一种新的实时多处理器系统的动态调度算法*

乔 颖, 王宏安, 戴国忠

(中国科学院 软件研究所 人机交互技术与智能信息处理实验室,北京 100080)
E-mail: qiaoyingbj@hotmail.com
http://iel.iscas.ac.cn

摘要: 实时多处理器系统的动态调度算法一直是实时系统研究中的重要课题,而评价实时调度算法性能的一个最重要的指标是调度成功率.在近视算法的基础上提出了一种新的实时多处理器系统的动态调度算法——节约算法.在该算法中,提出了一个新的处理器选择策略,从而提高了算法的调度成功率.同时,为了研究节约算法的有效性,对其进行了大量的模拟,分析了一些任务参数的变化对算法调度成功率的影响,并与近视算法的调度成功率进行了比较.模拟结果显示,节约算法的调度成功率要优于近视算法.

关 键 词: 多处理器;实时系统;动态调度;调度成功率;回溯
中图法分类号: TP301 文献标识码: A

随着实时应用的日趋复杂,多处理器系统由于其高性能及可靠性,逐渐成为处理这种复杂应用的有效计算手段.目前,实时多处理器系统已被广泛应用于航天控制、核反应堆控制等领域,而实时多处理器系统的动态调度算法则成为一个重要的研究课题.然而,Mok 等人[1]指出,对于实时多处理器的调度,并不存在最优算法.这一结论导致了我们需要使用启发式搜索来解决这类调度问题.

目前,已有多种使用启发式搜索的实时多处理器系统的动态调度算法.Krithi Ramamritham 等人[2]提出了针对动态实时多处理器调度的近视算法.它在传统的基于启发式搜索的基础上,限定了在一次调度中被考虑的任务数,从而降低了算法的复杂度.同时,该算法首次考虑了任务除了处理器以外还需其他资源的情况.在近视算法的基础上,G.Manimaran 等人[3]又提出了一种新的动态实时多处理器系统的调度算法.它利用了任务的并行性来提高调度算法的性能.Anita Mittal 等人[4]提出的实时多处理器系统中的集成动态调度算法也是基于近视算法的.它是针对在实时多处理器系统中硬实时与软实时任务并存的情况而提出的.但是,在以上这些算法[2~4]中,当选择了某一任务来扩充当前局部调度时,要为该任务分配一个处理器来运行,而这些算法所采用的选择方法是,在可满足该任务截止期的处理器中,选择一个最早可运行的.这种处理器的选择方法将延迟未被调度任务的可开始运行时间,从而可能会造成对这些任务调度的失败.这便阻碍了算法调度成功率的提高.同时,若系统中的任务均不具备并行性,或者系统中的任务均为硬实时任务,则文献[3,4]中的调度算法就不适用了.为了解决上述问题,本文在近视算法的基础上,提出了一种新的实时多处理器的动态调度算法——节约算法.该算法引入了一个新的处理器选择策略.这一策略提高了调度成功率.

本文第 1 节对节约算法所涉及的概念进行了定义.第 2 节是算法描述,并进行了算法举例及分析.第 3 节是对算法性能的模拟研究.第 4 节是结论.

* 收稿日期: 2000-04-12; 修改日期: 2000-07-19
基金项目: 国家自然科学基金资助项目(69896250;79931000)
作者简介: 乔颖(1973－),女,北京人,博士,主要研究领域为实时系统;王宏安(1963－),男,安徽和县人,博士,副研究员,主要研究领域为实时智能,用户界面;戴国忠(1944－),男,江苏无锡人,研究员,博士生导师,主要研究领域为用户界面,计算机图形学,实时智能.

1 基本概念

1.1 任务模型

我们假设一个实时多处理器系统有 m 个处理器($m>1$),且这些处理器是同构的.该系统中的实时任务具有以下特性:

(1) 每一个任务 T 都是非周期性的,其到达时间为 a_T,就绪时间为 r_T,最坏运行时间为 c_T,截止期为 d_T.

(2) 任务是不可抢占的,且相互间是独立的.

(3) 任务不具有并行性.

(4) 除了处理器以外,任务可能还需要其他一些资源,如变量及缓冲区等.同时,每个任务对资源的访问方式有两种:互斥访问和共享访问.

1.2 调度器模型

在这个调度算法中,我们采用集中式调度.在这种调度方案中,有一个处理器作为专门的调度器,所有任务都要先到达这个中心调度器,然后被分配到系统中其他的处理器去执行.每个处理器都有自己的一个调度队列(dispatch queues),这样,在它执行完当前任务之后,就从其调度队列中取出一个任务来执行.调度器与各处理器之间的通信通过这些调度队列来实现.同时,调度器与各处理器并行地运行,它对新到达的任务进行调度,并周期性地对调度队列进行修改[3].

1.3 相关定义

定义 **1**. 在一个调度中,若某任务的时间约束和资源需求都可以满足,则称该任务在这个调度中是可行的;任务的可行性则是指任务在调度中是"可行的"的机率.同时,如果一个任务集中的所有任务在调度中都是可行的,则称该调度对于此任务集来说是一个可行调度[2,3].

定义 **2**. 局部调度是指任务子集的一个可行调度.若一个局部调度被任务集中所剩的任何一个任务扩展后仍是可行调度,则称该局部调度是强可行调度[2,3].

定义 **3**. EAT^s_k是资源 R_k 在共享访问方式下的最早可用时间;EAT^e_k是资源 R_k 在互斥访问方式下的最早可用时间.

定义 **4**. $IEST(T)$为任务 T 的理想最早可运行时间.假设 PE 为一个处理器集合,R_T 为任务 T 所需的资源集合.于是,$IEST(T)=\mathrm{MAX}(r_T,\mathrm{MIN}_{P\in PE}(availtime(P)),\mathrm{MAX}_{R_k\in R_T}EAT^u_k)$.其中,$availtime(P)$表示处理器 P 最早可运行任务的时间;$\mathrm{MIN}_{P\in PE}(availtime(P))$为系统的处理器最早可用时间;$\mathrm{MAX}_{R_k\in R_T}EAT^u_k$表示任务 T 所需资源的最早可用时间,且当资源为共享访问方式时,$u=s$,当资源为互斥访问方式时,$u=e$.

定义 **5**. $AST(T)$为任务 T 的实际开始运行时间.

定义 **6**. $gaptime(T,P)$表示任务 T 的截止期与处理器 P 最早可运行任务的时间之差,即 $gaptime(T,P)=d_T-availtime(P)$,其中 $P\in PE$.

定义 **7**. $avail(T,P)$表示任务 T 在处理器 P 上运行能否满足其截止期.若 $gaptime(T,P)\geq c_T$,则 $avail(T,P)=1$,表示处理器 P 可以提供给任务 T 足够的运行时间来满足其截止期;否则,$avail(T,P)=0$,表示处理器 P 没有能力为任务 T 提供足够的运行时间来满足其截止期.

2 节约算法

2.1 算法描述

近视算法对动态到达、具有资源约束的任务进行了启发式搜索.搜索树中的一个结点表示了一个局部调度.而只有在这个局部调度是"强可行的"的时候,调度才从这个结点来进行扩充.同时,进行可行性检查是针对当前任务队列中的前 K 个任务的,即针对可行性检查窗口中的任务.

在近视算法[2]以及各类基于它的实时多处理器系统的动态调度算法[3,4]中,当选择了某一任务来扩充当前

局部调度且为该任务选择处理器来运行时,所采用的处理器选择方法将会影响调度成功率的提高.针对这一情况,本文提出了节约算法.该算法在近视算法的基础上提出了一个新的处理器选择策略.该策略增大了未被调度任务的可行性,从而提高了调度成功率.

设系统中的资源有 S 个.节约算法要求系统维护一个反映任务资源使用情况的任务-资源列表——RET 表.该表有 S 个入口,每个资源对应一个入口,且拥有两个记录变量.其中一个记录变量负责记录访问该资源的任务数,另一个则负责记录以互斥方式访问该资源的任务数.当生成一个新任务时,要根据任务使用资源的情况来修改各资源所对应的这两个记录变量.

我们将处理器的选择策略命名为一个函数 $choosep(T)$,该函数返回一个处理器的编号,参数 T 表示任务.新的处理器选择策略的基本思想是,在满足任务截止期的前提下,尽量延迟任务的开始运行时间,使所选处理器最早可运行任务的时间尽可能地接近任务的截止期.这样,当该任务运行以后,系统的处理器最早可用时间将可以提前,使得未被调度的任务有较早开始运行的机会,从而增大了未被调度任务的可行性,提高了算法的调度成功率.具体的处理器选择策略如下:

设 $ESTR_T$ 为任务 T 所需资源的最早可用时间.

(1) 若任务 T 除了处理器以外不需要其他资源,则为任务 T 选择一个处理器 P,使得 $gaptime(T,P)=\mathrm{MIN}_{pe\in\{pe|pe\in PE \text{ and } avail(T,pe)=1\}}(gaptime(T,pe))$.

(2) 若任务 T 除了处理器以外还需要其他资源,我们则采用以下方法:

(2.1) 检查系统的任务-资源列表 RET.

(2.2) 若 T 所需要的资源与当前等待调度的任何一个任务所需要的资源都没有交集,则任务 T 对处理器的选择方法与步骤(1)相同;

(2.3) 若 T 对资源的访问方式均为共享式,且等待调度的任务对 T 所需资源的访问方式也均为共享式,则任务 T 对处理器的选择方法与步骤(1)相同;

(2.4) 在其他情况下,分以下几种情况来处理:

(2.4.1) 若 $r_T\le ESTR_T$ 且 $ESTR_T=\mathrm{MAX}_{pe\in\{pe|pe\in PE \text{ and } avail(T,pe)=1\}}(availtime(pe))$,则任务 T 对处理器的选择方法与步骤(1)相同;

(2.4.2) 若 $r_T\ge ESTR_T$ 且 $r_T\ge\mathrm{MAX}_{pe\in\{pe|pe\in PE \text{ and } avail(T,pe)=1\}}(availtime(pe))$,则任务 T 对处理器的选择方法与步骤(1)相同;

(2.4.3) 若 $r_T\le ESTR_T$ 且 $\mathrm{MAX}_{pe\in\{pe|pe\in PE \text{ and } avail(T,pe)=1\}}(availtime(pe))\ge ESTR_T\ge\mathrm{MIN}_{pe\in PE}(availtime(pe))$,则为任务 T 选择一个处理器 P,使得 $avail(T,P)=1$,且 $availtime(P)=ESTR_T$;

(2.4.4) 若 $r_T\le\mathrm{MIN}_{pe\in PE}(availtime(pe))$ 且 $ESTR_T\le\mathrm{MIN}_{pe\in PE}(availtime(pe))$,则为 T 选择一个处理器 P,使得 $availtime(P)=\mathrm{MIN}_{pe\in\{pe|pe\in PE \text{ and } avail(T,pe)=1\}}(availtime(pe))$;

(2.4.5) 在其他情况下,为 T 选择一个处理器 P,使得 $avail(T,P)=1$,且 $r_T\text{-}availtime(P)=\mathrm{MIN}_{pe\in\{pe|pe\in PE \text{ and } avail(T,pe)=1 \text{ and } availtime(pe)\le r_T\}}(r_T\text{-}availtime(pe))$.

综上所述,我们可以对节约算法进行如下描述:

设 K 为可行性检查窗口的大小,d_T 为任务 T 的截止期,W 为所选择的权值.

(1) 将任务队列中的任务按截止期的非递减顺序排列.开始时,局部调度为空.

(2) 通过对可行性检查窗口中的 K 个(或少于 K 个)任务进行可行性检查来决定当前的局部调度是否为强可行的.若是,则 $feasible$=true;否则,$feasible$=false;

(3) if ($feasible$==ture)

(3.1) 计算可行性检查窗口中的任务的目标函数 H 的数值,$H(T)=d_T+W*IEST(T)$;

(3.2) 选择目标函数值最小的任务 T 扩充当前调度,并调用 $choosep(T)$为任务 T 选择处理器来运行.

Else

(3.3) 回溯到上一层调度;

(3.4) 在此层的可行性检查窗口中,选择目标函数值次优的任务 T' 扩充当前调度,并调用 $choosep(T')$为

任务 T' 选择处理器来运行;

(4) 将可行性检查窗口向后移动一个任务;

(5) 重复步骤(2)~(4)的操作,直到以下条件中的任意一个得到满足:

(a) 找到了一个完全的可行调度;

(b) 已经达到了最大的回溯次数或 H 函数的最大估算值;

(c) 已经没有再回溯的可能.

2.2 算法举例及分析

例 1:设任务队列中有 8 个任务,分别为 $T_1,T_2,T_3,T_4,T_5,T_6,T_7,T_8$;系统中有 3 个处理器,分别为 P_1,P_2,P_3;有两个资源,分别为 R_1,R_2,且每个资源只有一个实例.任务 T_1 和任务 T_6 均以共享的方式访问资源 R_1,而任务 T_2 则以互斥的方式访问资源 R_2.各任务的参数见表 1.

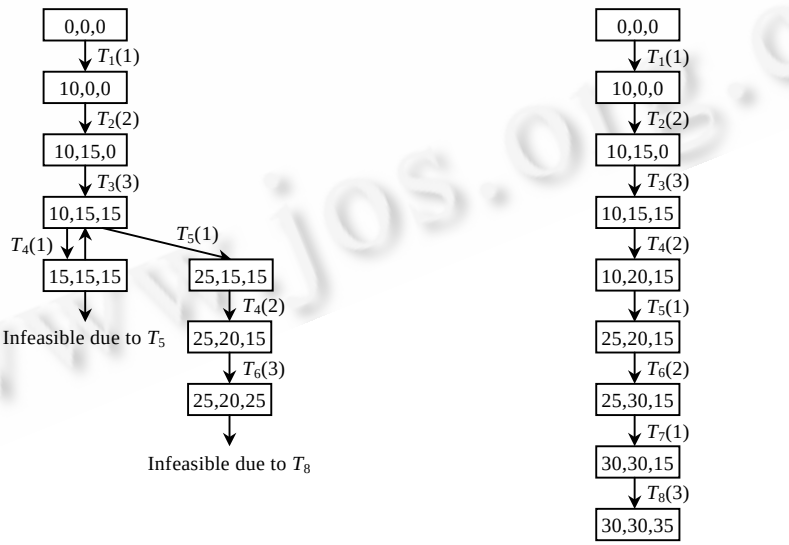
Table 1 List of task parameters

表 **1** 任务参数表

Task①	Ready time②	Execution time③	Deadline④	Resource request⑤
T_1	0	10	12	Shared⑥
T_2	0	15	18	Exclusive⑦
T_3	0	15	20	
T_4	3	5	23	
T_5	5	15	29	
T_6	6	10	30	Shared
T_7	10	5	32	
T_8	12	20	36	

①任务,②就绪时间,③运行时间,④截止期,⑤资源需求,⑥共享式,⑦互斥式.

设可行性检查窗口的大小 K=3,权值 W=1,最大回溯次数为 1,则使用近视算法和使用节约算法进行调度的结果如图 1 所示.图中的每一个方框表示了调度中的一个结点.每个方框中的 3 个数字分别表示系统中 3 个处理器的最早可运行任务的时间.箭头边上的标号 $T_a(b)$表示任务 T_a 在处理器 P_b 上运行.



(a) Search tree for myopic algorithm
(a) 近视算法的搜索树

(b) Search tree for thrift algorithm
(b) 节约算法的搜索树

Fig.1 Search tree for myopic algorithm and thrift algorithm

图 1 近视算法和节约算法的搜索树

在这个例子中,如果使用近视算法,则在对由任务 T_5,T_6,T_7 所组成的可行性检查窗口进行可行性检查时,由

于任务 T_5 是不可调度的,将导致一次回溯.而对由任务 T_7 和 T_8 所组成的可行性检查窗口进行可行性检查时,由于 T_8 的不可调度又将引发回溯,但此时已达到了最大回溯次数,这便导致了对 T_8 调度的失败,因此使用此算法未能找到该任务集的可行调度.如果使用节约算法对该例中的任务进行调度,则对任务 T_4 进行调度时,虽然 T_4 在处理器 P_1,P_2,P_3 上运行都可以满足其截止期,但节约算法却为其选择了 P_2,即在允许的范围内,尽量滞后任务的实际开始运行时间,使其后的任务有较早可以开始运行的机会,从而增大了任务被成功调度的机率.在这里,由于对任务 T_4 所运行的处理器进行了适当的选择,于是,在对由任务 T_5,T_6 和 T_7 所组成的可行性检查窗口进行可行性检查时,其中的任何一个任务都可以用来扩充当前的调度,使得当前的局部调度成为强可行调度,这就避免了近视算法在此处所发生的回溯问题.同理,在对任务 T_6 和任务 T_7 进行调度时,也作相同的考虑,避免了回溯的发生,从而找到了该任务集的一个可行调度.

此外,设任务队列中共有 n 个任务,可行性检查窗口的大小为 K,系统中处理器的数目为 M,资源数目为 S.由于算法要进行 n 步,且每次只对可行性检查窗口中的 K 个任务进行可行性检查和目标函数值的计算,同时,每步在调用 *choosep* 为任务选择处理器时,最多只需从 M 个处理器中进行选择,另外,对任务-资源列表进行检查时,也最多需要检查 S 个入口,因此,节约算法的复杂度为 $O((K+M+S)n)$,记为 $O(K'n)$.由于 M 和 S 均为常数,且在实际应用中一般远远小于 n,因此,节约算法的复杂度与近视算法的复杂度是相同的,并且都与任务队列中的任务数 n 成线形比例.

3 模拟研究

对于实时系统的调度算法来说,最重要的衡量标准是调度成功率.因此,我们在这里主要是对算法的调度成功率进行研究.

调度成功率是指可被算法调度的任务集数目与总共可调度任务集数目的比值.在这里,我们主要讨论算法的最大回溯次数、权值大小、可行性检查窗口的大小、任务使用资源的情况和任务的可延迟度对节约算法调度成功率的影响,并在各种情况下与近视算法的调度成功率进行比较.为此,我们对这两种算法进行了大量的模拟研究.在这些模拟中,我们采用文献[2]中的任务集生成方法,生成了 200 个可调度的任务集,每一个任务集包含 40~80 个任务.模拟参数见表 2.

Table 2 List of simulation parameters

表 **2** 模拟参数表

Parameters①	Explanation②
Procnum	Number of processors in the system③
Resnum	Number of resource types in the system④
Backnum	Number of backtracks permitted in the search⑤
K	Size of feasibility check window⑥
W	The value of weightiness⑦
Max_*exec*	Maximum execution time of tasks⑧
Min_*exec*	Minimum execution time of tasks⑨
R	Laxity of task, this parameter denotes the tightness of the deadline⑩
Use_P	Probability that a task uses a resource⑪
Share_P	Probability that a task uses a resource in shared mode⑫

①参数名,②含义,③系统中的处理器个数,④系统中的资源个数,⑤所允许的最大回溯数,⑥可行性检查窗口的大小,⑦权值大小,⑧任务的最大运行时间,⑨任务的最小运行时间,⑩任务的可延迟度,描述了任务截止期的急迫程度,⑪任务需要访问资源的概率,⑫任务以共享方式访问资源的概率.

在模拟中,我们固定地将调度长度[2]设置为 800.同时,将系统中的处理器个数设为 3,将系统中的资源个数设为 2,并假设每个资源只有一个实例.此外,将任务的最大运行时间和最小运行时间分别设定为 60 和 30.在这种参数取值下的模拟结果分别如图 2~图 6 所示.

3.1 最大回溯次数对调度成功率的影响

在这一模拟中,我们将 *Use_P*,*Share_P*,*K*,*W*,*R* 分别设为 0.2,0.5,7,8 和 0.2.图 2 显示了最大回溯次数对算法调度成功率的影响.从图中可以看出,当算法所允许的最大回溯次数变大时,两个算法的调度成功率都将随之增

加,但是变化的幅度都比较小.同时,在最大回溯次数的变化过程中,节约算法的调度成功率始终高于近视算法.

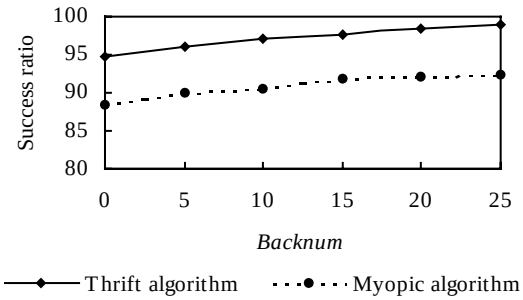


Fig.2 Effect of the maximunumber of backtracks

图 2 最大回溯次数的影响

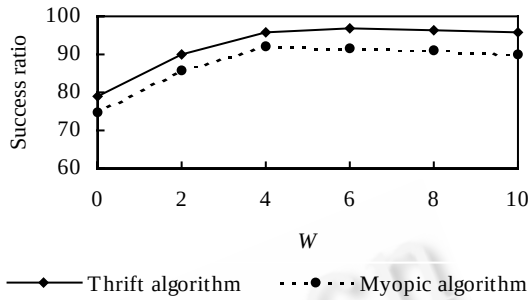


Fig.3 Effect of weightiness

图 3 权值的影响

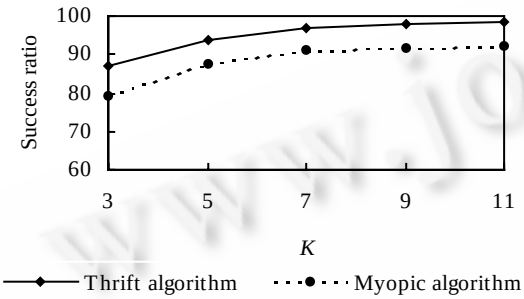


Fig.4 Effect of the size of feasibility check window

图 4 可行性检查窗口的大小的影响

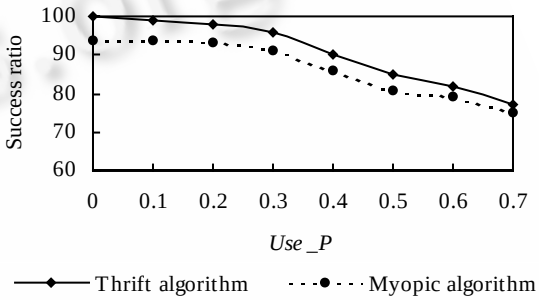


Fig.5 Effect of resource usage

图 5 任务的资源使用情况的影响

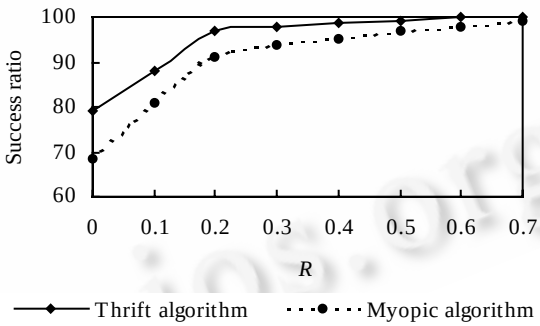


Fig.6 Effect of laxity

图 6 任务的可延迟度对调度成功率的影响

3.2 权值对调度成功率的影响

图 3 反映了权值对算法调度成功率的影响.在模拟中,我们对 *Use_P*,*R*,*Share_P*,*K*,*Backnum* 的取值分别为 0.2,0.2,0.5,7 和 10.从模拟结果可以看出,在权值的变化过程中,节约算法的调度成功率都比近视算法高.同时,当 *W* 在 0~4 之间时,近视算法的调度成功率随之增加,而在 *W*=4 以后,其调度成功率开始下降.节约算法的调度成功率则是在 *W* 为 0~6 之间不断增加,而在 *W*=6 以后开始下降,且下降幅度比近视算法要小.这是因为,当权值很大时,两个算法所采用的启发式搜索函数都将更注重于处理器和资源的使用情况,而任务截止期所起的作用则相对变小了,这就造成了算法性能的下降.而当 *W*=0 时,两个算法的启发式搜索函数又都退化成了最早截止期优先(earliest deadline first,简称 EDF),这时算法的调度成功率是比较低的.

3.3 可行性检查窗口的大小对调度成功率的影响

为了研究可行性检查窗口的大小对调度成功率的影响,我们在模拟中将 *Use_P*,*Share_P*,*W*,*Backnum*,*R* 分别设置为 0.2,0.5,8,10 和 0.2,其模拟结果如图 4 所示.从图中可以看出,在可行性检查窗口的大小相同时,节约算法的调度成功率要优于近视算法.而随着 *K* 的增加,两个算法的调度成功率也都随之增加.这是因为可行性检查窗口的加大会使算法在进行启发式搜索时的预见性更强,从而增加了其调度成功率.

3.4 任务的资源使用情况对调度成功率的影响

图 5 显示了当任务使用资源的情况发生变化时,算法调度成功率的变化情况.在这一模拟中,我们将 *Share_P*,*K*,*W*,*Backnum*,*R* 分别取值为 0.5,7,8,10 和 0.2.从模拟结果可以看出,当任务使用资源的概率增大时,两个算法的调度成功率都将随之降低.这是因为当任务使用资源的概率增加时,任务之间在使用资源时发生冲突的可能性将会增加.这样,与任务的就绪时间和系统的处理器最早可用时间相比,任务的理想最早可运行时间将更依赖于任务所需资源的最早可用时间.这就降低了算法的调度成功率.同时,在 *Use_P* 发生变化的过程中,节约算法的调度成功率始终高于近视算法.而且,*Use_P* 越小,节约算法的调度成功率比近视算法高得越多.这是因为,根据节约算法中的处理器选择策略,*Use_P* 越小,在为任务选择处理器来运行时,能够尽量提前任务运行后系统的处理器最早可用时间的机会就越大,从而越能增加未被调度任务的可行性.显然,这将使节约算法在调度成功率上的提高比近视算法更为明显.

3.5 任务的可延迟度对调度成功率的影响

图 6 表现了任务的可延迟度对调度成功率的影响.在模拟中,我们将 *Use_P*,*Share_P*,*K*,*W*,*Backnum* 分别设定为 0.2,0.5,7,8 和 10.模拟结果显示,当任务的可延迟度增加时,两个算法的调度成功率都将增加,而且节约算法的调度成功率始终高于近视算法.但随着任务可延迟度的增加,节约算法在调度成功率上的优势将逐渐减小.这是由于当 *R* 较大时,将会有更多的机会使得仅采用近视算法中的处理器选择方法而无需使用节约算法中的处理器选择策略就可以满足任务的截止期.

4 结 论

虽然评价实时调度算法的性能指标有很多,但最重要的性能评价指标是调度成功率.目前,已经提出了好几种基于近视算法的动态实时多处理器调度算法.但是,这些算法(包括近视算法本身),在为任务选择运行的处理器时所采用的选择方法将影响到算法调度成功率的提高.针对这一问题,我们提出了一种新的实时多处理器系统的动态调度算法——节约算法.

在节约算法中,我们提出了一个新的处理器选择策略.该策略通过对处理器的适当选择,在保证任务截止期的前提下尽可能地延迟了任务的开始运行时间.这样就提前了任务运行后系统的处理器最早可用时间,增加了未被调度任务的可行性,从而提高了调度成功率.

通过模拟可以看出,节约算法的调度成功率要优于近视算法的调度成功率.同时,该算法具有与近视算法相同的复杂度.

References:

[1] Mok, A.K. Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment [Ph.D.Thesis]. Cambridge, MA: Department of Electronic Engineering and Computer Sciences, MIT, 1983.

[2] Ramamritham, K.J., Stankovic, A., Shiah, P.-F. Efficient scheduling algorithms for real-time multiprocessor systems. IEEE Transactions on Parallel and Distributed Systems, 1990,1(2):184~194.

[3] Manimaran, G., Murthy, C.S.R. An efficient dynamic scheduling algorithm for multiprocessor real-time systems. IEEE Transactions on Parallel and Distributed Systems, 1998,9(3):312~319.

[4] Mittal, A., Manimaran, G., Murthy, C.S.R. Integrated dynamic scheduling of hard and QoS degradable real-time tasks in multiprocessor systems. In: Mok, A.K., Tokuda, H., eds. Proceedings of the 5th International Conference on Real-Time Computing Systems and Applications. Los Alamitos, CA: IEEE Press, 1998. 162~172.

Developing a New Dynamic Scheduling Algorithm for Real-Time Multiprocessor Systems*

QIAO Ying, WANG Hong-an, DAI Guo-zhong

(*Computer Human Interaction and Intelligent Information Processing Laboratory, Institute of Software, The Chinese Academy of Sciences, Beijing* 100080, *China*);

E-mail: qiaoyingbj@hotmail.com

http://iel.iscas.ac.cn

Abstract: Dynamic scheduling algorithms for real-time multiprocessor systems are important components of real-time systems. The most important metric for real-time scheduling algorithms is scheduling success ratio. In this paper, based on the traditional myopic algorithm, a new dynamic scheduling algorithm, called 'thrift algorithm', is proposed for real-time multiprocessor systems. In this algorithm, a new processor selection policy is developed to improve scheduling success ratio. To study the effectiveness of thrift algorithm, an intensive simulation study is made to analyze the impact of several task parameters on its scheduling success ratio and compare its performance to myopic algorithm. The simulation results show that the scheduling success ratio of our new scheduling algorithm is superior to that of myopic algorithm.

Key words: multiprocessor; real-time systems; dynamic scheduling; scheduling success ratio; backtracks

* Received April 12, 2000; accepted July 19, 2000

Supported by the National Natural Science Foundation of China under Grant Nos.69896250, 79931000

++

Pacific Graphics 2002 国际会议

征 文 通 知

Pacific Graphics 2002(PG2002)国际会议将于 2002 年 10 月 9 日~11 日在北京清华大学召开。Pacific Graphics 是在国际上有重要影响的图形学学术会议。会议论文集由 IEEE Computer Society 出版社出版,会议的优秀论文在 Graphical Models 和 Computer Graphics Forum 上以专辑形式发表。会议的语言为英语。有关论文投稿信息,请查阅会议网页。欢迎投稿并参加会议。

会议网页: http://ncc.cs.tsinghua.edu.cn/pg2002

征文截止时间: 2002 年 4 月 30 日

联系方式: (100084)清华大学计算机科学与技术系 胡事民 教授

电话: 010–62782052 传真: 010-62782052 E-mail: pg2002@tsinghua.edu.cn