

一种改进的强分区约束的实时调度算法

曹广旭¹, 蒋泽军¹, 王丽芳¹, 刘志强²

(1 西北工业大学 计算机学院, 陕西 西安 710129; 2 西北工业大学 软件与微电子学院, 陕西 西安 710072)

摘 要: 研究了实时操作系统在强分区约束下的两级分区的调度问题, 建立了基于时钟触发的两级分区调度算法的模型. 重点论述了调度开销和分区分配方法对系统可调度性的影响, 并进一步改进了算法. 仿真分析证实, 改进的算法能够更好地保障系统的可调度性.

关键词: 实时操作系统; 强分区约束; 时钟触发; 可调度性; 调度算法

中图分类号: TP316.86

文献标识码: A

文章编号: 1000-7180(2012)05-0108-05

Study of Real Time Scheduling Algorithm Under Strong Partition Constraints

CAO Guang-xu¹, JIANG Ze-jun¹, WANG Li-fang¹, LIU Zhi-qiang²

(1 School of Computer, Northwestern Polytechnical University, Xi'an 710129, China;

2 School of Software and Microelectronics, Northwestern Polytechnical University, Xi'an 710129, China)

Abstract: This paper studies two-level partition scheduling problem of strongly partitioned real time systems and establishes a two-level partition scheduling model based on tick triggering. Focusing on discussing the influence on system caused by expenses while scheduling and allocating method of partitions, we further improved the algorithm. The experiments' results show that the improved algorithm is able to provide a better schedulability of system.

Key words: real time system; strong partition constraints; tick triggering; schedulability; Scheduling Algorithm

1 引言

实时调度算法是保障实时系统强时限性和高可靠性的重要手段之一^[1]. 实时调度是指在有限的系统资源下, 为一系列任务决定何时以及在哪个处理器上运行, 并分配任务运行所需要的资源, 以保证其截止期限、顺序约束和资源约束都能得到满足^[2]. 在一个调度中, 若某任务集中的所有任务的实时约束条件都满足, 则称在此调度策略下, 该任务集是可调度的^[3]. 然而随着实时系统应用的不断复杂化、综合化, 传统意义上简单的调度越来越难以满足系统的实时性, 因此, 对多层次多分区模型下调度机制的研究具有很大的现实意义.

综合模块化的航空电子系统软件标准

ARINC653 标准中提出了一种两级分区的调策略, 这种调度模型带来了三个问题: (1) 如何在严格实时的约束条件下, 对分区任务的可调度性进行分析. (2) 给定分区任务, 分区的设计直接影响系统的实时特性. 如何合理地设计分区, 使分区中的任务都满足实时性要求, 又提供尽可能大的处理器利用率. (3) 当分区内调度算法的开销必须考虑时, 如何设计调度时机, 使各分区任务实际的可调度性得到保证.

2 基本概念

2.1 两级调度模型

系统层根据 ARINC653 标准, 采用轮转调度的方式激活每一个分区. 在分区内, 采用可预知性较强的单调速率算法. ARINC653 标准强调分区之间的

收稿日期: 2011-09-08; 修回日期: 2011-10-23

基金项目: 航空科学基金(2010ZD53, 2009ZD53044); 陕西省自然科学基金(2009JM8017, 2009JQ021, 2010JM8023); 西北工业大学种子基金(Z2011133)

相互独立性,即每一个分区内部的任务只能在当前分区处于激活状态才有可能被执行,因此必须通过一个足够精度的定时的任务只能在当前分区处于激活状态才有可能被执行,因此必须通过一个足够精度的定时器来实现分区时间片的轮转.时钟中断是一个周期性较高,并且可以周期大小可以调整(可降低至微秒级)的定时手段,通过时钟中断来实现定时触发分区调度是一个简单而有效的方法.两级调度模型如图1所示.

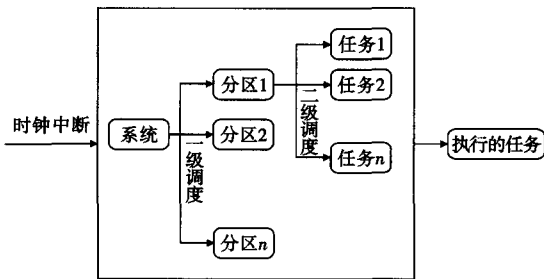


图1 两级调度模型

2.2 分区调度模型

假定分区 P^k 由 N 个任务组成,可表示为 Task_i^k ,表示第 k 个分区的第 i 个任务.在系统的第一级调度中,采用轮转调度激活各个分区,我们定义系统的轮转周期时长为 T_A .在每一个轮转周期内, T_k 表示分区 P^k 的执行时间,在其余 $T_A - T_k$ 时间段内,分区 P^k 处于阻塞状态.分区调度模型如图2所示.

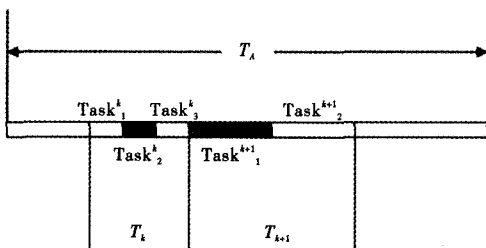


图2 分区调度模型

2.3 任务模型

假设有 n 个实时任务,分别用 $T_1, T_2, \dots, T_n$ 表示,它们组成一个任务集 T ,即 $T = \{T_1, T_2, \dots, T_n\}$.任务特征定义如下:

(1) 任务 T_i 的执行时间 C_i :表示第 i 个任务执行需要的时间.

(2) 任务 T_i 的周期 PT_i :表示第 i 个任务的任务周期.

(3) 任务 T 的截止期限等于任务的周期.

根据以上定义,任务由一个三元式表述: $T_i = \{R_i, PT_i, C_i\}$

任务 T_i 的利用率定义为: $U_i = C_i / T_i$

任务集 $T_1, T_2, \dots, T_n$ 的利用率为: $U = \sum U_i$,其中 $i \in T$.

3 支持两级分区调度策略的实现机制

Windows NT 操作系统实现可以通过时钟中断触发调度,修改任务状态并转入相应的任务队列中.本文讨论的分区模型下基于时钟调度的算法研究,通过对 windows 操作系统进行一定的扩展,能够在 windows 平台得到很好的支持.将整个系统划分为两个区域:(1)存放 windows 系统的非实时线程,由一个单独的处理器进行处理,我们称之为非实时区域;(2)存放需要处理的实时任务,由预留出来的其他的处理器专门进行处理,我们称之为实时区域.这种设计模型有两个好处:(1)实时任务可以获得更多的 CPU 资源,减少和非实时任务的竞争;(2)便于接管中断.将所有中断都只发往非实时区域,同时将时钟中断发往实时区域,这样既避免了中断对实时任务的影响,又可以直接利用时钟中断在实时区域进行基于时钟的两级分区调度.

由于 Windows NT 的时钟中断周期可以进行修改,并且实时区域只接收时钟中断.因此,为了使调度程序不仅能在一个可调节的周期点上准确执行,还能获得比任何实时任务更高的系统处理优先级,可以将调度程序放到 DPC 中,让每次时钟中断到来都插入一个 DPC 进行调度.当时钟中断退出, DPC 即获得执行的机会,保证了调度的时钟周期性和高优先级.

4 可调度性分析与改进

4.1 分区内考虑时钟中断调度开销的 RM 算法

基于时钟中断的调度,在可调度性分析中必须考虑两个因素:(1)在下一个时钟中断之前,调度程序有可能不会发现任务的释放并进行调度行为,等下一个时钟中断到来才发现时,可能造成任务完成时间的延迟.(2)调度程序对任务控制块进行的移动和修改任务状态等动作的时间也增加了调度的开销.

4.1.1 实例分析

用 P_0 表示时钟中断的周期大小, e_0 为处理时钟中断的时间.即使此次调度没有发现已释放的任务,这部分时间也要消耗.用 S_0 表示调度程序移动

任务和修改任务状态等动作所消耗的时间。

考虑这样一个系统,它包含 3 个任务 $T_1 = (0.1, 4, 1)$, $T_2 = (0.1, 5, 1.8)$, $T_3 = (0, 20, 5)$. 其中 T_3 的第一部分是不可抢占的. 这一部分的执行时间为 1.1. 任务的时限等于各自的周期,任务的优先级 $T_1 > T_2 > T_3$. 按照经典的 RM 算法进行分析,任务的首次执行结束时间分别为 2.1, 3.9, 14.4. 现假设调度程序只在时钟中断时执行,即 P_0 为 1, e_0 为 0.05 个时间单位, S_0 为 0.06 个时间单位. 于是这个系统的调度情况如图 3 所示.

从图 3 的系统调度情况可以清楚的看到, T_2 的

第一个执行周期实际截止时间为 6.29, 大于 5.1, 也就是它的截止时限, 按照理想情况下分析, T_2 的截止时间为 3.9, 那么按照理想情况进行分析就不能预知在时刻 5.1 会发生的错过时限的现象.

4.1.2 仿真验证

为了检验时钟中断以及调度开销对任务实际执行时间的影响, 现设计了一组任务 $\text{task}_1(500, 100)$; $\text{task}_2(800, 200)$, 优先级: $\text{task}_1 > \text{task}_2$. 在时钟周期为 15 ms, 调度开销为 20 μs 的系统中进行测试, 对 task_1 的实际运行时间进行跟踪, 得出结果如图 4 所示.

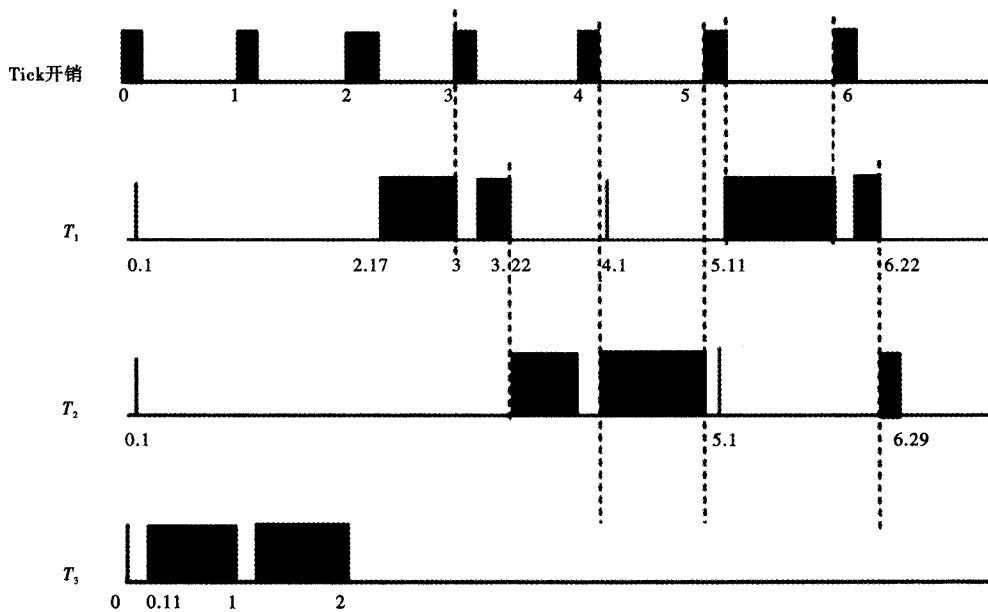


图 3 时钟调度开销对任务可调度性的影响示例图

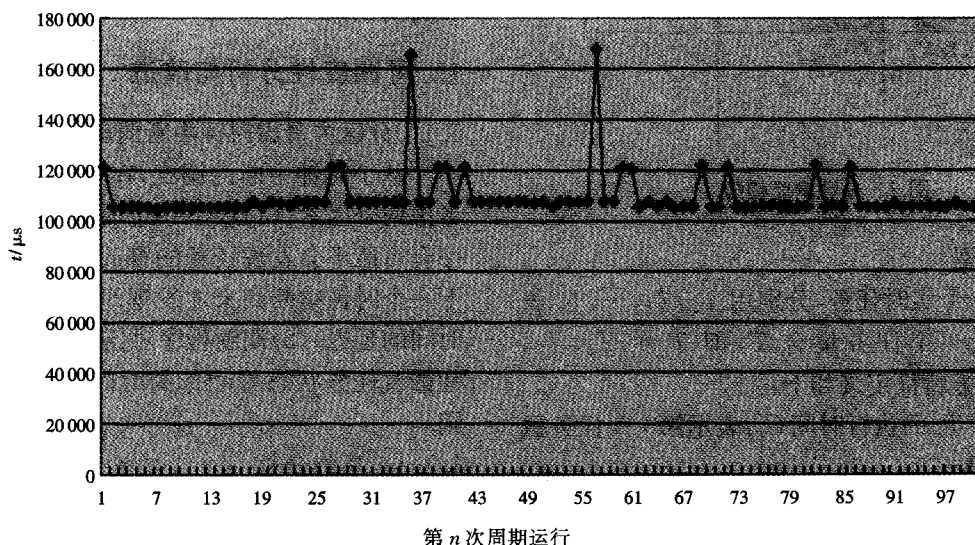


图 4 结果图

从图4可以明显的看出,在基于时钟中断的调度开销的影响下,task_i的实际执行时间平均在106 ms上下,比任务实际需要的运行时间100 ms要大一些,证实了时钟中断以及调度开销对任务执行时间具有一定的影响.因此这一算法在考虑时钟中断以及调度的开销情况下,需要进行必要的改进,以适应RM算法本身强预知性的初衷.

4.1.3 改进方法

综合前述现象,基于时钟的调度所引入的开销主要需要从以下几个方面进行考虑:

(1) 调度程序处理时钟中断的开销 e_0 .

(2) 每个任务就绪,或有更高优先级的任务就绪时,调度程序花费大约 S_0 个时间单位将就绪任务移到就绪队列.

(3) 调度程序将低优先级任务从挂起队列转移到就绪队列的开销(约为 S_0).

(4) 当低优先级任务有不可抢占的部分时,这个不可抢占的时间对高优先级任务造成的阻塞最坏是这段时间所能跨越的最大时钟中断周期数.

根据以上分析,分别采取相应的措施对每一个方面进行优化,具体方法如下:

(1) 在较高优先级任务集中加入任务 $T_a = (0, P_0, e_0)$;

(2) 每个较高优先级任务 T_k 的执行时间 e_k 增加 S_0 ;

(3) 对每个较低优先级的任务 T_k ,在较高优先级任务集中增加一个任务 $T_b(0, PT_k, S_0)$;

(4) 任务 T_k 的最大执行时间要包含低优先级任务的不可抢占部分造成的阻塞时间:

$((\beta_k/P_0)+2) * P_0$. (β_k 为低优先级任务 T_k 的不可抢占部分的最大执行时间.)

对4.1.1小节中的调度实例使用上述改进方法进行重新评估.考虑时钟中断及调度开销的情况下,计算T2的时间需求公式时,系统中有5个任务需要计算在内: $T_a = (0, 1, 0.05)$, $T_b = (0, 20, 0.06)$, $T_1 = (0.1, 4, 1.06)$, $T_2 = (0.1, 5, 1.86)$, $T_3 = (0, 20, 5)$.其中 T_a , T_b 和 T_1 的任务优先级高于 T_2 .带人文献[5]中推导出的任务 T_i 的最大执行时间函数:

$w(t) = e_i + b_i + (\sum t/PT_k)$;其中 b_i 为阻塞时间^[6].

迭代求得, T_2 的最大执行时间为7.44,由此可以得出正确的结论: T_2 是不可调度的.

4.1.4 更进一步探索

通过前述改进方法,可以准确地对任务的可调度性进行更实际的预测.经过进一步分析,不难发现,无论是任务集中新引入的任务造成的开销,还是不可抢占部分造成的阻塞时间,都与时钟中断的周期大小 P_0 存在着直接的关系.因此通过改变 P_0 的大小,会对任务实际的最大执行时间产生直接的影响,下面针对上述的实例,通过一组离散的数据,来说明 P_0 的大小与任务实际的最大执行时间之间的大致关系.如表1所示.

表1 调度周期与 T_2 执行时间关系

P_0	1	0.9	0.8	0.6	0.55	0.5	0.4	0.1
$w(t)$	7.44	7.14	6.89	6.34	6.29	6.74	6.49	7.74

可以看出,当 $P_0 = 0.55$ 时 task₂ 的最大执行时间达到最小值即为6.29.从本文举出的实例来看 T_2 是不可调度的,但是我们可以从分析结果中得出结论:

通过调整时钟周期的大小,一定能找到一个合适的时钟周期,使任务的最大执行时间最小,从而尽最大可能保证系统内整个任务集的可调度性.

4.1.5 仿真试验

通过C语言建模,对基于时钟中断的调度策略下的某个特定任务集在不同时钟周期下进行了仿真实验,实验平台采用VC++6.0.通过前述分析可以知道,时钟周期过大,某个任务的阻塞部分可能跨度的时间也就随之增大,对任务集的可调度性势必产生负面影响,为了较为突出的反映时钟调度开销对任务实际执行时间的影响,在仿真试验中,任务集中的任务均不包括不可抢占部分,这样可以更直观的看出时钟周期偏小的情况对任务可调度性的影响.现设计任务集如下: Task₁ (0 ms, 400 ms, 100 ms); Task₂ (0 ms, 800 ms, 200 ms); Task₃ (0 ms, 800 ms, 300 ms).任务的优先级 Task₁ > Task₂ > Task₃.在此任务集下,系统的负载可以达到87.5%以上.在调度开销固定的情况下,分别对不同的时钟周期调度下任务集的可调度性进行仿真测试.仿真结果如表2所示.

由表2可以看出任务优先级最低的Task₃的截止期限是800 ms,在时钟周期为50 ms以下时,task₃的运行过程中出现超期的次数比较多.当时钟周期为100 ms时,该任务集不会出现有任务超期的情况,任务集在系统运行过程中的可调度性最佳,证

实了上节的分析.

表 2 不同时钟周期的仿真测试

时钟周期/ ms	仿真时间/ ms	调度开销/ ms	Task ₃ 的截止期限/ ms	Task ₃ 超期任务数/ ms
33	120 000	5	800	20
40	120 000	5	800	13
50	120 000	5	800	11
80	120 000	5	800	2
100	120 000	5	800	0

(注:每次任务超过截止时限,该任务在 3 个任务周期时间后再次参与调度.)

4.2 分区设计方法

对于系统中运行的特定任务集,分区如何进行划分能兼顾系统的可调度性和 CPU 的利用率是一个关键的问题.如果每个分区设计得太大,则出现系统空闲的时间变长.如果分区设计得太小,则不能保证分区内任务的可调度性.

根据文献[7]中推导出的定理:

$\sum U_i \leq n_i [(2/(2-C_i))^{1/n_i} - 1]$ (n_i 为第 i 个分区的任务数; C_i 为分区 P^i 的执行时间权重)

可知,只要分区的任务总利用率 $\sum U_i$ 满足上述公式,那么分区 P^i 中的任务一定是调度的.在保证分区内所有任务可调度的前提下,为了使 CPU 的利用率尽可能大,应使分区 P^i 中任务的利用率之和尽可能的大,取 $\sum U_i$ 的上限 $\sum U_i = n_i [(2/(2-C_i))^{1/n_i} - 1]$,可得分区 P^i 的执行时间权重:

$$C_i = 2 - 2(\sum U_i / n_i + 1)^{-n_i}$$

对每个分区的执行时间权重按此规则进行设计可以保证系统内所有任务可调度的前提下,CPU 的利用率最大.

文献[7]给出了此种分区设计原则下任务可调

度性的测试结果.结果表明,此种分区设计原则下实时任务能够好的在任务期限内完成,更好地满足了实时系统两级分区调度的实时性要求.

5 结束语

本文根据 ARINC653 标准,建立了可行的两级分区调度模型并对一级调度的设计和二级调度的算法进行了可调度性的分析和改进,对 ARINC653 标准下系统的实时性保障能力进行了理论分析和仿真测试,是对多级分区系统实时调度问题的一次有益探索.

参考文献:

- [1] Burchard A, Liebeherr J. oh YF. New Strategies for assigning real-time tasks to multiprocessor systems [J]. IEEE Trans on Computers, 1995,44(12):1429-1442.
- [2] 王永吉,陈秋萍.单调速率及其扩展算法的可调度性判定[J].软件学报,2004(6):799-814.
- [3] 陈劲林,杨士元,胡东成.基于确定周期性任务的进程管理及可调度性分析[J].计算机研究与发展,2000,37(3):3-7.
- [4] Jane W. S. Liu. Real-Time System[M]. 北京:高等教育出版社,2003.
- [5] 李昕颖,顾健,何锋,等.硬实时系统在强分区约束下的双层分区调度[J].计算机学报,2010,33(6):6-11.

作者简介:



曹广旭 男,(1986-),硕士研究生.研究方向为嵌入式计算及其应用.