高优先级的线程**

线程优先级提升与时间配额调整

◎ Windows的调度策略

- 如何体现对某类线程具有倾向性？
- 如何解决由于调度策略中潜在的不公平性而带来饥饿现象？
- 如何改善系统吞吐量、响应时间等整体特征？

◎ 解决方案

- 提升线程的优先级
- 给线程分配一个很大的时间配额

线程优先级提升

◎ 下列5种情况，Windows 会提升线程的当前优先级：

- I/O操作完成
- 信号量或事件等待结束
- 前台进程中的线程完成一个等待操作
- 由于窗口活动而唤醒窗口线程
- 线程处于就绪态超过了一定的时间还没有运行——“饥饿”现象

针对可变优先级范围内(1至15)的线程优先级

I/O操作完成后的线程优先级提升

- ◉ 在完成I/O操作后，**Windows** 将临时提升等待该操作线程的优先级，保证该线程能更快上**CPU**运行进行数据处理
- ◉ 优先级的提升值由设备驱动程序决定，提升建议值保存在系统文件 “**Wdm.h**”或 “**Ntddk.h**”中
- ◉ 优先级的提升幅度与对I/O请求的响应时间要求是一致的，响应时间要求越高，优先级提升幅度越大
- ◉ 设备驱动程序在完成I/O请求时通过内核函数 **IoCompleteRequest**来指定优先级提升的幅度
- ◉ 为避免不公平，在I/O操作完成唤醒等待线程时会将该线程的时间配额减1

“饥饿”线程的优先级提升

- ◎ 系统线程“平衡集管理器(balance set manager)”
每秒钟扫描一次就绪队列，发现是否存在等待时间超过300个时钟中断间隔的线程
- ◎ 平衡集管理器将这些线程的优先级提升到**15**，并分配给它一个长度为正常值**4倍**的时间配额
- ◎ 当被提升的线程用完它的时间配额后，立即衰减到它原来的基本优先级

本讲重点

- ◎ 掌握处理器调度的相关概念
 - 调度时机、进程切换
 - 调度标准：吞吐量、周转时间、响应时间
 - 优先级/优先数、抢占/非抢占、I/O型与CPU型
- ◎ 掌握主要的调度算法
 - 先来先服务、短作业优先、最高相应比优先
 - 时间片轮转、最高优先级
 - 多级反馈队列
- ◎ 了解Windows、多处理器调度的基本思想

本周要求

重点阅读教材

第2章相关内容：2.4

第11章相关内容：11.4.3中的调度部分

重点概念

调度时机 进程切换 抢占/非抢占 时间片
优先级反转 饥饿 优先级与优先数 优先级提升
先来先服务 短作业优先 最高相应比优先
时间片轮转 最高优先级 多级队列反馈
吞吐量 周转时间 响应时间

THANKS

The End