Lecture4 进程线程调度

基本问题

1. 问题 1：How & When

调度这件事儿什么时候做？做的理由有哪些？

如果没有可被调度的进程，系统做什么呢？

上下文切换的过程？有哪些开销？

关于调度算法，我们都关心什么？

不同类型的操作系统都适用同一种调度算法吗？

对于一个调度算法，应该追求什么样的目标？

选进程时都考虑了哪些点？单一因素还是多因素？

2. 问题 2：What

(1) 适用批处理系统的调度算法有哪些？

(2) 适用交互式系统的调度算法有哪些？

(3) 适用实时系统的调度算法有哪些？

(4) 怎样理解抢占式和非抢占式？

(5) 从哪几方面对调度算法进行比较？

(6) 机制和策略分离的原则在调度算法中的应用

(7) 实例操作系统的调度算法都是什么？

What	When	How
• 依据何原则挑选进程/线程以分配处理器 • 调度算法	• 何时分配处理器 • 调度时机	• 如何分配CPU，即进程的上下文切换 • 调度过程

单处理器调度

一、 调度简介

1. 调度的概念

(1) 处理器调度——控制、协调进程对 CPU 的竞争

(2) 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

(3) 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

(4) 调度程序：挑选就绪进程的内核函数

系统场景

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

2. 调度的时机

内核对中断/异常/系统调用处理后返回到用户态前的最后时刻

(1) 典型时机

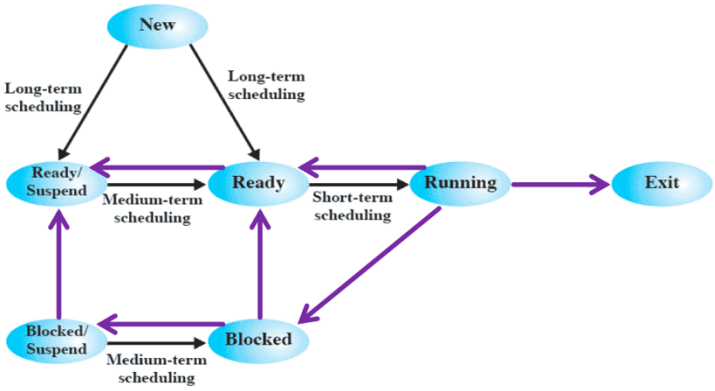
- ✓ 进程执行完毕并退出
- ✓ 进程由于某种错误或异常而终止
- ✓ 新进程的创建
- ✓ 运行进程要等待 I/O 操作或其他资源时进入阻塞态
- ✓ 被唤醒的阻塞进程重新回到就绪态
- ✓ 进程用完所分配到的时间片回到就绪队列

(2) 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

3. 调度的大致流程

(1) 事件发生 → 当前运行的进程暂停运行 → 硬件机制响应事件 → 进入内核处理相应的事件 → 结束处理后：某些进程的状态会发生变化，也可能又出现了一些新的进程 → 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



(2) 进程切换

a) 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

b) 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

4. 上下文切换

(1) 基本概念

a) 将 CPU 硬件状态从一个进程换到另一个进程的过程称为上下文切换

b) 进程运行时，其硬件状态保存在 CPU 上的寄存器（程序计数器、程序状态寄存器、栈指针、通用寄存器、其他控制寄存器...）中

c) 进程不运行时，这些寄存器的值保存在 PCB 中

(2) 主要过程

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

a) 切换全局页目录以加载一个新的地址空间

b) 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

(3) 具体步骤

场景：进程 A 下 CPU，进程 B 上 CPU

a) 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...)

b) 用新状态和其他相关信息更新进程 A 的 PCB

c) 把进程 A 移至合适的队列（就绪、阻塞...）

d) 将进程 B 的状态设置为运行态

e) 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...）

(4) 开销

a) 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）

b) 间接开销

✓ 高速缓存(Cache)、缓冲区缓存(Buffer Cache)和 TLB(Translation Lookaside Buffer)失效

二、 调度算法的设计

1. 不同操作系统追求的目标

(1) 交互式进程（interactive process）—— 响应时间、均衡性

a) 需要经常与用户交互，因此要花很多时间等待用户输入操作

b) 响应时间要快，平均延迟要低于 50~150ms

c) 典型的交互式程序： shell、文本编辑程序、图形应用程序等

(2) 批处理进程（batch process）—— 吞吐量、周转时间、CPU 利用率

a) 不必与用户交互，通常在后台运行

b) 不必很快响应

c) 典型的批处理程序：营业额计算、利息计算、索赔处理

(3) 实时进程（real-time process）—— 最后期限、可预测性

a) 有实时需求，不应被低优先级的进程阻塞

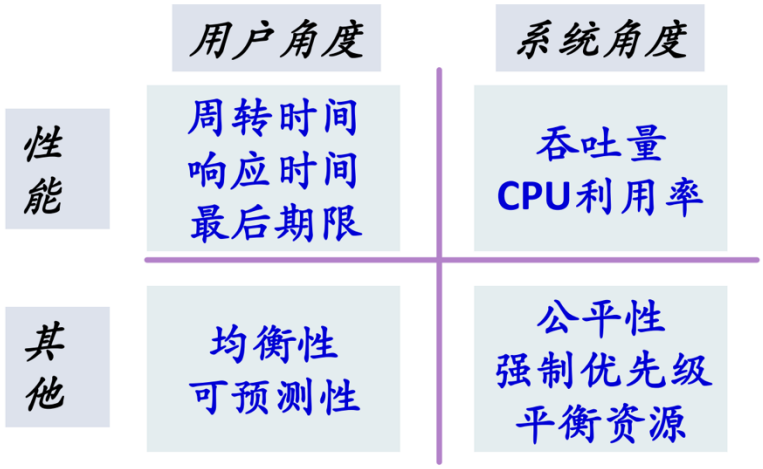
b) 响应时间要短

c) 典型的实时进程：视频/音频、机械控制等

2. 调度算法的设计考虑

(1) 系统的类型

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



(2) 算法衡量指标

a) 性能指标

- ✓ 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

b) 非性能指标

- ✓ 公平性（Fairness）
- ✓ 资源利用率

  CPU 利用率(CPU Utilization) ：CPU 做有效工作的时间比例

c) 特定场景考虑的指标

- ✓ 实时性
- ✓ 能耗

(3) 几个关键问题

a) 进程控制块 PCB 中需要记录哪些与 CPU 调度有关的信息？

b) 进程优先级及就绪队列的组织

c) 抢占式调度与非抢占式调度

d) I/O 密集型与 CPU 密集型进程

e) 时间片大小的选择

3. 基本概念

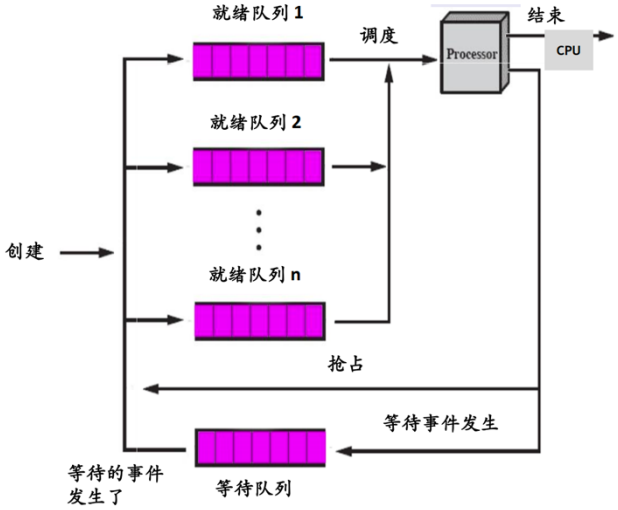
(1) 进程优先级(数)

a) 静态优先级：进程创建时指定，运行过程中不再改变

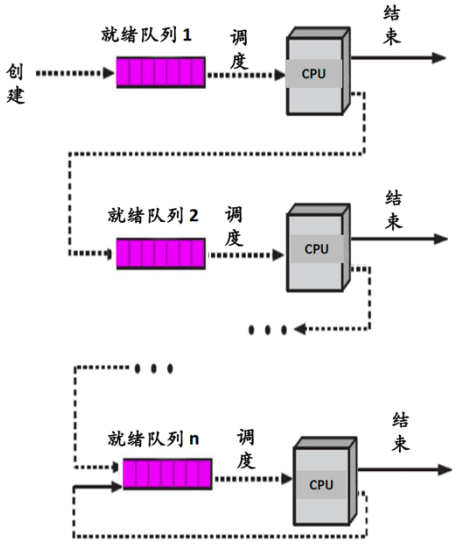
b) 动态优先级：进程创建时指定了一个优先级，运行过程中可以动态变化（e.g.等待时间较长的进程可提升其优先级）

(2) 进程就绪队列组织

a) 按优先级排队



b) 另一种排队方式



(3) 抢占与非抢占——指占用 CPU 的方式

a) 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

b) 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

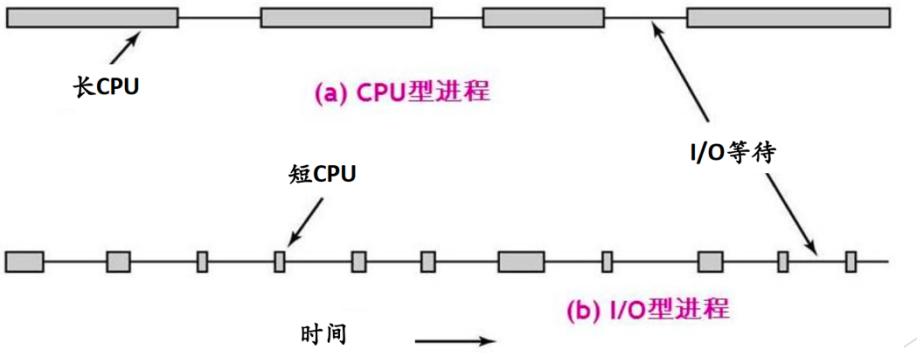
按进程执行过程中的行为划分

a) I/O 密集型或 I/O 型(I/O-bound)

- ✓ 频繁地进行 I/O，通常会花费很多时间等待 I/O 操作的完成
- ✓ 未来对 I/O 密集型进程的调度处理更重要

b) 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(Time slice / quantum)

长/短；固定/可变

a) 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

b) 选择时间片的考虑因素

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

三、 具体的调度算法

1. 批处理系统中采用的调度算法

(1) 先来先服务（FCFS-First Come First Serve）

a) 按照进程就绪的先后顺序使用 CPU，没有抢占

b) 优点

✓ 公平、简单

c) 缺点

✓ 长进程之后的短进程需要等很长时间，不利于用户的交互式体验

✓ I/O 资源和 CPU 资源的利用率较低

d) 示例

吞吐量：3 jobs/ 30s = 0.1 jobs/s
周转时间TT：P1:24；P2:27；P3:30
平均周转时间：27s

(2) 最短作业优先（SJF-Shortest Job First）

a) 具有最短完成时间的进程优先执行

b) 非抢占式（运行执行时间最短的进程，直到该进程结束或者被 I/O 阻塞）

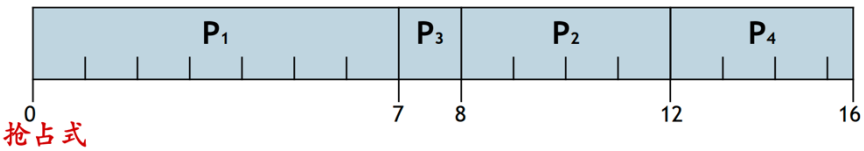
(3) 最短剩余时间优先（SRTN-Shortest Remaining Time Next）

a) SJF 的抢占版本

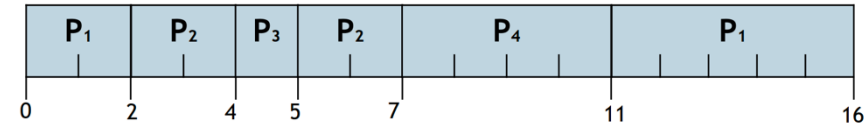
b) 当一个新就绪的进程比当前运行的进程具有更短的完成时间时，抢占当前进程，而选择新就绪的进程执行

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



- 总结：短作业优先调度算法

- ✓ 优点：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）
- ✓ 缺点：不公平（源源不断的短任务可能使长任务得不到任何处理器时间从而导致饥饿）

(4) 最高响应比优先（ HRRN-Highest Response Ratio Next）

a) 一个综合的算法

b) 计算每个进程的响应比 R，总是选择响应比最高的进程

c) 响应比 R = 作业周转时间 / 作业处理时间 =（作业处理时间+作业等待时间）/ 作业处理时间 =1+（作业等待时间 / 作业处理时间）

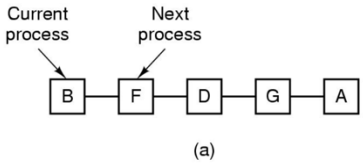
d) 可设计抢占/不可抢占版本

2. 交互式系统中采用的调度算法

(1) 轮转调度（RR-Round Robin）

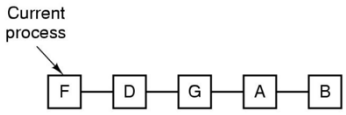
a) 思路

- ✓ 周期性任务切换
- ✓ 每个进程分配一个时间片
- ✓ 时钟中断 ➡ 轮换



(a)

可运行进程列表



(b)

B用完自己的时间片后

b) 目标

- ✓ 为短任务改善平均响应时间

示例：4个进程的执行时间如下

P1	**53**
P2	**8**
P3	**68**
P4	**24**

时间片为20的RR算法示例

等待时间 $P_1=(68-20)+(112-88)=72$
$P_2=(20-0)=20$
$P_3=(28-0)+(88-48)+(125-108)=85$
$P_4=(48-0)+(108-68)=88$

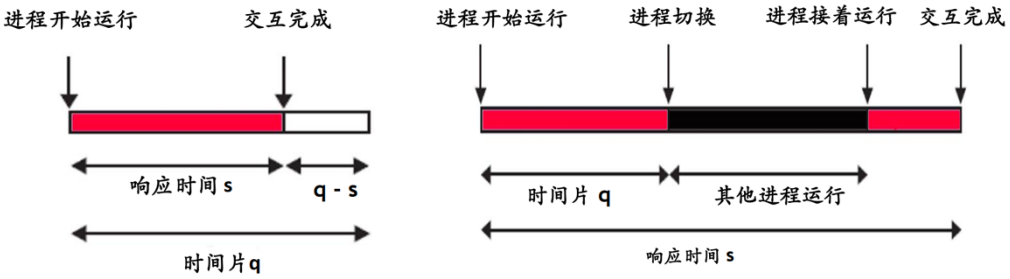
平均等待时间 = (72+20+85+88)/4=66.25

c) 如何选择合适的时间片？

- ✓ 太长——大于典型的交互时间

  问题：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

问题：进程切换浪费 CPU 时间



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

d) 优点

✓ 公平

✓ 有利于交互式计算，响应时间快

e) 缺点

✓ 由于进程切换，时间片轮转算法要花费较高的开销

✓ 对于相同大小的进程会导致周转时间很高

- 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

f) FCFS 和 RR 对比

示例：4个进程的执行时间

P1如下	53
P2	8
P3	68
P4	24

时间片	P_1	P_2	P_3	P_4	平均等待时间
RR(q=1)	84	22	85	57	62
RR(q=5)	82	20	85	58	61.25
RR(q=8)	80	8	85	56	57.25
RR(q=10)	82	10	85	68	61.25
RR(q=20)	72	20	85	88	66.25
BestFCFS	32	0	85	8	31.25
WorstFCFS	68	145	0	121	83.5

(2) 虚拟轮转调度(Virtual RR)

a) 对计算密集型进程如何分配时间片？

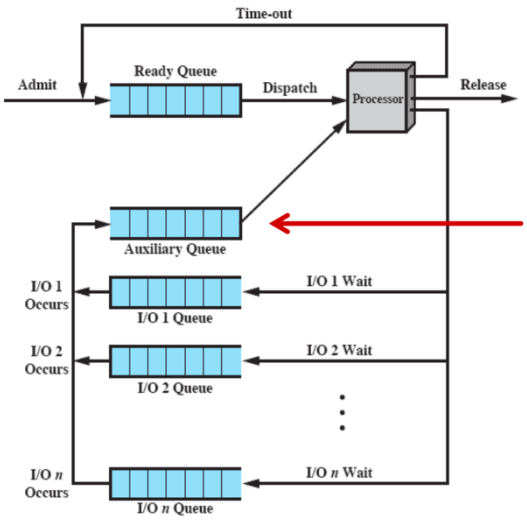
- ✓ Process A：需要 1000 ms 进行计算
- ✓ Process B：需要 1000 ms 进行计算
- ✓ Process C：while(1) {使用处理器 1 ms；使用 I/O 10 ms}
- ✓ 目标：保持处理器和磁盘忙
- ✓ FCFS：如果 A 或者 B 在 C 之前运行，它们将阻止 C 发出磁盘 I/O 请求多达 2000 ms
- ✓ RR

  时间片 100ms，CA---------B---------CA---------B---------...当 A 和 B 运行时，磁盘大多数时间处于空闲状态；每 200ms 中大约 10 ms 时间进行磁盘操作

  时间片 1ms，CABABABABABCABABABABABC…C 频繁得到调度，因此 C 可以在上次 I/O 请求执行完成后立即进行下次 I/O 操作，磁盘利用了接近 90%的时间；对 A 或是 B 的性能几乎没有影响

- ✓ SRTN

  当 C 的 I/O 磁盘操作一结束就运行 C（因为它具有最短的下次处理器突发），CA-------CA-------------CA--------- ...



虚拟轮转调度的队列图

(3) 优先级调度（HPF—Highest Priority First）

a) 选择优先级最高的进程投入运行（抢占式/非抢占式）

b) 一般情况下进程的优先级顺序

- ✓ 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

c) 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

d) 优缺点

✓ 实现简单

✓ 不公平——饥饿

e) 问题：优先级反转（优先级反置、翻转、倒挂）

✓ 基于优先级的抢占式调度算法具有的问题

✓ 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

✓ 影响

系统错误

高优先级进程停滞不前，导致系统性能降低

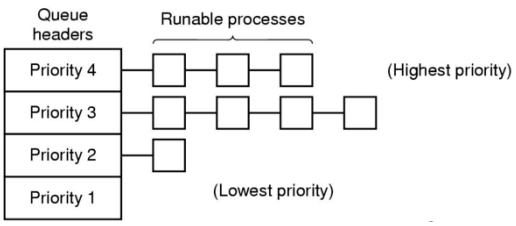
✓ 解决方案

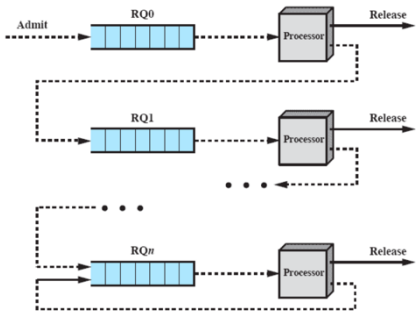
设置优先级上限（优先级天花板协议 priority ceiling protocol）

优先级继承

使用中断禁止

(4) 多级队列（Multiple queues）





a) 算法实现的细节问题

✓ 根据什么分队列？

✓ 排队方式？

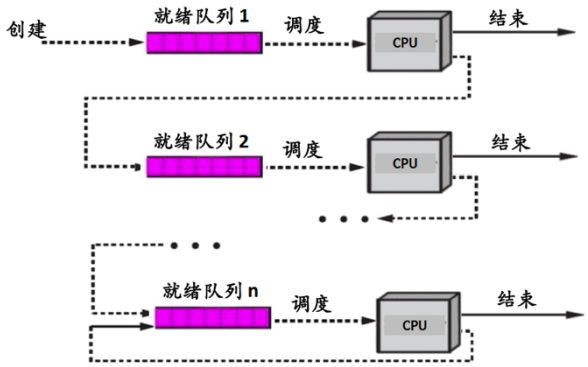
✓ 哪些进程优先级高？

✓ 每一级队列调度策略是否相同？

(5) 多级反馈队列（Multiple feedback queue）

a) 综合调度算法，非抢占式

b) 具体内容



- ✓ 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

c) 不同的实现策略

- ✓ 进入某一队列时是插入队首还是队尾？
- ✓ 再次被调度上 CPU 时对时间片的处理？

d) 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（或者队首）

(6) 其他调度算法

a) 公平共享调度（Fair-share scheduling）

b) 保证调度（Guaranteed scheduling）

c) 彩票调度（Lottery scheduling）

d) 最短进程优先（Shortest Process Next）

3. 各种调度算法的比较

调度算法	选择函数	决策模式	吞吐量	响应时间	开销	对进程的影响	饥饿问题
FCFS	max[w]	非抢占	不强调	可能很高，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密集型的进程不利	无
Round Robin	常数	抢占（时间片用完时）	如果时间片小，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	min[s]	非抢占	高	为短进程提供好的响应时间	可能较高	对长进程不利	可能
SRTN	min[s-e]	抢占（到达时）	高	提供好的响应时间	可能较高	对长进程不利	可能
HRRN	max((w+s)/s))	非抢占	高	提供好的响应时间	可能较高	很好的平衡	无
Feedback	见算法思想	抢占（时间片用完时）	不强调	不强调	可能较高	可能对I/O密集型的进程有利	可能

w：花费的等待时间；*e*：到现在为止，花费的执行时间；*s*：进程所需要的总服务时间，包括 *e*

4. 典型系统所采用的调度算法

(1) UNIX 动态优先数法

(2) 5.3BSD 多级反馈队列法

(3) Windows 基于优先级的抢占式多任务调度

(4) Linux 抢占式调度

(5) Solaris 综合调度算法

5. Windows 线程调度

(1) 调度单位是线程

(2) 采用基于动态优先级的抢占式调度，结合时间配额调整

a) 就绪线程按优先级进入相应队列（多个就绪队列）

b) 系统总是选择优先级最高的就绪线程让其运行

c) 同一优先级的各线程按时间片轮转进行调度

d) 多处理器系统中允许多个线程并行运行

(3) 引发线程调度的条件

a) 一个线程的优先级改变了

b) 一个线程改变了它的亲和(Affinity)处理器集合

c) 其他一般的条件

- ✓ 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

(4) 线程优先级

a) 实时优先级线程不改变其优先级

b) 可变优先级线程：其优先级可以在一定范围内升高或降低

- ✓ 基本优先级
- ✓ 当前优先级

c) 零页线程：优先级为 0，用于对系统中空闲物理页面清零

(5) 线程的时间配额

a) 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

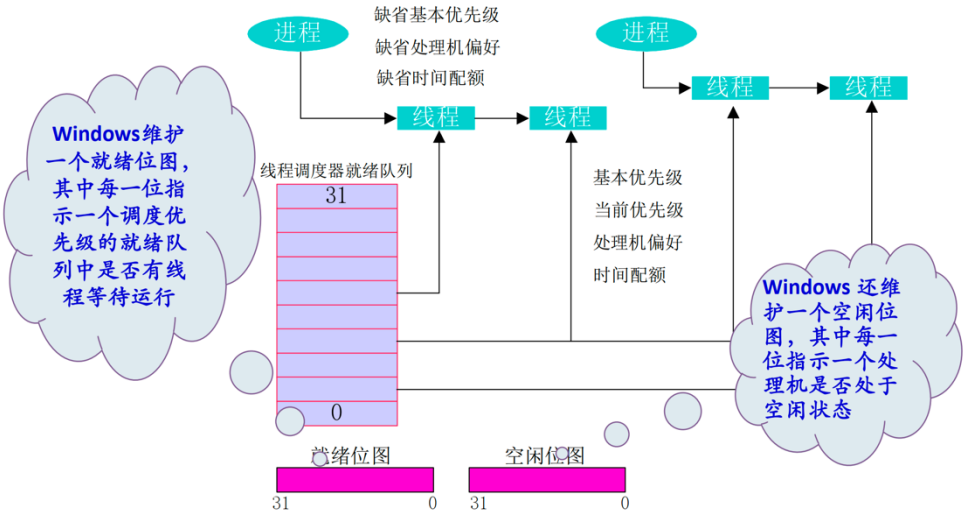
- ✓ KTHREAD：Quantum 和 QuantumReset
- ✓ #define CLOCK_QUANTUM_DECREMENT 3
- ✓ #define WAIT_QUANTUM_DECREMENT 1

b) 当一个线程用完了自己的时间配额时，如果没有其它相同优先级线程， Windows 将重新给该线程分配一个新的时间配额，并继续运行

c) 应用场景

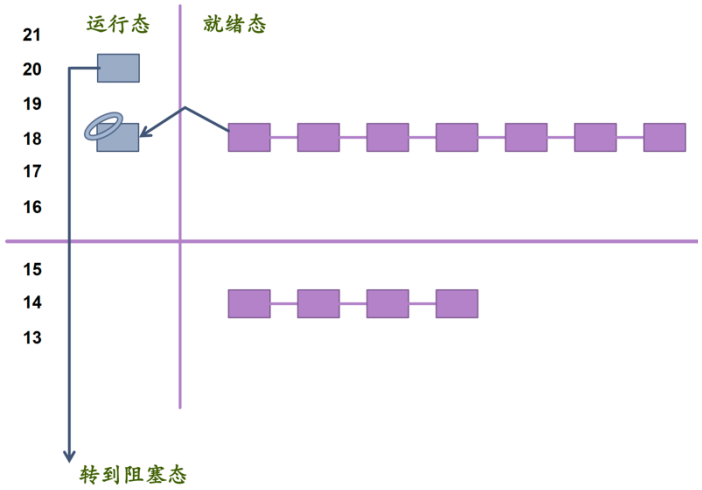
- ✓ 假设用户首先启动了一个运行时间很长的电子表格计算程序，然后切换到一个游戏程序(需要复杂图形计算并显示，CPU 型)
- ✓ 如果前台的游戏进程提高它的优先级，则后台的电子表格将会几乎得不到 CPU 时间
- ✓ 但增加游戏进程的时间配额，则不会停止执行电子表格计算，只是给游戏进程的 CPU 时间多一些而已

(6) 调度器数据结构

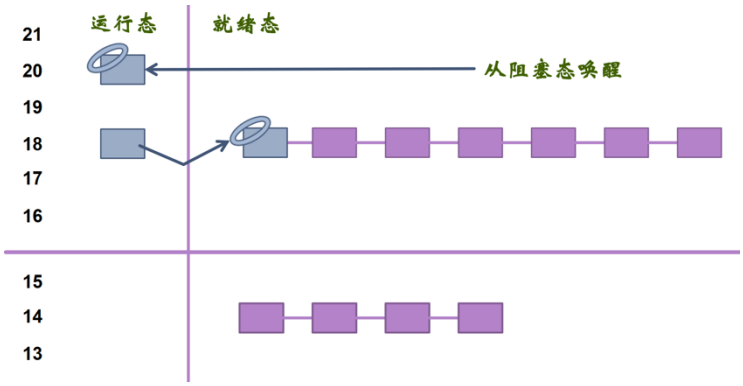


(7) 具体的调度策略

a) 主动切换



b) 抢占

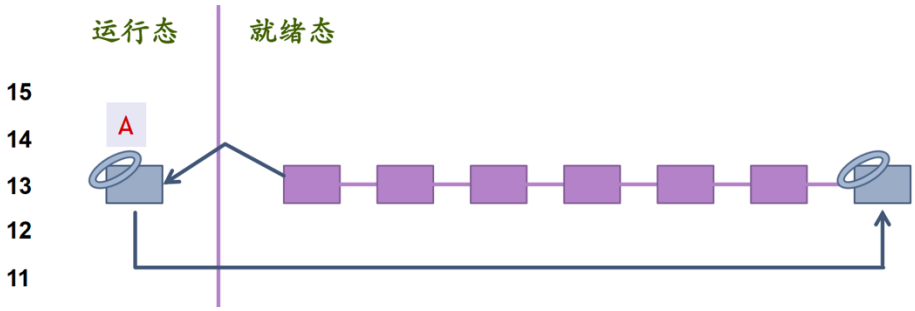


✓ 当线程被抢占时，它被放回相应优先级的就绪队列的队首

✓ 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，重新得到处理机后将运行剩余的时间配额

c) 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

(8) 改进 1：线程优先级提升

a) 针对可变优先级范围内(1 至 15)的线程优先级

b) 条件

- I/O 操作完成
  - ✓ 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，以保证该线程能更快上 CPU 运行进行数据处理
  - ✓ 线程优先级的实际提升值由设备驱动程序决定，提升建议值在文件“Wdm.h”或“Ntddk.h”中
  - ✓ 线程优先级的提升幅度与 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
  - ✓ 为了避免不公平，在 I/O 操作完成唤醒等待线程时将把该线

程的时间配额减 1

- 信号量或事件等待结束
  - ✓ 当一个等待事件对象或信号量对象的线程完成等待后，它的优先级将提升一个优先级
  - ✓ 阻塞于事件或信号量的线程得到的处理器时间比 CPU 型线程要少，这种提升可减少这种不平衡带来的影响
  - ✓ SetEvent、 PulseEvent、 ReleaseSemaphore 等函数调用可导致事件对象或信号量对象等待的结束
  - ✓ 提升是以线程的基本优先级为基准，提升后的优先级不会超过 15
  - ✓ 在等待结束时，线程的时间配额减 1，并在提升后的优先级上执行完剩余的时间配额；随后降低 1 个优先级，运行一个新的时间配额，直到优先级降低到基本优先级
- 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
  - ✓ 系统线程“平衡集管理器(balance set manager)”每秒钟扫描一次就绪队列，发现其中存在的排队超过 300 个时钟中断间隔的线程
  - ✓ 对这些线程，平衡集管理器将把该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
  - ✓ 当被提升线程用完它的时间配额后，该线程的优先级立即衰减到它原来的基本优先级

c) 空闲线程

- 空闲线程的功能：在一个循环中检测是否有要进行的工作
  - ✓ 如果在一个处理机上没有可运行的线程， Windows 会调度相应处理机对应的空闲线程
  - ✓ 由于在多处理机系统中可能两个处理机同时运行空闲线程，

所以系统中的每个处理机都有一个对应的空闲线程

- ✓ Windows 给空闲线程指定的线程优先级为 0，该空闲线程只在没有其他线程要运行时才运行

- 执行流程：循环中检测是否有要进行的工作
  - ✓ 处理所有待处理的中断请求
  - ✓ 检查是否有待处理的 DPC 请求。如果有，则清除相应软中断并执行 DPC
  - ✓ 检查是否有就绪线程可进入运行状态。如果有，调度相应线程进入运行状态
  - ✓ 调用硬件抽象层的处理机空闲例程，执行相应的电源管理功能

多处理器调度

一、 简介

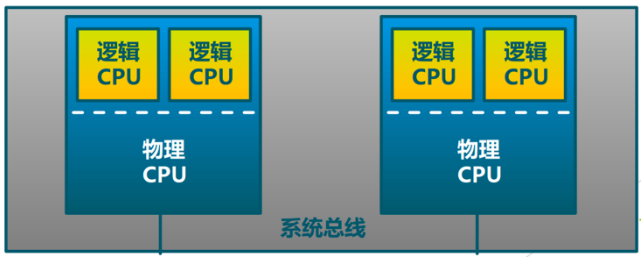
1. 特征

(1) 多个处理器组成一个多处理机系统

(2) 处理器间可负载共享

2. 对称多处理器(SMP, Symmetric multiprocessing)调度

(1) 每个处理器运行自己的调度程序，调度程序对共享资源的访问需要进行同步



(2) 对称多处理器的进程分配

a) 静态进程分配

- ✓ 进程从开始到结束都被分配到一个固定的处理机上执行
- ✓ 每个处理机有自己的就绪队列
- ✓ 调度开销小
- ✓ 各处理机可能忙闲不均

b) 动态进程分配

- ✓ 进程在执行中可分配到任意空闲处理机执行
- ✓ 所有处理机共享一个公共的就绪队列
- ✓ 调度开销大
- ✓ 各处理机的负载是均衡的

3. 调度算法设计

(1) 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

(2) 要考虑进程在多个 CPU 之间迁移时的开销

a) 高速缓存失效、TLB 失效

b) 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

(3) 考虑负载均衡问题