第16卷 第9期
2006年9月

计算机技术与发展
COMPUTER TECHNOLOGY AND DEVELOPMENT

Vol.16 No.9
Sep. 2006

Linux 2.6 内核进程调度分析

李正平，徐 超，陈军宁，谭守标，代广珍
（安徽大学 电子科学与技术学院，安徽 合肥 230039）

摘 要：Linux操作系统是一种支持多任务、多用户和多处理器的现代通用操作系统。2.6内核的Linux支持O(1)级进程调度算法，支持可抢占内核，相比于2.4内核具有更好的实时性能。文中基于Linux 2.6.10内核源代码，分析了Linux 2.6内核的进程调度系统。并在详细介绍关键数据结构的基础上，阐述了进程调度算法的原理，并对实时进程的支持作了分析。

关键词：Linux操作系统；进程调度；实时进程
中图分类号：TP316 **文献标识码**：A **文章编号**：1673－629X(2006)09－0076－03

Analysis of Process Scheduler in Linux 2.6 Kernel

LI Zheng-ping, XU Chao, CHEN Jun-ning, TAN Shou-biao, DAI Guang-zhen
(School of Electronic Science & Technology, Anhui University, Hefei 230039, China)

Abstract: Linux is a multi－users, multitasking general purpose operating system, which also supports SMP architecture. Linux 2.6 kernel supports O(1) process schedule algorithm and preemptive kernel, which offers better real time performance than 2.4 series kernel. Based on the 2.6.10 kernel source code, analyzes the process scheduler in linux 2.6 kernel in details. First, introduce the key data structures; then, a detailed address about process schedule algorithm is presented; finally, analyze the support for real time process in details.

Key words: Linux operation system; process scheduler; real time process

0 引 言

Linux操作系统是一种能运行于多种平台、源代码免费公开、功能稳定强大、符合POSIX规范与Unix兼容的操作系统。它已经成功应用于巨型机、小型机、PC机直到嵌入式系统的广泛领域，成为Windows操作系统强有力的竞争对手。

Linux操作系统支持多任务、多用户、多处理器。进程调度是所有支持多任务操作系统的关键部分，Linux操作系统具有一个高效优雅的基于优先级的进程调度器。2003年正式发布的2.6系列内核，相对2.4系列内核有很大的改进，特别在进程管理方面，2.6内核增加了对可抢占内核的支持，改进了进程调度算法，支持O(1)级调度算法[1,2]。文中就Linux 2.6内核的进程调度算法进行分析研究。

1 就绪进程队列

在Linux 2.4内核中，就绪进程队列是一个全局数据结构，所有的处理器共享同一个队列。调度器对它的所有操作都会因全局自旋锁而导致系统各个处理机之间的等待，使得就绪队列成为一个明显的瓶颈[3~5]。2.6内核重新设计就绪进程队列为每个CPU的数据结构，每个处理器都维护一个自己的就绪队列，这样就避免了2.4内核中的SMP性能瓶颈。

每个CPU的就绪进程队列由一个struct runqueue结构描述，其中最关键的子结构是优先级就绪数组[1]。每个runqueue包含两个优先级就绪数组：active和expired数组。active指向时间片没用完、当前可被调度的就绪进程；expired指向时间片已用完的就绪进程。描述优先级就绪数组的数据结构是prio_array_t，定义为：

```
struct prio_array {
  int nr_active; /* number of tasks in the queues */
  unsigned long bitmap[BITMAP_SIZE]; /* priority bitmap */
  struct list_head queue[MAX_PRIO]; /* priority queues */
};
```

优先级就绪数组包含MAX_PRIO个队列queue，每个queue维护了一个就绪进程列表，这些进程具有相同的优先级。此外，prio_array还包含一个优先级位图(bitmap)，用于快速定位优先级最高的进程。bitmap所有位的初始状态都是0，当一个优先级为N的进程变为TASK_RUNNING时，bitmap中对应的位N置1。此后内核可以通过调用sched_find_first_bit()函数寻找第一个非空的就绪进程链表。prio_array还包含一个nr_active变量用于记录该优先级就绪数组中进程的数目。

Runqueue结构中另两个重要的成员变量是best_ex-

收稿日期：2005－12－19
作者简介：李正平(1979－)，男，安徽宣城人，讲师，博士，研究方向为操作系统原理、计算机应用。

pired_prio和expired_timestamp。前者记录expired就绪进程组中的最高优先级,后者用来表征expired中就绪进程的最长等待时间。

2 task_struct结构

Linux用task_struct结构[1,2,5]表示进程,2.6内核的task_struct结构相对于2.4内核有很大变化。该结构记录了进程的重要信息,与进程调度有关的信息包括:

(1) state。

进程状态由state成员变量表示。一个进程共有7种可能状态,分别是:TASK_RUNNING,TASK_INTERRUPTIBLE,TASK_UNINTERRUPTIBLE,TASK_STOPPED,TASK_TRACED,EXIT_ZOMBIE和EXIT_DEAD。

(2) timestamp。

进程发生调度事件的时间(单位是nanosecond)。

(3) prio,static_prio。

进程的优先级和静态优先级。prio表示进程的动态优先级,与2.4内核中goodness()的计算结果相当。在0~MAX_PRIO-1之间取值(MAX_PRIO定义为140),其中0~MAX_RT_PRIO-1(MAX_RT_PRIO定义为100)属于实时进程范围,MAX_RT_PRIO~MX_PRIO-1属于非实时进程。数值越大,表示进程优先级越小。static_prio表示进程的静态优先级,相当于2.4内核中的nice。一个进程的初始时间片的大小完全取决于它的静态优先级。

(4) sleep_avg。

进程的平均等待时间,在0到NS_MAX_SLEEP_AVG之间取值,初值为0,相当于进程等待时间与运行时间的差值。它是动态优先级计算的关键因子,sleep_avg越大,计算出来的进程优先级也越高。

(5) interactive_credit。

该变量表示进程的交互程度。在-CREDIT_LIMIT到CREDIT_LIMIT+1之间取值,初始值为0,而后根据不同的条件加1减1,一旦该值超过CREDIT_LIMIT,即表示该进程是交互进程。

(6) array。

指向当前CPU的active就绪进程队列。

(7) run_list。

进程通过这个list_head变量连接自己到prio_array数组中queue队列,这样相同优先级的进程连接成一个双向列表,表头为prio_array结构中的queue变量。

3 进程调度算法

2.4内核中进程时间片的计算是在所有就绪进程的时间片耗尽的时候进行的,这个计算过程的耗时取决于系统中就绪进程的数目,因此是O(n)级的过程[3-5]。Linux 2.6内核的调度器为每个CPU维护两个优先级就绪数组prio_array。其中active数组包含当前CPU就绪进程队列中所有时间片未用完的进程,expired数组则包含所有时间片耗尽的进程。当一个进程的时间片耗尽后,内核在将其放入expired数组前单独计算该进程的时间片。而当active数组为空时(即active数组中所有进程的时间片全部耗尽),通过简单的调换active和expired指针实现所有进程时间片的重算,该过程是O(1)量级的,与系统中的进程数目无关。

进程调度由schedule()函数实现。首先,schedule()利用下面的代码定位优先级最高的就绪进程:

```
prev = current;
array = rq->active;
idx = sched_find_first_bit(array->bitmap);
queue = array->queue + idx;
next = list_entry(queue->next, struct task_struct, run_list);
```

schedule()通过调用sched_find_first_bit()函数找到当前CPU就绪进程队列runqueue的active进程数组中第一个非空的就绪进程链表。这个链表中的进程具有最高的优先级,schedule()选择链表中的第一个进程作为调度器下一时刻将要运行的进程。如果prev(当前进程)和next(将要运行的进程)不是同一个进程,schedule()调用context_switch()将CPU切换到next进程运行。

与Windows等操作系统一样,Linux的进程调度也是基于优先级的。在2.4内核中,进程的动态优先级取决于进程的nice值和剩余时间片,而2.6内核的调度器基于进程的静态优先级和进程的交互程度计算进程的动态优先级,取消了剩余时间片对进程动态优先级的影响。

进程的交互程度由task_struct结构中的sleep_avg和interactive_credit变量表征。当一个进程从睡眠状态唤醒后,内核调用recalc_task_prio()函数增加它的sleep_avg,直到达到MAX_SLEEP_AVG;而当进程每运行一个时钟tick,sleep_avg会被减小直到零。recalc_task_prio()函数同时还调整interactive_credit的数值,一旦它达到CREDIT_LIMIT即认为该进程是交互进程。

动态优先级的计算由effect_prio()函数完成,它根据进程的static_prio和sleep_avg数值计算出进程的动态优先级,sleep_avg越大,计算出来的优先级就越高。与先前的内核不同,2.6内核中优先级的计算不再集中在调度器选择候选进程的时候进行,只要进程状态发生改变,内核就有可能计算并设置进程的动态优先级。如进程被抢断时,内核在scheduler_tick()函数中调用effect_prio()计算动态优先级。

内核对交互式进程赋予了更高的优先。内核通过TASK_INTERACTIVE()宏判断一个进程是否交互式进程,如果一个进程足够交互,则当它的时间片耗尽后,它将被重新插入active数组,而不像非交互进程那样被插入expired数组,于是这个进程将得到更多的运行机会。为了防止非交互进程被交互进程长时间地阻塞在expired数

组中,保证调度策略的公平性,内核在一个进程时间片耗尽时,通过 EXPIRED_STARVING()宏判断当前 CPU 的 runqueue 中是否有进程在 expired 队列中等待过长时间。如果有这样的进程,那么即使当前进程是交互进程也不再将其插入 active 数组,而是排空 active 数组,切换到 expired 数组运行。

在多处理器系统中,由于存在多个就绪进程队列 runqueue,可能存在多个队列之间负载的不平衡状况:其中一部分 runqueue 包含的进程远远大于其余 runqueue,这种情况将影响系统的整体性能。Linux 通过 load_balance()函数实现各就绪队列间的负载平衡。该函数在两种情况下被调用:schedule()在当前 runqueue 为空时调用或者在系统空闲时由定时器每 1ms 调用一次,而在系统忙时则由定时器每 200ms 调用一次。当 load_balance()由 schedule()调用时,由于当前 runqueue 是空的,它简单地寻找任意就绪进程将其加入当前队列;而当由定时器调用时,load_balance()处理的工作相当复杂:

a. 首先,load_balance()调用 find_busiest_queue()确定最繁忙的 runqueue,即包含最多就绪进程的队列。

b. 其次,load_balance()决定从最繁忙 runqueue 的哪个 prio_array 数组"拉"出进程。除非该 runqueue 的 expired 数组为空,否则 load_balance()总是选择 expired 数组。

c. Load_balance()然后从选定的 prio_array 数组中找到优先级最高的进程列表。接着分析列表中的每个进程以确定是否适合被"拉"出,如果适合,则调用 pull_task()将该进程从原 runqueue"拉"到当前 runqueue。重复此过程直到负载平衡。

4 实时进程

Linux 2.6 内核对实时进程[1]的支持相对于以前版本的内核有很大的加强。其引入的两项新特性(分别是 O(1)调度算法和内核抢占支持)有利地提高了系统的实时性能,这两点都保证实时进程能在可预计的时间内得到响应。

Linux 支持两种实时进程调度策略:SCHED_RR 和 SCHED_FIFO,而普通的非实时进程的调度策略是 SCHED_NORMAL[2,4,5]。SCHED_FIFO 实现一个简单的无时间片先进先出的调度算法。一旦一个 SCHED_FIFO 实时进程获得运行,它将一直运行下去,除非它由于某种原因被阻塞,或者它自己放弃 CPU。在这种情况下,只有高优先级的实时进程可以抢断它。具有相同优先级的 SCHED_FIFO 进程以轮转的方式运行,而所有低优先级的进程将永远不会得到运行机会直到高优先级的进程全部运行结束。

SCHED_RR 策略与 SCHED_FIFO 相似,差别在于每个 SCHED_RR 进程被分配一个预定的时间片,当一个进程的时间片耗尽后,它将被抢占,CPU 以轮转的方式运行具有相同优先级的 SCHED_RR 实时进程。与 SCHED_FIFO 进程相同,高优先级的 SCHED_RR 进程总是立刻抢占低优先级的进程,而一个低优先级的进程永远不会抢占高优先级进程,即使高优先级进程的时间片已经耗尽。

实时进程的优先级范围为 0~MAX_RT_PRIO-1,对于实时进程来说,内核并不计算它的动态优先级,实时进程的动态优先级由 setscheduler()函数设定,而且一旦设定就不再改变。进程运行的顺序只取决于设定的动态优先级,而静态优先级与非实时进程一样决定进程的时间片长短。

5 结束语

Linux 操作系统经过十余年的发展,已经成为当今最成功的操作系统之一。其最新 2.6 版本的内核实现了一个高效的 O(1)级调度器,相对于 2.4 版内核具有更好的实时性能、重负载下更高的 CPU 使用率以及交互作业更快的响应等优良特性。但 2.6 版内核的实时性能仍然不能满足硬实时系统的要求,可抢占内核也只限于对 CPU 的抢占,还不支持对内存等其他资源的抢占。所有这些都将是今后 Linux 发展过程中值得深入研究的课题。

参考文献:

[1] Love R. Linux Kernel Development(2nd ed)[M]. 北京:机械工业出版社,2005.

[2] 倪继利. Linux 内核分析及编程[M]. 北京:电子工业出版社,2005.

[3] Bovet D P, Cesati M. Understanding the Linux Kernel(2nd ed)[M]. 北京:中国电力出版社,2004.

[4] 李善平. Linux 内核 2.4 版源代码分析大全[M]. 北京:机械工业出版社,2001.

[5] 毛德操,胡希民. Linux 内核源代码情景分析[M]. 杭州:浙江大学出版社,2001.

(上接第 75 页)

[J].武汉测绘科技大学学报,1999,24(3):209-211.

[2] 王晓东. 算法设计与分析[M]. 北京:清华大学出版社,2004.

[3] 严蔚敏,吴伟民. 数据结构[M]. 北京:清华大学出版社,1997.

[4] 金炳尧. 最优化问题中的若干新技术[J]. 科技通报,2002(2):119-124.

[5] 米涅卡 E. 网络和图的最优化算法[M]. 李家滢,赵关旗译. 北京:中国铁道出版社,1984.

[6] Zhan F B. Three Fastest Shortest Path Algorithms on Real Road Networks[J]. Journal of Geographic Information and Decision Analysis, 1997, 1 (1): 69-82.