改进的最小空闲时间优先调度算法*

金 宏+, 王宏安, 王 强, 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)

An Improved Least-Slack-First Scheduling Algorithm

JIN Hong+, WANG Hong-An, WANG Qiang, DAI Guo-Zhong

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

+ Corresponding author: Phn: +86-10-62559307 ext 8808, E-mail: hjin@iel.iscas.ac.cn, http://iel.iscas.ac.cn

Received 2003-05-15; Accepted 2003-09-26

Jin H, Wang HA, Wang Q, Dai GZ. An improved least-slack-first scheduling algorithm. *Journal of Software*, 2004,15(8):1116~1123.

http://www.jos.org.cn/1000-9825/15/1116.htm

Abstract: The LSF (least slack first) algorithm assigns a priority to a task according to its executing urgency. The smaller the remaining slack time of a task is, the sooner it needs to be executed. However, LSF may frequently cause switching or serious thrashing among tasks, which augments the overhead of a system and restricts its application. Assigning a preemption threshold in the scheduling policy can decrease the switching among tasks, however, the existing assigning methods are limited to the fixed priority such that they are not applied to the LSF algorithm. In order to relieve the thrashing caused by LSF, some applicable assigning schemes are presented to the LSF algorithm based on the preemption threshold. Every task is dynamically assigned a preemption threshold that is dynamically changing with the executing urgency of the task and is not limited by the number of tasks. Simulations show that, by using the improved LSF policy, the switching among tasks decreases greatly while the missed deadline percentage decreases. The proposed algorithm is useful for designing and implementing a real-time operating system.

Key words: scheduling; real-time operating system; thrashing; preemption threshold; missed deadline percentage

摘 要: 最小空闲时间优先(least slack first,简称 LSF)算法结合任务执行的缓急程度来给任务分配优先级.任务所剩的空闲时间越少,就越需要尽快执行.然而,LSF 算法造成任务之间的频繁切换或严重的颠簸现象,增大了系统开销,并限制了其应用.在调度策略中设置抢占阈值可以减少任务之间的切换,但现有的抢占阈值设置方法因受到固定优先级的限制而不适用于 LSF 算法.为了减轻 LSF 算法的颠簸现象,基于抢占阈值的思想,提出适用于 LSF 算法的

* Supported by the National Natural Science Foundation of China under Grant Nos.60374058, 60373055 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2001AA413020 (国家高技术研究发展计划(863))

作者简介: 金宏(1962－),男,江苏如皋人,博士,副研究员,主要研究领域为实时系统,智能调度,控制设计,实时数据库;王宏安(1963－),男,博士,研究员,主要研究领域为实时智能系统,实时调度;王强(1972－),男,博士,主要研究领域为实时数据库,实时系统;戴国忠(1944－),男,研究员,博士生导师,主要研究领域为人机交互技术,实时智能,软件工程.

抢占阈值分配方法,动态地给每个任务配置抢占阈值.任务的抢占阈值是随着任务执行的缓急程度不同而动态地变化的,而且不受任务个数的限制.仿真结果表明,通过对 LSF 算法的改进,任务之间的切换大大减少,同时降低了任务截止期错失率.该改进型算法对设计和实现实时操作系统具有一定的参考价值.

关键词: 调度;实时操作系统;颠簸;抢占阈值;截止期错失率

中图法分类号: TP316 **文献标识码**: A

在设计和实现嵌入式实时软件过程中,抢占多任务是一个普遍使用的结构,然而抢占多任务带来的代价包括 CPU 带宽的浪费,同时又增加了内存的总开销[1].当一个任务退出运行时,实时操作系统需要保存它的运行现场信息、插入相应的队列、并依据一定的调度算法重新选择一个任务使之投入运行,这一过程所需时间称为任务切换时间.任务切换时间,又称为上下文切换时间,是评价一个实时操作系统最重要的技术指标之一.几个比较著名的商用实时操作系统,如 QNX,VxWorks,LynxOS 等,对上下文切换、中断延迟、获得或释放信号量的延迟的要求都非常小(一般为几微秒),从而提高实时操作系统运行的及时性和高效性.衡量实时操作系统好坏的主要特点之一就是要求其核心能够支持快速多任务切换、抢占式任务调度等.因此,在保证一定调度性能指标(如截止期错失率)的条件下,尽量减少任务切换是我们在选择调度算法时所要考虑的.

最小空闲时间优先(least slack first,简称 LSF)[2,3]实时调度算法是结合任务执行的缓急程度来给任务分配优先级.任务所剩的空闲时间越少,就越需要尽快执行,这样保证了紧急任务(并非是截止期越早的任务)的优先执行.然而,由于等待执行任务(简称等待任务)的空闲时间是严格递减的,其等待执行的缓急程度也随时间越来越紧急,因此在系统执行过程中,等待任务随时可能会抢占当前执行的任务.LSF 算法造成任务之间的频繁切换或称为颠簸(thrashing)现象较为严重.颠簸现象增大了系统开销,并限制了 LSF 算法的应用.

为此,文献[4]提出一种新的增强 LSF 算法,针对多个具有相同最小空闲时间的任务,减小这些任务之间的上下文切换;对于相同的最小空闲时间,截止期越早就越先执行,执行过程中不进行切换.文献[5]提出一种修改的 LSF 算法,在有两个或更多任务具有最小空闲时间的情况下,它用一个更复杂的算法确定任务执行的顺序.但这些措施都需要对具有相同最小空闲时间的任务进行单独管理,增加了任务调度的复杂度.

利用具有抢占阈值的调度模型,通过控制不必要的任务抢占,可降低由于任务抢占引起的系统开销和过多的上下文切换[1].目前采用的抢占阈值调度模型是在任务集及其任务个数能够预先确定的条件下实现的,没有两个任务具有相同的优先级,且任务的优先级及其抢占阈值的分配都是固定的整数[6].然而,在 LSF 算法中,等待任务的空闲时间是严格递减的,其优先级也是不断提高的,这样无法采用现有的方法[6,7]来确定任务的优先级和抢占阈值.

本文针对任务的空闲时间随时间减小的特点,提出适用于 LSF 算法的抢占阈值动态确定方法,动态地分配任务的抢占阈值,且分配策略不受任务个数和整数抢占阈值的限制.将这种动态抢占阈值设计方法集成到 LSF 算法中,通过仿真表明,与 LSF 算法相比,这种改进的 LSF 算法(以下简记为 ILSF(improved least slack first))减少了任务上下文之间的切换次数(context switching number,简称 CSN),降低了任务的截止期错失率(missed deadline percentage,简称 MDP).MDP 和 CSN 将作为本文后面仿真讨论中的两种性能指标.

1 LSF 调度及颠簸现象

1.1 空闲时间

记 T_{ik} 为任务 T_i 的第 k 个任务样例,其空闲时间 S_{ik} 定义为 T_{ik} 从当前时刻 t 直到其截止期 d_{ik} 的时间与其剩余执行时间 $c_{ik}(t)$之间的差[4],或等价地定义为 T_{ik} 在 t 时刻的相对截止期 $D_{ik}(t)$与其剩余执行时间之间的差[8],

$$S_{ik}(t)=D_{ik}(t)-c_{ik}(t) \tag{1}$$

其中,$D_{ik}(t)=d_{ik}-t$,$d_{ik}=a_{ik}+(k-1)P_i+D_i$,$a_{ik}$ 为 T_{ik} 从第 k 个周期开始到该任务到达所需要的时间,P_i 和 D_i 分别是任务 T_i 的周期和初始相对截止期.显然,$D_{ik}(a_{ik}+(k-1)P_i)\equiv D_i$,$c_{ik}(a_{ik}+(k-1)P_i)\equiv c_i$,其中 c_i 为 T_i 的执行时间.T_{ik} 的初始空闲时间为 $S_{ik}(a_{ik}+(k-1)P_i)=D_i-c_i$.$c_{ik}(t)$又可用下式来表示:

$$c_{ik}(t)=c_i-M_{ik}(t) \quad (2)$$

其中,$M_{ik}(t)$为 T_{ik} 在 t 时刻已经执行的时间,$t\in[a_{ik}+(k-1)P_i,kP_i]$,$M_{ik}(a_{ik}+(k-1)P_i)\equiv 0$.

LSF 调度策略按照任务的空闲时间的非降顺序动态地分配优先级,空闲时间越短,任务的优先级就越高,然后选择具有最小空闲时间的任务进行优先调度.空闲时间算法的意义在于,测量任务需要被调度的缓急程度.假设一个特定任务在变为活动任务时,处理器正被具有更高优先级的其他任务所占用,从而阻止了该任务接受调度处理.随着时间的推移,这个等待任务的空闲时间稳定地减小直至小于正占用 CPU 的任务的空闲时间时,按 LSF 调度策略,处理器必须切换到调度执行该等待任务;当该等待任务的空闲时间减小到 0 时,意味着该任务在其截止期前刚好能够完成,此时,该任务需要立即抢占 CPU 资源以便在其截止期前完成,如果处理器在此关键时刻还不开始调度该任务,那么该任务的空闲时间将变成负值;当空闲时间为负时,意味着该任务不可能在其截止期前完成,此时,没有必要让空闲时间为负的任务占用 CPU 资源,以免造成 CPU 资源浪费.

1.2 颠簸现象

由于等待任务的空闲时间是随时间严格递减的,而当前执行任务(假设为 T_i)的空闲时间保持不变,并假设不为 0(即 $S_i\neq 0$),因此随着调度的执行,总会有某个等待任务 T_j(如果有任务等待调度的话)的空闲时间 S_j 稳定地下降到小于 S_i(即 $S_j\leq S_i-1$).此时,根据 LSF 调度策略,将发生一次任务切换,T_j 将抢占 T_i 占用 CPU 资源.经此任务切换后,T_j 成为当前执行任务,而 T_i 成为新的等待任务.在 T_j 执行期间,其空闲时间 S_j 保持不变,而 T_i 的空闲时间 S_i 将随时间严格递减;当 S_i 减小到小于 S_j 时,T_i 将抢占 T_j 重新占用 CPU 资源,从而又发生一次任务切换,这样 T_i 与 T_j 相互抢占执行;当 T_i 与 T_j 相互抢占执行时,其他等待任务的空闲时间也一直保持严格递减,这样将会出现多个任务相互交叉抢占执行的现象,每次抢占都发生一次任务切换,这种频繁切换现象称为颠簸现象.

在这种多个任务具有最小空闲时间的情况下,LSF 调度将产生严重的颠簸现象,并将引起大量的、不必要的任务切换.如果它们之间没有抢占执行,它们就将一个接一个地满足其截止期.任务切换次数的增加意味着浪费计算资源,从而显示出较差的调度性能,这将对实时操作系统提出更高的实时性能要求.由于任务切换操作需要的时间是不可忽略的,它使得离线可调度性分析变得更加复杂.

2 ILSF 算法

为了减轻颠簸现象造成计算资源的浪费,需要对 LSF 算法进行改进,这种改进将确保任务在尽可能少相互抢占的情况下连续被执行;这种缓减颠簸现象不能通过简单的临时非抢占来实现,因为在任务的空闲时间降为 0 时利用非抢占 LSF 算法可能会造成任务错失截止期.本节将介绍基于抢占阈值的 ILSF 算法.

2.1 抢占阈值

考虑抢占阈值的调度算法集中并包含了完全抢占(或称为纯抢占)和非抢占的特点,当每个任务的抢占阈值与其优先级相同时,模型就还原成完全抢占调度模型,LSF 算法属于完全抢占调度模型;当每个任务的抢占阈值都充分大时,就变成非抢占调度模型.ILSF 算法属于有条件抢占调度模型,通过选择适当的抢占阈值后,我们可以潜在地利用完全抢占和非抢占调度的各自优点,因此研究具有抢占阈值的调度方法具有很重要的意义.

早期的抢占阈值概念是用来扩展固定优先级的抢占调度[6].在具有抢占阈值的调度模型中,每个任务除了分配优先级以外,还分配一个抢占阈值,从而构成一个双优先级系统.一旦任务得到 CPU 资源,它的优先级就提升到它的抢占阈值水平,直到它在执行结束或被其他任务抢占后,再恢复到原来的优先级水平.

不失一般性,假设任务的优先级值越大,其优先级就越高,这样,任务的抢占阈值(h)就应大于其优先级值(p),即 $h\geq p$.根据第 1.1 节的分析,我们允许调度的任务的最小空闲时间为 0,而且任务的优先级和抢占阈值需要是可以变化的;此外,对调度算法来讲,任务的到达时间、夭折时间或提交时间是不知道的,这样,任务集中任务的个数不是预先能够确定的,从而无法采用现有的方法来分配任务的抢占阈值.在第 3 节,我们将介绍适用于 ILSF 算法的抢占阈值分配方案.

2.2 调度过程

记任务 T_i 的优先级值和抢占阈值分别是 p_i 和 h_i,任务 T_j 的优先级值和抢占阈值分别是 p_j 和 h_j.当 T_i 占用 CPU 资源时,T_j 等待执行,这时 T_i 的优先级可看成是提高到其抢占阈值的等级(其实际优先级值仍然是 p_i).随着时间推移,T_j 的空闲时间逐渐递减,从而 p_j 值逐渐递增.当 $p_j>p_i$ 时,T_j 的优先级值已经超过 T_i 的优先级值,这时若按 LSF 调度策略,T_j 将抢占 T_i 执行.但引入抢占阈值后,条件 $p_j>p_i$ 不再是 T_j 抢占 T_i 的充分条件,虽然 T_i 名义上的优先值仍然是 p_i,但其优先级等级已提高到 h_i 层次;只有当 $p_j>h_i$ 时,T_j 才可抢占 T_i,从而延缓了 T_j 抢占 T_i 的进度(有条件抢占的 ILSF 调度).当 $h_i=p_i$ 时,抢占过程变为完全抢占(LSF 调度);当 $h_i=\infty$ 时,变为非抢占调度.

我们可以通过适当选择任务的抢占阈值,减少抢占的次数,减轻颠簸的严重性;还可以根据任务的执行情况,合理地确定任务是否要抢占.此外,在具有同样最小空闲时间的任务组中,我们将选择截止期最早的任务优先进行调度;在具有相同最小空闲时间和相同最早截止期的任务组中,选择先到达任务优先进行调度.

3 抢占阈值的确定

3.1 优先级大小

由第 2.1 节的假设,优先级值越大意味着任务的优先级越高,而任务的空闲时间越短则意味着任务越是紧急,因此任务的优先级与其空闲时间之间是逆反关系.另外,我们考虑最小的空闲时间为 0(也即 $S_{\min}=0$),与之相对应的任务优先级值为最大,记为 $p_{\max}$.记任务的初始空闲时间为 S_0,与之对应的优先级值为 p_0.任务的空闲时间是递减的,即 $S:S_0\downarrow S_{\min}$;而任务的优先级值相应是递增的,即 $p:p_0\uparrow p_{\max}$.这里,$p_{\max}$ 可预先确定.

为了符合任务的空闲时间(S)与其优先级值(p)之间的这种互逆对应关系,不失一般性,我们取

$$p=-S+p_{\max} \tag{3}$$

任务的初始优先级值($p_0=p_{\max}-S_0$)可正可负.特别地,我们可以通过选取足够大的 $p_{\max}$,如 $p_{\max}>\max\{S_0(T_i)\}$,使得 $p_0>0$,从而保证所有任务的优先级值始终大于 0,这里,$S_0(T_i)$表示为任务 T_i 的初始空闲时间.由于 $S\geq 0$(不考虑调度空闲时间为负值的任务,即 $S_{\min}=0$),因此 $p\leq p_{\max}$,这样优先级值为 $p_{\max}$ 的任务为最高优先级任务.一般地,如果没有任务的先验知识,我们可取 $p_{\max}=0$,也即优先级值为 0 的任务最紧急,其优先级为最高.

3.2 确定抢占阈值的几种方案

抢占阈值的确定是 ILSF 算法的关键,直接影响到任务切换的频率,也影响到任务截止期的错失率,还影响到 CPU 的有效利用率.根据前面关于优先值越大,任务的优先级就越高的假设,任务的抢占阈值需要设计成不小于相应的优先级,也即 $h\geq p$.如果取 $h\geq p_{\max}$,则调度策略变为非抢占模型,因此在 ILSF 算法中,抢占阈值的取值范围为$[p,p_{\max}]$;在任务空闲时间不为 0 时,如果不考虑完全抢占和非抢占模式,则抢占阈值的取值范围为$(p,p_{\max})$.

由于任务的优先级 p 是随其空闲时间变化而变化的,因此,任务的抢占阈值不能事先确定(或者离线确定),且随着任务优先级值的变化而变化,这样,现有的关于抢占阈值的取值方法在这里都是不可行的.根据抢占阈值的取值范围以及优先级值计算公式(3),我们给出如下几种适用于 ILSF 算法的抢占阈值确定方案:

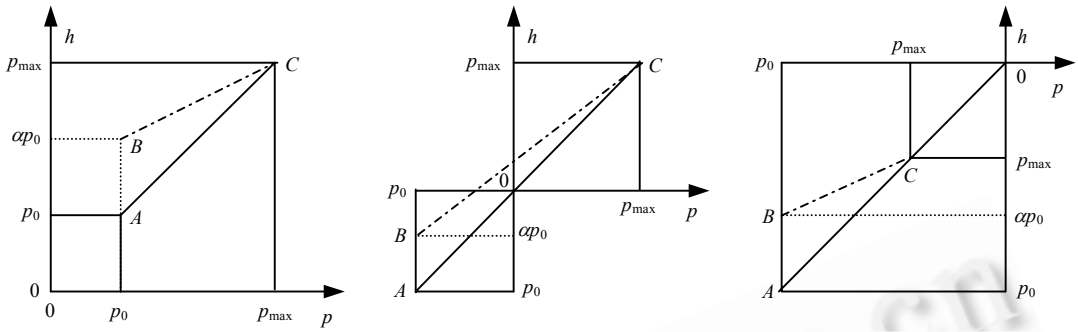
方案 1(比例法). 当取 $p_{\max}=0$ 时,$h=ceil(\alpha\cdot p)$,其中比例系数$\alpha\in(0,1)$为给定常数,函数 $ceil(x)$表示取大于 x 的最小的整数.

方案 2(线性法). 对于任意的 $p_{\max}$,h 值由下式计算:

$$h=\alpha(p_0)p_0+slope\cdot(p-p_0) \tag{4}$$

其中,p,p_0 分别为任务的当前优先级值和初始优先级值,$slope=(p_{\max}-\alpha(p_0)p_0)/(p_{\max}-p_0)$,$\alpha(p_0)$是根据 p_0 的正负来确定的常数.在图 1 中,$A(p_0,p_0)$表示在 LSF 的完全抢占情况下,任务的初始优先级和初始抢占阈值都是 p_0;$B(p_0,\alpha p_0)$表示在 ILSF 的有条件抢占情况下,任务的初始优先级和初始抢占阈值分别是 p_0 和αp_0;$C(p_{\max},p_{\max})$表示任务在其空闲时间为 0 时的优先级值和抢占阈值都是 $p_{\max}$(此时该任务不能被其他任务抢占).图中的实直线 AC 为完全抢占情况下的取值($h=p$),点划线 BC 为有条件抢占情况下由式(4)确定的 h 取值.为确保 $h\in(p,p_{\max})$,当 $p_0>0$ 时,α取为满足 $1<\alpha<p_{\max}/p_0$ 的常数,如图 1(a)所示;当 $p_0<0$ 时,无论 $p_{\max}$ 大于 0 还是小于 0,α都取为满足

0<α<1 的常数,如图 1(b)~图 1(c)所示.在初始执行时刻,$p=p_0$,$h=\alpha p_0$;在任务的空闲时间等于 0 时,$p=p_{\max}$,$h=p_{\max}$.



(a) p_0>0 and 1<α<$p_{\max}/p_0$ (b) p_0<0, $p_{\max}$>0 and 0<α<1 (c) p_0<0, $p_{\max}$<0 and 0<α<1

Fig.1 Assignment scheme of preemption threshold based on Scheme 2

图 1 基于方案 2 的抢占阈值确定方案

方案 3(混合法). 当任务的剩余执行时间小于一个统一的临界值ε时,不允许被抢占,即低优先级任务接近完成时不允许被其他任务抢占;或当任务的相对剩余时间(剩余执行时间/总的执行时间)小于一个统一的临界值ε时,不允许被抢占;或当任务完成量不到α%时采取完全抢占策略,否则采取非抢占策略,其中α为给定常数.

4 运行模型

运行模型是不确定优先级的,具有抢占阈值的抢占调度,这是固定优先级调度模型的推广,也是传统 LSF 调度模型的推广.当一个任务就绪时,它将根据自己的优先级来竞争 CPU.当任务 T_i 占用 CPU 后,它可以被另一个任务 T_j 所抢占当且仅当 T_j 的优先级 p_j 超过 T_i 的抢占阈值 h_i(即 p_j>h_i).在 T_i 执行过程中,由于 p_j 逐步增大而 h_i 保持不变,这时当 p_j 增大到超过 h_i 时,T_j 将抢占 CPU 资源以取代 T_i 调度执行.

对于当前任务集,按任务优先级值的大小,从小到大排列构成当前任务优先级序列表 Q={1,2,...,n},n 表示当前任务集中的任务个数.记 $Pos(i)=k$ 表示任务 T_i 的优先级值在 Q 表中所排的位置,对于任务 T_i,都有(p_i,h_i)与 $Pos(i)$相对应.随着任务的接收、完成提交或夭折,对当前任务优先级序列表 Q 进行插入和移除操作,并从 Q 表中选择最高优先级所对应的任务进行调度.

假设在当前时间单元内,任务 T_i 在执行,并假设其优先级值 p_i 在优先级序列表 Q 中的当前位置为 $Pos(i)=k$.在进入下一时间单元后:

步骤 1. 由于等待任务的空闲时间减小了,相应的任务优先级值增大了,因此需要对 Q 表进行重新排队.

步骤 2. 如果有一个新任务实例加入,则按任务接收策略对 Q 表进行重新排序,这时 $Pos(i)$位置可能发生变化.如果新任务的优先级值高于 p_i,则插入到 $Pos(i)$位置之后,此时,$Pos(i)$位置仍保持为 k;如果新任务的优先级低于 p_i,则插入到 $Pos(i)$位置之前,此时,$Pos(i)$位置变为 $Pos(i)=k+1$.如果有多个新任务实例加入,$Pos(i)$位置则可能要移动多次.不妨假设经多次移动后,$Pos(i)$的新位置为 $Pos(i)=m$.

步骤 3. 在对新加入任务调整结束后,判断 Q 表中在 $Pos(i)=m$ 位置之后有没有比当前正在执行的任务 T_i 具有更高优先级的任务,如果有,则 $Pos(i)$不是 Q 表中的最后一位,排在它后面位置所对应的任务都具有比 T_i 更高的优先级.如果 $Pos(i)$在新 Q 表中位于最后一位,说明 T_i 的优先级为当前任务集所有任务优先级中最高的,这时没有任务抢占发生的可能;如果 $Pos(i)$在新 Q 表中不是位于最后一位,即在 $m+1$,$m+2$,…位置上有其他任务与之对应,则说明当前任务集中有优先级比任务 T_i 的优先级还要高的新加入任务或等待任务存在,这时需要考虑是否有抢占调度的可能性发生.

步骤 4. 不妨假设任务 T_j 的优先级值 p_j 在 Q 表中的位置 $Pos(j)$排在 $Pos(i)$之后(即 $Pos(j)$>$Pos(i)$),并且优先级最高.如果 p_j>h_i,则任务 T_j 将抢占 T_i 所占有的 CPU 进行执行.

步骤 5. 如果 p_j 不大于 h_i,则任务 T_i 继续执行.

步骤 6. 在进入下一个时间单元前,对当前任务集判断有没有任务已经完成,需要提交,或者到了其截止期

还未完成需要放弃时,则按任务完成或夭折策略对 Q 表进行重新调整.如果任务 T_i 完成或需要放弃,则从 Q 表中删除 T_i 的信息,并在进入下一个时间单元后选择任务集中具有最高优先级的任务加以执行;如果 T_i 还未完成且其截止期还未到,则其信息仍将保留在 Q 表中等待进入下一时间单元后继续执行或被其他任务抢占.

步骤 7. 循环往复,直到仿真实验时间结束为止.

5 性能仿真

5.1 仿真条件

(1) 最坏情况执行时间 C_i 在 2~5 个时间单元之间均匀、随机地选择;

(2) 周期 P_i 按照公式 $P_i=N*C_i/\rho$计算[9],其中 N 表示任务集中的总任务数,ρ 表示期望产生的工作负载;

(3) 任务实例 T_{ij} 的周期和最坏情况执行时间分别为 P_i 和 C_i,其相对截止期和初始空闲时间分别为 $D_{ij}=P_i$ 和 $S_{ij}=D_{ij}-C_i$.

不失一般性,在仿真中,假设第 1.1 节中的 $a_{ik}\equiv0$,且用最坏情况执行时间作为任务的实际执行时间进行仿真,所有仿真结果都是由 100 次独立仿真实验获得的统计平均值,每次仿真运行的持续时间为 1 000 个时间单元.为了节省篇幅,这里只采用由方案 1 来确定抢占阈值的 ILSF 算法进行调度.在仿真中,所有任务包括每个周期任务的所有任务实例,一个任务实例代表一个任务.在图 2~图 4 中,实线表示由 ILSF 算法得到的调度结果,虚线表示由 LSF 算法得到的调度结果,后面不再一一说明.

5.2 两种性能指标

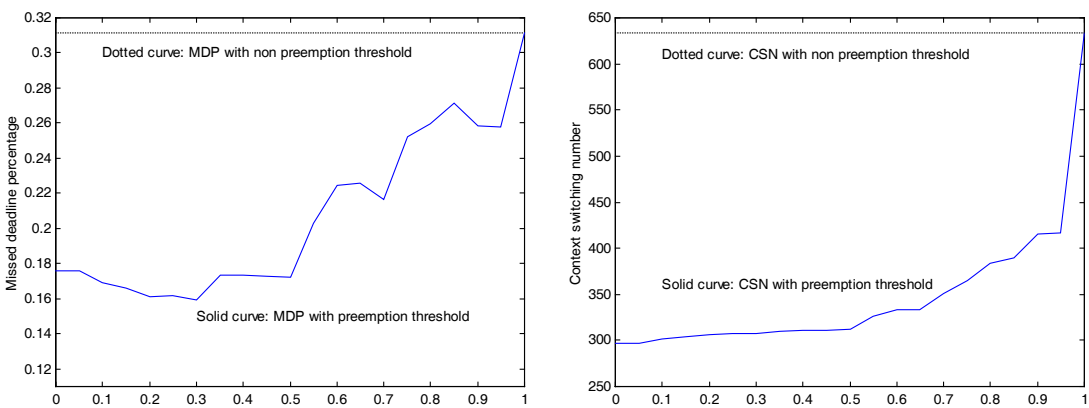
(1) 截止期错失率(MDP)=所有截止期错失的任务数/任务总数.这里,每个截止期错失的任务实例都计算在内;

(2) 上下文切换次数(CSN)=“从未完成任务到抢占任务的切换次数”+“从完成提交任务到另一执行任务的切换次数”.

5.3 仿真比较

(1) 对于不同的比例系数α

取 N=5 和ρ=1.2,图 2(a)和图 2(b)分别给出不同比例系数α(X 轴)时的 MDP 比较和 CSN 比较.在 LSF 和 ILSF 的两种调度模式下:(1) 如图 2(a)所示,由 ILSF 算法得到的平均 MDP 远远低于由 LSF 算法得到的平均 MDP,在比例系数α比较小的情况下,这种差别更大;(2) 由图 2(b),在 LSF 和 ILSF 的两种调度模式下,由前者得到的平均切换次数要高,并远远高于由 ILSF 调度模式下得到的平均切换次数.



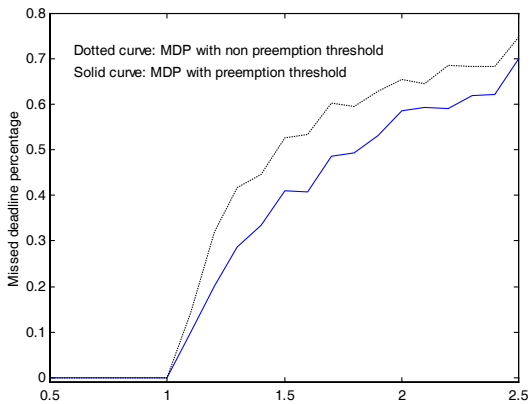
(a) MDP comparison
(a) MDP 比较

(b) CSN comparison
(b) CSN 比较

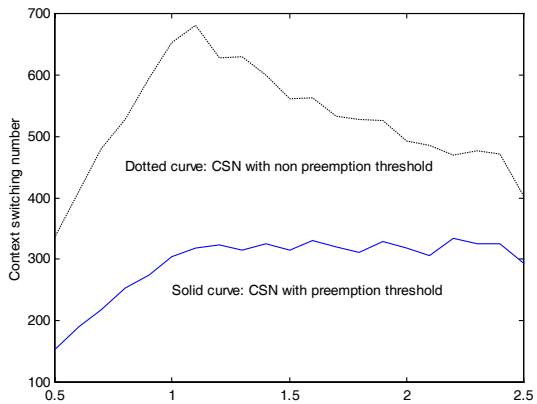
Fig.2 Comparison between MDPs or CSNs for different proportion coefficient
图 2 不同比例系数时的 MDP 比较和 CSN 比较

(2) 对于不同的工作负载ρ

取 N=5 和α=0.5,图 3(a)和图 3(b)分别给出不同工作负载ρ(X 轴)时的 MDP 比较和 CSN 比较.(1) 由图 3(a),当$\rho\leq 1$ 时,由 LSF 和 ILSF 调度模式下得到的平均 MDP 都是 0,这时系统中的所有任务都能得到完成并提交;当$\rho>1$ 以后,由 ILSF 调度模式下得到的平均 MDP 低于由 LSF 调度模式下得到的平均 MDP;在中等工作负载情况下,采用 ILSF 算法进行调度具有明显的改善效果.(2) 由图 3(b),在不同的工作负载情况下,在 ILSF 调度模式下得到的平均切换次数要远远低于在 LSF 调度模式下得到的平均切换次数,并且在 ρ=1 时差别达到最大.



(a) MDP comparison
(a) MDP 比较

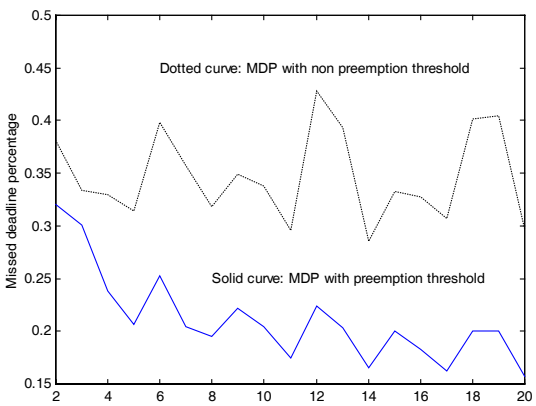


(b) CSN comparison
(b) CSN 比较

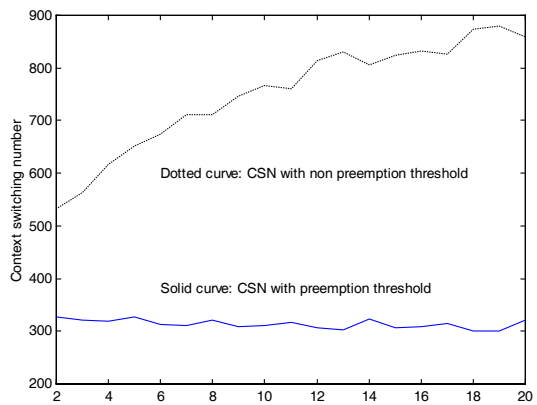
Fig.3 Comparison between MDPs or CSNs for different workload
图 3 不同工作负载时的 MDP 比较和 CSN 比较

(3) 对于不同的周期任务个数 N

取 ρ=1.2 和α=0.5,图 4(a)和图 4(b)分别给出不同任务个数 N(X 轴)时的 MDP 比较和 CSN 比较.对于一定的工作负载和不同的周期任务个数:(1) 由图 4(a),在 ILSF 调度模式下得到的平均 MDP 远低于在 LSF 调度模式下得到的平均MDP,任务个数越大,改善的效果越明显;(2) 由图4(b),在ILSF调度模式下得到的平均切换次数远远低于在 LSF 调度模式下得到的平均切换次数,任务个数越大,改善的效果越明显,而且平均切换次数保持在同一个水平上.



(a) MDP comparison
(a) MDP 比较



(b) CSN comparison
(b) CSN 比较

Fig.4 Comparison between MDPs or CSNs for different task number
图 4 不同周期任务个数时的 MDP 比较和 CSN 比较

6 结 论

由以上分析可以看出,在 ILSF 调度模式下,任务的截止期错失率和任务的切换次数得到大幅度降低.通过对抢占阈值的有条件的限制,减少了由于过多的任务之间的频繁抢占所造成的任务截止期错失,由此可以看出,并不是抢占次数越多调度效果就越好,因此,我们需要适当地控制任务之间的频繁抢占次数.

7 结 语

本文针对不确定优先级的任务集,提出一种改进的具有动态抢占阈值的ILSF动态抢占调度算法.对于ILSF算法来讲:(1) 每个任务实例的到达时间、当前等待调度的任务个数可以是不确定的;(2) 在任务执行过程中,任务实例的优先级、抢占阈值可以是动态变化的;(3) 在任务调度过程中,由于不断有任务实例的加入、完成提交或放弃,当前任务集中的任务总数也是不断变化的.在以上 3 种情况下,现有的抢占阈值确定方法是不可行的.针对 LSF 算法的特点,我们给出 3 种适用于 ILSF 算法的抢占阈值分配方案.

本文将所提出的抢占阈值计算集成到 ILSF 调度策略中,并进行仿真比较.仿真结果显示,过多的任务之间的切换,造成了 CPU 资源的浪费,影响了任务调度的性能保证;在有条件抢占调度模式下,通过对抢占阈值的有条件的限制,任务的切换次数得到大幅度降低,减少了由于过多的任务之间的频繁抢占所造成的 CPU 资源浪费和任务的截止期错失,极大地降低了任务的截止期错失率.由此可见,并不是抢占次数越多,CPU 资源的利用就越好,因此,我们需要适当控制任务之间的频繁抢占次数,从而达到更好的调度效果.

References:

[1] Saksena M, Wang Y. Scalable real-time system design using preemption thresholds. In: Jeffay K, ed. Proc. of the 21st IEEE Real-Time Systems Symp. Los Alamitos: IEEE Computer Society Press, 2000. 25~34.

[2] Abbott R, Garcia-Molina H. Scheduling real-time transactions: A performance evaluation. In: Stankovic JA, Ramamritham K, eds. Advances in Real-Time Systems. Los Alamitos: IEEE Computer Society Press, 1993. 652~663. http://www.vldb.org/conf/1988/P001.PDF

[3] Dertouzos ML, Mok AK. Multiprocessor on-line scheduling of hard-real-time tasks. IEEE Trans. on Software Engineering, 1989, 15(12):1497~1506.

[4] Hildebrandt J, Golatowski F, Timmermann D. Scheduling coprocessor for enhanced least-laxity-first scheduling in hard real-time systems. In: Proc. of the 11th Euromicro Conf. on Real-Time Systems. Los Alamitos: IEEE Computer Society Press, 1999. 208~215.

[5] Oh SH, Yang SM. A modified least-laxity first scheduling algorithm for real-time tasks. In: Gakkai JS, Kwahakhoe HC, eds. Proc. of the 5th Int'l Conf. on Real-Time Computing Systems and Applications. Los Alamitos: IEEE Computer Society Press, 1998. 31~36.

[6] Wang Y, Saksena M. Scheduling fixed-priority tasks with preemption threshold. In: Gakkai JS, ed. Proc. of the 6th Int'l Conf. on Real-Time Computing Systems and Applications. Los Alamitos: IEEE Computer Society, 1999. 328~335.

[7] Kim S, Hong S, Kim TH. Perfecting preemption threshold scheduling for object-oriented real-time system design: From the perspective of real-time synchronization. In: Proc. of the Languages, Compilers, and Tools for Embedded Systems (LCTES 2002) and Software and Compilers for Embedded Systems (SCOPES 2002). Berlin, 2002. 223~232. http://redwood.snu.ac.kr/PAPERS/source/02-lctes.pdf

[8] Jackson LE, Rouskas GN. Deterministic preemptive scheduling of real-time tasks. Computer, 2002,35(1):72~79.

[9] Buttazzo G, Spuri M, Sensini F. Value VS. deadline scheduling in overload conditions. In: Proc. of the 16th IEEE Real-Time Systems Symp. Los Alamitos: IEEE Computer Society Press, 1995. 90~99. http://csdl2.computer.org/dl/proceedings/rtss/1995/7337/00/73370090.pdf