操作系统随笔

《有关操作系统 CPU 调度算法的随笔》

基于吴国伟老师的课堂讲义

CPU 调度的概念是 CPU 对进程（线程）处理顺序的安排，这一安排的根本目的是使 CPU 能够在每一个时刻都在工作，并且这些工作可以公平地处理进程。

- 公平在这里的含义可以类比于人，社会主义社会的公平指的是按需分配。进程要求在某个时间被处理（因为它自己清楚自己如果不被处理会对整个操作系统或者用户产生影响），它就应该被 CPU 在某个时间 running，而不是 CPU 在运行一个和它相比并不是那么重要的进程。
- 高效的意思从 CPU 的角度就是说，它始终在运行一个在某一时间最为重要的进程。

所以，这就是调度。为什么需要调度、调度在何时发生、调度应该基于怎样的原则才能使得 CPU 实现公平高效运行、以及如何实现这种调度是我们要关心的问题（这里有一个思维缺陷是为什么我们要关心这些问题）。

进程并不是所有时间都需要被计算的。他可能在不被计算的时候做这些事：进入 I/O 队列等待；它的生命结束了而被删除；在这两种情形下，进程会自愿的（符合预期的）让出 CPU 资源，让其他进程使用。这种调度是非抢占式的。

进程有时候没有执行完但是 CPU 选择让另一个进程代替它来使用 CPU 资源。这可能是由于：它受到了一个更加优先的进程的中断而不得不为老大哥让路；还可能是它使用 fork 创建了一个子进程然后“慈爱的”将 CPU 资源禅让给子进程。这种调度是非自愿的（被强占的进程无法提前预知是否会出现抢占者），因而是抢占式的。

除了以上两点直接原因，还有更加根本的原因在于为了满足实现 CPU 高效性所必须的并发性。所谓并发就是串行，但是表面上看类似于并行，因为微观上的交替速度极快，以至于宏观上让人的感知出现了“并行”的假象。如果要实现这种高效（准确、快速）的交替，那么 CPU 调度是需要的。

但是我们发现抢占式有一个潜在的缺陷（需要情况极端但也不是不可能出现），那就是如果优先级高的进程连续出现，那么它会因总被强占而无法再次回到 running 状态。这一状态的概念我们称之为“饥饿”。而饥饿到达极端的状态我们称之为“饿死”。

该如何实现这种调度呢？这一问题的主体依然是 CPU。CPU 每次都只能从就绪队列中选择一个进程交给 CPU，使其占有 CPU 的资源。在 CPU 内部发生了新老交替的过程，这一过程有一个专用的英文单词 dispatch 来描述。Dispatch 是 CPU 一系列对进程执行状态进行更替

的操作统称，就好像上级在反复使唤一个下级跑东跑西一样。由于进程的执行状态是作为一个变量被存储在 PCB 中的，所以派遣的实现也就是对于 PCB 中进程一系列状态信息的转存、变化。这些行为有：切换上下文（PCB 中信息的交换）、从内核模式转换至用户模式、以及跳转到用户程序中的某一位置。对于 CPU 调度而言，派遣是指将等待队列中的 CPU 即将代替老进程的进程选择出来，并利用 PCB 进行上下文交换，使老进程的信息得以保留（如果需要保留的话）。调度选择算法以及 PCB 的上下文交换都有一定的时间开销，这两部份开销直接决定了派遣的时间。

最后的问题就变成了：如何挑选一个代替老进程，使得 CPU 能兼顾公平与高效的进程呢？我们需要首先明确选择的指标是什么。CPU 利用率是在一个时间周期内，CPU 空闲（发呆）的时间与 CPU 运行非空闲进程的时间之比。在 CPU 进程中还真有一个叫做系统空闲进程的进程在，正是这个进程占用的时间可以让我们推算出在一个时间周期内 CPU 的利用率。第二个指标是 CPU 的吞吐率，它是面向选择过程而言的，这个概念是单位时间内由就绪态转变为运行态的进程占所有就绪态进程的百分比，也就是从就绪队列的视角看待调度算法的效率究竟怎样。周转时间是指一个进程从创建到进入运行状态所需要的平均时间。这是从进程的视角对调度算法的服务情况做一个平均的估计。如果一个进程从创建到运行所需要的平均时间很长，我们认为调度算法在设计时有可能有所怠慢，CPU 的效率可能不是特别高。等待时间和周转时间的相似之处在于都是对于进程的追踪。等待时间是进程从进入就绪队列到运行的时间，这一时间长短的意义和周转时间是十分类似的，但是它兼顾了我们上面说到的需要调度的四种情况。最后的响应时间针对交互式系统，描述的是从用户发起请求（连接请求、访问请求等）到请求被应答（开始响应）的时间，而不是从发起请求到相应被输出的时间。

为了符合这些指标，我们设计出了对应的几种调度。其中最为朴素的是先到先服务调度。这种调度基于优先队列的 FIFO 原则。它有一个问题在于，进程的平均等待时间受到进程的进入顺序严重影响。之所以如此是因为如果一个很长的进程先进入，而一些相对它而言小的多的进程后进入，那么这些小的多的进程的等待时间就都至少是这个长进程等待时间。所以先到先服务调度的补强是最短优先调度。

最短优先调度首先估计出每一个进程可能消耗的时间，这种估计是基于进程类型和经验的粗略估计。然后以这些时间作为权重利用排序算法对进程进行排序，按时间小的优先调度的原则进行。事实证明，这的确是等待时间最小的优先调度算法。但是这无法解决饥饿问题，如果一直出现时间较短的进程，那么时间最长的进程依旧会被饿死。

如果我们在创建进程时就为它们标注好了优先级（依据实时性、时间等因素），然后按照优先级由低到高（Linux 是优先级数字越小代表优先级越高）进行排序。面对饥饿问题，优先级调度算法给出了自己的解决方式，它自动增加在队列中等待时间很长的那些进程的优先级，越饥饿的进程其优先级就越可能高。这是优先级动态分配的实例，类似于计算机组成原理中的中断屏蔽字技术。

最后还有一种非常“公平”的调度方式即轮转法调度。它的关键在于预先设置好一个时间片，这个时间片对于等待队列中的每一个进程都是相等的，该等待队列中的一个进程进入 CPU 以后时间片便开始倒计时，待倒计时结束时自动放弃 CPU 资源，做好派遣准备，被等待队列中的下一个进程取代，首先轮转每一个进程，然后再对第一轮中没有完成的进程进行第二轮的轮转。时间片的长度十分关键，如果选择过长会导致时间上毫无必要的浪费。如果选择

过段，则派遣过于频繁，派遣的时间开销过大，这都降低 CPU 的效率，虽然合乎了公平性。依据以往的经验，我们可以大约计算户时间片的合适长度，这种计算一定是简单的，因为过于复杂的计算算法同样会导致时间上的浪费。

第五种调度算法是多级优先队列。这种队列依旧无法解决饥饿问题，它像计算机网络中网络层路由器的转发队列一样按类创建队列，并且每个队列的调度算法依类型而异。队列之间存在优先级或者时间片，按照优先级对于队列进行轮转或者使用时间片对于队列进行轮转。但是一个队列中被处理的元素依旧是那些高优先级的进程，所以依旧无法在每个队列中解决掉饥饿的问题。