

Linux 内核调度器算法研究与性能分析

杨 嘉, 王移芝

(北京交通大学 计算机科学与技术学院, 北京 100044)

摘 要: Linux 操作系统正在向嵌入式系统和高端服务器领域发展。提高调度器的调度性能, 支持实时应用以及支持多处理器并行性的研究工作显得非常重要。文中对 Linux 2.4.22 和 2.6.10 两个版本的内核调度器进行比较分析, 重点分析了两种调度器的调度算法、调度时机、优先权计算方法和时机以及调度性能。

关键词: Linux; 内核; 调度器; 性能

中图分类号: TP316.8

文献标识码: A

文章编号: 1005-3751(2006)03-0095-03

Research and Analysis of Linux Scheduler's Algorithm and Performance

YANG Jia, WANG Yi-zhi

(School of Computer Science and Technology, Beijing Jiaotong University, Beijing 100044, China)

Abstract: Linux OS is developing into the fields of embedded systems and high-end servers. So, the research of enhancing scheduler's performance, supporting real-time application and multi-processor parallelizability are important. In this paper, have compared the scheduler in Linux kernel v2.4.22 with that in linux kernel v2.6.10, and especially analyzed the scheduling algorithm and opportunity, the priority computing method and opportunity, and also the performance of the two schedulers.

Key words: Linux; kernel; scheduler; performance

0 引言

Linux 操作系统的发展非常迅速, 不论是在桌面系统和低端服务器领域, 还是在嵌入式系统和高端计算领域, 它都是其他操作系统的强劲竞争对手。Linux 最初实现在 Intel 80×86 系列 CPU 的计算机上, 是分时操作系统, 系统目标是保持较好的响应时间和较高的吞吐量, 因此 Linux 内核优先考虑交互式进程和批处理进程, 对实时应用和多处理机的并行工作支持不够。

根据操作系统的概念, 进程是具有独立功能的程序在某个数据集上的运行过程。进程是资源分配的独立单位, 线程是进程执行的单位。在 Linux 中, 把进程称为任务 task 或线程 thread。多个线程是多个执行流, 它们具有相同的进程号 PID, 可共享同一内存地址空间, 同一打开文件集, 但是它们由内核独立调度^[1]。

进程调度是指什么时候以怎样的方式选择新进程运行以及在何时进行进程切换。实质是资源分配并保证系统有较短的相应时间和较高的吞吐量。调度算法的核心是在进程的就绪队列中确定最佳候选者^[2]。

1 Linux 2.4.22 和 2.6.10 内核调度器的比较分析

1.1 调度器涉及的关键数据结构和变量

Linux 操作系统内核中的进程、轻量级进程或线程的进程控制块 PCB 均用结构 task_struct 来表示。系统中的所有进程被组织成以 init_task 为表头的双向链表, 全系统唯一^[3]。

Linux 2.4 的 task_struct 中与进程优先级相关的变量有: 用户可支配优先数 nice, 在 -20 到 +19 之间取值; 当前时间片内进程剩余时间 counter; 实时进程优先数 rt_priority。这 3 个变量都将参与就绪进程选择算法。Linux 2.6 重新设计与进程优先级相关的变量: 动态优先数 prio 从 0~139 之间取值, 0~99 属实时进程范围, 100~139 属普通进程范围; 静态优先数 static_prio 决定了进程初始时间片大小, 但不参与优先级计算; 进程交互程度变量 interactive_credit 判断一个进程是否是交互进程; 进程平均睡眠时间变量 sleep_avg 是优先级计算的决定因素, 值越大优先级越高, 即系统优先考虑睡眠时间较长的进程。

Linux 2.4 内核的系统中所有就绪进程被组织成以 runqueue_head^[4] 为头指针的双向链表, 全系统唯一, 调度器将从中选择最合适的进程投入运行。Linux 2.6 内核设计了全新的就绪进程队列数据结构。其中最关键的数据结构是 prio_array_t^[5], 每个 CPU 维护一个就绪队列, 每

收稿日期: 2005-06-13

作者简介: 杨 嘉 (1981—), 男, 山西运城人, 硕士研究生, 研究方向为计算机网络与数据库; 王移芝, 教授, 硕士生导师, 研究方向为计算机网络与数据库。

个就绪队列按时间片是否用完分为 active 队列(时间片尚未用完的就绪队列)和 expired 队列(时间片已经用完的就绪队列)。prio_array_t 的数据结构如下:

```
struct prio_array{
    int nr_active; //进程数
    struct list_head queue[MAX_PRIO]; /* 以优先数为下标的数组,共 140 个元素,每个数组元素是具有相同优先数的进程的双向链表头指针 */
    unsigned long bitmap[BITMAP_SIZE]; /* 以优先数为索引的 HASH 表,某优先数双向链表为空,则对应位为 0 */
}prio_array_t;
```

1.2 调度器的工作流程和工作时机

1.2.1 调度器的工作流程

(1)Linux2.4 的调度器工作流程:

- a. 清理当前运行进程;
- b. 设置调度下去的进程的状态;
- c. 计算就绪进程的权值,从中选择具有最大权值的进程为将要投入运行的进程;
- d. 设置新进程的运行环境;
- e. 进行进程切换;
- f. 为被调度下去的进程寻找新的 CPU。

(2)Linux2.6 的调度器工作流程:

- a. 清理当前运行进程;
- b. 修改调度下去的进程的运行时间;
- c. 设置调度下去的进程的状态;
- d. 如果当前 CPU 的就绪队列没有进程则从 SMP 系统其他 CPU 上“拉”进程过来;如果就绪队列有进程但是 SMP 系统有 CPU 正在运行实时进程,则当前 CPU 将运行 idle 进程保证实时进程被调度下来后有 CPU 可用;否则从有序的 active 就绪队列中找出优先级最高的第一个进程;

e. 计算选出进程的平均睡眠时间(sleep_avg),在计算优先数后插入有序的就绪进程队列;

- f. 设置新进程的运行环境;
- g. 进行进程切换;
- h. 后期整理。

1.2.2 调度器的工作时机

(1)Linux2.4.22 中调度器有两种工作时机:

- a. 主动式:进程因等待核心事件主动调用 schedule()^[4]将自己挂起,以方便其它进程使用 CPU。
- b. 被动式:在系统调用结束后由核心态返回用户态时,内核将检查当前运行进程的 need_resched 位。如果是 1,则调用 schedule()。

(2)Linux2.6.10 调度器的工作时机有以下几种:

- a. 核心应用主动调用调度器,比如核心态下的设备驱动等待事件;
- b. 从中断或系统调用返回;
- c. 进程主动放弃 CPU,将自己挂起;
- d. 允许抢占 CPU,preempt_enable()^[5]调用 preempt-

schedule()^[5],后者再调用调度器。

1.3 就绪进程选择算法

1.3.1 Linux2.4.22 就绪进程选择算法

调度器将遍历整个就绪队列,调用 goodness()^[4]计算每个就绪进程的权值,选择具有最大权值的进程投入运行。计算进程权值时将进程分为实时进程和非实时进程。

非实时进程的初始权值等于进程在当前时间片内剩余的运行时间;如果运行进程的 CPU 就是当前运行调度器的 CPU,则权值增加 15,表示调度器优先考虑不迁移 CPU 的进程,因为 CPU 的 Cache 中的信息还活跃(有效);如果进程没有内存工作区或工作区就是当前运行进程的工作区,则权值加 1,表示调度器优先考虑不切换内存的进程;进程的最终权值为当前权值加上 20 - nice。

实时进程的权值为其优先数加 1000,即如果遇到实时进程,调度器将把实时进程的优先级提到最高,保证实时进程能尽快得到响应。

1.3.2 Linux 2.6.10 就绪进程选择算法

优先数的计算由一个非常简单的 effect_prio()^[5]函数完成。

对于实时进程,effect_prio()将直接返回其优先数,从 1 到 99 之间取值。

对于非实时进程,其优先数决定于该进程的平均睡眠时间(sleep_avg)和静态优先级(static_prio)。计算公式为:prio = p - >static_prio - bonus^[5]。其中 bonus 是由进程 p 的平均睡眠时间计算出来的奖励值,sleep_avg 越大,bonus 越小,prio 值也越小,优先级越高.prio 计算完成后再将其值限制在 100 到 139 之间。

2.4 内核调度器的进程权值计算在选择候选进程时进行。在 2.6 内核中,这种集中计算的情况不复存在了。优先数计算在进程状态发生改变时进行,比如进程创建、休眠进程被唤醒、进程时间片耗尽、CPU 间负载平衡和使用系统调用 set_nice()^[5]主动修改进程优先数。

2 调度器性能分析

2.1 就绪队列的组织

Linux2.4.22 中就绪进程的 PCB 被组织成以 run_queue_head 为头指针的双向链表,这一全局的数据结构供整个系统使用。在对就绪队列操作时为了保证同步,访问它的进程需要持有自旋锁(同步的一种实现,当临界资源被上锁时,进程将反复执行一条紧凑的循环指令,直到自旋锁被释放^[1]),当系统中其它处理机上的进程也要访问就绪队列时必须等待锁的释放,这样一来,就绪队列成了内核性能的瓶颈。

Linux2.6.10 重新设计了就绪队列数据结构,每个 CPU 维护自己的就绪队列,这大大减小了进程对临界资源的竞争。

2.2 对交互式进程优先的改进

Linux 内核保证交互式进程的响应时间。2.4 内核的

调度器对交互式进程的支持并不精确,比如数据库管理系统和一些网络应用需要从硬盘和网络上收集很多数据,这样的交互式进程并不需要短的相应时间,但它们也是由调度算法推进的,这导致真正和用户交互的进程让用户觉得反应迟钝^[1]。对此,2.6内核做了以下改进。

首先,交互式进程的休眠次数多、时间长,而进程优先级主要决定于进程平均睡眠时间(sleep-avg),这样计算出来的进程优先级相应高一些。

其次,在进程的task_struct结构中引入了变量interactive-credit,表示进程的交互程度。如果进程存在内存工作区,进程不是从TASK_UNINTERRUPTIBLE状态(进程等待IO)被唤醒而且进程的睡眠时间超出一定限度,那么该进程的interactive-credit值将加1。此外进程的sleep-avg在调整之后大于内核给定的最大睡眠时间,上述变量值也会加1。如果被调度下来的进程的sleep-avg在修正后不大于0,且interactive-credit值在取值范围内,则其值要减1。一旦进程的interactive-credit值达到上限,该进程就被永久地认做交互式进程。

最后,内核将尽量保证交互式进程处于active就绪队列上以提高响应速度。但是前提是expired就绪队列中进程的等待时间没有超过某一给定的阈值,一旦超过,交互式进程也要转移到expired就绪队列上来。

2.3 对高负载系统的支持

在负载很高的情况下,系统中可能存在上百个进程,2.4内核的调度器一次性重新计算所有就绪进程权值的做法是相当低效的,权值计算时间复杂度为 $O(n)$, n 为就绪进程数目;而2.6中进程优先数计算以进程为单位进行,时间复杂度为 $O(1)$,而且计算机分散在多种情况下,避免了集中计算带来的巨大时间损耗^[1]。其次,交互式进程因为睡眠时间相对较长很少被执行,如果系统负载很重,这些进程执行的机会更小,响应时间较长,在用户看来系统反应慢。最后,在nice值(从-20到+19之间取值)为0时,2.4内核和2.6内核中进程的初始时间片大约为60ms和100ms^[1],这样的初始时间片对重负载系统来说显得太长了。

2.4 实时性能

2.4内核区分实时进程和非实时进程。实时进程的调度策略又分为SCHED-RR(时间轮转调度)和SCHED-FIFO(先入先出调度)。调度器选择下一个要投入运行的进程时,如遇到实时进程将其权值加到最大,这种做法保证了实时进程优先被选择。但是2.4内核不支持内核抢占,调度器的运行时机很特殊,当进程在核心态下运行时,没有办法打断它,除非主动让出CPU^[3]。比如低优先级进程正在核心态执行内核系统调用或中断服务,这一过程要花费掉几毫秒。在这段时间里,就绪态的实时进程就不得不等待,这对于要求短响应时间的实时应用是无法接受的。

2.6内核调度器运行时如果发现SMP系统中CPU

上有实时进程运行,则当前CPU上将运行idle进程,保证实时进程被调度下来后仍然有CPU可用。

2.6内核实现了内核抢占运行,没有锁保护的任何代码段的执行都能被中断^[6]。进程执行中不论是返回核心态还是用户态下,只要变量preempt-count的值为0,且进程的需要调度位need-resched为1,调度器就会运行,当前进程不论是核心线程还是用户进程,都会被高优先级进程抢占。2.6内核调度器的改进保证了实时进程的限时响应(软实时),但是离立即响应(硬实时)尚有距离。

此外,2.6内核调度器在用动态优先级判断一个进程是否是实时进程时,使用unlikely()告诉编译器这一判断并不是经常执行。从这一微小细节上以及Linux操作系统的现实表现上能够看出Linux内核发展到现在仍然着眼于桌面系统和低端服务器。支持实时应用并不是Linux的主要发展目标。

2.5 对多处理机并行运行的支持

Linux系统最初实现在Intel的80×86系列CPU的计算机上,主要针对单处理器(Unique Processor)系统。此后,Linux逐步发展到支持多处理器,主要支持SMP(对称多处理器)系统。使用2.4内核的Linux系统,多个处理器共享就绪队列,而临界资源共享必须有同步操作,这导致了多处理机并行运行瓶颈。另外,2.4的调度器在当前CPU的Cache中信息尚有效的情况下,为被调度下来的进程选择其他CPU,在进程数量很大时,进程在CPU之间的迁移过于频繁,这又是一个性能瓶颈。

2.6内核新设计了就绪队列的数据结构,每个CPU维护自己的就绪队列,不存在共享一个就绪队列带来的性能问题。调度器在发现当前CPU就绪队列没有进程时,将从其他CPU上“拉”进程过来。在每毫秒一次的时钟中断时,所有CPU都将执行rebalance_tick()^[5]函数,平衡CPU间的负载。在进程改变执行它的CPU时,当前CPU将执行migration_thread()^[5]函数为进程选择新的CPU。这些改进更好地支持了多处理器并行性。

3 结束语

文中在比较Linux2.4.22和Linux2.6.10调度器的基础上,详细分析了调度器的5个方面的性能问题。可以看出,Linux调度器主要面向交互式进程,2.6调度器设计开发重点在于使调度器更加简单精练以及更好地满足实时性和多处理机并行性。在桌面系统和低端服务器系统中,Linux操作系统是非常好的选择。如果Linux想在嵌入式和高端计算领域变得更有竞争力的话,必须要在增强实时性和多处理机并行性上下功夫。

参考文献:

- [1] BOVET D P, CESATI M. 深入理解linux内核(第2版)[M]. 陈莉君,等译. 北京:中国电力出版社,2004. 79-113,

(下转第100页)

真后得到的任务分配矩阵为 $R_{7 \times 7}$, 求得 $D_{\text{Min}} = 167$, 实验结果见表 1。

$$C_{7 \times 7} = \begin{bmatrix} 68 & 68 & 93 & 38 & 53 & 83 & 4 \\ 6 & 53 & 67 & 1 & 38 & 7 & 42 \\ 68 & 59 & 93 & 84 & 53 & 10 & 65 \\ 42 & 70 & 91 & 76 & 26 & 5 & 73 \\ 33 & 65 & 75 & 99 & 37 & 25 & 98 \\ 72 & 75 & 65 & 8 & 63 & 88 & 27 \\ 44 & 76 & 48 & 24 & 28 & 36 & 17 \end{bmatrix}$$

$$R_{7 \times 7} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

表 1 实验结果

| 实验次数 | D_{Min} | N_C |
|---------|------------------|-------|
| 1 | 167 | 7 |
| 2 | 167 | 5 |
| 3 | 167 | 3 |
| 4 | 167 | 2 |
| 5 | 167 | 2 |
| 平均值 3.8 | | |

实验 2^[5]: 将 10 个任务分配给 10 个节点 ($n = 10$), 其代价矩阵为 $C_{10 \times 10}$, 已知总的代价的最优解为 29。仿真时采用参数为: $N_{\text{CMax}} = 500$, $Q = 1$, $\alpha = 1$, $\beta = 5$, $\rho = 0.2$, 仿真后得到的任务分配矩阵为 $R_{10 \times 10}$, 求得 $D_{\text{Min}} = 29$, 实验结果见表 2。

$$C_{10 \times 10} = \begin{bmatrix} 3 & 3 & 10 & 9 & 5 & 2 & 11 & 2 & 11 & 5 \\ 6 & 2 & 7 & 11 & 4 & 10 & 4 & 4 & 5 & 4 \\ 9 & 7 & 9 & 10 & 4 & 4 & 5 & 5 & 4 & 5 \\ 8 & 6 & 7 & 8 & 8 & 8 & 10 & 6 & 3 & 9 \\ 7 & 2 & 8 & 6 & 10 & 9 & 6 & 6 & 11 & 10 \\ 5 & 11 & 3 & 6 & 10 & 3 & 6 & 7 & 2 & 10 \\ 4 & 11 & 11 & 5 & 9 & 11 & 7 & 9 & 10 & 11 \\ 11 & 10 & 5 & 4 & 11 & 4 & 7 & 8 & 7 & 3 \\ 11 & 5 & 5 & 3 & 2 & 5 & 7 & 10 & 7 & 3 \\ 10 & 4 & 5 & 2 & 11 & 6 & 11 & 7 & 8 & 2 \end{bmatrix}$$

$$R_{10 \times 10} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 4 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

表 2 实验结果

| 实验次数 | D_{Min} | N_C |
|---------|------------------|-------|
| 1 | 29 | 10 |
| 2 | 29 | 5 |
| 3 | 29 | 9 |
| 4 | 29 | 2 |
| 5 | 29 | 6 |
| 平均值 3.8 | | |

根据仿真实验结果可以看出:

- (1) 算法简练、易懂;
- (2) 具有并行算法的特点, 易于扩展, 适合网格环境动态变化的特点;
- (3) 收敛速度快, 能在较短的时间内求得最优解。

5 结 论

文中利用蚂蚁算法在求解 NP-Complete 问题方面的优势, 以及其可扩展的特点, 将其应用于解决异构、动态的网格环境中的任务分配问题, 通过仿真实验的验证, 取得了良好的效果。

参考文献:

- [1] 都志辉, 陈 渝, 刘 鹏. 网络计算[M]. 北京: 清华大学出版社, 2002.
- [2] Dorigo M, Gambardells L M. Ant Colonied for the Travelling Salesman Problem[J]. Biosystems, 1997, 43(2): 73-81.
- [3] 侯向丹. 蚂蚁算法扩展性及应用研究[J]. 河北工业大学学报, 2002(3): 15-16.
- [4] Wang Xiuhong, Wang Zheng'ou, Qiao Qingli. Solving assignment problems with chaotic neural network[J]. Journal of Systems Engineering, 2001, 16(2): 146-150.
- [5] Foulds L R. Combinatorial Optimization for Undergraduates [M]. Shen Minggang Translated. Shanghai: Shanghai Translation and Publishing Corporation, 1988.

(上接第 97 页)

- 168-196, 354-373.
- [2] 汤子瀛. 计算机操作系统[M]. 西安: 西安电子科技大学出版社, 2001. 26-101.
- [3] 毛德操, 胡希明. Linux 内核源代码情景分析[M]. 杭州: 浙江大学出版社, 2001. 263-414.

- [4] linux 内核源代码, v2. 4. 22 [EB/OL]. www.kernel.org, 2003-08-25.
- [5] linux 内核源代码, v2. 6. 10 [EB/OL]. www.kernel.org, 2004-12-24.
- [6] 赵 炯. linux 内核完全注释[M]. 北京: 机械工业出版社, 2004. 36-129.