Linux 内核 2.6 进程调度分析与改进

杨 静,李 炜,万峰松,吴建国
(安徽大学 计算智能与信号处理重点实验室,安徽 合肥 230039)

摘 要:对 Linux 内核 2.6 进行了进程调度分析,阐述了 Linux 内核 2.6 提高实时性的各方面因素。同时针对 Linux 内核 2.6 三种基本的调度策略 SCHED_OTHER, SCHED_FIFO,SCHED_RR 存在调度实时性不强的问题,提出了四种改进调度实时性的调度算法:静态优先级的 RM 调度算法,动态优先级的 EDF,LSF 调度算法及一种混合的调度算法。这四种方法都在不同程度上提高了 Linux 内核 2.6 的实时性能。为了让 Linux 更好地应用到实时系统中去,今后应当研究更切实有效的调度算法来提高 Linux 实时性。

关键词:Linux 内核 2.6;进程;调度算法;实时性
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2009)07-0105-03

Analysis and Improvement of Linux Kernel 2.6 Process Scheduler

YANG Jing, LI Wei, WAN Feng-song, WU Jian-guo
(Key Lab. of Intelligent Computing & Signal Processing, Anhui University, Hefei 230039, China)

Abstract: Expresses the analysis of Linux kernel 2.6 process scheduler, elaborates the reasons of the improvement of Linux kernel 2.6's real-time performance. And point out three basic scheduling policies: SCHED_OTHER, SCHED_FIFO, SCHED_RR have the problem of real-time scheduler, then expresses four kinds of scheduling algorithm to improve this defect: RM algorithm based on static priority, EDF, LSF algorithm based on dynamic priority and a mixed algorithm. These four algorithms all improve Linux kernel2.6's real-time performance on different level. For letting Linux use better in real-time system, now should study more effective and useful scheduling algorithm to improve Linux real-time performance.

Key words: Linux kernel 2.6; process; scheduling algorithm; real-time

0 引 言

Linux 系统经过多年的发展日益成熟,并且由于它的源码开放性,Linux 系统在生活生产各个领域得到了广泛的应用。然而 Linux 系统本身只是一个通用的分时系统,面对现今实时系统的广泛应用和发展,Linux 在实时性方面存在较大的不足[1]。为了弥补实时性的不足,需要做的就是对 Linux 进行实时性改进。文中就是针对 Linux 内核 2.6 在实时性方面进行了进程调度分析和改进。

1 Linux2.6 内核进程调度分析与改进

Linux2.6 内核具备两大新的特色:$O(1)$ 调度算法和可抢占式内核。

1.1 进程调度分析

1.1.1 可运行队列 runqueue

Linux2.6 中每个 CPU 都拥有一个自己的可运行队列,且每一个可运行队列都有一个自旋锁,使得各个 CPU 都各自独立使用自己的可运行队列。可运行队列由结构 runqueue[2~8]表示如下:

```
struct runqueue
{
  ...
  prio_array_t * active, * expired, arrays[2];
  ...
};
```

可运行队列根据时间片是否被用完分为活动的 active 队列和过期的 expired 队列。分别存放那些时间片没用完,当前可被调度的就绪进程和时间片用完的进程。其中 prio_array_t * active 是指向活动优先级数组(active 队列)的指针, * expired 是指向过期优先级数组(expired 队列)的指针,array[2]是实际优先级

收稿日期:2008-11-12;**修回日期**:2009-02-12
基金项目:安徽省自然科学资助计划项目(2006KJ013A)
作者简介:杨 静(1983-),女,硕士研究生,研究方向为嵌入式软件技术;李 炜,副教授,硕导,研究方向为嵌入式系统和 CIMS 技术;吴建国,教授,博导,研究方向为中文信息处理及智能 CDA/EDA。

数组。这样的好处是，当 active 队列中的进程一旦用完了自己的时间片，就重新设置新的初始时间片，并转移到 expired 队列中；当 active 队列为空时，则表示当前所有进程的时间片都消耗完了，此时，active 队列和 expired 队列进行一次互换，重新开始下一轮的时间片消耗过程。

对于每一类的队列都用 struct prio_array[1] 表示：

```
struct prio_array
{
  int nr_active; /* 本进程组中的进程数 */
  struct list_head queue[MAX_PRIO]; /* 每个优先级的可运行进程队列，MAX_PRIO 默认值为 140，数值越大优先级越低 */
  unsigned long bitmap[BITMAP_SIZE];/* 优先级位图，BITMAP_SIZE 默认值为 5，每一位与 queque[i]对应 */
}
```

1.1.2 数据结构 task_struct

```
Struct task_struct[2~8]
{
  int prio,static_prio;
  /* prio 是动态优先级，取值范围 0～139，0～99 为实时进程，100～139 为非实时进程；static_prio 是静态优先级取值范围 0～139，主要用于进程时间片的计算和动态优先级 prio 的计算 */
  prio_array_t *array;/* 记录当前 CPU 的活动就绪队列 */
  Unsigned long 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。sleep_avg 既可用于评价该进程的交互程度，又可用于表示该进程需要运行的紧迫性 */
}
```

1.1.3 $O(1)$ 调度算法

$O(1)$ 调度算法[2~8]是 Linux2.6 内核中的一大特色，将进程的相关操作运行的时间复杂度从 2.4 内核的 $O(n)$ 降到了 $O(1)$。保证了调度时间确定性，提高了调度的实时性。具体表现在：

(1) 在 Linux2.4 内核中由于是利用调度器 schedule() 查找最佳可运行进程的，这种查找与当前就绪进程数有关，所以查找时间是 $O(n)$[9]，这无疑是对实时性的一个限制。而目前的 Linux2.6 中，可运行队列 array 中有一个优先级队列数组 queue[MAX_PRIO]，其中每个元素对应一个相同优先级的队列链表，具有相同优先级的就绪进程都放在同一优先级链表中。另外还有一个优先级位图数组 bitmap 与其对应。当 queue[i] 中有就绪进程时 bitmap 第 i 位就置 1，否则置 0。这样一来，可以快速地查找到最高优先级进程链表。而查找 bitmap 的时间只是个常数，把时间控制在了 $O(1)$ 量级。

(2) 在 Linux2.4 内核中，进程的优先级计算采用的是集中式计算，运算的时间复杂度与就绪进程数相关，达到了 $O(n)$，而在 Linux2.6 中，采用分散式来计算进程优先级，一但进程状态发生改变就重新计算优先级。这样时间复杂度降到 $O(1)$ 量级。

(3) Linux2.6 中，时间片的计算变成了在各个进程消耗完时间片的时候单独进行，并利用 active 队列和 expired 队列的简单对调完成了时间片的轮转，从而调度时间控制在了 $O(1)$ 量级。

1.1.4 内核抢占式调度

相对 2.4 内核，2.6 内核实现了可抢占式调度[2,6]，没有锁保护的任何代码段都有可能被中断，这使得一些后来的优先级高的实时进程能够得到及时响应的时间，增强了系统的实时性。

1.1.5 调度策略的缺陷

Linux2.6 内核中保持了三种调度策略[4,6]：一种是非实时进程调度策略 SCHED_OTHER，另外两种是实时进程调度策略 SCHED_FIFO，SCHED_RR。对于实时进程的这两种策略，前者是先进先出算法，后者是时间片轮转法。这两种都是基于静态优先级的调度算法，即实时进程始终是以固定的优先级进行调度的。然而在实际的实时系统当中，实时进程的优先级是随时间等各种因素不断变化的，调度器需要对实时进程的动态优先级进程计算，所以目前这两种静态的调度算法是不能满足实时应用的需要的。

1.2 进程调度算法的改进

就上述调度策略上的缺陷，在 Linux 内核当中加入新的改进的调度算法来改善存在的实时性缺陷。

1.2.1 RM 算法

RM 算法[10]，也叫单调速率调度算法，是一种基于静态优先级的调度算法。RM 算法给系统中每个任务分配一个根据任务运行周期来设定的静态优先级。遵循周期越短，截止期越紧，优先级越高。因此对于 RM 算法，优先级最高的任务就是周期最短的任务。当有多个任务可供执行时，周期最短的任务最先得到执行权。对于周期性独立的任务集，每个任务不超过它的截止期限，即 CPU 利用率满足下面的条件就能被 RM 算法调度。

$$U = \frac{C_1}{T_1} + \frac{C_2}{T_2} + \cdots + \frac{C_n}{T_n} \leqslant n(2^{\frac{1}{n}} - 1) \quad (n > 1)$$

其中 C_i 为每个 P_i 任务在每个周期的最大执行时间，T_i 为 P_i 任务的运行周期。C_i/T_i 表示 CPU 利用率。

1.2.2 EDF 算法

EDF 算法[10,11]，也叫最早截止期限优先算法，这是一种动态优先级的调度算法。算法将最高优先级分配给具有最早截止期的周期任务。当有多个任务可供

执行时,具有最早截止期的周期任务最先得到执行权。其中的截止期不再是一个不变值,而是一个随着任务执行而动态改变的时间段。因此,EDF 算法调度的任务的优先级也是随时间动态变化的。对于每一个可调度时刻,将根据任务的最早截止期限,动态改变任务的优先级。对于一个给定的任务集,当且仅当 CPU 利用率满足下面条件,可应用 EDF 调度:

$$\frac{C_1}{T_1}+\frac{C_2}{T_2}+\cdots+\frac{C_n}{T_n}\leqslant 1$$

其中 C_i 为每个 P_i 任务在每个周期的最大执行时间,T_i 为 P_i 任务的运行周期。C_i/T_i 表示 CPU 利用率。

1.2.3　LSF 算法

虽然动态优先级的 EDF 算法在一定程度上提高了进程调度的实时性能,可是它也存在着一定的缺陷。在实际的任务调度过程中,如果某些进程因为截止期晚优先级低一直处在等待状态,它很有可能就会出现错过截止期的情况,这样一来,这个进程就会因来不及处理而夭折。而 LSF 算法[12],就是对 EDF 算法这个缺陷的一个改进。

LSF 算法也叫最小裕度算法。任务根据裕度的大小决定优先级,裕度越小优先级越大。具有最小裕度的任务最先得到调度。裕度定义如下:

记 t 为当前系统时间,E 为任务估算执行时间,P 为任务实际执行时间,d 为截止期,则裕度

$$d_i = d-(t+E-P)$$

当 $d_i \geqslant 0$ 时,调度任务,否则夭折。LSF 算法中,正在运行的任务裕度是不变的,而在等待的就绪进程的裕度将随时间的推移而减小,这样就会使得等待进程的优先级动态地随裕度的变小而变高。在中断时,除了递减运行进程的执行时间,还要对就绪进程的裕度进行累减。

1.2.4　混合调度算法

对于上述说的几种调度算法,无论是动态的还是静态的,都有它的优缺点,因此可以想到将它们的优缺点相结合,采用一种基于权重的混合调度算法。这样一来能更有效地利用 CPU,具有更好的调度性能。

将静态调度算法 RM 算法和动态调度算法 LSF 算法按一定权重比结合,形成一个动静结合的调度算法。可将每个实时任务都设置两种优先级:一种是 RM 优先级,一种是 LSF 优先级。并为每个设定一个权重系数,每个系数与优先级相乘,之后相加得到一个真正的任务优先级,调度器会按照这个真正的优先级调度任务。优先级关系式如下:

RM 优先级 $*\ a$ + LSF 优先级 $*(1-a)$

其中 a 就是权重系数,系统可根据需要灵活调整 a 的比例,对 RM 和 LSF 算法进行不同侧重的使用。

上面所说到的这四种改进的算法,都在一定条件下不同程度上对 Linux 内核 2.6 的进程调度策略所缺乏的实效性进行了改善。更好地利用了 CPU,加强了 Linux 系统的实时性能。

2　结束语

Linux 越来越多地应用于实时系统,面对这样的需求,Linux 系统的关键性问题在于提高它的实时性,Linux 内核 2.6 相对于以前的内核在调度实时性上做出了改进。可是 Linux 内核 2.6 却仍然存在实时系统所不能满足的实时性能的问题。对此,提出一些新的静态或动态的调度算法来提高 Linux 实时性能势在必行。文中提出了四种改进调度算法来提高 Linux 实时性。今后研究的重点仍然是改进 Linux 调度实时性。研究出更好更有效的算法来提高 Linux 的实时性能,为其更好的应用于实时系统打下基础。

参考文献:

[1] Eikermann M. C - LAB Cooperative Computing & Communication Laboratory, Real - Time Linux[M]. [s.l.]: University of Paderhorn, 2001.

[2] Love R. Linux 内核设计与实现[M]. 北京:机械工业出版社,2006.

[3] Claudia, Rodriguez S, Fischer G, et al. Linux 内核编程[M]. 北京:机械工业出版社,2006.

[4] 李正平. Linux2.6 内核进程调度分析[J]. 计算机技术与发展,2006,16(9):76 - 78.

[5] 邹治峰,张曦煌. Linux2.6 进程调度[J]. 微计算机信息,2006,22(1 - 2):77 - 79.

[6] 张同光. Linux2.6 内核分析[J]. 长春工业大学学报,2006,27(4):333 - 337.

[7] 王　刚. Linux2.6 内核进程调度分析[J]. 计算机应用与软件,2007,24(9):162 - 164.

[8] 於时才. Linux2.6 调度系统的分析与改进[J]. 软件时空,2007,24(5 - 3):252 - 254.

[9] Bovet D P, Cesati M. Understanding the Linux kernel[M]. 北京:中国电力出版社,2004.

[10] 黄　健. 基于调度策略的 Linux 实时性改进[D]. 成都:电子科技大学,2007.

[11] Haritsa J R, Livny M, Carey M J. Earliest deadline scheduling forReal - time database systems[C] // In: Proceedings of the 12th IEEE Real - TimeSystems Symposium. Los Alamitos, CA: IEEE Computer society Press, 1991.

[12] 赖　娟, Linux 内核分析及实时性改造[D]. 成都:电子科技大学,2007.