Linux CFS 调度算法分析

简岩 （贵州省遵义师范学院计算机与信息科学学院）

摘要：一个操作系统的核心是进程调度。那么操作系统最重要的程序之一则是进程调度程序，同时进程调度程序也是多任务操作系统执行频度最高的一部分，操作系统的整体性能也决定于进程调度程序的性能。本文剖析了从 O(1)算法到 CFS 算法的演变。最后用测试工具对 CFS 的稳定性和计算速度进行了分析。

关键词：Linux 调度　O(1)　CFS 调度器　红黑树

1 linux 进程调度的概述

所谓进程调度就是指操作系统正确的匹配 CPU 时间之后来准确的执行等待中的进程。怎样从若干个可运行的进程里面找到其中最优先的进程执行的同时又能保证响应时间短、吞吐量高是进程调度的核心所在。进程调度有何时启动调度器和调度器执行什么调度算法两部分。

进程调度的要求就是吞吐量大、响应时间快、周转时间短以及效率高。由进程的响应时间 Linux 内核可以把进程分为三类：实时进程、批处理进程和交互进程。根据这三类进程内核又产生了三种不同的调度方法：先进先出策略（SCHED_FIFO）、轮转策略（SCHED_RR）和适合交互分时的程序（SCHED_OTHER）。

2 进程调度算法

①当 CPU 运行进程时，调度是被禁止的；只有当 CPU 处于无进程运行时，可以进入调度。

②当准备队列空闲时，执行缺省的 idle-task、进程。

③当准备队列非空闲时，执行的进程需要调度器在准备队列中挑选出来。这时，goodness() 函数将会从调度器在准备队列中挑选出来的进程中计算其权值，只有权值最大才能有执行的优先级。

④如果经过 goodness() 函数计算之后每个进程的权值均是 0，则说明 CPU 所提供给实时进程的准备队列中进程的时间片全部用光了，需要进行重置之后返回步骤①，继续执行调度。

⑤当实时进程都执行完成之后，CPU 将对普通进程开始支持。当每一个普通进程的权值均是 0 时，则说明 CPU 所提供给普通进程的准备队列中进程的时间片全部用光了，需要进行重置之后返回步骤①，继续执行调度。

3 linux 从 O(1)调度器到 CFS 调度器

3.1 O(1)调度器

在 Linux 新版本 Linux2.6.22 中，其内核采用的是 O（1）调度器，其不仅仅能够支持 SMP 并且可以确保系统的负载和处理器的数目如何变化，其判断相对应的任务所匹配 CPU 所利用的时间是不变的。

有 2 种不同的任务是 O（1）调度器的分内工作：

①计算动态任务优先级。利用公式 dynamic　priority =max（100，min（static　priority - bonus+5，139））进行计算。

②当拥有最高优先级的进程执行过程中，选择出下一个需要进行执行的进程。CPU 利用调度器对所有进程进行了任务排队：expired 数组与 active 数组。其数组中的某一元素寄存着该任务队列某一优先级的指针，如果需要判断下一个执行的进程时，并不需要将所有的队列进行遍历，只需要在 active 数组排列好的队列中直接选择优先级最高的进程进行执行。上述方法的复杂度是 O（1）。

为了让交互式任务的响应速度变得更快，任何一次时钟中断里，处于执行的任务的时间片减一，如果时间片是

（上接第 235 页）

的 E-R 图如下：

6 系统安装与使用的具体步骤

本文仅以 Windows 2003 为例，详细说明高职院校学生信息管理系统开发与设计的具体步骤：

打开“控制面板”，双击“添加 / 删除程序”一项，然后点击“添加 / 删除 Windows 组件”一项，再选择程序安装，待安装结束后，双击“计算机管理”一项，再从“服务和应用程序”任务栏中找到“Internet 信息服务”一项，打开后再右击“默认 Web 站点”，选择“新建 / 虚拟目录”，并将其名称设定为“asp”接下来先选择一个具体的位置，再将文件夹也命名为 asp 并按照系统指示进行依次操作。在进行系统的主目录设置时，先将 ITS 服务器打开，右击 Web 站点，点击“属性”，选择系统目录，在选择文档标签时，选择 index.asp。设置完成后，还要安装 SQL Server 2000，还原原来数据库，再设置新的登录名和密码，规定其登录名为“cxj”，而将服务器角色规定为“System Administrators”。在一切完成之后，即可输入网址“http://localhost/，然后链接至 index.asp。

参考文献：

[1]胡华.浅谈高职信息管理专业课程设置的问题[J].科技创新导报，2010(33).

[2]李琳.高职学生信息管理系统设计与实现[J].电脑编程技巧与维护，2012(14).

[3]毕晓彬.基于 ASP.NET 的学生信息管理系统的设计与实现[J].科技信息，2011(14).

O 的时候,将对其类型进行判断,若是交互式任务,需要将时间片进行重置然后将 active 数组再次插入其中;若不是交互式任务,需要把 active 数组转到 expired 数组中,如此便可以让 CPU 优先被交互式任务所使用。当然,并不能够将进城长期的放在 active 数组里面,当 CPU 被交互式任务占用达到了一定的数值时,将会把任务转到 expired 数组中去。如果 active 数组处于空的状态时,则将两个数组进行互换,从而执行下一轮调度。

O (1) 调度器的优点已经显而易见了,与此同时其算法也存在一些不可避免的缺陷,如执行导交互式任务的时候反应速度并不理想。多任务队列和动态优先级是 O (1) 调度器所应用的相对繁琐的方法,这样就迫使了调度器较为繁琐以及对代码维护的时候难度非常之大。此外,在实际使用的时候发现,相对于在类似于服务器等不存在较大的交互性应用需求的时候,在桌面应用这种对交互性要求很高的环境下,O (1) 调度器的效果表现非常不理想。基于这种缺点,Ingo Molnar 研发了新的完全公平调度器(Completely Fair Scheduler,CFS),此款调度器与 O (1) 调度器的框架完全不同,在 Linux2.6.23 版本中,就将 CFS 作为默认调度器进行使用。

3.2 CFS 调度器

3.2.1 算法的主要思想

CFS 并没有采用以往的将进程进行排列分组以及进行动态的优先级分类,也没有采用睡眠时间的概念,同时也没有将任务分为交互任务和其他,CFS 则是引入了一个新的概念——红黑树,所谓红黑树就利用时间来计算一个键值从而来选择下一个执行进程,进而利用全部进程所对 CPU 所利用的时间状态来调度任务。全部的准备状态中的进程被赋予的键值数值被放在红黑树的叶子节点上,按照键值从小到大一次排列在从左到右。在任何一个调度点上,调度器会从红黑树排列好的进程从左至右的对 CPU 进行进程的调度,这种执行的复杂度为 O (log2N)。

在所有的时钟中断上面,其首要的任务是更新调度信息,其次是对红黑树上的排列好的进程进行进一步的调整。当检测到正在执行的进程并不是最左边的进程时,将对其进行 need_resched 标记,当执行中断返回的时候就要利用 scheduler_tick () 来完成对进程的切换,如果没有这个过程,CPU 将会被一直占用。

3.2.2 红黑树

CFS 依靠进程的虚拟运行时间作为其变量,进而使用红黑树把每一个准备好执行的进程排列成“runqueue”。CFS 将之前一直运用的 FIFO 以及 Hash 映射形式的线性“qunqueue”彻底移除,每一个 runqueue 都是利用红黑树来进行组成的。

红黑树的特点也是非常明显的:首先,红黑树上的每一个节点都能够保证一项规律,那就是该节点的左边节点永远小于该几点,相对的右边的节点就大于该节点,这就是之所以红黑树从左至右依次增大的效果;其次,红黑树上分布的所有节点都是均匀的,绝对不会出现不均匀的情况;最后,其操作代价非常低,插入、查找以及删除的代价都是 0 (logn)。在 CFS 实际的应用中,表现最为突出的还是第一条特点。

该算法中,每一个节点都作为 virtual_runtime 键值植入到红黑树里面,当有进程需要进行执行的时候,将键值进行更新之后植入到红黑树,之后从左至右的一次进行执行进程。

当 CFS 进行调度的过程中,其复杂度是 O (logn),可是因为 CFS 具有实现代价低的特点,实际执行的过程中,反应速度反而快了很多。此外,Linux 中的 CFS 还赋予了很多可操作性的功能,像可以进行灵活配置的调整选项等,这样可以让用户在任何情况之下都能够得到最好的性能体验。

3.2.3 源代码分析

我们参考内核 2. 6. 24 源代码中的 sched. c 和 sched_fair. c 来分析:

Scheduler_tick () 函数是 Sched.c 中的一个函数,Scheduler_tick () 函数改变当前的时间值与 clock 值后 CPU 会刷新当前的负载情况,最后调用 sched_class 函数和 task_tick( ) 函数。

```
static void task_tick_fair (struct rq* rq,struct task_struct * curr)
{
struct cfs_rq* cfs_rq;
struct sched_entity* se = &curr - > se;
for_each_sched_entity(se) {
cfs_rq = cfs_rq_of(se);
entity_tick(cfs_rq,se);
}
}
```

task_tick_fair 函数的目的是使待调度任务执行 entity_tick()函数:

```
static void entity_tick ( struct cfs_rq * cfs_rq, struct sched _entity *
curr)
```

用函数 dequeue_entity( )与 enqueue_entity( )的主要目的是在红黑树里把任务删除在插入到红黑树里用来调整任务在红黑树里的位置。_pick_next_entity()函数是返回到 CFS 中红黑树的最左边的叶子节点,如果发现这个节点它不是当前的任务,那么就调用 _check_preempt_curr_fair ( ) 设置调度标志,当中断返回时就会调用 schedule_tick() 进行调度,替换当前任务。

参考文献:

[1]杜慧江.Linux 内核 2.6.24 的 CFS 调度器分析[J].计算机应用与软件,2010(2).

[2]冯宇.Linux 进程调度算法分析[J].计算机与现代化,2009(6).

[3]张永选.Linux 2.6 内核调度机制剖析与改进[J].计算机系统应用,2009(11).