Windows的线程调度算法

典型操作系统的调度算法

下面我们介绍一下典型操作系统的调度算法 UNIX

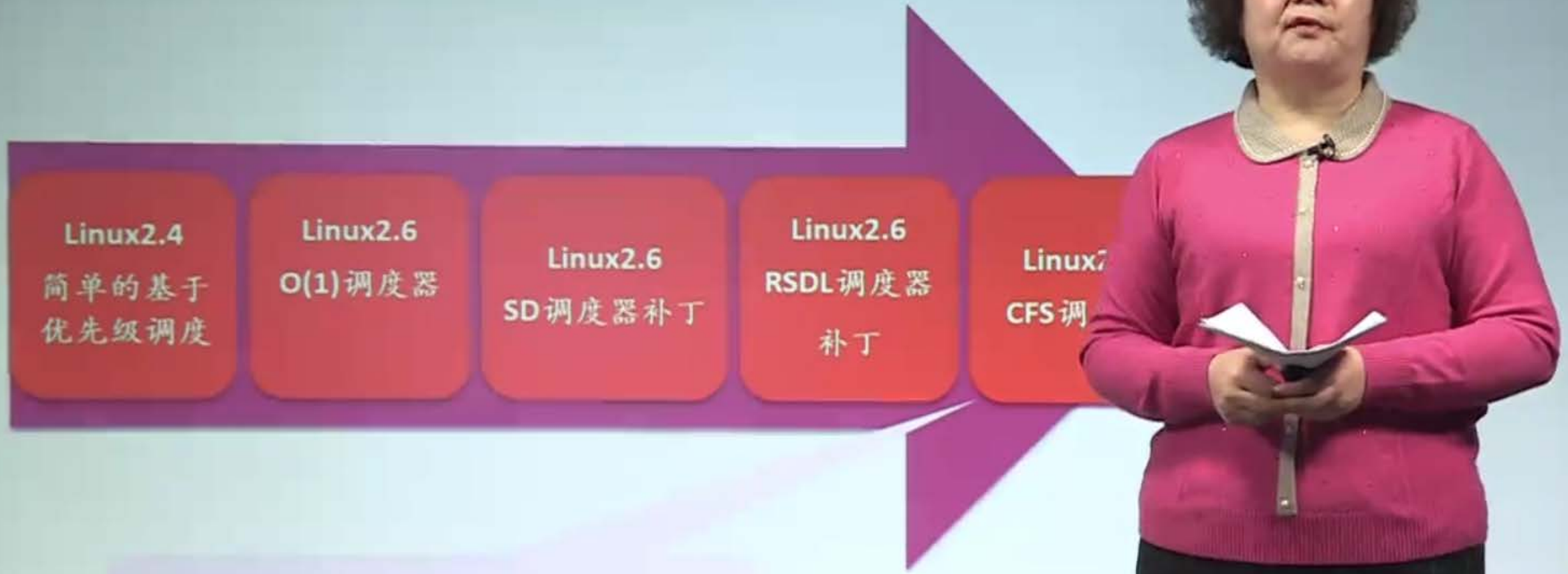
典型系统所采用的调度算法

- 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

Solaris 操作系统采用的是综合调度算法

LINUX调度算法的发展历史

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

CPU 系统中呢允许多个线程并行执行 在

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

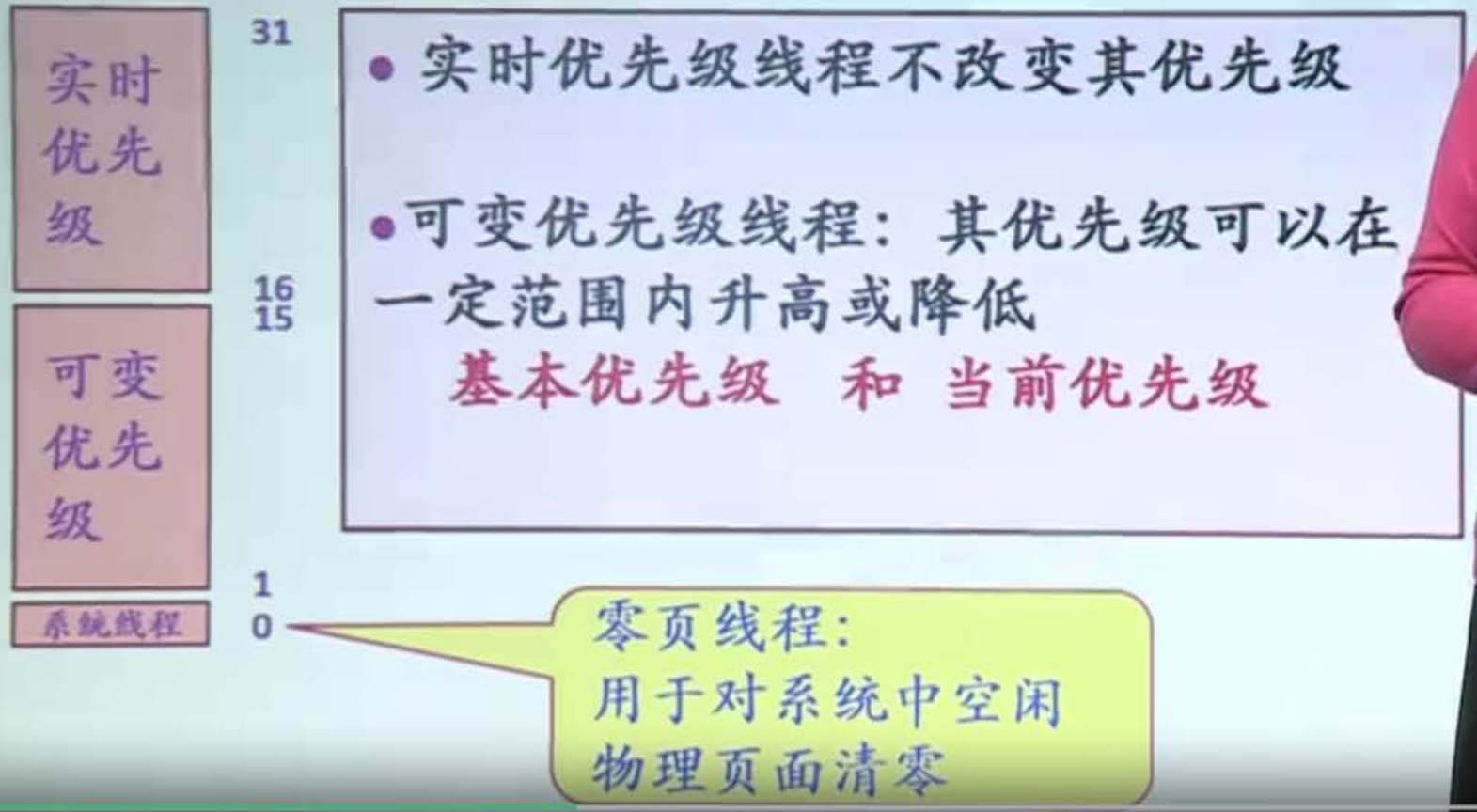
- 线程正常终止 或 由于某种错误而终止
- 新线程创建 或 一个等待线程变成就绪
- 当一个线程从运行态进入阻塞态
- 当一个线程从运行态变为就绪态

比如说增加了一个新的处理机在这个集合当中，那么就可以引发新的线程调度

Windows

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

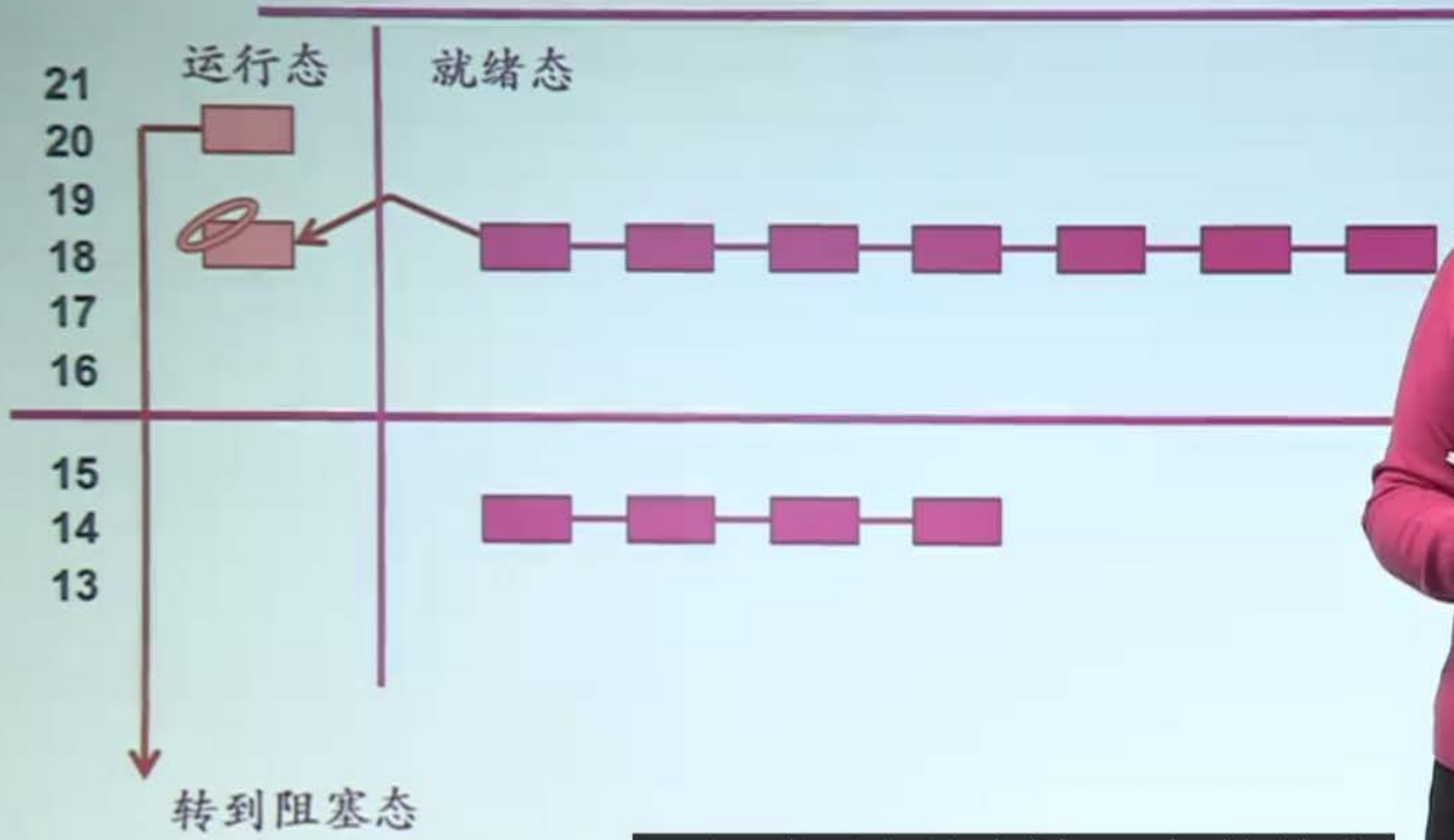
- 假设用户首先启动了一个运行时间很长的电子表格计算程序，然后切换到一个游戏程序(需要复杂图形计算并显示，CPU型)
- 如果前台的游戏进程提高它的优先级，则后台的电子表格计算进程就几乎得不到CPU时间了
- 但增加游戏进程的时间配额，则不会停止执行电子表格计算，只是给游戏进程的CPU时间多一些而已

调度策略

- 主动切换
- 抢占
- 时间配额用完

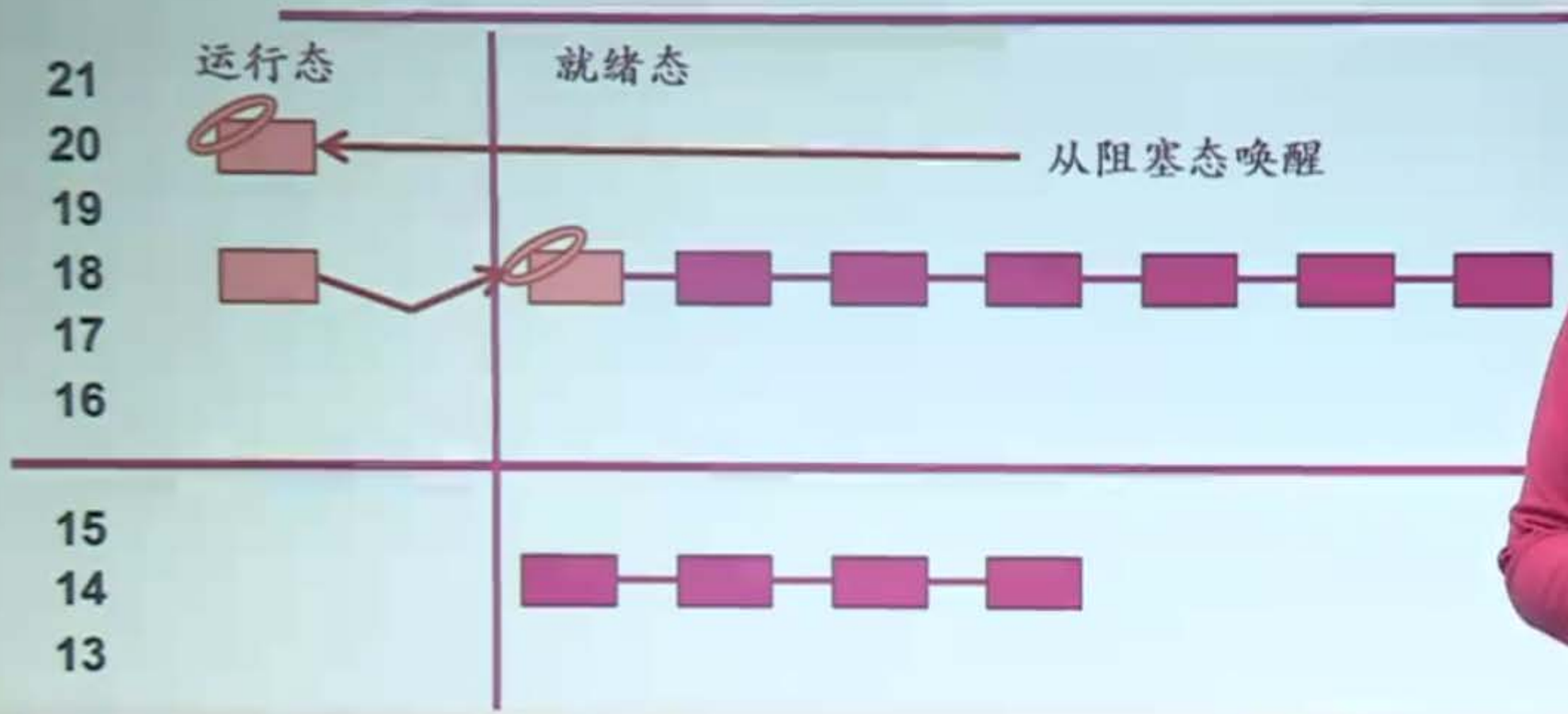
时间 下面我们讨论一下 Windows 的线程调度策略

(1)主动切换



运行 如果刚才被阻塞的线程

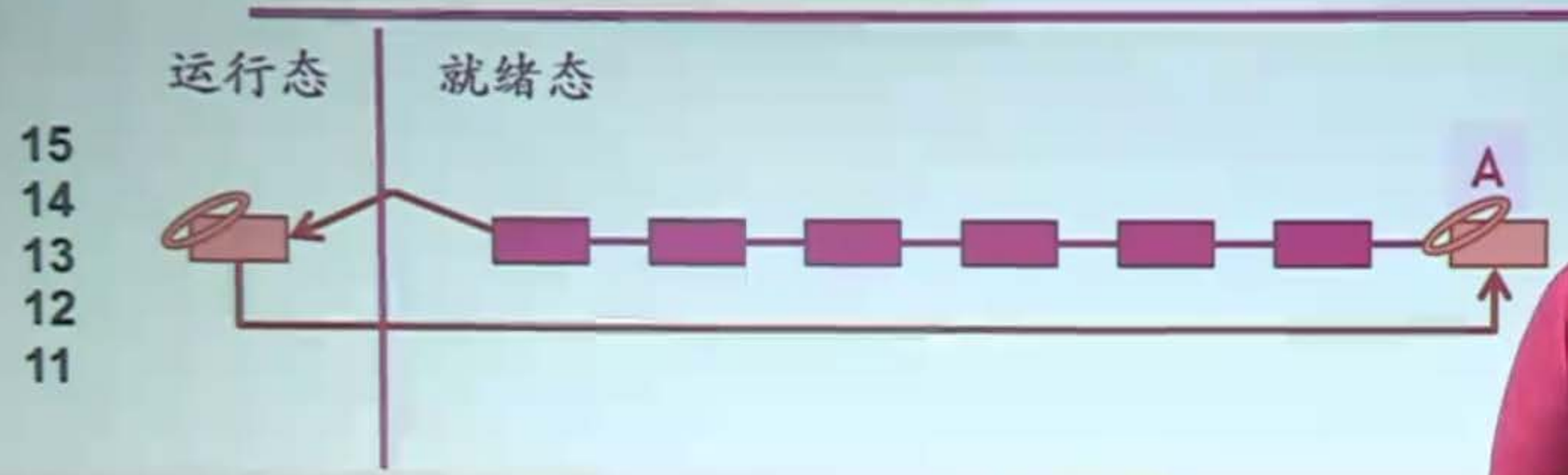
(2)抢占



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

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

会提升它的优先级？ Windows

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

当中 线程优先级的提升只是针对可变优先级这一类线程

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

线程会把它的时间配额减 1 这也是一种平衡。

“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

那么这个线程会立即衰减 回到它原来的基本优先级队列中。

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

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
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

0:00 下面我们介绍一下典型操作系统的调度算法 UNIX 操作系统采用的是动态优先数法 5.3BSD 采用的是 我们前面介绍过的多级反馈队列调度算法 Linux 操作系统呢采用的是抢占式调度策略 而 Windows 操作系统采用的是基于优先级的抢占式多任务调度策略 Solaris 操作系统采用的是综合调度算法 我们首先 来看一下 LINUX 调度算法的演化过程 Linux2.4 调度算法 采用的是一个简单的基于优先级的一个调度策略 Linux2.6 版本呢 对 2.4 的调度算法进行了改进 首先推出的是 O(1) 调度算法，之后 2.6 的几个小版本呢 推出了 SD 调度器补丁、 RSDL 调度器补丁 最后呢推出了 CFS 调度算法 一直沿用到今天。 那么 CFS 呢 是一个完全公平 调度算法。 下面我们介绍一下 Windows 的线程调度算法 由于 Windows 操作系统支持 内核级线程 所以 CPU 的调度单位呢是线程。 Windows 的 线程调度采用的是基于动态优先级的、 抢占式调度 同时结合了时间配额的调整。 我们简单介绍一下 Windows 线程调度的基本思想 就绪线程是按照 优先级进入不同的就绪队列 操作系统总是选择优先级最高的线程投入运行 同一优先级的各个线程呢是按照时间片轮转的方式进行调度的 多 CPU 系统中呢允许多个线程并行执行 在 Windows 操作系统中，引发线程调度的条件 除了之前我们介绍的四个条件之外 还增加了两个条件：第一个是，线程的优先级改变 会引发线程调度。 第二个是 一个线程改变了它的亲和处理机集合，我们简单解释一下 什么是一个线程的亲和处理机集合？ 那么有这样一个处理机的集合 有几个处理机，那么允许这个线程在这几个处理机上执行 那么除了这几个处理机之外 其他的处理机空闲，那么这个线程也不能执行 这就是这个集合就是这个线程的亲和处理机集合 那么如果这个集合改变了，也就是说 比如说增加了一个新的处理机在这个集合当中，那么就可以引发新的线程调度 Windows 把线程分成了 32 个优先级 一个分成了三类。 第一类呢 是实时优先级，优先级是从 16 到 31 第二类呢是可变优先级，优先级是从 1 到 15 最后一类呢只有一个级别 0 那么实时优先级的线程呢 一旦确定了优先级，就不会再改变了 而可变优先级的线程呢，它的优先级可以在一定的范围内 提升或者是降低。 因此 对于可变优先级的线程呢，我们可以区分它的基本优先级 和当前优先级 通常呢，系统会安排一个零页线程，也就是 给物理内存清零的这么一个线程，把它的级别呢安排在零集 那么当没有其他进程运行的时候呢，那么有一个线程可以做这样的工作 下面我们介绍一下时间配额的概念，时间配额不是一个时间长度值 而是一个称为配额单位的这么一个整数 一个进程用完了，系统分配给它的一个时间配额之后 如果没有其他相同优先级的线程 那么 Windows 会重新给这个线程分配

一个新的时间配额，让它继续运行 时间配额还有一种特殊的作用 我们来看一个例子，假如用户首先启动了一个 运行时间很长的一个电子表格计算程序 然后呢切换到了一个前台的游戏进程 这个游戏进程由于要进行复杂的图形计算 所以是一个 CPU 型进程。 如果前台的游戏进程提高了它的优先级 那么后台的这个电子表格计算进程呢就几乎得不到 CPU 时间了 所以可以通过增加游戏进程的时间配额 这样既不会停止后台的电子表格计算程序的执行 也会让前台的游戏进程得到更多的 CPU 时间 下面我们讨论一下 Windows 的线程调度策略 第一种情况是主动切换 有一个正在运行的线程，优先级是 20 在运行过程中，由于需要等待输入输出的结果 那么这个线程呢就转到了阻塞态 让出了 CPU。 那么调度程序呢会去选择一个新的线程上 CPU 运行 如果刚才被阻塞的线程 被唤醒了，那么由于它的优先级高，它会去抢占 CPU，上 CPU 去运行 那么被抢占的这个线程呢就回到了就绪队列 我们来看一下，那么当线程被抢占时它被放回到相应优先级的就绪队列的队首 除此之外，还有不同的策略 对于处于实时优先级的线程在被抢占的时候 那么对它的时间配额是重新分配给它一个完整的时间配额 因此，它再度上 CPU 运行的时候呢，它运行的是一个完整的时间配额 而对于处于可变优先级的线程，在它被抢占后 那么回到了就绪队列的队首，它再度被调度上 CPU 后 那么它只能够运行完刚才剩余的时间配额 那么这就区别出两种不同类型的线程 第三种情况是时间配额用完 好，我们假设线程 A 的时间配额用完 那么可能有两种情况 第一种情况是线程 A 的优先级没有降低 那么如果队列里头 还有其他的就绪线程 调度程序会去选择下一个线程执行 而 A 就回到了原来就绪队列的末尾 如果队列中没有其他的就绪线程 那么系统会给线程 A 重新分配一个时间配额，然后让它继续执行 这是第一种情况。 第二种情况呢 是 A 的优先级会降低，因此呢 Windows 会去选择一个更高优先级的线程执行 为什么一个线程用完了它的时间配额后 它的优先级会被降低？那是因为 这个线程在此之前，它的优先级曾经被提升过 我们来看一下，在 Windows 的调度策略中 如何体现对某类线程具有倾向性呢？ 如何解决由于调度策略中 潜在的不公平性而带来的一些饥饿现象呢？ 如何改善 系统的吞吐量、响应时间等整体特征呢？ 通常的解决方案就是 提升线程的优先级 给线程分配一个更大的时间配额 我们来看一下对哪些线程 会提升它的优先级？ Windows 会针对下列 5 种情况提升线程的当前优先级 这 5 种情况是 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了一

个等待操作 或者由于窗口活动而唤醒了窗口线程 最后一种情况是，线程处于 就绪态超过了一定的时间还没有被调度 运行，这也就是说产生了"饥饿"现象 那么在 Windows 当中 线程优先级的提升只是针对可变优先级这一类线程 下面我们举两个例子来讨论一下线程优先级的提升 第一个例子是 I/O 操作完成后的线程优先级的提升 在完成了 I/O 操作后 Windows 会临时提升等待该操作的线程的优先级 以确保这个线程能尽快上 CPU 运行去处理相应的数据 优先级的提升值呢 是由设备驱动程序决定的 会有建议值保存在系统文件中 优先级提升的幅度 与对 I/O 操作的响应时间的要求是一致的 响应要求越高，那么提升的幅度就越大 那么设备驱动程序在完成 I/O 请求之后呢 会去调用内核函数来指定优先级提升的幅度 另外为了避免不公平，在 I/O 操作完成后唤醒的 线程会把它的时间配额减 1 这也是一种平衡。 第二个例子呢 是"饥饿"线程的优先级提升 有一个系统线程叫"平衡集管理器" 它每秒钟扫描一次就绪队列 去查看一下是否存在 等待时间超过了 300 个时钟周期的这样的线程 如果存在这样的线程，平衡集管理器 就会将这样线程的优先级提高到 15 也就是在可变优先级的范围内的最高优先级 不仅如此呢，它还会给这个线程 一个长度为正常值 4 倍的一个时间配额 因此当被提升的线程 调度上 CUP 用完了时间配额之后 那么这个线程会立即衰减 回到它原来的基本优先级队列中。 那么这就是 解决饥饿问题的一种方法 本讲的重点呢分为 3 个方面 首先要掌握处理器调度的相关概念 其次要掌握主要的 调度算法，像先来先服务、 短作业优先 最短剩余时间优先、 最高响应比优先、 时间片轮转 最高优先级和多级反馈队列 还要了解 Windows 多处理器调度的基本思想 这里给出了重点阅读的材料，那么一个是第 2 章的相关内容 还有第 11 章 11.4.3 当中 关于 Windows 的调度部分。 这里也 给出了本章所需要掌握的重点概念 好 CPU 调度这一讲就介绍到这里，谢谢大家！