Linux操作系统处理机管理实现方式

Linux是一个多任务操作系统，支持程序的并发运行。

它因此必然会有一套的多进程的处理方式 ，Linux从2.5版本开始引入一种名为的调度器，后在2.6版本中将公平的调度概念引入了调度程序，又可称作公平的调度算法，将其代替之前的调度器。

Linux操作系统对于进程的调度采用了如下的几个原则，或者叫做策略。

处理机中的**I/O消耗型进程**和**处理器耗费型**的进程，他们一般都一直在运行，除非被抢占，其中I/O消耗型继承和处理器消耗型进程是指那些大部分时间都在等待I/O操作的进程，处理器耗费型的进程则是指把大多数时间用于执行代码的进程。当然为了保证交互式应用和桌面系统的性能，一般Linux更倾向于优先调度I/O消耗型进程。

Linux的处理机也是可以对进程配置优先级的，操作系统将优先级范围分为两种，其一是**固定优先级**，使用nice值：越大的nice值意味着更低的优先级，其二是使用**动态优先级**，根据系统当前的情况对各进程的优先级进行实时改变，优先级越高进程就越有可能优先被执行。除去给进程划分优先级之外，Linux操作系统还会为其划分适当的时间片，其中每个时间片的长度并不是固定的，而是根据当前所获得的实际CPU时间就和系统的负载密切相关。

Linux操作系统使用调度算法是通过以**模块化**的Linux**调度器**实现的，这样使得不同类型的进程按照自己的需要来选择不同的调度算法。

当然这样处理进程是存在很多问题的，其中nice值必须映射到处理器的绝对时间上去，这意味着同样是瓜分100ms时间片的两个同样优先级的进程，发生上下文切换的次数并不相同，可能会差别很大。优先级越低的进程分到的时间片单位越小，但是实际上它们往往是需要进行大量后台计算的，这样做并不合理。相对nice值也会引发新的问题，两个nice值不同但差值相同的进程，分到的时间片的大小是受到其nice值大小影响的：比如nice值18和19的两个进程分到的时间片是10ms和5ms,nice值为0和1的两个进程分到的却是100ms和95ms，这样分配也无法做到公平的调度各个进程。还有如果要进行nice值到时间片的映射，我们必须能够拥有一个可以测量的“绝对时间片”（这牵扯到定时器和节拍器的相关概念）。实际上，时间片是会随着定时器的节拍而改变的，同样的nice值最终映射到处理器时间时可能会存在差异。最后为了能够更快的唤醒进程，需要对新的要唤醒的进程提升优先级，但是这可能会打破“公平性”

因此为了解决上述问题Linux在2.6版本更新之后，引入了CFS（**公平调度算法**），CFS对时间片的分配方式进行了根本性的重新设计，摒弃了时间片，用处理器使用比重来代替它。

出发点：进程调度的效果应该如同系统具备一个理想的多任务处理器——我们可以给任何进程调度**无限小的时间周期**，所以在任何可测量范围内，可以给n个进程同样多的运行时间。

举个例子来区分Unix调度和CFS：有两个运行的优先级相同的进程，在Unix中可能是每个各执行5ms，执行期间完全占用处理器，但在“理想情况”下，应该是，能够在10ms内同时运行两个进程，每个占用处理器一半的能力。

CFS的做法是：在所有可运行进程的总数上计算出一个进程应该运行的时间，nice值不再作为时间片分配的标准，而是用于处理计算获得的处理器使用权重。

接下来我们考虑调度周期，理论上，调度周期越小，就越接近“完美调度”，但实际上这必然会带来严重的上下文切换消耗。在CFS中，为能够实现的最小调度周期设定了一个近似值目标，称为“目标延迟”，于此同时，为了避免不可接受的上下文切换消耗，为每个进程所能获得的时间片大小设置了一个底线——**最小粒度**（通常为1ms）。

在每个进程的平均运行时间大于最小粒度的情况下，CFS无疑是公平的，nice值用于计算一个进程在当前这个最小调度周期中所应获得的处理器时间占比，这样就算nice值不同，只要差值相同，总是能得到相同的时间片。我们假设一个最小调度周期为20ms，两个进程的nice值差值为5：

* 两进程的nice值分别为0和5，后者获得的时间片是前者的1/3，因此最终分别获得15ms和5ms
* 两进程的nice值分别为10和15，后者获得的时间片是前者的1/3，最终结果也是15ms和5ms

接下来介绍Linux的进程调度的实现，首先所有的调度器都必须对进程的运行时间记账，换句话说就是要知道当前调度周期内，进程还剩下多少个时间片可用（这将会是是否抢占的一个重要标准），其中在调度器中有这样一个变量vruntime，这里我们称之为虚拟时间，也就是v就是virtual的意思。该运行时间的计算是经过了所有可运行进程总数的标准化（简单说就是加权的）。它以ns为单位，与定时器节拍不再相关。

可以认为这是CFS为了能够实现理想多任务处理而不得不虚拟的一个新的时钟，具体地讲，一个进程的vruntime会随着运行时间的增加而增加，但这个增加的速度由它所占的权重来决定。

结果就是权重越高，增长越慢：所得到的调度时间也就越小 —— CFS用它来记录一个程序到底运行了多长时间以及还应该运行多久。

调度的核心部分是进程的选择，Linux系统选择进程，就是**选择具有最小vruntime的进程**作为下一个需要调度的进程。为了实现选择，当然要维护一个可运行的进程队列（教科书上常说的ready队列），CFS使用了**红黑树**来组织这个队列。

红黑树是一种非常著名的数据结构，但这里我们不讨论它的实现和诸多特性（过于复杂），我们记住：红黑树是一种**自平衡二叉树**，再简单一点，它是一种以树节点方式储存数据的结构，每个节点对应了一个键值，利用这个键值可以快速索引树上的数据，并且它可以按照一定的规则自动调整每个节点的位置，使得通过键值检索到对应节点的速度和整个树节点的规模呈指数比关系。

在进程选择的过程中这个进程可能处在睡眠状态，我们需要唤醒它，在Linux中进程的睡眠是指，进程将自己标记成休眠状态，然后从可执行红黑树中移除，放入等待队列，然后调用选择和执行一个其他进程。进程的唤醒是指，进程被设置为可执行状态，然后从等待队列移到可执行红黑树中去。

进程进入睡眠状态，可能是因为被其它进程抢占了当前处理机，在Linux操作系统中，抢占分为用户抢占和系统抢占，内核为每个进程设置了一个标志来表明是否需要重新执行一次调度，当某个进程应该被抢占时，会设置这个标志，当一个优先级高的进程进入可执行状态的时候，也会设置这个标志位，内核检查到此标志位就会调用重新进行调度。内核即将返回用户空间的时候，如果标志位被设置，会导致被调用，此时就发生了用户抢占。内核抢占是指：只要重新调度是安全的，内核就可以在任何时间抢占正在执行的任务，这个安全是指，只要没有持有锁，就可以进行抢占。

为了支持内核抢占，Linux做出了如下的变动：

* 为每个进程的引入了计数器，用于记录持有锁的数量，当它为0的时候就意味着这个进程是可以被抢占的。
* 从中断返回内核空间的时候，会检查和的值，如果被标记，并且为0，就意味着有一个更需要调度的进程需要被调度，而且当前情况是安全的，可以进行抢占，那么此时调度程序就会被调用。

除了响应中断后返回，还有一种情况会发生内核抢占，那就是内核中的进程由于阻塞等原因显式地调用来进行显式地内核抢占：当然，这个进程显式地调用调度进程，就意味着它明白自己是可以安全地被抢占的，因此我们不用任何额外的逻辑去检查安全性问题。

至于在进程调度中出现的死锁问题，基本上和操作系统课本上的方法别无二致，因此不再赘述。

以上即为Linux处理机处理进程的实现方式。

参考文献：

[1] 陈科.Linux内核分析及应用.北京：机械工业出版社[M], 2018-8-1.

[2] yz764127031.《Linux内核分析》笔记与课件整理. https://blog.csdn.net/yz764127031/article/details/71036935. 2017-04-30.

[3] Sandra Henry-Stocker. Using multitail on Linux[J]. Network World (Online),2019.

[4] 汤小丹，梁红兵，哲凤屏，汤子瀛.计算机操作系统第四版[M].西安:西安电子科技大学出版社, 2014-05.

[5] 刘振洪，吴敏凤．Linux操作系统实用教程．天津:天津科学技术出版社，2016：4-20

[6] Marky Lumin. Linux内核调度分析（进程调度）. https://cloud.tencent.com/developer/article/1027448, 2018-01-25.