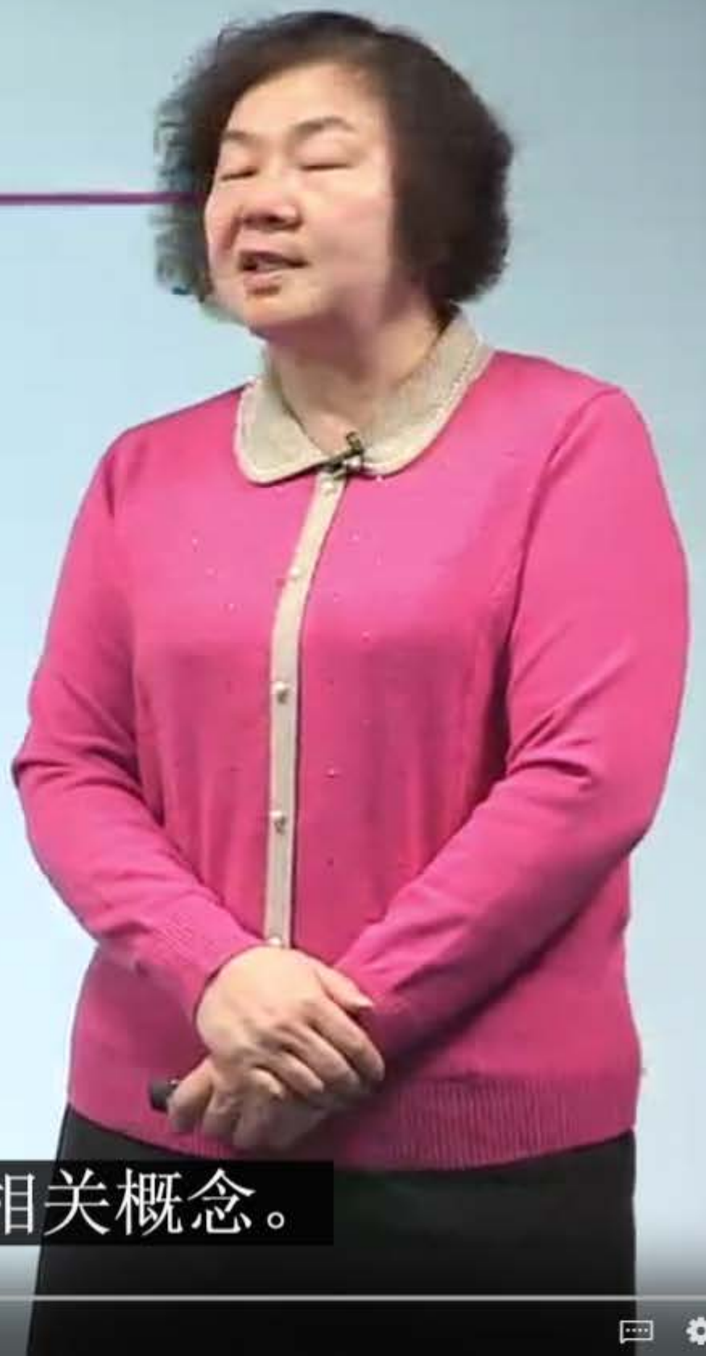
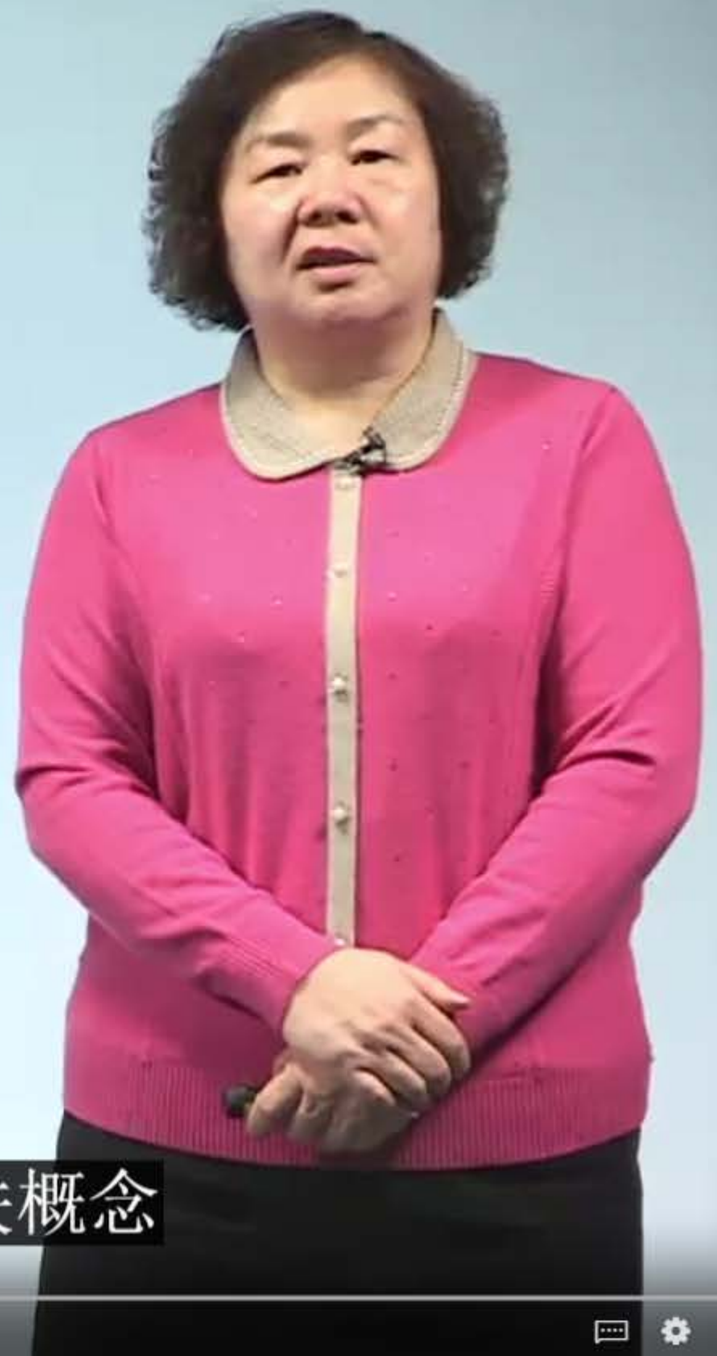
CPU调度

- 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



我们主要介绍以下这些内容 CPU 调度的相关概念。

CPU调度的相关概念



首先，我们介绍一下 CPU 调度的相关概念

CPU调度

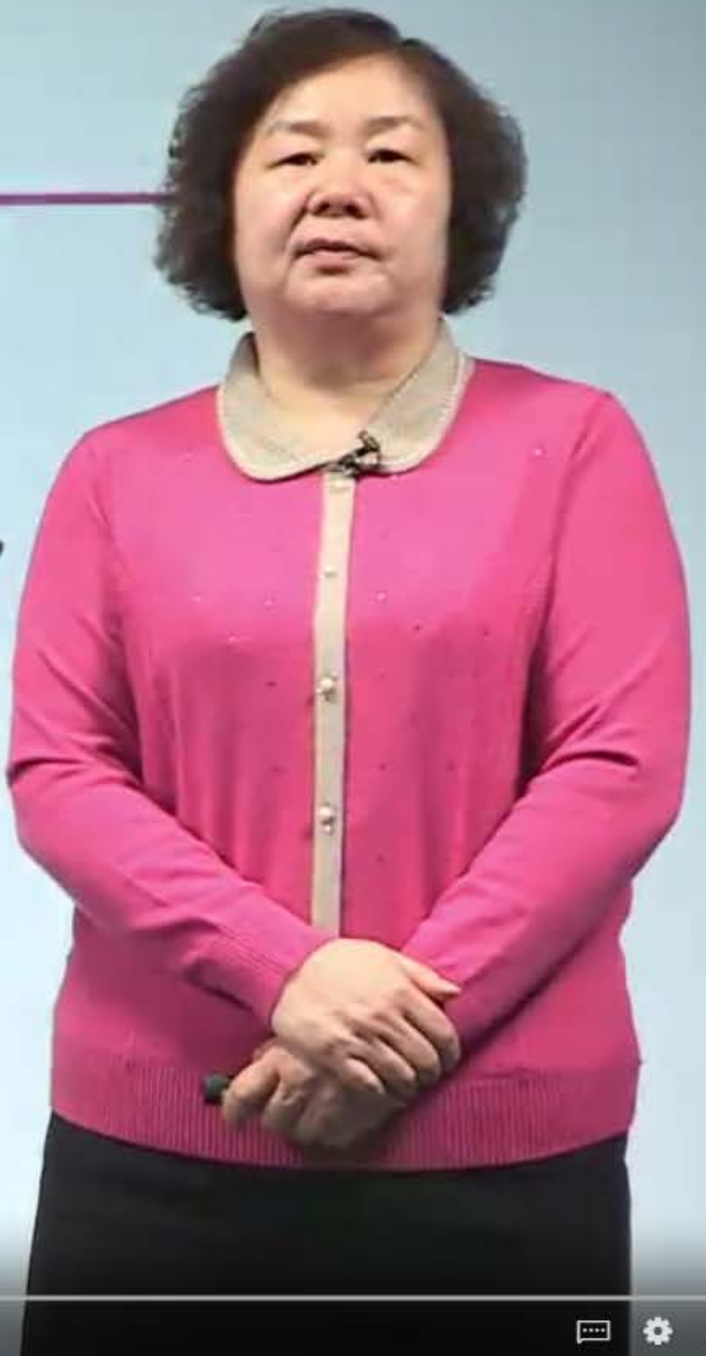
- **CPU调度**

——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 ≥ 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

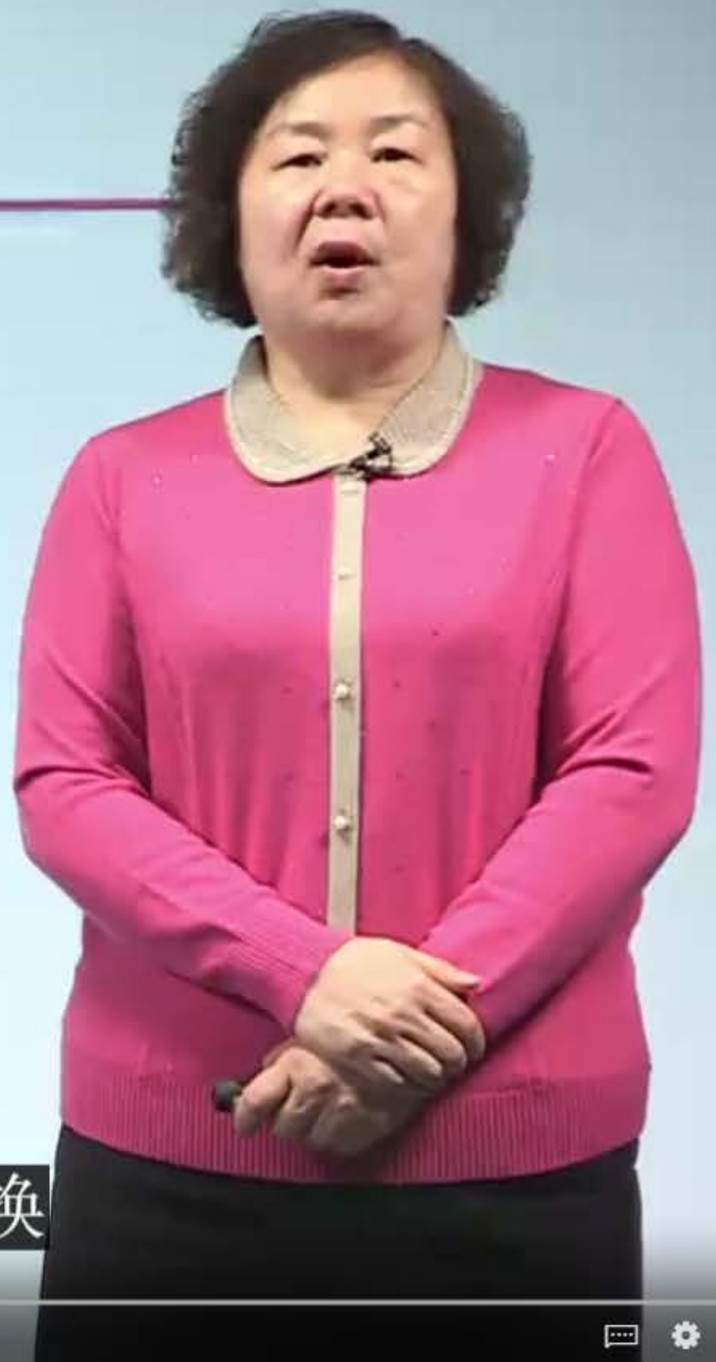
— 调度算法

WHEN： 何时进行选择

— 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）



主要内容呢就是 进程的上下文切换

CPU调度的时机(1/2)

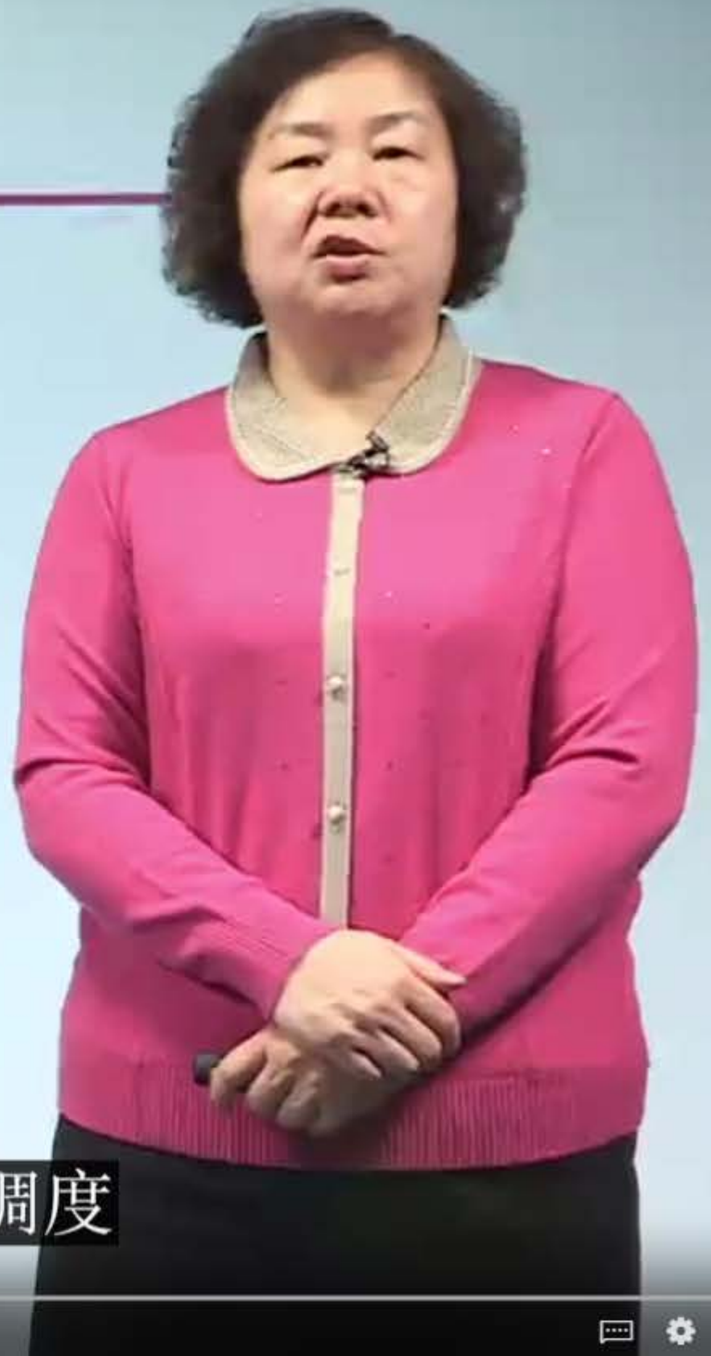
事件发生 → 当前运行的进程暂停运行 → 硬件机制响应后 → 进入操作系统，处理相应的事件 → 结束处理后：

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

→ 就绪队列改变了 → 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

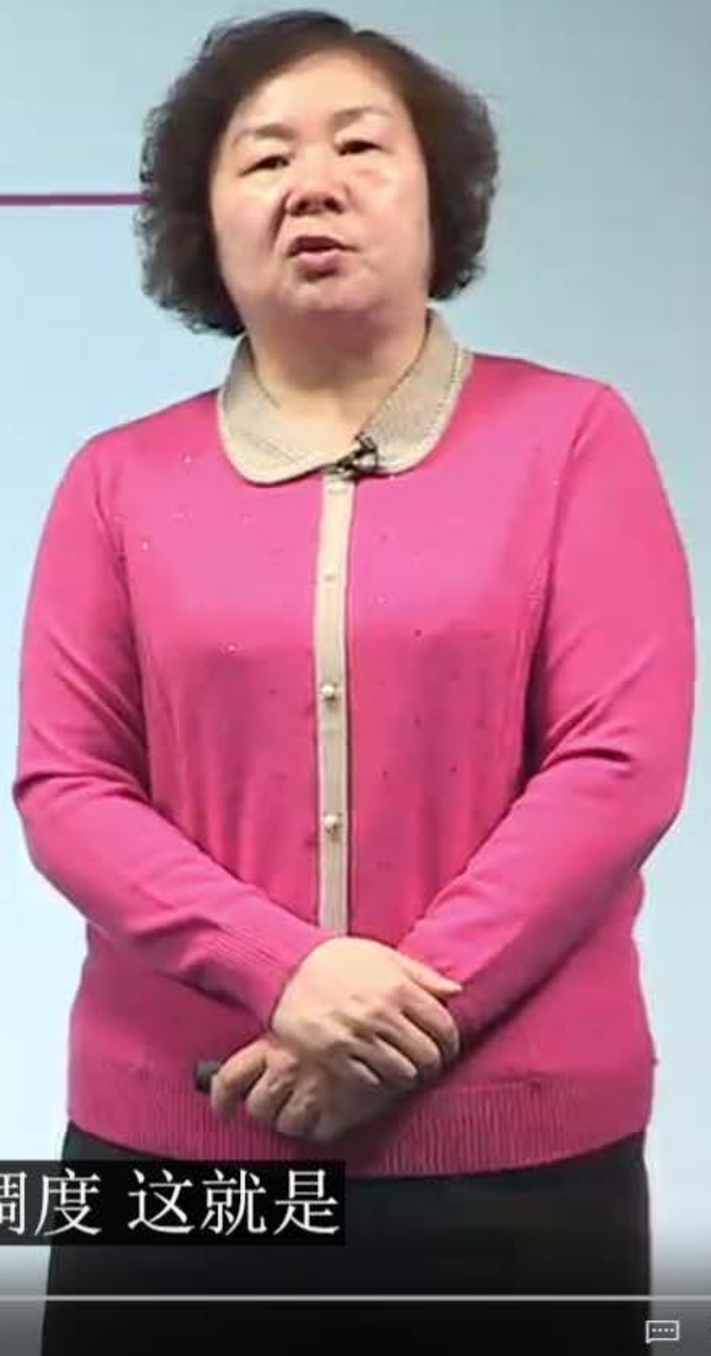


也就是就绪队列的改变 引发了重新调度

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



返回到用户态的时候，这个时候要重新调度 这就是

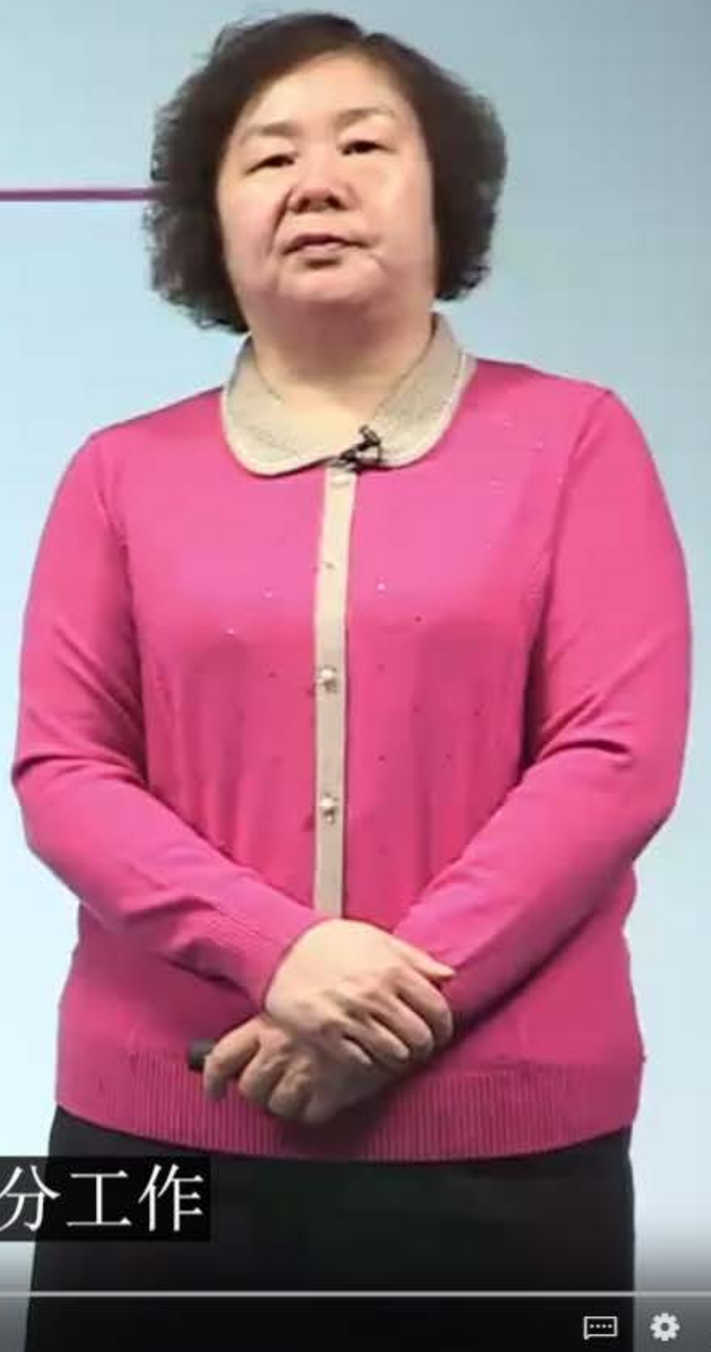
调度过程——进程切换(1/2)

- 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

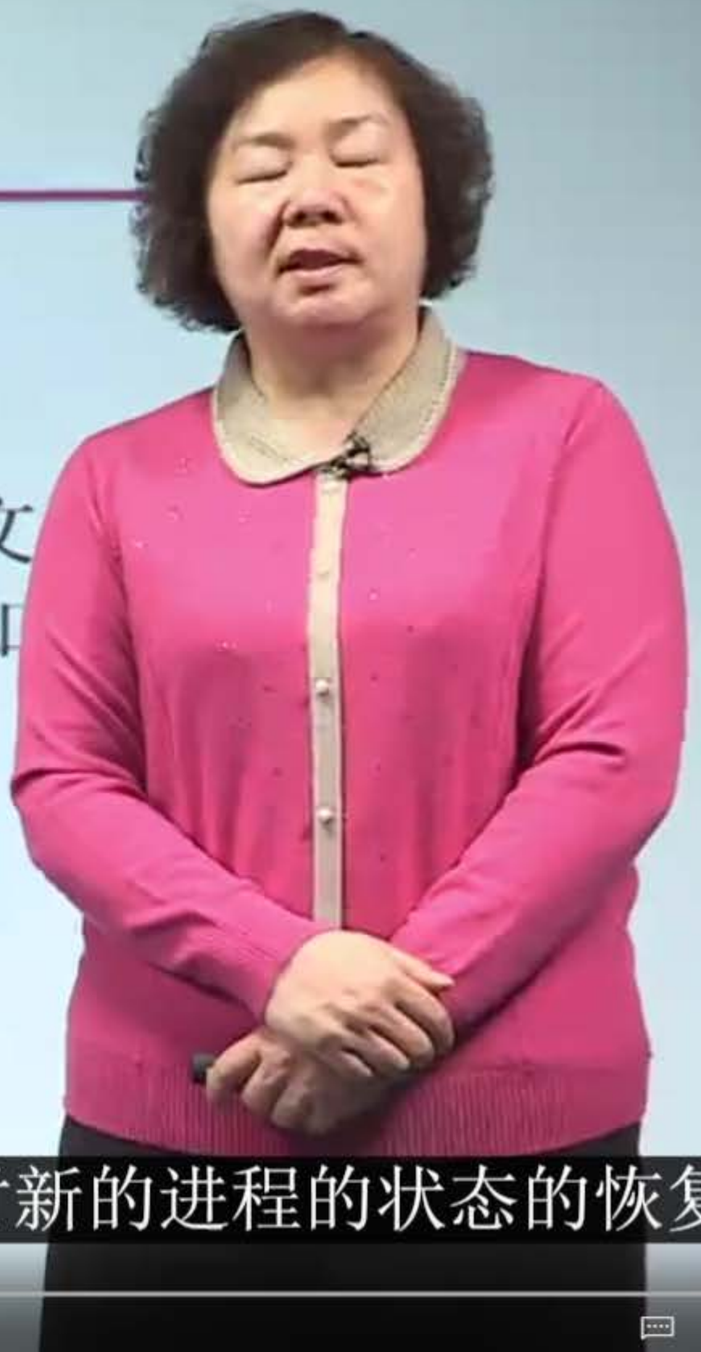


的一个过程 进程切换呢，主要包括两部分工作

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

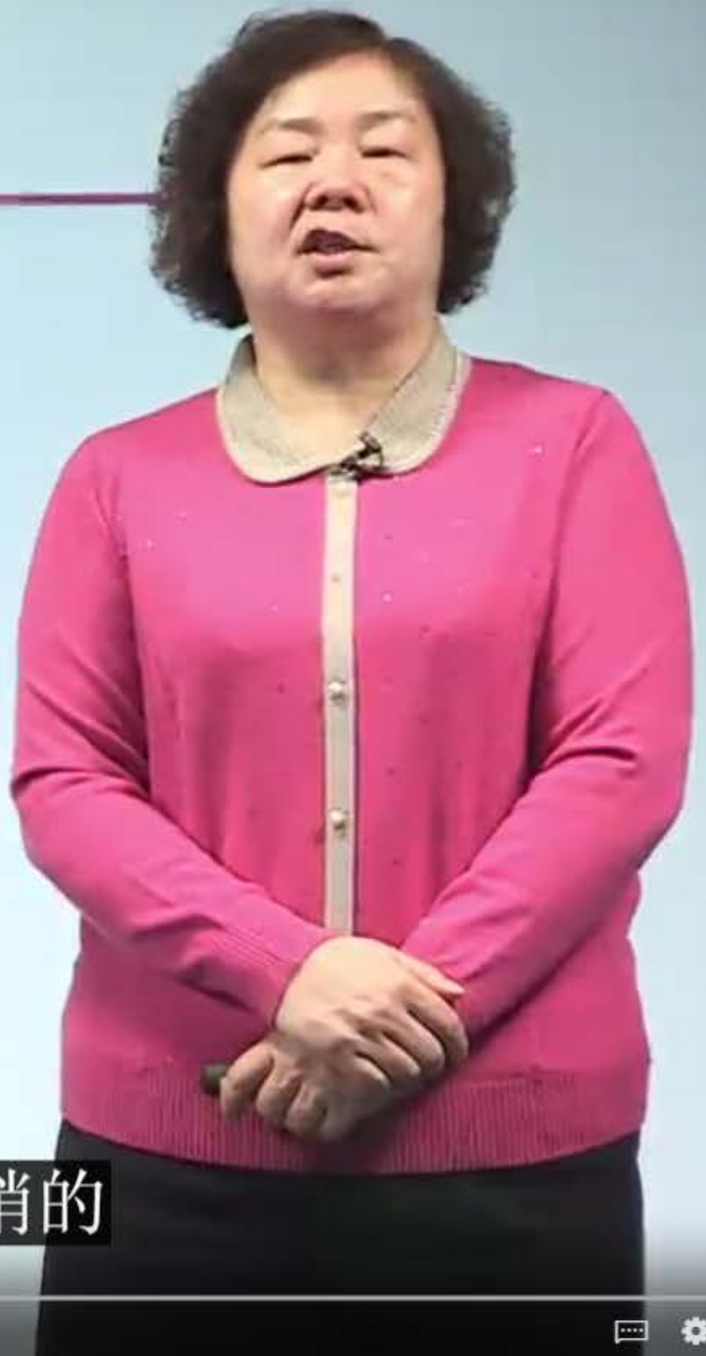


过程当中，包括了对原有进程的各种状态的保存 以及对新的进程的状态的恢复，这样一个过程

上下文切换具体步骤

场景：进程A下CPU，进程B上CPU

- 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.....）
- 用新状态和其他相关信息更新进程A的PCB
- 把进程A移至合适的队列（就绪、阻塞.....）
- 将进程B的状态设置为运行态
- 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.....）

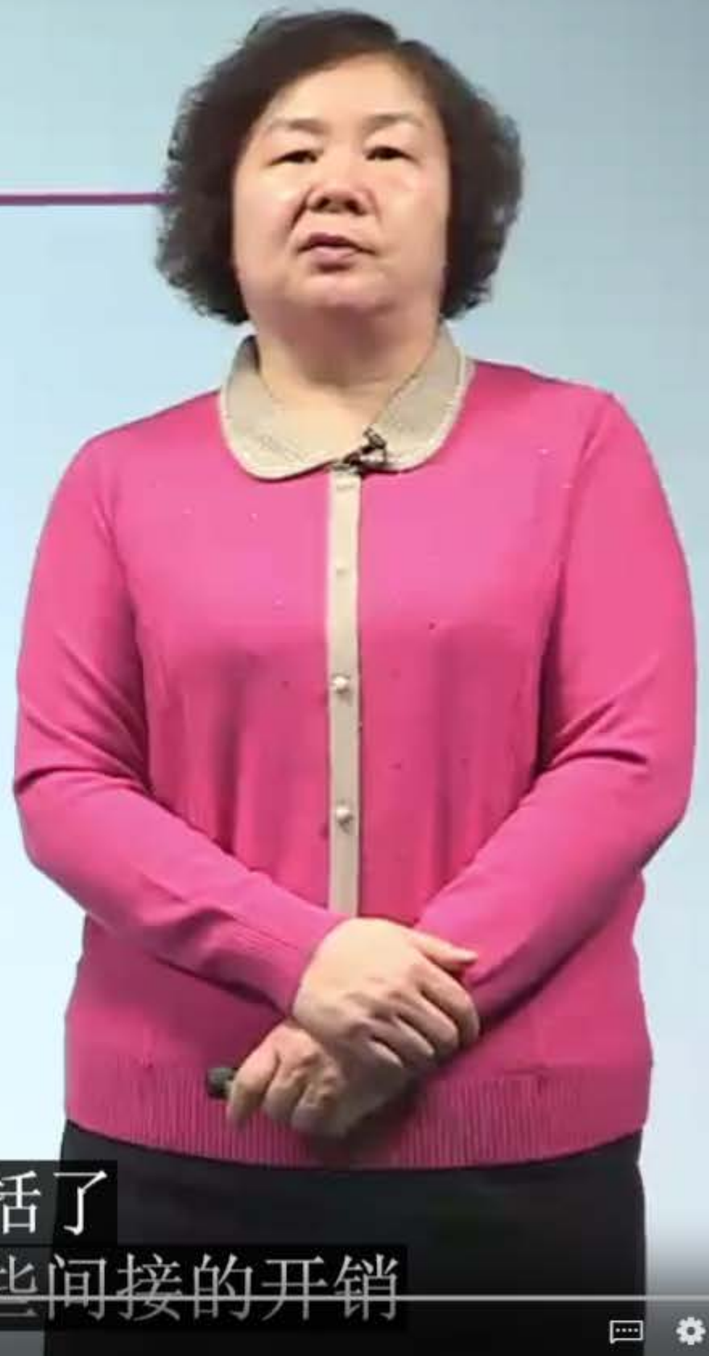


CPU 接着执行 上下文切换是有开销的

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 Lookup Buffer)失效

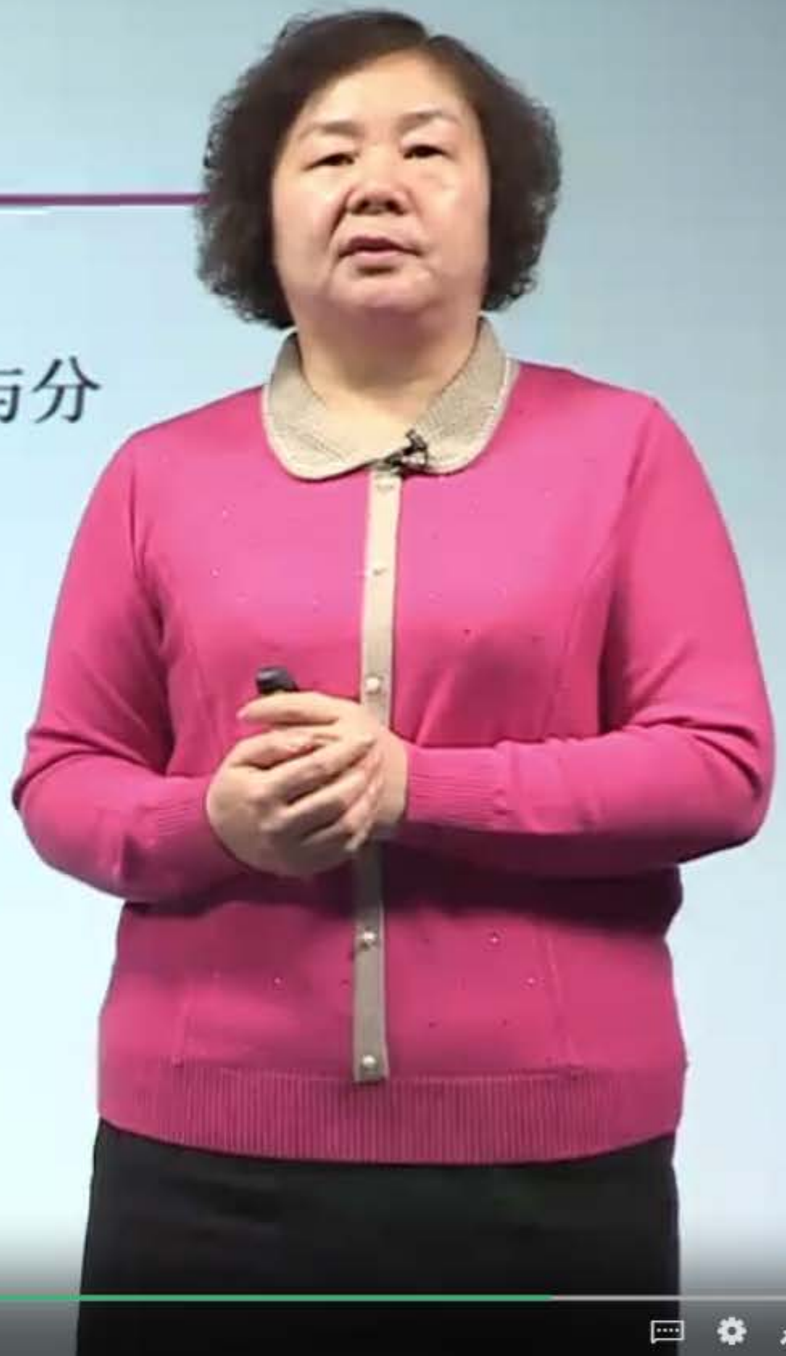


所以，上下文切换的开销呢，既包括了
直接的开销，也包括了一些我们看不到的一些间接的开销

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)

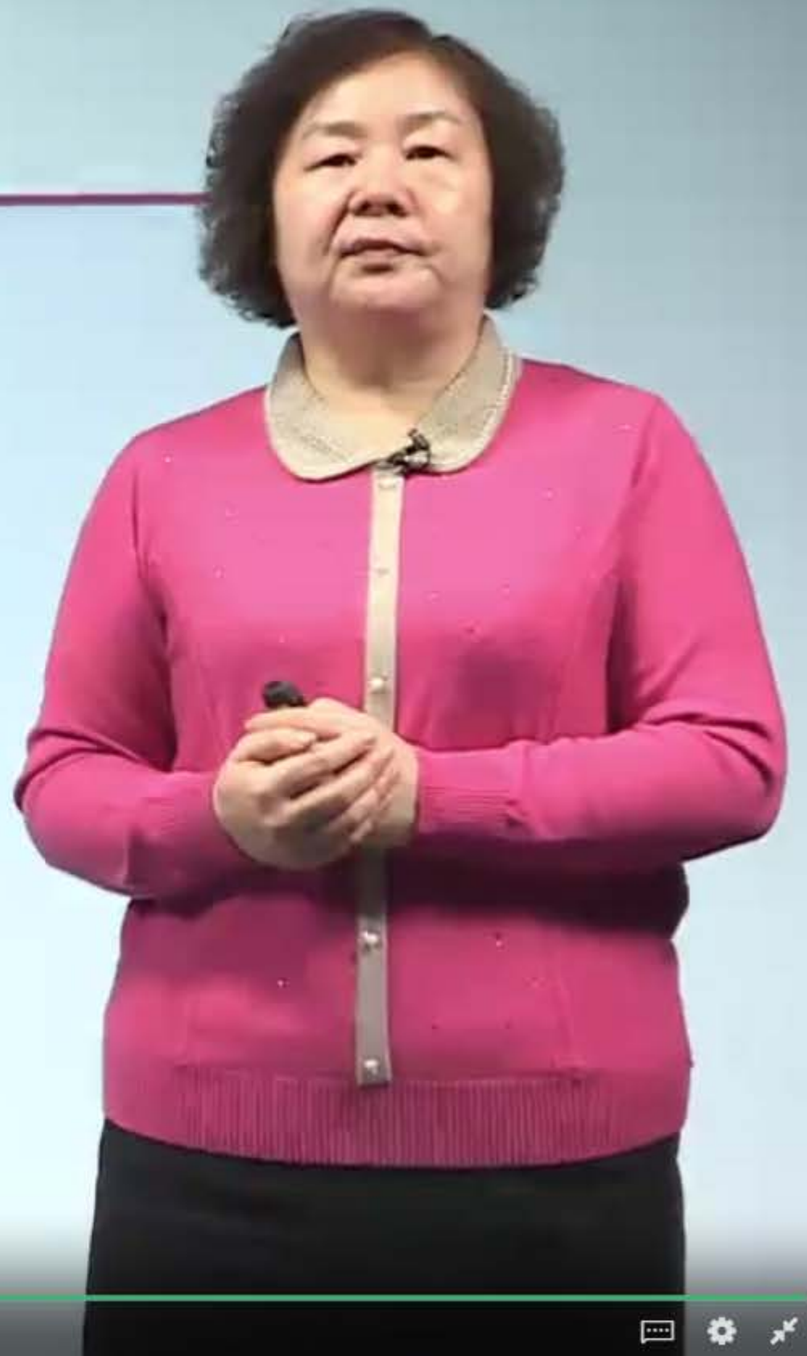
  每个进程从提出请求到运行完成的时间
- 响应时间RT(Response Time)

  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



0:00 大家好！今天我给大家带来的是操作系统原理课的第四讲 处理器调度，也就是 CPU 调度 我们主要介绍以下这些内容 CPU 调度的相关概念。 然后我们讨论了 在设计调度算法的时候，要考虑的几个因素 接着呢，我们会去介绍批处理操作系统 或者是交互式系统所采用的各种调度算法 我们也简单地要介绍一下，Windows 操作系统所涉及的线程调度算法 首先，我们介绍一下 CPU 调度的相关概念 什么是 CPU 调度？ CPU 调度的任务是控制、 协调 多个进程对 CPU 的竞争 也就是按照一定的调度算法，从就绪队列中选择一个进程，然后把 CPU 的控制权交给被选中的进程 如果就绪队列没有其它的进程 那么系统会安排一个空闲进程 或者也称为 idle 进程，上 CPU 运行 CPU 调度所面临的一个场景呢，是这样一个场景 系统中呢，有 N 个进程 它们就绪等待上 CPU 运行，而系统呢 有多余一个 CPU ，因此呢，操作系统的调度模块 要决策，究竟 给哪个进程分配哪一个 CPU 这就是操作系统 CPU 调度所要决策的事情 CPU 调度呢，主要要解决三个问题 第一个问题， WHAT 按什么样的原则，选择下一个要运行的进程 这就是调度算法。 这也是我们这一讲当中的重点内容 第二个问题呢，是 WHEN 何时进行选择？这就是调度时机 第三个问题，是 HOW 选中了一个进程之后 如何让被选中的这个进程上 CPU 运行？ 这就是调度的过程。 主要内容呢就是 进程的上下文切换 我们首先来介绍 CPU 调度的时机 系统运行时，会发生很多的事件 比如说，一些进程的操作 比如说， I/O 中断 时间中断。 那么这些事件发生以后 系统要做相应的处理。 这是我们第二讲 所讲的主要内容。 我们来回顾一下。 事件的发生 使得当前正在运行的进程暂停 硬件机制呢去响应这个事件 进入了操作系统，操作系统呢处理相应的事件 当事件处理完后，那么，会发生什么呢？事件处理完后，我们会看到 某些进程的状态发生了变化 也可能呢，又创建出了一些新的进程 这就导致了就绪队列改变了 就绪队列的改变，就需要进程调度 按照事先预定的算法 从就绪队列中重新选择一个进程 那么，这就是调度的时机。 也就是就绪队列的改变 引发了重新调度 那么，我们总结一下，进程调度的时机有 4 个 第一个，进程正常终止 或者由于某种错误终止。 第二个 创建了新的进程，或者是一个等待的进程变为了就绪 第三个时机 是一个进程从运行态进入了等待态 第四个时机，是一个进程由于时间片到等因素 从运行态变为了就绪。 总而言之 什么时候重新调度？往往是 内核对中断、 陷入 系统调用等处理之后 返回到用户态的时候，这个时候要重新调度 这就是 CPU 调度的时机 当有一个新的进程被选中之后 那

么这个进程，可以是刚刚被暂停执行的进程 也可以是一个新的进程。 如果是一个新的进程 那么就要发生一个进程切换 所谓进程切换呢，就指的是 一个进程让出 CPU ，另外一个进程占用 CPU 的一个过程 进程切换呢，主要包括两部分工作 首先，要切换全局页目录 它的目的是加载一个新的地址空间 因为，新的进程上 CPU ，那么它要用自己的地址空间 第二个，是切换内核栈和硬件上下文 因此，我们说进程切换实际上这个 过程当中，包括了对原有进程的各种状态的保存 以及对新的进程的状态的恢复，这样一个过程 我们举一个例子，如果进程 A 下 CPU 进程 B 上 CPU ，那么进程切换主要做的工作呢，是这样一些步骤 首先要保存进程 A 的上下文环境 程序计数器呀，程序状态字呀，还有其它的一些寄存器 然后，用新的状态和其它的 相关信息来更新进程 A 的 PCB 把进程 A 移到一个合适的队列，可能是就绪队列，可能是某个等待队列 进程 B 被选中之后 把进程 B 的状态，把它改变成运行态 然后从进程 B 的 PCB 当中，恢复上下文 做完了这些事情，那么进程 B 就上 CPU 运行了 而进程 A 的所有信息保存好之后，以后它还可以继续上 CPU 接着执行 上下文切换是有开销的 那么，什么是进程的上下文切换的开销呢？ 通常包括两部分。 第一部分开销呢 是直接的开销，是内核 完成上下文切换，所花费的 CPU 的时间 这些时间呢，用于保存和恢复寄存器 用于切换地址空间 特别是切换地址空间的这些指令呢，是非常昂贵的 另一部分开销呢，是间接开销 它就指的是，高速缓存失效 缓冲区缓存失效，还有 TLB 快表的失效 这里呢，我稍微做一些解释 这部分内容呢，在存储管理里头还会去介绍 那么，什么是高速缓存呢？ 高速缓存里头存放了一些 刚才执行的这些，进程当中的一些指令和数据 那么， TLB 快表里头呢，存放了这个进程的一些页表表项 那么，新的进程上 CPU 之后，原来的这些内容 都没用了，都失效了。 还要把新的 进程所需要的指令数据，送入高速缓存 或者是把新的进程的页表表项呢，送入 TLB 快表里 那么，又需要花一些时间，因此呢，这就指的是这部分开销 所以，上下文切换的开销呢，既包括了 直接的开销，也包括了一些我们看不到的一些间接的开销 下面我们讨论 一下CPU 调度算法的设计时要考虑的一些问题 那么什么情况下，需

要仔细来斟酌调度算法呢？我们来看一下调度算法的一些演变 早期的批处理对调度算法的要求不高 也比较简单，那么因为它是一个程序执行完了以后接着执行下一个程序 但是当有了多道程序设计系统之后，那么多个程序都要想 进内存，都上 CPU，那么调度算法就要去选择 推敲一下让哪些进程早一点上 CPU 了 那么到了批处理与分时系统 的一个混合系统的过程中，那我们会看到可能 调度算法既要照顾到前台的进程 也要照顾到那些后台的进程，它可以合理地来安排 调度的顺序，让不同的进程都能有机会上 CPU 执行 个人计算机，可能刚开始的时候调度算法非常简单 后来慢慢慢慢演化到现在，那么桌面操作系统里头那么调度算法也变得非常复杂 对于网络服务器由于有多个客户端同时 向它提出服务请求，所以要在调度算法上下一些功夫 要斟酌不同的调度算法，那么在设计调度算法的时候呢 从用户的角度和系统的角度对调度算法的要求是不一样的 我们来看一下，好我们来看一下关于性能方面 用户对性能的要求呢，第一，周转时间 短一点，响应时间快一点 如果有些紧急的任务应该能够在最后的期限之内 能够完成。 那么从系统的角度上看呢 它可能追求的是吞吐量 追求的是 CPU 的利用率和资源的利用率 在其他方面呢 那么从用户的角度呢，他希望他能够对自己的程序什么时候运行有一个可预测 那么从系统来看呢 更多的是想说，是不是公平，是不是对所有的 进程都是公平处理的。 如果有一些 进程更重要，更紧急一些，是不是能够 让它尽快地执行。 另外 系统中各种资源是不是都能够平衡地使用 所以从这张图中，我们可以看到，用户的角度和系统的角度 对于调度算法有不同的要求。 而这些要求，往往 有的时候是相互矛盾的。因此，在设计调度算法的时候呢，是要在各种 因素当中呢，折中权衡，下面我们给出 衡量调度算法的一些典型的指标 第一个，吞吐量，也就是单位时间里头 完成进程的数量。 那么这个吞吐量是越大越好 第二个指标，是周转时间 也就是从进程提出它请求 到完成这个进程的整个的时间，我们称之为周转时间 对于每个进程来讲，周转时间是越短越好 第三个指标呢，是响应时间，是从进程 提出请求到首次回应这样一个时间 当然也是越短越好。 除了这三个指标之外呢 实际上呢，我们还要考虑到其它两个指标，一个是 CPU 的利用率，也可以是其它资源的利用率 也就是在 CPU 上有多长时间 是空闲的，有多长时间是忙碌的。 忙碌的时间越多越好 第二个是等待时间，对于一个进程来讲 它在就绪以后，等待的时间越少越好 那么我们在设计调度算法的时候呢，要考虑到对 这个几个指标，那么这个调度算法是个什么样的情况