Linux处理机管理方式研究

摘要：Linux是一套免费使用和自由传播的类Unix操作系统，是一个基于POSIX和Unix的多用户、多任务、支持多线程和多CPU的操作系统。自1991年发布至今在开源社区获得了巨大的用户量支持，也经历了多个版本的发展，是一个性能稳定的现代多用户网络操作系统。本文将阐述Linux处理机管理即进程调度与死锁的处理方式。

关键词： Linux内核；操作系统；调度器；算法发展；死锁处理

1 Linux简介

Linux是类Unix系统，其本质为内核即操作系统最核心的功能部分。一般常见的操作系统如Ubuntu、Fedora、Arch等都是基于Linux内核的。Linux相比较Windows、OS X等系统最显著的特征是其开源性。在Linux操作系统的诞生、发展成长过程中，主要依赖于以下五个重要支柱：UNIX操作系统、MINIX操作系统、GNU计划、POSIX标准与Internet网络。

2 处理机管理

处理机管理主要有进程的调度管理与算法以及死锁的预防与接触等。

2.1 进程控制

进程控制的主要任务是为作业创建进程，撤销已结束的进程，以及控制进程在运行过程中的状态转换。

2.2进程同步

进程同步的主要任务是对诸进程的运行进行调节。

2.3 进程通信

进程通信的任务是实现在相互合作进程之间的信息交换。

2.4 进程调度

进程调度分为作业调度和进程调度.作业调度的基本任务是从后备队列中按照一定的算法，选择出若干个作业，为它们分配必要的资源。进程调度的任务是从进程的就绪队列中，按照一定的算法选出一新进程，把处理机分配给它，并为它设置运行现场，是进程投入运行。

2.5 死锁处理[[1]](#endnote-1)

死锁即两个以上的进程互相都要求对方已经占有的资源导致无法继续运行下去的现象。产生原因：有进程推进顺序不当、同类资源分配不当、PV操作使用不当等。产生的必要条件为互斥条件、请求保持条件、不可剥夺条件、环路条件。对于死锁的处理主要有以下四种方式：一、预防：破坏死锁产生的必要条件之一。二、避免：银行家算法：分配资源后系统进入不安全的状态，则不予分配，若安全，则分配。三、检测：系统定时运行一个死锁检测程序，判断系统是否发生死锁。四、解除：资源剥夺法和撤销进程法。

3 Linux中进程管理调度与算法研究

3.1 进程分类

Linux把进程区分为实时进程和非实时进程，其中非实时进程进一步划分为交互式进程和批处理进程。

3.1.1 交互式进程

此类进程经常与用户进行交互, 因此需要花费很多时间等待键盘和鼠标操作. 当接受了用户的输入后, 进程必须很快被唤醒, 否则用户会感觉系统反应迟钝。如shell与图形化应用程序等。

3.1.2 批处理进程

此类进程不必与用户交互, 因此经常在后台运行. 因为这样的进程不必很快相应, 因此常受到调度程序的怠慢。如程序语言的编译程序，数据库搜索引擎等。

3.1.3 实时进程

这些进程由很强的调度需要, 这样的进程绝不会被低优先级的进程阻塞. 并且他们的响应时间要尽可能的短。如视频音频应用程序及传感器收集数据的程序等。

3.2不同进程采用不同调度策略

3.2.1 实时进程调度

对于实时进程，采用FIFO或者Round Robin的调度策略。对于普通进程，则需要区分交互式和批处理式的不同。传统Linux调度器提高交互式进程的优先级，使得它们能更快地被调度。而CFS和RSDL等新的调度器的核心思想是”完全公平”。这个设计理念不仅大大简化了调度器的代码复杂度，还对各种调度需求的提供了更完美的支持。注意Linux通过将进程和线程调度视为一个，同时包含二者。进程可以看为是单个线程，但是进程可以包含共享一定资源的多个线程。因此进程调度也包含了线程调度的功能.

3.2.2非实时进程调度

目前非实时进程的调度策略比较简单, 因为实时进程值只要求尽可能快的被响应, 基于优先级, 每个进程根据它重要程度的不同被赋予不同的优先级，调度器在每次调度时, 总选择优先级最高的进程开始执行. 低优先级不可能抢占高优先级, 因此FIFO或者Round Robin的调度策略即可满足实时进程调度的需求.

3.2.3 CFS调度

普通进程（实时进程）的调度策略就比较麻烦, 普通进程不能简单的只看优先级, 必须公平的占有CPU, 否则很容易出现进程饥饿, 这种情况下用户会感觉操作系统卡顿, 响应速度缓慢，因此在linux调度器的发展历程中经过了多次重大变动, linux总是希望寻找一个最接近于完美的调度策略来公平快速的调度进程。如今的Linux使用的调度器为CFS调度器。

1. CFS调度器

4.1 CFS基本原理

cfs定义了一种新调度模型，它给cfs运行队列中的每一个进程都设置一个虚拟时钟值（vruntime）。如果一个进程得以执行，随着执行时间的不断增长，其时钟值也将不断增大，没有得到执行的进程时钟值将保持不变。而调度器将会选择最小的时钟值的进程来执行。这就是所谓的“完全公平”。不同优先级的进程其时钟值增长速度不同，优先级高的进程时钟值增长得慢，所以它可能得到更多的运行机会。

* 1. CFS算法设计

CFS根据各个进程的权重分配进程运行时间。进程的运行时间计算公式为:分配给进程的运行时间 = 调度周期 \* 当前进程权重 / 所有进程权重总和 (公式3.1)

调度周期为将所有处于TASK\_RUNNING态进程都调度一遍的时间,在O(1)调度算法中就是运行队列中进程运行一遍的时间。所以进程权重与分配给进程的运行时间成正比。

vruntime的计算公式为： vruntime = 实际运行时间 \* NICE\_0\_LOAD/ 当前进程权重 (公式3.2)

如果分配给进程的运行时间等于实际运行的时间时，将推到出另一vruntime计算公式。把公式3.2中的分配给进程的运行时间 与公式3.1中实际运行时间替换，将得出以下结果：vruntime = (调度周期 \* 当前进程权重 / 所有进程权重总和) \* NICE\_0\_LOAD/ 当前进程权重 = 调度周期 \* NICE\_0\_LOAD/ 所有进程权重总和

* 1. 初步结论

当分配给进程的运行时间等于实际运行的时间时，虽然每个进程的权重不同，但是它们的 vruntime增长速度均相同，与权重无关。上文已述用vruntime来选择将要运行的进程，vruntime值较小表明它以前占用cpu的时间较短，受到了“不公平”对待，因此下一个运行进程就是它。如此一来既能公平选择进程，又能保证高优先级进程获得较多的运行时间。如果分配给进程的运行时间不等于实际运行的时间时：CFS的思想就是让每个调度实体的vruntime增加速度不同，权重越大的增加的越慢，这样高优先级进程就能获得更多的cpu执行时间，而vruntime值较小者也得到执行。每一个进程或者调度组都对应一个调度的实体，每一个进程都通过调度实体与CFS运行对列建立联系，每次进行CFS调度的时候都会在CFS运行对列红黑树中选择一个进程（vruntime值较小者）。cfs\_rq代表CFS运行对列，它可以找到对应的红黑树。进程task\_struct ，可以找到对应的调度实体。调度实体sched\_entity对应运行队列红黑树上的一个节点。

5 Linux中死锁处理

5.1 D状态死锁检测

进程长时间（系统默认配置120秒）处于TASK\_UNINTERRUPTIBLE 睡眠状态，这种状态下进程不响应异步信号。如：进程与外设硬件的交互（如read），通常使用这种状态来保证进程与设备的交互过程不被打断，否则设备可能处于不可控的状态。对于这种死锁的检测linux提供的是hungtask机制，主要内容集中在Hung\_task.c文件中。

5.2 R状态死锁检测

所谓R状态死锁：进程长时间（系统默认配置60秒）处于TASK\_RUNNING 状态垄断cpu而不发生切换，一般情况下是进程关抢占后长时候干活，有时候可能进程关抢占后处于死循环或者睡眠后，这样就造成系统异常。对于这种死锁的检测机制linux提供的机制是softlockup。主要集中在softlockup.c文件中。系统创建一个fifo的进程，此进程周期性的清一下时间戳（per cpu），而系统的时钟中断中会被softlockup挂入一个钩子（softlockup\_tick）,这个钩子就是每个时钟中断到来的时候都检查是否每cpu的时间戳被touch了，若在阀值60秒内都没有被touch，系统就打印调试信息。

5.3 长时间关中断检测

长时间关中断检测可以有几种实现机制，而利用nmi watchdog来检查这种长时间关中断情况，是比较简单的。其原理是需要软硬件配合，硬件通常提供一个计数器（可以递增也可以递减），当记数到某个值得时候，系统就硬件复位。而nmi watchdog就定期（小于这个计数到达系统复位的时间）的去清理系统的计数，若是某个进程长时间关中断，则可能导致nmi watchdog得不到清，最终系统复位。

6 总结与展望

Linux自从发布以来就备受开源社区的喜爱，在全球开发者共同努力下不断发展完善，其处理机调度与死锁处理机制也在不断发展完善，到如今较完善的CFS调度器机制，已成长为现代网络操作系统，被越来越多的人在越来越多的行业里应用，随着处理器性能的提高，相信Linux一定会不断发展，发展出更加完善的处理机调度算法和死锁处理机制。

参考文献

[1]汤小丹，梁红兵，哲凤屏，汤子瀛.计算机操作系统[M].西安：西安电子科技大学出版社2014：5-4

[2] yc\_wj. 操作系统之处理机管理 [blog]. https://blog.csdn.net/yc\_wj/article/details/62037146

[3] JeanCheng. Linux进程调度策略的发展和演变--Linux进程的管理与调度(十六） [blog]. https://blog.csdn.net/gatieme/article/details/51701149

[4] Bystander\_J. 浅析Linux中完全公平调度——CFS [blog]. <https://blog.csdn.net/weixin_42092278/article/details/83959440>

[5] shuai\_wen. 浅谈linux的死锁检测 [blog]. <https://blog.csdn.net/u011279649/article/details/11523741>

1. [↑](#endnote-ref-1)