单调速率及其扩展算法的可调度性判定*

王永吉[1+], 陈秋萍[1,2]

[1](中国科学院 软件研究所 互联网软件技术实验室,北京 100080)

[2](中国科学院 研究生院,北京 100039)

On Schedulability Test of Rate Monotonic and Its Extendible Algorithms

WANG Yong-Ji[1+], CHEN Qiu-Ping[1,2]

[1](Laboratory for Internet Software Technologies, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Graduate School, The Chinese Academy of Sciences, Beijing 100039, China)

+ Corresponding author: Phn: +86-10-62581294, E-mail: ywang@itechs.iscas.ac.cn, http://itechs.iscas.ac.cn

Received 2003-09-10; Accepted 2004-01-07

Wang YJ, Chen QP. On schedulability test of rate monotonic and its extendible algorithms. *Journal of Software*, 2004,15(6):799~814.

http://www.jos.org.cn/1000-9825/15/799.htm

Abstract: Schedulability test is an essential issue in real-time scheduling theory. Rate monotonic(RM) algorithm is one of the scheduling algorithms in real-time systems However, a general review on this field can hardly be seen. This paper presents a review of the various schedulability tests under RM algorithm, starting from the simplest ideal RM scheduling model and then going into the more complicated ones. Three subjects are covered: (1) schedulable tasks' CPU utilization least bound and the sufficient and necessary conditions; (2) condition of schedulable tasks taking the scheduling time cost into account; (3) condition of schedulable tasks taking priority inversion into account. Some appropriate examples are provided to illustrate and compare the relative merits among the discussed algorithms.

Key words: real-time system; real-time operating system; real-time scheduling; RM algorithm; schedulability test

摘 要: 任务可调度性判定是实时系统调度理论研究的核心问题.单调速率(RM)算法是实时调度的重要算法,自其提出以来已被广泛研究.然而到目前为止,尚缺乏专题性的文章来系统而深入地探讨 RM 及其扩展算法的可调度性判定,以及各种现实条件和实现方式(包括任务调度的时间开销和任务同步问题等)对可调度性的影响.围绕 RM 算法下的可调度性判定问题,由浅入深,系统性地讨论各种不同假设和实现方式对可调度性的影响,具

* Supported by the National Natural Science Foundation of China under Grant No.60373053 (国家自然科学基金); the Hundred Talents of the Chinese Academy of Sciences (中国科学院"百人计划"); the Chinese Academy of Sciences and the Royal Society of the United Kingdom under Grant Nos.20030389, 20032006 (中国科学院与英国皇家学会联合资助项目); the State Education Ministry Scientific Research Foundation for the Returned Overseas Chinese Scholars under Grant No.[2003]406 (留学回国人员科研启动基金)

作者简介: 王永吉(1962－),男,辽宁盖州人,研究员,博士生导师,主要研究领域为实时系统,网络优化,智能软件工程,优化理论,机器人,控制理论;陈秋萍(1980－),女,硕士生,主要研究领域为实时系统,智能软件工程.

体分为下述 3 大类问题:(1) 理想的 RM 算法下的可调度性判定的 CPU 利用率最小上界及可调度的充分必要条件;(2) 考虑调度时间开销情况下的可调度性判定条件;(3) 优先级反转协议及其对可调度性的影响.给出了具体实例来阐述上述问题,并从算法复杂度和可检测率两方面来比较各种算法的优劣.

关键词: 实时系统;实时操作系统;实时调度;RM 算法;可调度性判定

中图法分类号: TP316 **文献标识码**: A

与非实时系统相比,实时系统的一个显著特点是,它们试图同时实现计算在逻辑和时间上的正确性.在实时系统中,计算的正确性不仅取决于计算的逻辑结果,也取决于结果产生的时间[1].实时调度算法是保障实时系统两个必备特性——时限性和高可靠性的重要手段之一.实时调度是指在有限的系统资源(如 CPU 等)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,以保证其时间约束(即截止期限)、时序约束和资源约束得到满足.

实时调度一直都是实时系统研究中的热点问题,在国内外学术界倍受关注[2~8].在实时系统调度理论研究中,可调度性判定是核心问题[1,2,9~15].这是因为,实时任务具有时限要求,在一个或多个处理器之间调度实时任务,需要判断是否每个任务的执行都能够在其截止期限内完成.如果每个任务的执行都能够在其截止期限内完成,则称该调度是可行的.可调度性判定(或称调度可行性判定)就是判定给定的 n 个实时任务在应用某种调度算法的前提下能否产生一个可行的调度.调度算法的设计要尽可能满足任务可调度性的要求.

由于实时系统的侧重点不同,实时调度亦有多种分类方式.常见的分类有,根据任务实时性要求的重要程度,分为硬实时调度和软实时调度——在硬实时调度中任务必须在其截止期限(deadline)内执行完毕,否则将产生严重后果.而对于软实时任务,当系统负载过重的时候,允许发生错过截止期限的情况;根据任务是在一个或多个处理器上运行,分为单处理器实时调度和多处理器实时调度,多处理器实时调度又可分为集中式调度和分布式调度;根据调度算法和可调度性判定是在任务运行之前还是运行期间进行的,分为静态调度、动态调度和混合调度;根据被调度的任务是否可以互相抢占,分为抢占式调度和非抢占式调度;根据任务请求到达的情况不同,分为周期性任务调度和非周期性任务调度.不同调度方式具有各自的优缺点,适用于不同类型的实时系统.

1973 年,Liu 和 Layland 提出了一种适用于可抢占的硬实时周期性任务调度的静态优先级调度算法——速率单调(rate monotonic,简称 RM)调度算法,并对其可调度性判定问题进行了研究[2].RM 算法自从提出以来得到了广泛的研究和应用,目前已有大量关于 RM 算法及其各种扩展情况下的调度算法以及实时任务在这些算法下的可调度性判定研究的文献[9,11,12,14,16~21].理论研究的问题集中在如何找到更快、更好的可调度性判定方法,以及如何扩展 RM 算法,使之更好地满足现实实现的需求.尽管已有少量关于实时系统方面的专著[22~24],但是至今尚缺乏这方面的专题性文章来系统而深入地探讨 RM 及其扩展算法的可调度性判定,研究各种现实条件和实现方式对可调度性的影响.这不利于实时操作系统的开发,因为引入时间开销及死锁防止机制下的可调度性判定是实时操作系统实现的重要理论基础.

本文围绕 RM 算法下的可调度性判定问题,由浅入深,系统性地讨论各种不同假设及实现方式对可调度性的影响,具体分为下述 3 大类问题:(1) RM 算法在理想情况下的可调度性判定;(2) 引入时间开销情况下的可调度性判定条件;(3) 优先级反转问题及其对可调度性的影响.本文给出具体实例来说明上述问题,同时从算法复杂度和可检测率两方面来比较各种算法的优劣.文章最后给出了结论及未来的工作.

1 RM 算法及其可调度性判定

1.1 理想的RM调度模型

定义 $S=\{t_1,t_2,\ldots,t_n\}$ 为一个含有 n 个周期性任务的集合,集合中的任务用 $t_i=(T_i,C_i,D_i,P_i,U_i)(i=1,\ldots,n)$ 来表示,其中,T_i 表示 t_i 的周期,C_i 表示 t_i 的最坏执行时间,D_i 表示 t_i 的截止期限,即 t_i 的运行必须在时间 D_i 内完成,P_i 表示 t_i 在该任务集中的优先级(优先级值越小优先级越高),U_i 表示 t_i 的 CPU 利用率,即 $U_i=C_i/T_i$.整个任务集的 CPU 利用率定义为

$$U=\sum_{i=1}^{n}(C_i/T_i) \tag{1}$$

所谓的 RM 调度[2],就是为每一个周期任务指定一个固定的优先级,该优先级按照任务周期的长短顺序排列,任务周期越短,其优先级越高,调度总是试图最先运行周期最短的任务.也就是说,若 $T_i<T_j,1\le i\le n,1\le j\le n$,则 $P_i\le P_j$.为方便计,我们假定:若 $i<j,1\le i\le n,1\le j\le n$,则 t_i 的优先级高于 t_j,也就是说,$t_1,t_2,\ldots,t_n$ 按照优先级由高到低的顺序排列.

Liu 和 Layland 在文献[2]中证明了 RM 算法是最优的,即对于在任何其他静态优先级算法下可调度的任务集合,在 RM 算法下也是可调度的*,证明思想如下.假设一个任务集 S 采用其他静态优先级算法可以调度,设 t_i 和 t_j 是其中两个优先级相邻的任务,$T_i>T_j$,而 $P_i\le P_j$.将 t_i 和 t_j 的优先级互换,可以证明这时 S 仍然可以调度.按照这样的方法,其他任何静态优先级调度最终都可以转换成 RM 调度.这样便证明了 RM 算法在所有静态优先级算法中是最优的.RM 算法的最优性保证了其能够广泛地应用于各种静态优先级调度的情况,这是 RM 算法受到重视的重要原因.

需要特别指出的是,理想的 RM 调度模型是基于一系列假设的基础上的,这些假设在理想的 RM 模型中缺一不可.所谓的 RM 扩展模型则意味着对这些基本假设进行修改.RM 模型的基本假设如下[2]:

(A1) 所有的任务请求都是周期性的,具有硬时限要求,即必须在限定的时限内完成;

(A2) 任务的时限要求仅限于任务必须在该任务的下一个请求发生之前完成,即 $D_i=T_i,i=1,\ldots,n$;

(A3) 任务之间都是独立的,每个任务的请求不依赖于其他任务请求的开始或完成;

(A4) 每个任务的运行时间是不变的,这里任务的运行时间是指处理器在无中断情况下用于处理该任务的时间;

(A5) 调度和任务切换的时间忽略不计;

(A6) 任务之间是可抢占的;

(A7) 所有任务的分配都在单处理器上进行.

Liu 和 Layland 在以上基本假设的基础上对 RM 算法的可调度性判定进行了研究,提出了一个 RM 算法下的可调度性判定条件.在后人对 RM 算法的研究中,既有保持以上假设提出的更为精确的判定条件,又有放松上述的某些基本假设,在 RM 模型中引入新的影响因素,在扩展了的 RM 模型的基础上研究任务可调度性判定条件.在本节中我们主要讨论理想的 RM 调度模型及其可调度性判定的研究,RM 扩展模型将在下一节中讨论.

1.2 Liu和Layland的RM算法可调度性判定条件

首先解释一下任务的临界时刻(critical instant)这一重要概念,它是Liu和Layland提出来的[2].一个任务的临界时刻是指这个任务在这一时刻提出请求将会有最长的响应时间.如果一个任务在其临界时刻能够被调度,那么它在任何时刻都能够被调度.本文所讨论的 RM 算法可调度性判定条件都是在任务的临界时刻进行考察的.

直观地看,当一个任务被触发时,如果有比其优先级更高的任务同时被触发,则该任务将会因被抢占而延长其响应时间,因而一个任务的临界状态应该出现在所有比其优先级高的任务被同时触发的时候.Liu 和 Layland 证明了这一点,即定理 1[2].

定理 1. 任何任务在与比其优先级高的所有任务同时被触发时达到其临界时刻.

根据定理 1,对于 n 个固定优先级的任务,对这些任务进行调度的最坏情况是这 n 个任务同时就绪.此时,所有的任务都在其临界时刻,亦即所有的任务响应时间均为最长,如果在这种情况下这 n 个任务能够被调度,则在任何时刻这些任务都能被调度.在定理 1 的基础上,Liu 和 Layland 给出了一个 RM 算法的可调度性判定条件,该条件是通过 CPU 的利用率给出的.见定理 2[2].

定理 2. 给定任务集 $S=\{t_1,t_2,\ldots,t_n\}$,如果这 n 个任务的 CPU 利用率满足下面的条件:

$$U<n(2^{1/n}-1)=L(n) \tag{2}$$

则该任务集 S 用 RM 算法是可调度的.这里,$L(n)$表示 n 个任务的 CPU 利用率的最小上界.由式(2)可知,$L(n)$与

* 文献[4]中最优化证明的结论是正确的,但用其中公式(1)的证明是不对的,正确的应该是用文献[9]中的充分必要条件.

C_i,T_i无关,它随 n 的变化关系如图 1 所示.

Fig.1 Relation of $L(n)$ and n

图 1 $L(n)$随 n 单调变化关系趋势

由图 1 可见,$L(n)$随 n 单调递减.当 n=1 时,$L(1)$=1.当 $n\to\infty$时,$L(n)=\ln 2\approx 0.693$.

下面我们举 3 个例子来说明定理 2 的应用及其优缺点,并在后面用来与其他判定条件进行比较.

例 1:假设一个周期性任务集 $S_1=\{t_1,t_2,t_3\}$,其中 t_1:C_1=20,T_1=100,U_1=0.200;t_2:C_2=40,T_2=150,U_2=0.267;t_3:C_3=100,T_3=350,U_3=0.286.

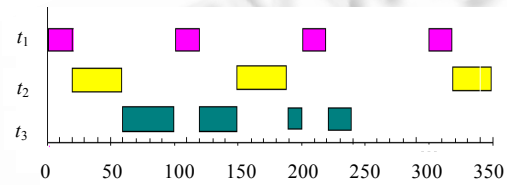

Fig.2 Scheduling of S_1

图 2 S_1的调度示意图

根据式(1)计算得到该任务集的 CPU 利用率 U=0.753<$L(3)\approx$0.780,根据定理 2 可以判定该任务集是可调度的.S_1的调度如图 2 所示.

例 2:假设一个周期性任务集 $S_2=\{t_1,t_2,t_3\}$,其中 t_1:C_1=8,T_1=32,U_1=0.250;t_2:C_2=15,T_2=40,U_2=0.375;t_3:C_3=20,T_3=80,U_3=0.200.

计算得到该任务集的 CPU 利用率 U=0.825>$L(3)$.对该任务集,用定理 2 无法判定其可调度性.

例 3:假设一个周期性任务集 $S_3=\{t_1,t_2,t_3\}$,其中 t_1:C_1=40,T_1=100,U_1=0.400;t_2:C_2=50,T_2=250,U_2=0.200;t_3:C_3=100,T_3=400,U_3=0.250.

计算得到该任务集的 CPU 利用率 U=0.825>$L(3)$.对该任务集,定理 2 无法判定其可调度性.

从上面的例子我们可以看出,对于一些任务集,用定理 2 可以很容易地判定其可调度性,简便、快速是定理 2 的优点.而对于另一些任务集,用定理 2 则无法判定其是否可调度(在后面的章节中,我们将用其他方法判定例 2 和例 3 都是可调度的).对于一个含有 n 个任务的任务集来说,如果 CPU 利用率大于 $L(n)$,则用定理 2 无法判断该任务集合是否可调度.这是因为定理 2 给出的判定条件是在最坏情况下考察其可调度性而得出的,它只是一个充分但不必要的条件.定理 2 因此被称为"悲观的",这是其不足之处.

1.3 更高的CPU利用率最小上界

正如我们在第 1.2 节中所看到的,定理 2 并不能适用于所有的 RM 可调度性判定.后来,Burchard 等人给出了另一个可调度性判定的 CPU 利用率最小上界[1].这个最小上界比定理 2 中的最小上界要高,从而扩大了 RM 可调度性判定的范围.

定理 3. 给定任务集 $S=\{t_1,t_2,\ldots,t_n\}$,定义 H_i,β如下:

$$H_i=\log_2 T_i-\lfloor\log_2 T_i\rfloor,1\le i\le n \tag{3}$$

注:$\lfloor x\rfloor$表示不大于实数 x 的整数,下文中$\lceil x\rceil$表示不小于实数 x 的整数.

$$\beta=\max_{1\le i\le n}H_i-\min_{1\le i\le n}H_i \tag{4}$$

(1) 若$\beta<1-1/n$,且 CPU 利用率满足:

$$U\le(n-1)(2^{\beta/(n-1)}-1)+2^{1-\beta}-1=B(n) \tag{5}$$

(2) 若$\beta\geq 1-1/n$,且 CPU 利用率满足:

$$U\leq n(2^{1/n}-1)=L(n) \tag{6}$$

则该任务集 S 在 RM 算法下是可调度的.

定理 3 中的条件(2)与文献[2]中的结果相同,Burchard 等人的改进在于式(5),提出了一个更高的 CPU 利用率的最小上界 $B(n)$.下面我们证明:当$\beta<1-1/n$ 且 $n\geq 2$ 时,总有 $B(n)>L(n)$.

证明:设函数 $f(x)=x(2^{1/x}-1)$,当 $x>0$ 时,有 $f'(x)=2^{1/x}\ln^2 2/x^3>0$,可知函数 $f(x)$是一个凸函数.根据凸函数性质,对 $\forall x>0,\forall y>0,\forall \lambda\in(0,1)$,如下不等式总成立:

$$f(\lambda x+(1-\lambda)y)\leq \lambda f(x)+(1-\lambda)f(y).$$

现假设$\lambda=\beta,x=(n-1)/\beta,y=1/(1-\beta)$代入上式,则有,

左式 $$f(\lambda x+(1-\lambda)y)=f(n)=n(2^{1/n}-1),$$

右式 $$\beta f((n-1)/\beta)+(1-\beta)f(1/(1-\beta))=(n-1)(2^{\beta/(n-1)}-1)+2^{1-\beta}-1.$$

可见,当$\beta\in(0,1)$时,总有

$$(n-1)(2^{\beta/(n-1)}-1)+2^{1-\beta}-1\geq n(2^{1-n}-1);$$

当$\beta=0,1$,上述结论亦成立.又根据已知条件有$\beta\in[0,1]$,综上得证. □

下面我们仍用前面所举的例子来说明定理 3 的应用及其优缺点.首先以例 2 为例,根据式(3)和式(4)计算得到:$H_1=0,H_2=0.322,H_3=0.322,\beta=0.322-0<1-1/3\approx 0.667$.对该任务集 S_2 应依据定理 3 中的条件(1)来判断.因为 $U=0.825\leq(3-1)(2^{\beta/(3-1)}-1)+2^{1-\beta}-1=0.836=B(3)$,即在这种情况下不等式(5)成立,由此可以断定,该任务集 S_2 是可调度的.S_2 的调度如图 3 所示.

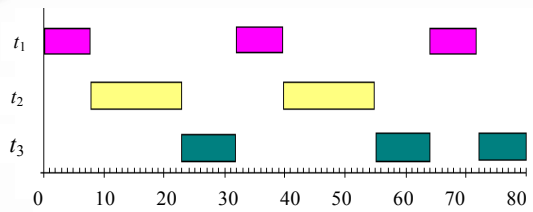

Fig.3 Scheduling of S_2

图 3 S_2 的调度示意图

下面我们再来看例 3.根据式(3)和式(4)计算得到:$H_1=0.644,H_2=0.966,H_3=0.644,\beta=0.966-0.644=0.322<1-1/3\approx 0.667$.对该任务集 S_3 应依据定理 3 中的条件(1)来判断,因为 $U=0.850>(3-1)(2^{\beta/(3-1)}-1)+2^{1-\beta}-1=0.836=B(3)$,即在这种情况下不等式(5)不成立.因此,用定理 3 无法判定该任务集 S_3 是否可调度.

从上面的例子我们可以看出,尽管定理 3 比定理 2 的判定范围要广,但定理 3 仍不能从根本上解决所有的 RM 可调度性判定问题,它存在着与定理 2 相同的问题.定理 3 中的判定条件也是一个充分但不必要条件,对某些任务集仍然无法判定其可调度性.对于一个给定的任务集 $S=\{t_1,t_2,\ldots,t_n\}$,如果$\beta\geq 1-1/n$ 而 $U>B(n)$,或者$\beta<1-1/n$ 而 $U>L(n)$,在这两种情况下用定理 3 无法判定该任务集 S 的可调度性.从这个角度来看,定理 3 也是“悲观的”.

1.4 RM算法可调度性的充要条件

正如我们在第 1.3 节中所看到的,定理 3 并不能适用于所有的 RM 可调度性判定.即使后来提出的更高的 CPU 利用率最小上界,例如 Bini 的双曲线的 CPU 利用率最小上界[16,17],也存在同样的问题.大量的事实证明,应用 RM 算法时,CPU 的利用率在 0.90 左右的可调度任务集并非少见.这说明 RM 算法的平均情况比最坏情况要好得多.一个任务集合的可调度性完全取决于该集合中任务的周期和运行时间.

在 Liu 和 Layland 的研究基础上,Lehoczky 等人对固定优先级算法的特征进行了更为精确的研究,并提出了 RM 算法可调度性判定的充要条件[9].

对于任务集 $S=\{t_1,t_2,\ldots,t_n\}$,用

$$W_i(t)=\sum_{j=1}^{i}C_j\lceil t/T_i\rceil \tag{7}$$

表示在时间段$[0,t]$之间该任务的前 i 个任务对 CPU 的累计需求量,其中时刻 0 是临界时刻(假定在时刻前 i 个任务同时就绪).接下来定义:

$$L_i(t)=W_i(t)/t \tag{8}$$

$$L_i=\min_{\{0<t\leq T_i\}}L_i(t) \tag{9}$$

$$L=\max_{\{1\le i\le n\}}L_i \tag{10}$$

由上面的定义,Lehoczky 等人给出了 RM 算法可调度性的充要条件:

定理 4. 给定任务集 $S=\{t_1,t_2,\ldots,t_n\}$,

(1) 第 $i(1\le i\le n)$个任务 t_i 在 RM 算法下能够被调度,当且仅当 $L_i\le 1$;

(2) 该任务集 S 在 RM 算法下是可调度的,当且仅当 $L\le 1$.

从表面上看,用定理 4 进行可调度性判定的时候,要对每一个任务 t_i 求 L_i,需要在$[0,T_i]$这一连续区间内的无穷多个点上求 $L_i(t)$的最小值.而实际上,只需要在有限的一些点上对 $L_i(t)$进行求值,然后比较这些值,其中的最小值就是 L_i.由式(7)和式(8)可以看出,在$[0,T_i]$的任一子区间$[a,b]$中$(a,b$ 是$\{kT_j|j=1,\ldots,i;k=1,\ldots,\lfloor T_i/T_j\rfloor\}$中两个值相邻的数),$W_i(t)$首先是不变的,然后在该区间的右端点跳跃到一个更高的值,而 t 在整个区间中单调递增,因此,$L_i(t)$在该区间首先是单调递减的,最后在该区间的右端点跳跃到一个更高的值.也就是说,当 t 是 $T_j(1\le j\le i)$的倍数时,$L_i(t)$具有局部最小值.因此,要求 $L_i(t)$的最小值 L_i 只需比较这些离散点上对应的局部最小值.

令

$$E_i=\{kT_j|j=1,\ldots,i;k=1,\ldots,\lfloor T_i/T_j\rfloor\} \tag{11}$$

则

$$L_i=\min_{\{t\in E_i\}}L_i(t) \tag{12}$$

定理 4 等价于定理 5.

定理 5. 给定任务集 $S=\{t_1,t_2,\ldots,t_n\}$,

(1) 第 $i(1\le i\le n)$个任务 t_i 在 RM 算法下能够被调度,当且仅当

$$L_i=\min_{\{t\in E_i\}}L_i(t)\le 1 \tag{13}$$

(2) 该任务集在 RM 算法下是可调度的,当且仅当

$$L=\max_{\{1\le i\le n\}}L_i\le 1 \tag{14}$$

由以上的分析,我们得到 RM 算法可调度性判定的充要条件的直观意义:对于每一个任务 $t_i,1\le i\le n$,在$(0,T_i)$(时刻 0 是临界时刻)中,存在至少一个这样的时刻,使得优先级不低于 t_i 的所有任务请求都能够完成.

下面,我们仍用前面给出的例 3 来说明定理 5 的应用及其优缺点.根据定理 2 可以判定 t_1 和 t_2 是能够被调度的,因此只需判断 t_3 能否被调度.根据定义(11)计算出 $E_3=\{T_1,2T_1,T_2,3T_1,T_3\}=\{100,200,250,300,400\}$.如果 $L_3=\min_{\{t\in E_i\}}L_i(t)\le 1$ 成立,即只要下面 5 个不等式之一成立,就可以判定该任务集 S_3 可调度,否则 S_3 不可调度:

(a) $C_1+C_2+C_3\le T_1$;

(b) $2C_1+C_2+C_3\le 2T_1$;

(c) $2C_1+2C_2+C_3\le T_2$;

(d) $3C_1+2C_2+C_3\le 3T_1$;

(e) $4C_1+2C_2+C_3\le T_3$.

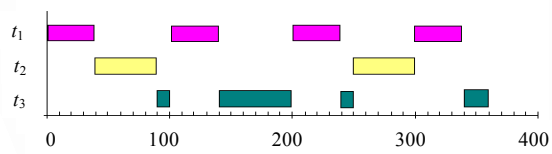

Fig.4 Scheduling of S_3

图 4 S_3 的调度示意图

计算得出,不等式(a),(b),(c),(d)皆不成立,但不等式(e)成立:$4C_1+2C_2+C_3=160+100+100=360<T_3=400$.因此,该任务集 S_3 是可调度的.S_3 的调度如图 4 所示.

1.5 多项式时间的判定算法

RM 算法的可调度性判定充要条件的提出是一个重大的突破,它提供了适用于任何情况的 RM 算法可调度性判定.但是,定理 5 是一个伪多项式时间的算法,复杂度较高.后来,Han 和 Tyan 提出了一种比较简单的多项式时间判定算法[11,14].

1.5.1 Han 和 Tyan 的构造性判定方法

依据第 1.4 节的充分必要条件,Han 和 Tyan 证明了如下定理[14]:

定理 6.对于给定的任务集 $S=\{t_1,t_2,\ldots,t_n\}$,如果存在另一个任务集 $S'=\{t_1',t_2',\ldots,t_n'\}$,满足:$T_i'\le T_i,C_i'=C_i$,$i=1,2,\ldots,n$,并且任务集 S'在 RM 下是可调度的,则该任务集 S 在 RM 下是可调度的.

下面的定理是定理 6 的一个特例[14].

定理 7. 对于给定的任务集 $S=\{t_1,t_2,\ldots,t_n\}$,如果存在另一个任务集 $S'=\{t_1',t_2',\ldots,t_n'\}$满足下列条件:

$$T_i'\le T_i, i=1,2,\ldots,n,\text{且 } T_i'\mid T'_{i+1}(T_i'\mid T'_{i+1}\text{表示 } T_i'\text{整除 } T'_{i+1}), i=1,2,\ldots,n-1 \tag{15}$$

$$U(S')=\sum_{i=1}^{n} C_i/T_i'\le 1 \tag{16}$$

则该任务集 S 在 RM 下是可调度的.

1.5.2 两种多项式算法:S_r 和 DCT

定理 7 为 RM 下的算法可调度性判定提供了一种新的构造性方法.对于一个给定的任务集 $S=\{t_1,t_2,\ldots,t_n\}$,如果能够找到另一个任务集 $S'=\{t_1',t_2',\ldots,t_n'\}$满足定理 7 中的条件,则可以判定该任务集 S 是可调度的.因此,剩下的一个重要问题就是如何找出一个任务集 S'满足定理 7 中的条件.Han 和 Tyan 提出了两种算法:S_r 和 DCT[11,14].

(1) S_r 算法

S_r 算法的构造思想是:找出一个特定的 r,使得 S'中的每一个任务 t_i'满足下列条件:

$$T_i'=r\cdot 2^{m_i}\le T_i<r\cdot 2^{m_i+1}=2T_i',\ i=1,\ldots,n \tag{17}$$

其中 r 是实数,m_i 是整数.从式(17)可以推出:$T_i<r\cdot 2^{m_i+1}=2T_i'<2T_i$.该不等式两边同时除以 r 然后取对数可得:$\log_2(T_i/r)-1<m_i<\log_2(T_i/r)$.因为 m_i 是整数,所以 m_i 只能取$\lfloor\log_2(T_i/r)\rfloor$.从而得出结论:当 r 确定时,对于每个 i,$1\le i\le n$,m_i 也就确定了,于是 T'_i 也就随之确定了:

$$m_i=\lfloor\log_2(T_i/r)\rfloor \tag{18}$$

$$T_i'=r\cdot 2^{\lfloor\log_2(T_i/r)\rfloor} \tag{19}$$

另外,由式(17)可以推出 $U(S')=\sum_{i=1}^{n}C_i/T_i'\ge U(S)=\sum_{i=1}^{n}C_i/T_i$.根据定理 7 我们知道,如果 $U(S')\le 1$,则可以判定 S 是可调度的.因此,S_r 的问题便进一步转化为如何找到一个最好的 r,使得 $U(S')$达到最小的问题.

从式(19)可以看出,对于每一个任务 t_i',在任一区间$[T_i/2^{m+1},T_i/2^m]$(m 是整数且 $m\ge 0$)中,T_i'先随着 r 的增大而增大,然后在区间的右端点跳跃到一个更高的值.也就是说,当 $r=T_i/2^m$(m 为整数且 $m\ge 0$)时,T_i'具有局部最大值,因此,S'的 CPU 利用率 $U(S')$在所有这些 r 的离散点上具有局部最小值.因此,要求 $U(S')$的最小值只需比较这些局部最小值.这里所用的方法和第 1.4 节中求 $L_i(t)$的最小值 L_i 有异曲同工之妙.

剩下来的问题是求 $U(S')$在上面提到的离散点上的局部最小值.用 $U_S(r)$表示由任务集 S 和 r 确定的任务集 S'的 CPU 利用率 $U(S')$,r^*表示使 $U_S(r)$取得最小值的 r 值.首先作如下定义:

$$l_i=T_i/2^{\lceil\log_2(T_i/T_1)\rceil} \tag{20}$$

接着将 $\{l_1,l_2,\ldots,l_n\}$ 按照非递减的方式排序,并除去重复的值,令 $(k_1,k_2,\ldots,k_u)$ 为所得到的序列,则 $k_u=l_1=T_1$.$\{k_1,k_2,\ldots,k_u\}$就是所要进行求值的离散点 r 的集合.对该集合中的每一个数 k_v,$1\le v\le u$,利用等式(19)的构造方法构造一个任务集 S',然后计算其 CPU 利用率 $U_S(k_v)$,并比较所得到的值,其中的最小值就是 $U_S(r^*)$,此时的 k_v 即为 r^*.若 $U_S(r^*)\le 1$,则可以判定任务集 S 是可调度的.

S_r 算法的伪码如下:

```
/*输入:任务集 S={t1,t2,...,tn},其中 t1,t2,...,tn 的周期从高到低排列.
/*输出:由任务集 S 构造出的满足定理 7 中条件(1)的任务集 S'={t1',t2',...,tn'}及 US(r*).
1.  US(r*)=0;r*=1;
2.  for i=1 to n do li=Ti/2^⌈log2(Ti/T1)⌉.
3.  将{l1,l2,...,ln}按照非递减的方式排序,并除去重复的值,令(k1,k2,...,ku)为所得到的序列.
4.   for v=1 to u do{
        US(kv)=0;
        for i=1 to n do{
            Ti'=kv·2^⌊log2(Ti/kv)⌋;
            Us(kv)=Us(kv)+Ci/Ti';
        }
        if i=1 then US(r*)=US(kv);
        else if US(kv)<US(r*) then
                r*=kv;
```

$U_S(r^*)=U_S(k_v)$;

}

5. 输出 $U_S(r^*)$及 $T_i'=r^*\cdot 2^{\lfloor \log_2(T_i/r^*)\rfloor}$,$i=1,\ldots,n$.

(2) DCT 算法

DCT 算法采用的是另一种构造方法.它是这样构造任务集的:对每一个 f,$1\le f\le n$,首先令

$$T_f'=T_f \tag{21}$$

然后令

$$T_i'=T_{i-1}'\cdot\lfloor T_i/T_{i-1}'\rfloor,i=f+1,f+2,\ldots,n \tag{22}$$

$$T_i'=T_{i+1}'/\lfloor T_{i+1}'/T_i\rfloor,i=f-1,f-2,\ldots,1 \tag{23}$$

这样,对于每一个 f 就构造出了一个满足定理 7 条件的任务集 S'.计算每一个任务集 S'的 CPU 利用率,其中 CPU 利用率最小的任务集就是最终所要找的任务集 S'.这一构造方法比 S_r 算法简洁、易懂.

DCT 算法的伪码如下:

```
/*输入:任务集 S={t1,t2,...,tn},其中 t1,t2,...,tn 的周期从高到低排列.
/*输出:由任务集 S 构造出的满足定理 7 中条件(1)的任务集 S'={t1',t2',...,tn'}.
 min_f=1;min_utilization=∞;
 for f=1 to n do {
     Zf=Tf;
     for i=f+1 to n do Zi=Zi-1·⌊Ti/Zi-1⌋;
     for i=f-1 to 1 do Zi=Zi+1·⌊Zi+1/Ti⌋;
     utilization=∑_{i=1}^{n} Ci/Zi ;
     if utilization<min_utilization then
         min_utilization=utilization;
         min_f=f;
         for i=1 to n do Ti'=Zi;
     endif
 }
```

从上面的算法描述容易证明,S_r 算法的时间复杂度是 $O(n\cdot\log n)$,DCT 算法的时间复杂度是 $O(n^2)$.DCT 算法的思想比 S_r 算法简单,但 DCT 算法复杂度比 S_r 算法复杂度要高.

S_r 和 DCT 算法的优点有两个:一个是它们比第 1.4 节中讨论的充分必要判定算法快速;另一个优点是它们的使用范围比第 1.3 节中的定理 3 更广——因而也就比第 1.2 节中的定理 2 更广.可以证明,如果一个任务集用定理 3 判定是可调度的,则用 S_r 和 DCT 算法判定也一定是可调度的;如果一个任务集用定理 3 无法判定其可调度性,则用 S_r 算法和 DCT 算法却有可能判定出其可调度性.

但是,S_r 算法和 DCT 算法比定理 3 的判定算法复杂度要高.另外,由于 S_r 算法和 DCT 算法的根本依据都是定理 7,而定理 7 给出的是一个充分但不必要的条件,因此 S_r 算法和 DCT 算法并不能取代定理 5 的充要判定条件.也就是说,对于一个给定的任务集,即使找不到另一个任务集满足定理 7 中的条件,也不能判定该任务集是不可调度的.下面,我们来看一个例子.

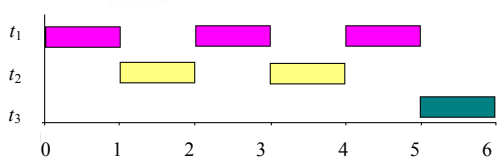

Fig.5 Scheduling of S_4

图 5 S_4 的调度示意图

例 4:假设一个周期性任务集 $S_4=\{t_1,t_2,t_3\}$,其中,t_1:$C_1=1$,$T_1=2$,$U_1=0.500$;t_2:$C_2=1$,$T_2=3$,$U_2=0.333$;t_3:$C_3=1$,$T_3=6$,$U_3=0.167$.

根据定理 5 可以很容易地判定该任务集 S_4 是可调度的.S_4 的调度如图 5 所示.

对于 S_4,用 S_r 算法和 DCT 算法只能找到两个满足式(15)的任务集 S',其任务的周期分别为 2,2,2 或 2,2,4,运行时

间与S_4中的任务运行时间相等.但显然这两个任务集都不满足式(16).因此,对于S_4,用构造性方法S_r算法和DCT算法无法找到满足定理7中的条件的一个任务集S'.

Han和Tyan利用随机产生的周期性任务集对S_r和DCT两种算法进行仿真比较,任务个数n从2~1000,所有的任务集经定理5判定均是可调度的,且其CPU利用率大于相应的$L(n)$和$B(n)$.这些任务集用定理3是无法判定其可调度性的.仿真结果见文献[14].仿真结果表明,通过判定的任务集的百分率随着 n 的增大而减少.当n=2时,所有的可调度任务集都通过判定;当n超过100时,通过判定的任务集的百分率集中在83%左右.这证明了S_r算法和DCT算法的适用范围比定理3要广,但不能取代定理5.另外,从仿真结果也可以看出,当n比较小时,DCT算法的判定结果比S_r算法要好,而当n增大到一定的数时,两者的判定结果不相上下.

1.6 理想的RM调度模型及其可调度性判定条件小结

RM 算法是一种静态优先级调度算法.它在运行前确定任务的执行顺序,运行时调度开销很小,并且平均CPU利用率较高,易于保持瞬间负载过重情况下的系统稳定性,即当系统不能保证所有任务的时间要求时,调度算法能使其中一部分关键任务始终满足时限要求.并且,RM 算法的最优性保证了其能够广泛地应用于各种静态优先级调度的情况.同时,RM 算法是一种最优的调度算法,最优性保证了其能够广泛地应用于各种静态优先级调度的情况.由于上述特点,RM算法得到了最广泛的研究和应用[25].

Liu和Layland的工作是奠基性的工作,极大地推进了实时调度的研究.但是,Liu和Layland考虑的是任务在临界时刻即最坏情况下的可调度性,他们所提出的 RM 算法可调度性判定条件是一个很“松”的条件,在很多情况下并不适用.虽然后来Burchard等人给出了一个更高的CPU利用率最小上界,也不能解决问题,因为两者都是充分但不必要条件.

Lehoczky 等人给出的 RM 算法的可调度性判定充要条件是一个重大的突破,它提供了一种可用于任何情况下的RM算法可调度性判定的方法.但这一算法的时间复杂性是伪多项式时间,复杂度较高.后来Han和Tyan提出了多项式时间的判定算法,但他们的判定也只是充分判定,而不是充要判定.

另外,上面提到的 RM 算法可调度性判定条件都对调度环境作了一系列理想的假设,忽略了一些重要的影响因素,例如任务调度的时间开销和任务相关性,这就使理想的调度模型比现实情况要简单得多,从而影响了其适用范围.在实际的实现中,这些影响因素都是应当考虑的.引入时间开销及死锁防止机制下的可调度性判定是实时操作系统实现的重要理论基础.

下面,我们将对理想的 RM 模型进行扩展,引入新的影响因素——任务调度的时间开销和任务相关性,即分别去掉两个基本假设,不再假设中断处理、调度、任务切换等时间开销可以忽略不计,并且不再假设任务之间都是不相关的,然后考察 RM 扩展模型的可调度性判定条件.

2 考虑时间开销的 RM 可调度性判定条件

任务调度时处理中断、调度、任务切换等所需要的时间开销在实时调度的实现时是应当考虑的.在可调度性判定时引入这些因素,任务集的可调度性将发生很大的变化.在本节中我们把任务调度的时间开销引入理想的 RM 模型,考察在这种情况下的任务可调度性判定条件.

首先,需要明确任务调度的时间开销都有哪些.这需要对操作系统及其硬件平台的实现方式有一个深入的理解.一般地,任务调度涉及到的时间开销主要有:

C_{int}:中断处理时间;

C_{sched}:确定下一个将被执行的任务所需要的调度时间;

C_{resume}:返回之前被挂起的活动任务所需要的时间;

C_{store}:将活动任务的状态保存到任务控制块所需要的时间;

C_{load}:从运行队列加载一个新的活动任务所需要的时间;

C_{trap}:处理一个任务正常结束所产生的陷入所需要的时间.

不同的操作系统和硬件实现方式造成任务调度所需要的时间开销不同.因此,操作系统的实现方式对任务

可调度性影响很大.根据驱动任务调度发生的方式,实时系统调度可分为中断事件驱动调度和时钟驱动调度[12].

中断事件驱动的实现方式是依赖一个外部的硬件设备,通过产生中断的方式来为任务周期提供信号.这种调度方式又可细分为两种:集成中断事件驱动调度和非集成中断事件驱动调度.在集成中断事件驱动的调度系统中,每一个任务周期都会有一个中断被发布给处理器,中断的优先级与任务的优先级相对应,并且该中断只有在其优先级高于正在执行的任务时才会被处理器响应.所谓集成的含义就是,中断的优先级是取决于其对应任务的优先级的.在非集成中断事件驱动的调度系统中,任务都是通过外部中断启动的.每一个新任务到来时,当前任务都会被中断,即中断的优先级与相关的到达任务的优先级是没有关系的.如果到达的新任务优先级高于正在执行的任务,那么新任务抢占正在执行的任务,否则不发生抢占,但是即使没有发生抢占,中断仍然会发生,并且调度器也会运行.

时钟驱动的实现方式是利用一个可编程的时钟发出的周期性时钟中断来控制调度器的运行,并且调度器维护一个内部的时间值来计算任务周期和决定何时调用这些任务,这种实现方式可以继续被细分为时钟驱动调度和有计数器的时钟驱动调度.时钟驱动调度使用一个周期时钟发出的中断来中断系统运行并调用调度器来更新系统时间,并在需要时重新调度.有计数器的时钟驱动调度维护一个计数器用来限制调度点的数量.在每一个调度点,计数器的值被初始化为到达下一个任务截止期限的时钟周期数.当每一次时钟中断发生时,计数器的值都会递减;当计数器的值减为 0 时,调度器被触发.

下面我们针对这 4 种不同的系统实现,考察引入时间开销的 RM 算法可调度性判定条件.更多的细节可参见文献[12].

2.1 集成中断事件驱动调度的可调度性判定条件

仔细分析集成中断事件驱动调度的过程可以知道,在这种情况下,与调度有关的时间开销有:

$$C_{preempt}=C_{int}+C_{sched}+C_{store}+C_{load},\ C_{exit}=C_{trap}+C_{load}.$$

可以证明:

引理 1. 在集成中断事件驱动调度下,每一个固定优先级周期任务用于调度的最坏时间开销是

$$C_{preempt}+C_{exit} \tag{24}$$

将这些时间开销引入定理 4,得到定理 8.

定理 8. 给定任务集 $S=\{t_1,t_2,\ldots,t_n\}$在集成中断事件驱动调度条件下使用 RM 算法是可调度的,若:

$$\forall i,1\le i\le n,\ \min_{0\le t\le D_i}\left\{\sum_{j=1}^{i}\frac{C_j+C_{preempt}+C_{exit}}{t}\left\lceil\frac{t}{T_j}\right\rceil\right\}\le 1 \tag{25}$$

2.2 非集成中断事件驱动调度的可调度性判定条件

在非集成中断事件驱动调度下,与调度有关的时间开销有

$$C_{preempt}=C_{int}+C_{sched}+C_{store}+C_{load},\ C_{nonpreempt}=C_{int}+C_{sched}+C_{resume},C_{exit}=C_{trap}+C_{load}.$$

可以证明:

引理 2. 在非集成中断事件驱动调度下,相对于 n 个到达的优先级相等或较低的任务,一个固定优先级周期任务 t_i 在时间间隔 t 中最坏情况下的阻塞时间为

$$\sum_{j=i+1}^{n}\lceil t/T_j\rceil\cdot C_{nonpreempt} \tag{26}$$

同样地,将这些时间开销引入定理 4,得到定理 9.

定理 9. 任务集 $S=\{t_1,t_2,\ldots,t_n\}$在集成中断事件驱动调度条件下使用 RM 算法是可调度的,如果:

$$\forall i,1\le i\le n,\ \min_{0<t\le D_i}\left(\sum_{j=1}^{i}\frac{C_j+C_{preempt}+C_{exit}}{t}\left\lceil\frac{t}{T_j}\right\rceil\right)+\frac{\sum_{j=i+1}^{n}\left\lceil\frac{t}{T_j}\right\rceil\cdot C_{cnonpreempt}}{t}\le 1 \tag{27}$$

定理 9 中的判定公式可进一步简化为

$$\forall i,1\le i\le n,\min_{0<t\le D_i}\left(\sum_{j=1}^{i}\frac{C_j+C_{preempt}+C_{exit}}{t}\left\lceil\frac{t}{T_j}\right\rceil\right)+\frac{(n-i)C_{cnonpreempt}}{t}\le 1 \tag{28}$$

2.3 时钟驱动调度的可调度性判定条件

在时钟驱动调度下,与调度有关的时间开销有

$$C_{timer}=C_{int}+C_{sched}+C_{resume},\ C_{preempt}=C_{store}+C_{load},C_{exit}=C_{trap}+C_{load}.$$

在时钟驱动调度下,一个任务被阻塞的时间与时钟中断周期有关,因为调度只有在时钟中断到来时才能发生.假设时钟中断每隔 T_{tic}(T_{tic}称为时钟的解析率)发生 1 次,调度也就每隔 T_{tic} 时间才能发生 1 次.因此,可以证明:

引理 3. 在时钟驱动调度机制下,一个固定优先级周期任务在最坏情况下的阻塞时间为 T_{tic}.

将这些时间开销引入定理 4,得到定理 10.

定理 10. 任务集 $S=\{t_1,t_2,\ldots,t_n\}$在集成中断事件驱动调度条件下使用 RM 算法是可调度的,如果:

$$\forall i,1\le i\le n,\min_{0<t\le D_i}\left(\sum_{j=1}^{i}\frac{C_j+C_{preempt}+C_{exit}}{t}\left\lceil\frac{t}{T_j}\right\rceil\right)+\left\lceil\frac{t}{T_{tic}}\right\rceil\frac{C_{timer}}{t}+\frac{T_{tic}}{t}\le 1 \tag{29}$$

注意,定理 10 中的判定公式多计了一次 C_{timer},因为有一个 C_{timer} 与 T_{tic} 是重叠的.

由定理 10 可以得到,在 RM 算法下,时钟解析率 T_{tic} 允许的最大值是由优先级最高的任务 t_1 决定的:

$$T_{tic}\le T_1-(C_1+C_{timer}\lceil T_1/T_{tic}\rceil+C_{preempt}+C_{trap}) \tag{30}$$

T_{tic} 的最大值可以用迭代的方法求出.首先令 $T_{tic}=T_1$,带入不等式的右边求出一个值,再将所求得的值带入不等式右边,如此循环迭代,直到求出的值不再减小为止,则该值为 T_{tic} 的最大值.下面举个例子说明计算过程:

例 5:假设一任务集 $S=\{t_1,t_2,\ldots,t_n\}$,$T_1=40$,$C_1=25$,$C_{timer}=1$,且 $C_{preempt}+C_{trap}=2$.首先令 $T_{tic}=40$,则

$$T_{tic}<40-(25+1\lceil 40/40\rceil+2)=12,T_{tic}<40-(25+1\lceil 40/12\rceil+2)=9,$$

$$T_{tic}<40-(25+1\lceil 40/9\rceil+2)=8,T_{tic}<40-(25+1\lceil 40/8\rceil+2)=8,$$

所以,T_{tic} 的最大值为 8.

2.4 有计数器的时钟驱动调度的可调度性判定条件

在计数器的时钟驱动调度下,与调度有关的时间开销有:

$$C_{timer}=C_{int}+C_{resume},\ C_{preempt}=C_{sched}+C_{store}+C_{load},\ C_{nonpreempt}=C_{sched},\ C_{exit}=C_{trap}+C_{load}.$$

可以证明:

引理 4. 在一个固定优先级的有计数器的时钟驱动调度实现中,在有 n 个任务的任务集合中某个任务 t_i,其在时间间隔 t 以及时钟解析度 T_{tic} 条件下的最坏阻塞时间为

$$\sum_{j+1}^{n}\lceil t/T_j\rceil C_{nonpreempt} \tag{31}$$

同样地,将这些时间开销引入定理 4,得到定理 11.

定理 11. 在一个固定优先级的有计数器的时钟驱动调度实现中,同时在每一个调度点计数器的值都被初始化为到达下一个任务死线的时钟周期数,任务集 $S=\{t_1,t_2,\ldots,t_n\}$在集成中断事件驱动调度条件下使用 RM 算法是可调度的,如果

$$\forall i,1\le i\le n,\min_{0<t\le D_i}\left(\sum_{j=1}^{i}\frac{C_j+C_{preempt}+C_{exit}}{t}\left\lceil\frac{t}{T_j}\right\rceil\right)+\left\lceil\frac{t}{T_{tic}}\right\rceil\frac{C_{timer}}{t}+\sum_{j=i+1}^{n}\left\lceil\frac{t}{T_j}\right\rceil\frac{C_{nonpreenpt}}{t}+\frac{T_{tic}}{t}\le 1 \tag{32}$$

定理 11 的判定公式可进一步简化为

$$\forall i,1\le i\le n,\min_{0<t\le D_i}\left(\sum_{j=1}^{i}\frac{C_j+C_{preempt}+C_{exit}}{t}\left\lceil\frac{t}{T_j}\right\rceil\right)+\left\lceil\frac{t}{T_{tic}}\right\rceil\frac{C_{timer}}{t}+(n-i)\frac{C_{nonpreenpt}}{t}+\frac{T_{tic}}{t}\le 1 \tag{33}$$

同样可以得到,在 RM 算法下,时钟解析率 T_{tic} 的最大值由优先级最高的任务 t_1 决定:

$$T_{tic}\le T_1-(C_1+C_{preempt}+C_{exit}+C_{timer}\lceil T_1/T_{tic}\rceil+\sum_{j=2}^{n}C_{nonpreempt}) \tag{34}$$

2.5 引入时间开销的可调度性判定条件的仿真和分析

Katcher 等人利用随机产生的 50 个周期性任务集对上述 4 种情况进行仿真比较,任务集的任务个数 n 在 5~80 之间,任务周期之比从 1~100 不等.仿真结果见文献[12].从仿真结果可以看出,同理想情况相比,考虑时间开销的可调度任务集的 CPU 利用率大大降低,特别是当任务个数比较大的时候.即使在任务个数比较小(5 或 10)时,可调度任务集的 CPU 利用率也比理想情况低 20%左右.在非集成中断事件调度下,当任务个数 n 达到 80 以上时,任务集已无法调度,这是因为低优先级任务造成连续性的阻塞.

3 考虑优先级反转的 RM 可调度性判定条件

理想的 RM 算法可调度性条件的一个重要的基本假设(A3)是任务的独立性,即假设任务之间都是不相关的,每个任务的请求不依赖于其他任务请求的开始或完成.但实际上,任务之间往往存在着种种联系,由于共享资源而引起的同步问题是其中十分重要的一种.在多个共享某一个互斥资源的任务当中,如果有一个任务已经进入该资源的临界区,其他任务就必须等待该任务退出临界区,即使它们的优先级比该任务的优先级高,这样实际上就改变了原来由 RM 算法确定的优先级配置(这种情况称为优先级反转,下面我们还会详细讨论).优先级驱动的实时调度都存在这样的问题.

可见,任务的相关性会对实时调度产生极大的影响,进而也影响了任务的可调度性判定,因此必须在可调度性判定中考虑任务相关性.本节中我们将讨论优先级反转对实时调度造成的重要影响及其解决办法,并研究考虑优先级反转的 RM 可调度性判定条件.

3.1 优先级反转

所谓优先级反转是指共享资源的任务在到达其临界区时,因为共享资源的竞争而造成高优先级任务被低优先级任务阻塞的情况.举例来说:

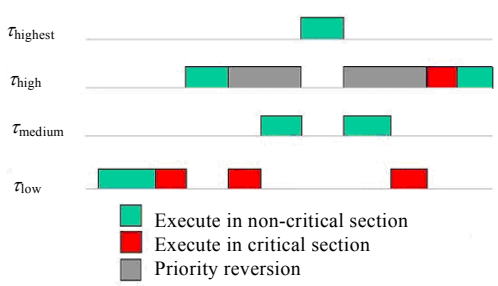

Fig.6 An example of priority reversion
图 6 优先级反转示例

例 6:假设有 3 个任务进程 A,B,C,优先级由高到低排列.A 和 B 同时等待一个事件发生,C 在执行.在某一时刻,C 收到信号量,使得 C 可以访问共享资源 R.C 在该共享资源上进行某些操作,中途被 A 抢占.A 需要访问共享资源 R,但访问该共享资源所需的信号量被 C 拥有,所以 A 被挂起,重新调度后,C 继续执行,中途又被优先级高的 B 抢占,因为 B 等待的事件发生.B 处理该事件直到结束.这时,C 接着运行到结束,释放信号量,A 终于可以访问该共享资源,并继续运行.这样优先级最高的进程 A 由于等待 C 占有的资源,事实上变得比 C 的优先级还要低.更糟糕的状况是在 B 抢占 C 后,更加推迟了 A 的执行.图 6 表示了一个更复杂的优先级反转示例.

在基于优先级的多任务操作系统中,这种情况若不加以处理,就会影响优先级高的任务截止期限的满足.特别是,在 RM 算法中如果出现这种情况,那么优先级从高到低的单调排列的次序就会被打乱,从而使得上面的可调度性判定规则失效.

3.2 优先级反转的几种解决方案

为了解决优先级反转,研究人员提出了几种解决方案,主要有以下几种[26~28].

(1) 不允许任何任务在临界区中执行时被抢占

这种方法简单,但它会影响与临界区无关的其他高优先级的任务的执行.

(2) 优先级继承协议(priority inherence protocol)[26,27]

优先级继承协议的基本思想是:当一个任务进程 J 阻塞一个或多个优先级更高的任务进程时,将 J 的优先级

暂时提高到被它阻塞的所有进程中具有的最高优先级,从而使 J 能够抢占它所阻塞的所有进程而进入临界区,并且不影响与它所进入的临界区无关的其他高优先级的任务的执行.当 J 退出临界区时就恢复原来的优先级.优先级的继承是可以传递的(transitive).例如,假设 J_1,J_2,J_3 是 3 个优先级从高到低排列的任务,如果 J_3 阻塞 J_2,J_2 阻塞 J_1,则 J_3 将通过 J_2 而继承 J_1 的优先级.

例如,对于例 6,优先级继承协议的解决办法是暂时地提高 C 的优先级至 A,使它比其他竞争该资源的进程有更高的优先级,比如进程 B,使得 C 不会被 B 抢占,让 C 执行到释放该信号量,这时再将 C 的优先级降低到原有的级别,A 就可以在 B 之前被调度运行,从而减小了 A 的执行时间.

在优先级继承协议下,一个高优先级的任务只可能在两种情况下被低优先级的任务阻塞.一种情况是直接阻塞(direct blocking),发生在高优先级的任务试图对已经被加锁的信号量加锁的时候.直接阻塞保证了共享数据的一致性.另一种情况是越级阻塞(push-through blocking),如一个优先级居中的进程 J_1 可以被低优先级的进程 J_2 阻塞,如果 J_2 继承了优先级比 J_1 高的进程 J_0 的优先级.越级阻塞则避免了高优先级的任务 J_0 因为优先级较低的任务 J_1 的执行而被间接地抢占.

优先级继承协议在一定程度上解决了优先级反转的问题,实现起来也比较简单.优先级的继承和恢复都应该是不可分割的原子操作,以保证实时系统内部数据结构的一致性.这一过程可以在用户程序中实现,也可以由内核的调度程序来自动实行.但是,该方法存在两个重要的缺陷.首先是会造成死锁.例如,假设在 t_1 时刻,J_2 对信号量 S_2 上锁并进入其临界区.在 t_2 时刻,J_2 试图在 S_2 的临界区内对信号量 S_1 上锁,当此刻优先级更高的 J_1 已经就绪,J_1 抢占了 J_2,对 S_1 上锁.若接下来 J_1 试图再对 S_2 上锁,则将发生死锁.另外,由于允许任务在临界区内嵌套地锁定其他信号量,因此可能引起连续的阻塞链.

(3) 优先级上限协议(priority ceiling protocol)[26,27]

在基本的优先级继承协议的基础上,Sha 等人提出了一个改进的方案——优先级上限协议.与优先级继承协议不同的是,优先级上限协议定义了信号量的优先级上限(priority ceiling)——一个信号量的优先级上限与可能锁定该信号量的所有任务中优先级最高的任务相等;调度的时候不仅比较任务之间的优先级,还要比较信号的优先级上限.当一个进程 J 要进入一个临界区的时候,如果 J 的优先级不比已经被其他进程锁定的所有信号量的优先级上限高,则 J 被阻塞.

下面我们举例来说明优先级上限协议的应用.

例 7: 假设 J_0,J_1,J_2 是优先级由高到低排列的 3 个任务进程.J_0 中包含两个临界区的代码:{…,$P(S_0)$,…,$V(S_0)$…,$P(S_1)$,…,$V(S_1)$,…};J_1 中只包含一个临界区的代码:{…,$P(S_2)$,…,$V(S_2)$,…};J_2 中包含两个临界区的代码:{…,$P(S_2)$,…,$P(S_1)$,…,$V(S_1)$,…,$V(S_2)$,…}.根据优先级上限协议的定义,信号量 S_0 和 S_1 的优先级都等于 J_0 的优先级 P_0,S_2 的优先级都等于 J_1 的优先级 P_1.假设:

1) 在 t_0 时刻,J_2 开始执行,对 S_2 上锁;

2) 在 t_1 时刻,J_1 就绪,抢占 J_2 开始执行;

3) 在 t_2 时刻,J_1 试图对 S_2 上锁,由于 S_2 已被 J_2 锁定,J_1 被阻塞,J_2 继承 J_1 的优先级,以优先级 P_1 进入 S_2 的临界区;

4) 在 t_3 时刻,J_2 对 S_1 上锁,进入嵌套的临界区——此时 S_1 未被其他进程锁定;

5) 在 t_4 时刻,J_2 仍在其嵌套的临界区内执行,而此时 J_0 就绪,抢占 J_2——由于此时 J_0 的优先级 P_0 高于 J_2 的继承优先级 P_1;

6) 在 t_5 时刻,J_0 试图进入其临界区,这时虽然 S_0 没有被其他进程锁定,但是,由于 J_0 的优先级并不比被 J_2 锁定的 S_1 的优先级要高(都是 P_0),因而 J_0 被 J_2 阻塞,J_2 继承 J_0 的优先级,以优先级 P_0 恢复执行;

7) 在 t_6 时刻,J_2 退出临界区,释放 S_1,优先级恢复为 P_1.这时,J_0 抢占 J_2,对 S_0 上锁,进入其临界区,之后退出临界区,释放 S_0;接着进入 S_1 的临界区,然后退出;

8) 在 t_7 时刻,J_0 执行完毕,J_2 恢复执行,优先级为 P_1;

9) 在 t_8 时刻,J_2 退出临界区,释放 S_2,并恢复原先的优先级 P_2,J_1 被唤醒;这时 J_1 抢占 J_2,对 S_2 上锁,进入其临界区,而后退出,进入其非临界区;

10) 在 t_9 时刻,J_1 执行完毕,J_2 恢复执行,直至结束.

整个调度过程请见文献[27].

为了避免死锁和阻塞链,优先级上限协议在直接阻塞和越级阻塞之外引入了一种新的阻塞:上限阻塞.例 7 中的 t_5 时刻,J_0 被 J_2 阻塞就是一种上限阻塞.避免死锁和阻塞链是优先级上限协议的一个重要的优点,但这也意味着引进了不必要的阻塞,因此优先级上限协议是次优化(suboptimal)的协议.另外,优先级上限协议的实现,除了要支持对信号量的锁定和释放之外,还要维护一个按优先级排序的进程队列和一个按优先级排序的当前被锁定的信号量队列.

3.3 在优先级反转情况下的可调度性判定条件

由于优先级反转的解决方案暂时地改变了 RM 算法原来的优先级配置,因而其可调度性条件也发生了变化.在本节中,我们以优先级上限协议为例来考察在优先级反转情况下的可调度性条件.

Sha 等人在理想的 RM 模型研究的基础上,提出了在优先级反转情况下的可调度性判定条件[27].首先假定:任意两个临界区 $z_{i,j}$ 与 $z_{i,k}$ 之间,都有 $z_{i,j}\subset z_{i,k}$,或 $z_{i,j}\supset z_{i,k}$,或 $z_{i,j}\cap z_{i,k}=\varnothing$.令 B_i 为能够对任务 t_i 造成阻塞的所有临界区的集合(嵌套的临界区只计最外一层),$t(x)$为执行临界区 x 所花费的时间,$b_i=\max_{x\in Bi}t(x)$.对于优先级最低的任务 t_n,$b_n=0$.Sha 等人证明了引理 5.

引理 5. 在优先级上限协议下,任务 t_i 的最大阻塞时间为 b_i.

在定理 2 的基础上,Sha 等人证明了定理 12.

定理 12. 一个由 n 个周期任务 $t_1,t_2,\ldots,t_n$ 组成的任务集合能够用 RM 算法调度,如果:

$$\forall i,1\le i\le n,\sum_{j=i}^{i}\frac{C_j}{T_j}+\frac{b_i}{T_i}\le i(2^{1/i}-1) \tag{35}$$

在定理 5 的基础上,Sha 等人证明了定理 13.

定理 13. 一个由 n 个周期任务 $t_1,t_2,\ldots,t_n$ 组成的任务集合能够用 RM 算法调度,如果:

$$\forall i,1\le i\le n,\ \sum_{(k,l)\in E_i}^{i}U_j\frac{T_j}{lT_k}\left\lceil\frac{lT_k}{T_j}\right\rceil+\frac{C_i}{lT_k}+\frac{b_i}{lT_k}\le 1,\ E_i=\{(k,l)|1\le k\le i;\ l=1,\ldots,\lfloor T_i/T_k\rfloor\} \tag{36}$$

需要注意的是,定理 5 的判定条件是充分必要条件,而不等式(36)只是一个充分条件,因为 b_i 表示的是最坏情况.

下面举例说明定理 13 的应用.

例 8:假设一个周期性任务集 $S=\{t_1,t_2,t_3\}$,其中 t_1:$C_1=40,T_1=100,b_1=20,U_1=0.400$;$t_2$:$C_2=40,T_2=150,b_2=30,U_2=0.267$;$t_3$:$C_3=100,T_3=350,b_3=0,U_1=0.286$.

我们用定理 13 可以判定该任务集在 RM 下是可调度的,步骤与用定理 5 对例 2 进行判定的方法类似.

4 结论及未来工作

本文深入系统地讨论了实时系统中最重要的一类静态优先级调度算法——RM 及其扩展算法的可调度性判定问题.

本文的研究表明,在理想的 RM 算法调度判定方法中,算法复杂度和可检测率总是互相矛盾的.算法的复杂度越低,算法的可检测率也就越低.Liu 和 Layland 提出的判定条件最为简单,其可检测率也就最低;Burchard 等人给出更高的 CPU 利用率最小上界,可检测率有所提高,但其算法也复杂了一些;Han 和 Tyan 提出的多项式时间判定算法,其可检测率比前两者都高,但其算法复杂度也比前两者高得多;Lehoczky 等人给出的 RM 算法的可调度性判定充要条件是一个重大的突破,可用于任何情况下的 RM 算法可调度性判定,可检测率最高,在理想的 RM 调度模型中达到 100%,但这一算法的时间复杂性也最高.因此,在实际的应用当中,这 4 种判定算法是互相补充的.对一个任务集进行可调度性判定的时候,可以首先选择复杂度比较低的算法,如果无法判定才选择其他复杂度更高的判定方法.其次,当考虑任务调度的时间开销这一影响因素时,可调度任务集的 CPU 利用率大大降低,特别是当任务个数比较大的时候.即使任务个数比较小,可调度任务集的 CPU 利用率也比理想情况低 20%

左右.在非集成中断事件调度下,当任务个数 n 达到 80 以上时,任务集已无法调度,这是因为低优先级任务造成连续性的阻塞.因此,在实际的实现中,任务调度的时间开销是不容忽略的一个重要因素.引入任务调度时间开销的可调度性判定条件,为固定优先级调度算法在操作系统中的实现提供了有力的可调度性判定工具,具有重大的理论和实践意义.另外,本文还研究了优先级反转情况下 RM 算法可调度性判定的充分条件.优先级上限协议解决了优先级驱动的可抢占性调度下的优先级反转问题,任务优先级反转情况下 RM 算法可调度判定条件的提出,具有重要的意义.引入任务相关性这一条件之后,RM 算法及其可调度性判定的复杂程度将增加.

未来的工作主要有以下几个方面: (1) 多处理器分布状态下基于 RM 算法的可调度性判定问题.(2) 考虑任务相关性和不确定因素的算法及其可调度性判定问题.(3) 开发实用的实时任务可调度性判定的计算机辅助软件,进一步验证算法与实际实现之间的关系.(4) 把优化设计思想(如文献[29])引入到实时系统理论中来.目前的判定方法都是基于 C,T 等给定条件,从实时系统设计角度来讲,一个更有效的方法是如何快速找到 C,T 同时满足可调度和使得某种优化指标最小.这将是一个较为复杂的优化问题,这方面的工作仍很少见.(5) 综合研究可调度性判定与时间开销的度量问题.当考虑时间开销对可调度性的影响时,一些开销在实际系统中不容易度量,如 C_{store} 和 C_{load},在现代计算机系统中有的时间开销甚至无法预先估计.时间开销的度量已有一些研究(参见文献[30]),但在可调度性判定问题上,有关时间开销度量问题的研究尚且不多.

致谢 感谢中国科学院软件研究所互联网软件技术实验室实时系统组的刘军祥、赵欣培、刘晗、赵玉柱等博士、硕士研究生,感谢他们与作者的有益讨论.感谢审稿人对本文提出的宝贵意见.

References:

[1] Burchard A, Liebeherr J, Oh YF, Son SH. New strategies for assigning real-time tasks to multiprocessor systems. IEEE Trans. on Computers, 1995,44(12):1429~1442.

[2] Liu CL, Layland JW. Scheduling algorithms for multiprogramming in a hard-real-time environment. Journal of ACM, 1973,20(1):174~189.

[3] Batptiste P, Pape CL, Nuijten W. Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems. Boston: Kluwer Academic Publishers, 2001. 1~198.

[4] Liu JF, Chou PH, Bagherzadeh N, Kurdahi F. A constraint-based application model and scheduling techniques for power-aware systems. In: Proc. of the 9th Int'l Symp. on Hardware/Software Codesign. Copenhagen: ACM, 2001. 153~158.

[5] Kumar PR, Seidman TI. Dynamic instabilities and stabilization methods in distributed real-time scheduling of manufacturing systems. IEEE Trans. on Automation Control, 1990,35(3):289~298.

[6] Kim YD, Shim SO, Choi B, Hwang H. Simplication methods for accelerating simulation-based real-time scheduling in a semiconductor wafer fabrication faCility. IEEE Trans. on Semiconductor Manufacturing, 2003,16(2):290~298.

[7] Zou Y, Li MS, Wang Q. Analysis for scheduling theory and approach of open real-time system. Journal of Software, 2003, 14(1):83~90 (in Chinese with English abstract). http://www.jos.org.cn/14/83.htm

[8] Jin H, Wang HA, Wang Q, Dai GZ. An integrated design method of task priority. Journal of Software, 2003,14(3):376~382 (in Chinese with English abstract). http://www.jos.org.cn/14/376.htm

[9] Lehoczky JP, Sha L, Ding Y. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In: Proc. of the 10th IEEE Real-Time Systems Symp. Santa Monica: IEEE Computer Society Press, 1989. 166~171.

[10] Shepard T, Gagne JAM. A pre-run-time scheduling algorithm for hard real-time systems. IEEE Trans. on Software Engineering, 1991,17(7):669~677.

[11] Han CC, Lin KJ, Hou CJ. Distance-Constrained scheduling and its anplcations to real-time systems. IEEE Trans. on Computers, 1996,45(7):814~826.

[12] Katcher DI, Arakawa H, Strosnider JK. Engineering and analysis of fixed priority schedulers. IEEE Trans. on Software Engineering, 1993,19(9):920~934.

[13] Jeffay K, Stone DL. Accounting for interrupt handling costs in dynamic priority task systems. In: Proc. of the IEEE Real-Time Systems Symp. Raleigh Durham: IEEE Computer Society Press, 1993. 212~221.

[14] Han CC, Tyan H. A better polynomial-time schedulability test for real-time fixed-priority scheduling algorithms. In: Proc. of the 18th IEEE Real-Time Systems Symp. San Francisco: IEEE Computer Society Press, 1997. 36~45.

[15] Hsueh CW, Lin KJ. Schedulability comparisons among periodic and distance-constrained real-time schedulers. In: Proc. of the 4th Int'l Workshop on Real-Time Computing Systems and Applications(RTCSA'97). Taipei: IEEE Computer Society Press, 1997. 60~66.

[16] Bini E, Buttazzo GC, Buttazzo G. A hyperbolic bound for the rate monotonic algorithm. In: Proc. of the 13th Euromicro Conf. on Real-Time Systems (ECRTS 2001). Delft: IEEE Computer Society Press, 2001. 933~942.

[17] Bini E, Buttazzo GC, Buttazzo G. Rate monotonic analysis: the hyperbolic bound. IEEE Trans. on Computers, 2003,57(7):59~66.

[18] López JM, Díaz JL, García DF. Minimum and maximum utilization bounds for multiprocessor RM scheduling. In: Proc. of 13th Euromicro Conf. on Real-Time Systems (ECRTS 2001). Delft: IEEE Computer Society Press, 2001. 67~75.

[19] Chen Y, Xiong GZ. ImpreCise computation fault-tolerant rate-monotonic scheduling. In: Proc. of the 5th Int'l Conf. on Algorithms and Architectures for Parallel Processing (ICA3PP 2002). Beijing: IEEE Computer Society Press, 2002. 293~296.

[20] Naghibzadeh M, Kim KH. A modified version of rate-monotonic scheduling algorithm and its efficiency assessment. In: Proc. of 7th IEEE Int'l Workshop on Object-Oriented Real-Time Dependable Systems (WORDS 2002). San Diego: IEEE Computer Society Press, 2002. 289~294.

[21] Bini E, Buttazzo GC, Buttazzo G. The space of rate monotonic schedulability. In: Proc. of the 23rd IEEE Real-Time Systems Symp. (RTSS 2002). Austin Texas: IEEE Computer Society Press, 2002. 169~180.

[22] Liu JWS. Real-Time Systems. Beijing: Higher Education Press, 2000.

[23] Zheng ZH. Software Basics of Real-Time Systems. Beijing: Tsinghua University Press, 2002 (in Chinese).

[24] Krishna CM, Shin KG. Real-Time Systems. McGraw-Hill Companies, Inc., 1997.

[25] Atlas A, Bestavros A. Design and implementation of statistical rate monotonic scheduling in KURT Linux. In: Proc. of the 20th IEEE Real-Time Systems Symp. Phoenix: IEEE Computer Society Press.1999. 272~276.

[26] Sha L, Rajkumar R, Lehoczky JP. Priority inheritance protocols: An approach to real-time synchronization. IEEE Trans. on Computers, 1990,39(9):1175~1185.

[27] Goodenough JB, Sha L. The priority ceiling protocol: A method for minimizing the blocking of high priority Ada tasks. ACM Ada Letters, 1988,8(7):20~31.

[28] Wu J, Kuo TW. Real-Time scheduling of CPU-bound and I/O-bound processes. In: Proc. of the 6th Int'l Conf. on Real-Time Computing Systems and Applications. Hong Kong: IEEE Computer Society Press, 1999. 303~310.

[29] Wang YJ, Lane DM. Solving a generalized constrained optimization problem with both logic AND and OR relationships by a mathematical transformation and its application to robot path planning. IEEE Trans. on System, Man and Cybernetics, Part C: Application and Reviews, 2000,30(4):525~536.

[30] Christopher AH, Whalley DB. Automatic detection and exploitation of branch constraints for timing analysis. IEEE Trans. on Software Engineering, 2002,28(8):763~781.

附中文参考文献:

[7] 邹勇,李明树,王青.开放式实时系统的调度理论与方法分析.软件学报,2003,14(1):83~90.http://www.jos.org.cn/14/83.htm

[8] 金宏,王宏安,王强,戴国忠.一种任务优先级的综合设计方法.软件学报,2003,14(3):376~382.http://www.jos.org.cn/14/376.htm

[23] 郑宗汉.实时系统软件基础.北京:清华大学出版社,2002.