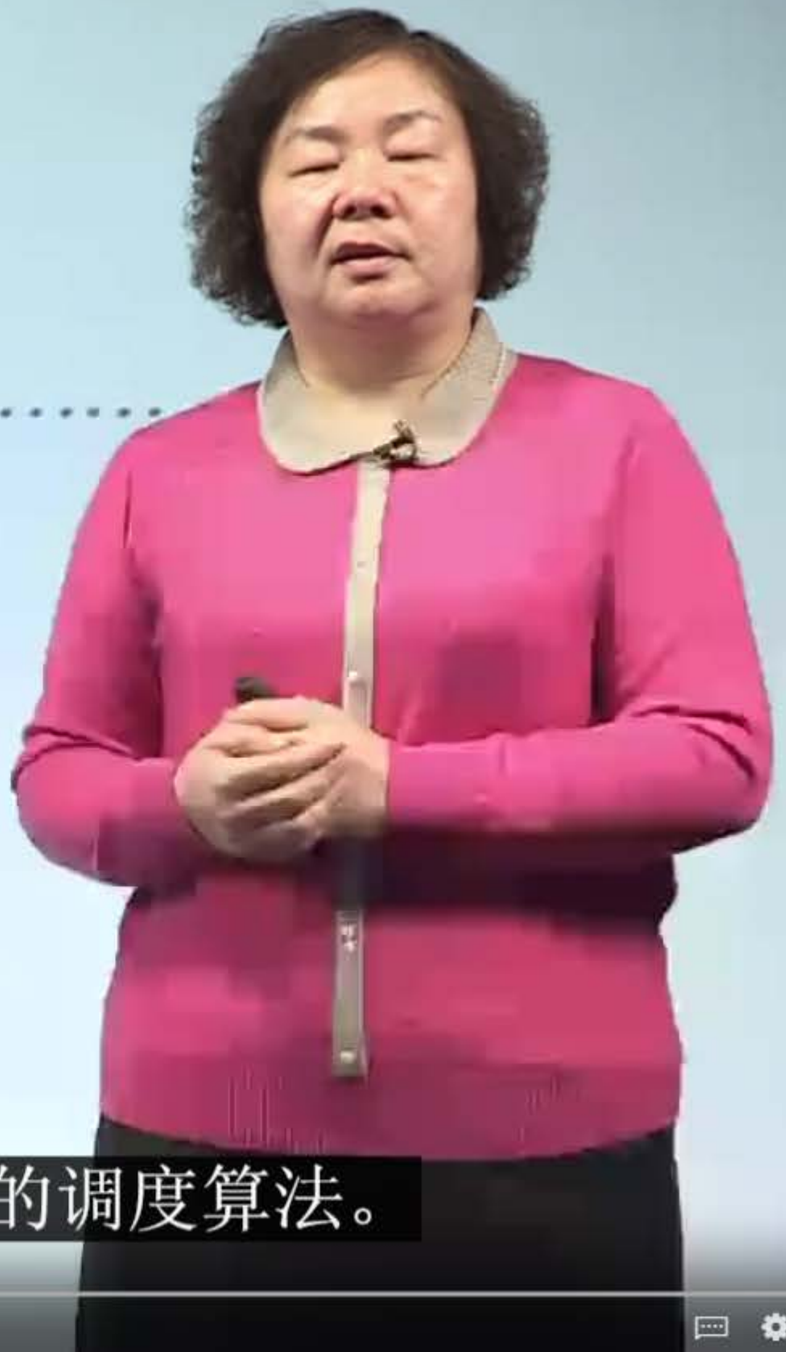
先来先服务、短作业优先、……

批处理系统的调度算法



下面呢我们来介绍批处理系统中的常用的调度算法。

批处理系统中采用的调度算法

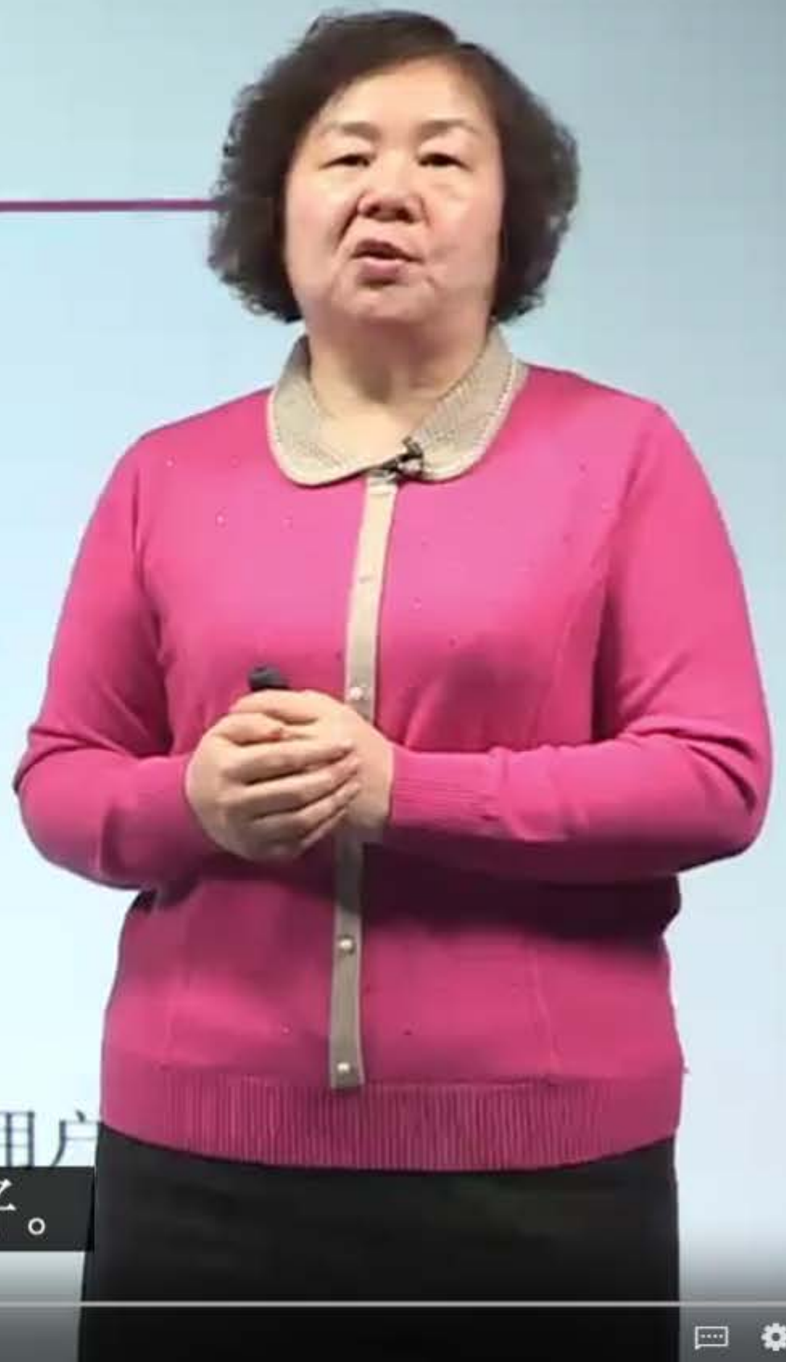
- 先来先服务（FCFS-First Come First Serve）
- 最短作业优先（SJF-Shortest Job First）
- 最短剩余时间优先
  （SRTN-Shortest Remaining Time Next）
- 最高相应比优先
  （HRRN-Highest Response Ratio Next）

当然也要考虑到公平和平衡的问题。

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

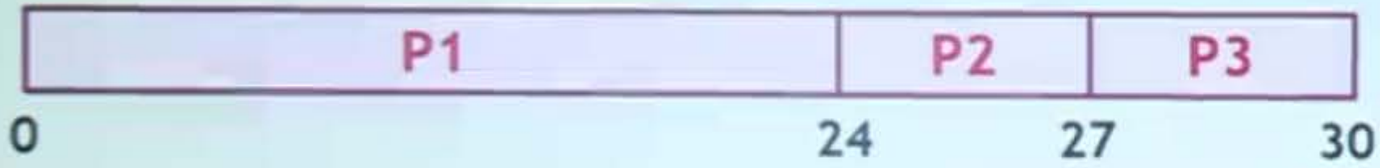


这样呢使得用户体验不够友好。

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 jobs/ 30s = 0.1 jobs/s

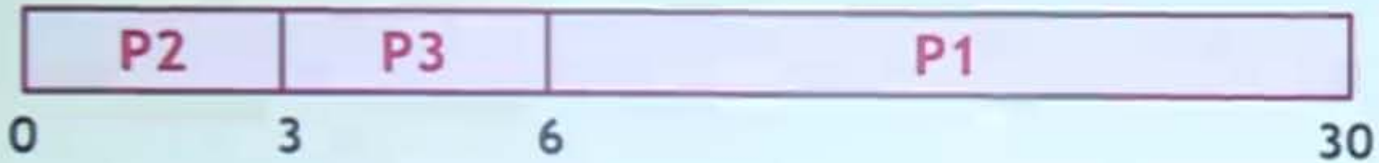
周转时间TT：P1:24；P2:27；P3:30

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 jobs/ 30s = 0.1 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

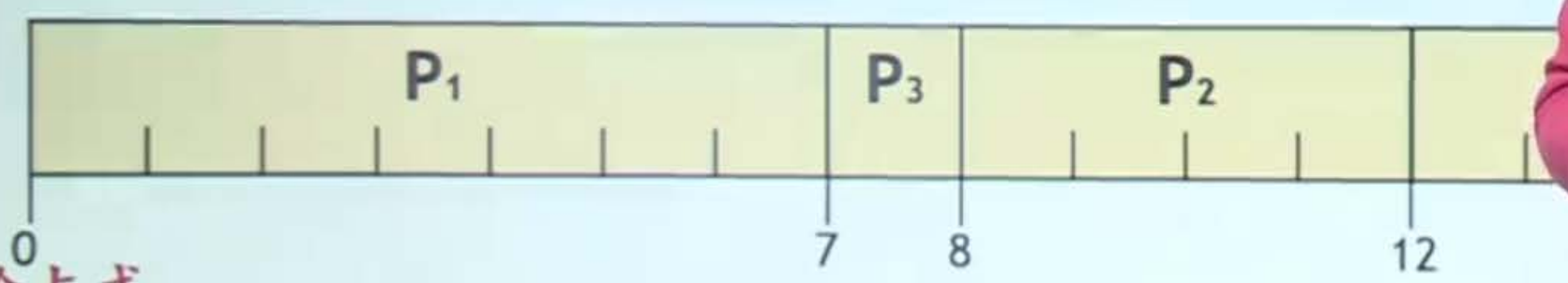
  Shortest Remaining Time Next(SRTN)
- **SJF**抢占式版本，即当一个新就绪的进程比当前运
  进程具有更短的完成时间时，系统抢占当前进程，
  选择新就绪的进程执行

思路：先完成短的作业
改善短作业的周转时间

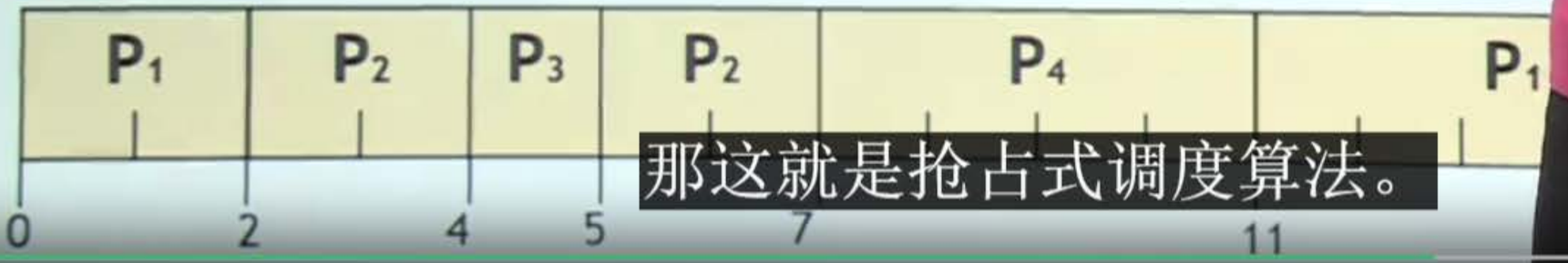
短作业优先SJF (2/3)

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



那这就是抢占式调度算法。

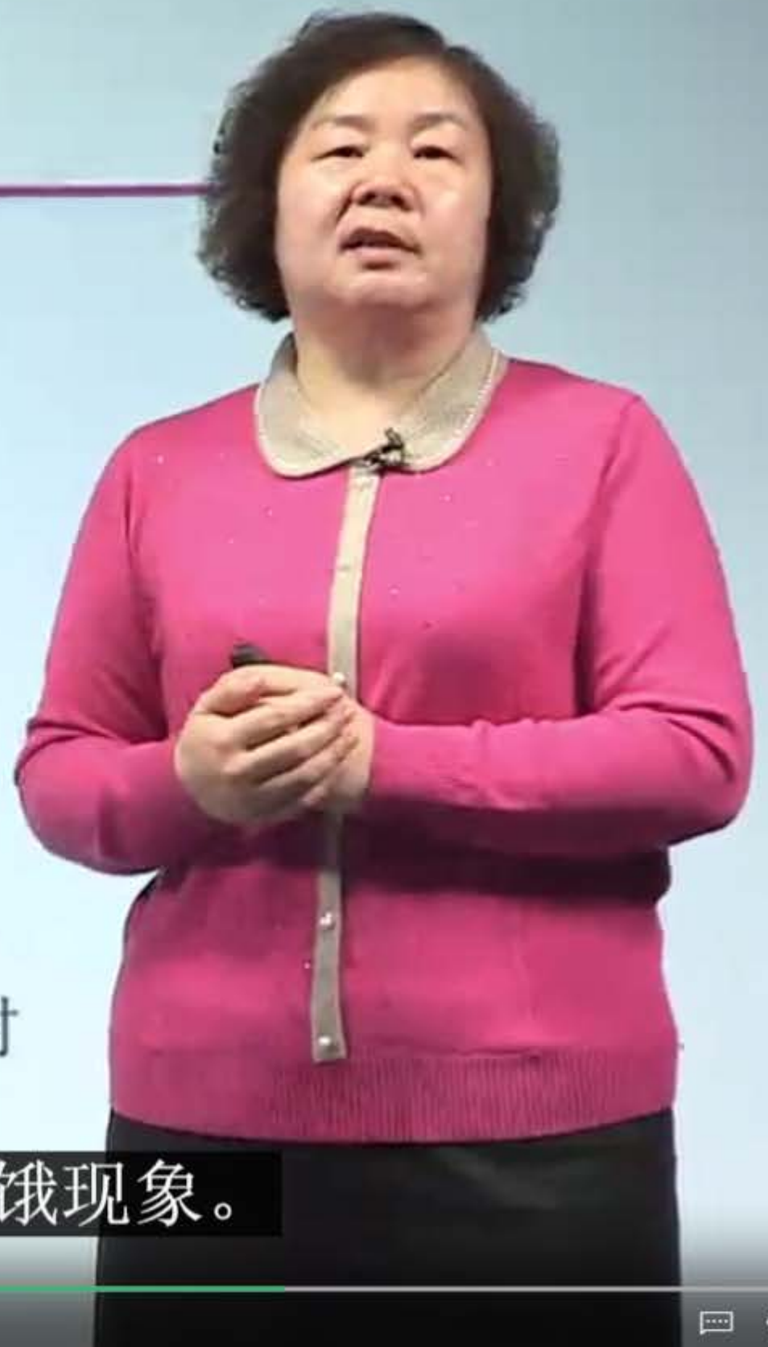
短作业优先调度算法(3/3)

- 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)



带来一种饥饿感，这就是所谓的饥饿现象。

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

折衷权衡
tradeoff

响应比R = 周转时间 / 处理时间
= （处理时间 + 等待时间） / 处理时间
= 1 + （等待时间 / 处理时间）

0:00 下面呢我们来介绍批处理系统中的常用的调度算法。 通常包括了先来先服务、 最短作业优先、 最短剩余时间优先，还有最高响应比优先这样几个典型的算法。 在批处理操作系统当中，对于算法的要求呢往往是 看它的带来的吞吐量是不是大， 周转时间和 CPU 利用率这几个因素。 当然也要考虑到公平和平衡的问题。 先来先服务调度算法也称为先进先出。 它指的是按照进程就绪的先后顺序 来使用 CPU。 先进先出调用算法是一个非抢占式的调度算法。 它的优点呢是公平， 谁先就绪谁先上 CPU。 实现起来也非常简单，维护一个队列就可以了。 那么选择进程的时候呢从队首选择， 那么新就绪的进程呢 排在队尾。 但是先来先服务调度算法呢也有一些缺点。 比如说如果有一个短的进程 排在了一个长时间运行进程的后面， 也就是说长进程后面的短进程它需要等待的时间比较长， 这样呢使得用户体验不够友好。 我们来举一个例子。 假设有三个进程按照 P1、 P2、 P3 就绪了， 那么 P1 进程它需要运行 24 秒， P2 和 P3 呢分别需要 3 秒计算。 如果我们采用先来先服务的调度算法， 那么怎么样去来调度呢？首先先运行 P1，运行了 24 秒，接着运行 P2，再接着运行 P3。 花了 30 秒三个进程运行完了。 我们来看一下有关的指标。 吞吐量 0.1。 周转时间，那么 P1 的周转时间呢是 24 秒； P2 呢由于要等 P1 运行完才能运行，所以它要 先等24 秒，再运行 3 秒，所以它的周转时间呢是 27 秒； P3 呢是 30 秒。 所以平均下来 平均的周转时间是 27 秒。 那我们可以看到 P2 运行时间很短，但是呢它要等 P1 很长的运行完之后才能去运行。 那么能不能通过改变这个 调度的顺序来减少我们的周转时间呢？ 我们来看一个讨论。 同样还是这个例子， 如果我们改变了调度的顺序，那么先运行 P2、 P3，再运行 P1。 那么 P2 运行 3 秒，接着是 P3 运行 3 秒， 然后呢 是 P1 运行它的 24 秒。 同样是 30 秒，三个进程运行完， 我们来看一下相关的指标。 我们看到 P1 的周转时间是 30 秒，P2 就是 3 秒，P3 是 6 秒。 所以平均下来平均的周转时间降低到了 13 秒。 所以我们可以看到改变调度的顺序，也就是调度算法的改变 会改善平均周转时间， 进而可能改善了响应时间这些指标。 于是我们得到了一个新的调度算法： 短作业优先调度算法。 所谓短作业优先指的是具有最短完成时间的 进程优先执行。 它也是一个非抢占式的。 那么如果 我把短作业优先调度算法再改进一下， 加了抢占，变成了一个抢占式的版本呢？ 那就是最短剩余时间优先调度算法。 所谓最短剩余时间呢，就是 如果当一个新就绪的进程 比正在当前运行的进程它的 运行时间，剩余的运行时间短的时候， 系统就会去抢占当前的进程，然后呢把 CPU 呢交给这个 运行时间更短的进程。 因为在一个短

作业运行的过程中，可能有一个新的进程 创建出来，或者是一个进程从等待变成就绪，那么它 剩余的时间比当前正在运行的这个进程剩余的这个运行时间还短， 就要抢占了。 那么不管是 短作业优先还最短剩余时间优先这样的调度算法， 它的主要思想就是首先先把短的作业完成， 这样就可以改善短作业的一个周转时间。 我们来看一个例子。 四个进程它们的到达时刻不是同时到达的，有不同的时刻， 那么运行的时间呢是 这么多。 我们来看看如果采用的是非抢占式的短作业优先调度算法， 它的运行是什么样的呢？零时刻那么 P1 进程到达， 那么它被调度上 CPU ，因为没有其他的进程存在。 在它运行的这个 7 个单位， 这个我们一般就是 7 个单位吧，运行的 7 个单位期间， 那么剩下三个进程都已经到位了。 接着当 P1 运行完成， 那么系统就要在这三个进程中去挑选， 哪个进程运行的时间短，哪个进程上 CPU。 那我们看 P3 运行一个单位，所以呢 P3 上 CPU。 然后在 P2 和 P4 中去选，那它俩是一样，那么 主要是先选谁先来的，谁排在前面就选谁，因此呢 P2 上 CPU。 然后呢是 P4 上 CPU。 那么这是这种情况，就是非抢占式的 短作业优先调度算法。 如果我们用一个 抢占式的最短剩余时间优先调度算法呢？我们来看一下它的调度的顺序是什么样的。 P1 上 CPU， 在它运行完 2 个单位之后，那么 P2 到达。那 P2 呢需要运行 4 个单位， 而 P1 呢还剩余 5 个单位，所以呢 换进程了，P2 上 CPU。 当 P2 也运行完 2 个单位之后， 这个时候 P3 到达了，再去重新调度， P3 还剩下 1 个单位，P2 剩 2 个单位， P1 剩 5 个单位，所以呢 P3 上 CPU 运行。 所以 最短剩余时间呢那么选择的是 P3 上 CPU 运行。 它运行完一个单位之后，这个时候系统中有 P1、 P2 和 P4， P4 呢需要 4 个单位，P2 呢还剩下 2 个单位， 而 P1 呢还剩下 5 个单位，所以呢又选择了 P2。 然后 P4 ，最后 P1 接着执行之后的 5 个单位。 那这就是抢占式调度算法。我们来看一下 短作业优先调度算法它的优缺点。 首先， 短作业优先调度算法可以得到最短的平均周转时间。 当然这是有前提的，在所有进程同时可运行时， 采用短作业优先调度算法可以得到 最短的平均周转时间。 那如果不满足这个条件， 可能的这个平均周转时间不是最短，当然也是比较短的。 那么短作业优先调度算法总是选择 运行时间最短的那个作业，那就会有一些不公平的现象发现。 那么当新的 源源不断的短任务到来，就会使得一些长的 任务长时间地推后执行， 得不到执行，那么就会产生一个叫做 "饥饿"的 现象。 这比较形象，因为长的作业、 长的任务总也得不到 CPU 的一个资源，所以一直拖延、 拖

延， 带来一种饥饿感，这就是所谓的饥饿现象。 那么刚才我们谈到了先来先服务调度算法 它的优点和缺点，也总结了短作业优先调度算法的优缺点， 那么我们能不能既具有先来先服务的优点， 又具有短作业优点的这样的调度算法呢？ 实际上这也是操作系统当中的一个常用的手段，就是折衷权衡。 也就是既保持先来先服务的优点， 克服其缺点，又能够保持 短作业优先的优点，克服其缺点，这样一个调度算法。 这个调度算法呢其中一个呢叫最高响应比优先。 它是一个综合的算法。 它的基本思想是这样的： 在调度的时候呢，首先呢先计算每个进程的一个数值响应比，一个参数， 响应比一个参数，也可以看成是一个权值。 那么计算完之后，在 所有的进程当中选择 这个响应比最高的进程，让它来执行。 那么到底怎么去计算呢？计算公式是什么呢？那么可以有不同的设计。 我们给出一种方案。 那么响应比呢 是周转时间除以处理时间来计算出响应比。 周转时间呢由两部分组成，一部分呢是它的处理时间，一部分呢是等待时间。 所以呢我们把它代进来，代进公式， 在经过化简之后呢我们得到了一个响应比的计算公式，就是 1 +（等待时间／处理时间）。 好，那么这个公式我们很明显地看到，如果 处理时间短，也就是短作业，那么分母小， 那么整个的值就大，因此呢响应比就大。 如果你在很早就就绪了，在就绪 队列中等的时间非常长，那你的等待时间就变得越来越大， 那么直到最后你的响应比变得非常大，被调度上 CPU。 所以我们可以看到通过等待时间和调整，那么 那些长的等待的作业，长时间等待的 作业就会慢慢慢慢变得响应比越来越大，那么就有机会上 CPU。 如果你是短作业，那么一上来你的分母 很小，那么你的响应比就很高，你很快就有机会上 CPU。 所以呢每次动态地来计算就得到了一个很好的 结果，这是最高响应比优先调度算法。