一种面向硬件线程的实时调度算法研究与设计

尹震宇[1], 赵 海[1], 林 恺[1], 孙佩刚[1], 王金英[1,2]
(1. 东北大学信息科学与工程学院, 辽宁沈阳 110004; 2. 中国科学院沈阳计算技术研究所, 辽宁沈阳 110004)

摘 要: 本文根据硬件线程的特征, 为硬件线程调度建立了一个周期与非周期混合线程集的调度模型. 在数学层面描述了硬件多线程调度中每个线程被成功调度的条件判据. 并在此基础之上, 提出一种以截止时间与最坏执行时间差为基本因子的 DR-EDF 算法, 提供了一种实现这种 DR-EDF 算法的硬件多线程控制器的设计原理. 最后用 FPGA 为载体, 实现了一款硬件多线程处理器, 通过实际测试的分析结果, 得出这种面向硬件多线程的实时调度算法在不影响线程集错失率前提下, 提高了嵌入式系统中紧急任务的可调度性.

关键词: 抢占模型; 最早期限优先; 变级最早期限优先; 硬件线程; 实时调度
中图分类号: TN393 文献标识码: A 文章编号: 0372-2112 (2007) 08-1467-05

Research and Design of a Real-Time Scheduling Algorithm for Hardware-Thread

YIN Zhen-yu[1], ZHAO Hai[1], LIN Kai[1], SUN Pei-gang[1], WANG Jin-ying[1,2]
(1. *School of Information Science and Engineering, Northeastern University, Shenyang, Liaoning* 110004, *China*;
2. *Shenyang Institute of Computing Technology, Chinese Academy of Science, Shenyang, Liaoning* 110004, *China*)

Abstract: Based on the characteristics of the hardware scheduling, a scheduling model for the periodic and unperiodic threads in hardware scheduling is proposed. The criterion of the successfully scheduled threads is presented by mathematical description. On this basis, the DR-EDF algorithm based on the deadline and the difference of the worst executing time is proposed. The design of thread-scheduling controller using the DR-EDF is brought forward. And the processor based on the DR-EDF algorithm is achieved on an FPGA. The results of experiment show that the scheduling failure rate will not be affected and scheduling ability for emergency threads is improved.

Key words: Preemption model; earliest deadline first (EDF); dynamic rate earliest deadline first (DR-EDF); hardware-thread; real-time scheduling

1 引言

在嵌入式领域当中, 对线程或者任务调度算法改进已经停留在微调这个阶段. 而且大部分的研究算法改进都着眼于通过减少线程调度过程中所产生的系统资源占用率, 以增加线程的可调度性[1~3]. 然而通过上面这种方式所改进的无论是 RM[4] 算法还是 EDF[5] 算法性能的提升都非常有限. 当 CPU 占用率很高时, 这些改进的算法都不能很好地起到作用.

同时, 由于 EDA 技术的发展以及可编程逻辑器件的集成化程度越来越高, "软件硬件化"的趋势正在成为一个研究的新热点, 因此本文提出了一种用硬件逻辑实现线程调度的功能, 用这种方式避免软件编程所存在的弊端. 从而极大地提高了线程的可调度性、嵌入式系统的实时性, 尤其增加了对紧急事件响应的处理能力.

在本文中, 线程和任务均是指处理器硬件调度的最小程序执行单元, 不做区分.

2 硬件线程抢占式模型

2.1 硬件线程特点

所谓硬件线程, 即通过硬件逻辑进行调度的一系列指令集合. 它们所起到的作用与普通的线程是相同的, 都是为了增加 CPU 的执行效率, 提高系统并行处理能力. 但是由于它们是由硬件逻辑进行调度的, 所以与软件编程实现的线程调度相比, 硬件线程调度有其自身的特点:

(1) 由于线程切换的过程, 主要是对线程所用到的资源进行保护, 会在这种操作过程中浪费大量的时间[4,6]; (2) 硬件线程切换对所需要保护的资源, 可以采取多种方式对其进行, 简单的按页切换只需要一个时钟周期, 复杂一些保护方式也会在几个时钟周期内结束, 因此线程切换时间可以做到很小; (3) 用软件实现的线

收稿日期: 2007-01-08; 修回日期: 2007-04-20
基金项目: 国家 863 高技术研究发展计划(No. 2001AA415320)

程调度, 在抢占调度发生时, 对所需要保护资源是不可预知的, 因此线程切换时间也会不可预知[7,8]. 而对于硬件逻辑实现的线程调度, 每次保护资源的总量不变, 因而切换时间可以预知, 抢占式切换与正常调度时间没有区别.

2.2 硬件线程抢占式模型

硬件线程是线程的一个特殊类别, 普通线程的抢占式调度模型也适用于硬件线程的调度, 只是其中的一些属性发生了变化.

硬件线程在抢占式的调度中存在着周期性与非周期性, 因此定义一个具有 n 个周期性调度线程集 $T=\{T_i \mid i=0,1,\cdots,n-1\}$; 定义一个具有 m 个非周期性调度线程集 $L=\{L_j \mid j=0,1,\cdots,m-1\}$, 从而抢占式周期与非周期总线程集为 $\Omega=T\cup L$, $\Omega=\{\Omega_k \mid k=0,1,\cdots,n+m-1\}$, 其中共有 $m+n$ 个线程参加调度.

另外为了增加建立模型的通用性, 我们把周期调度的线程作为一种非周期线程的特殊类别看待, 通性地描述出周期线程与非周期线程的共同特征, 进一步对周期线程 T_i 特殊属性给予说明.

对这个 Ω 线程集的属性符号定义如下:

(1) F_k 是 Ω_k 的启动时间, $k=0,1,\cdots,n+m-1$.

(2) D_k 是 Ω_k 的截止时间, $k=0,1,\cdots,n+m-1$, 这里设周期线程的截止时间与执行周期为同一个参数.

(3) C_k 是 Ω_k 的最坏执行时间, $k=0,1,n+m-1$.

(4) t 是系统运行时间, 理论上 $0<t\rightarrow\infty$, 但是为了模型的可用性, 设 t 为 $\max(D_k)$.

(5) λ_k 是 Ω_k 在系统运行时间 t 里所运行的次数. $k=0,1,\cdots,n+m-1$. 对于周期线程的执行次数 $\lambda_k=\lfloor t/D_k\rfloor$. 而对于非线程周期在系统时间 t 内的理论最多执行次数为 $\lambda_k=\lfloor t/C_k\rfloor$.

(6) $E_{k,v}$ 是 Ω_k 第 v 次执行所经历的时间, $k=0,1,\cdots,n+m-1$. $E_{k,v}$ 含盖了所有第 v 次执行当中抢占它的线程执行时间、系统调度时间, 以及线程 Ω_k 本身最坏执行时间 C_k.

(7) $\alpha_{k,u}$ 是 Ω_k 在一次执行的过程中被 Ω_u 直接抢占的次数, $k=0,1,\cdots n+m-1$, $u=0,1,\cdots,k-1$.

(8) Π_k 是 Ω_k 在一次执行的过程中抢占此线程的周期线程序列, $\Pi_k[i]$ 表示 Π_k 中第 i 个元素, $k=0,1,\cdots,n+m-1$, $i<k$, β 为 Π_k 中序列的长度, $\beta<k$.

(9) Γ_k 是比 Ω_k 优先级高的非周期线程队列, $\Gamma_k[j]$ 表示 Γ_k 中第 j 个元素, $k=0,1,\cdots,n+m-1$, $j<k$, δ 为 Γ_k 中序列的长度, $\delta<k$.

在实时系统当中, 一个线程 Ω_k 是否可调度的条件是这个线程必须在规定的时间 D_k 完成执行. 如果超过这个时间段 D_k, 那么线程 Ω_k 就是不可调度的, 也就是必须满足下面这个判据(1):

$$F_k+C_k+\sum_{i_1=0}^{\beta-1}(\alpha_{k,\Pi_k[i_1]}\cdot E_{k,\Pi_k[i_1]})+\sum_{i_2=0}^{\delta-1}(\alpha_{k,\Gamma_k[i_2]}\cdot E_{k,\Gamma_k[i_2]})\leqslant D_k \tag{1}$$

Ω_k 可调度判据所描述的内容为从 Ω_k 启动时间起, Ω_k 的执行时间与直接抢占 Ω_k 的线程执行所经历时间的总和, 必须小于截止时间 D_k. 对于线程集 Ω 如果是可调度的, 那么 Ω_k, $k=0,1,\cdots,n+m-1$ 则都是可调度的. 不难推算, 周期线程抢占 Ω_k 的次数是可预知为 $\lfloor\sum_{i_1=0}^{\beta-1}D_k/D_{\Pi_k[i]}\rfloor$, 而非周期抢占次数是不可预知的, 因此对于非周期抢占只能定性描述, 定义一个非周期线程的执行频率空间 $h_k\in\left(0,t/D_k\right)$, 总抢占线程 Ω_k 的次数

$$N_k\in\left[\left\lfloor\sum_{i=0}^{\beta-1}D_k/D_{\Pi_k[i]}\right\rfloor,\left\lceil\sum_{i=0}^{\beta-1}D_k/D_{\Pi_k[i]}+\sum_{j=0}^{\delta-1}D_k/D_{\Gamma_k[j]}\right\rceil\right] \tag{2}$$

所以在系统运行时间 t 内一共发生的抢占次数为 $\sum_{k=0}^{n+m-1}N_k$.

从 N_k 的取值范围可以看到, 由于存在对非周期线程 L_j 的调度, 因此发生的抢占次数范围与偶然性非常大. 传统用软件编程实现的线程调度方式在线程切换过程中需要耗费大量的处理时间, 如果线程切换过于频繁将导致线程可调度性急剧下降. 因而, 现存的很多调度算法都是以减少线程切换次数为主要改进方法.

而对于硬件线程调度则不存在线程切换所带来的CPU 处理能力消耗, 它是最接近理想模型的线程调度方式. 因此在嵌入式实时系统中存在的多硬件线程调度的问题, 线程切换次数已经不再是增加可调度性的主要参数, 而转变为研究如何满足每个线程都能够在截止时间前执行完毕的问题. 因此对于硬件线程调度模型, 为了满足线程集 Ω 的可调度性, 问题已经转化为已知硬件线程集 Ω 中线程 Ω_k 的 F_k、D_k、C_k, 根据公式(1)求解一个序对集合 O, 其中包含元素为 $o_g<k,\rho>$, 其表示硬件线程 Ω_k 的优先级是 ρ, $\rho=0,1,\cdots,n+m-1$, $g=0,1,\cdots,n+m-1$. 用这个序对集合 O 调度就可以最大程度地保证线程集 Ω 是可调度性以及在实际工程中的实用性.

3 DR-EDF 算法

根据文献 4 可知 RM 算法是以最短 C_k 优先, 它采用的是离线计算序对集合 O, 然后在线按照 O 中的序对所表示的优先级调度线程集 Ω. 而由文献 5 可得 EDF 算法是采用最早时限 D_k 优先, 动态的改变序对集合

O,实时调整处于就绪队列中线程的优先级,从而提高整个 Ω 的可调度性. 但是这两种算法在嵌入式实时系统中缺乏工程实际应用性. 在硬件线程调度中虽然可以把线程的切换时间忽略,但是 RM 算法没有考虑线程的截止时间 D_k,这导致这种算法的实时性无法得到保障. 而动态的 EDF 算法虽然在理论上满足了实时性能,但是实际应用却缺乏对紧急事件线程的处理能力. 在实际的嵌入式系统中存在这样的线程,它们的执行时间与截止时间的差距非常小,而它们的值又相对较大. 这种特殊的线程通常都是为了嵌入式系统中处理紧急任务而设定的,它们一般都是非周期任务,并不经常出现. 但是一旦它们出现就必须满足其调度性. 所以在 Ω 定义一种紧急事件线程 Δ_w,它的特性为:$(D_w - C_w) \rightarrow 0$, $\omega \ll n+m$. 为了使 EDF 算法更加适合嵌入式系统的实际工程应用,为 Δ_w 的可调度性增加权重,把 EDF 算法改进为 DR-EDF 算法.

由于 DR-EDF 算法是面向硬件逻辑所实现的线程调度,所以此算法的设计需要考虑到硬件逻辑实现的可行性. 因此 DR-EDF 算法描述如下:

在硬件线程调度对线程切换次数不敏感的条件下可以假设在 Ω 中的所有周期调度线程都可认为在时刻 0 时同时启动,$F_k = 0$,假定在时刻 0 时,虽然最多可产生 $m+n-1$ 次线程切换,但是并不会影响线程调度的有效性. 所以 DR-EDF 算法忽略了 F_k,采取以线程的执行时间 C_k 与线程截止时间 D_k 之间的差 ξ_k 作为基本因子,动态的构造序对集合 O,使整个硬件线程调度最先满足对 Δ_w 的可调度性.

∂t 为序对集合 O 重新构造时间片,$\partial t > 0$;q_k 为 Ω_k 从启动到执行完毕所经历的时间片数,$q_k > 0$.

定义一个 $\boldsymbol{J}$ 矩阵作为序对集合 O 的重构判据,$\boldsymbol{J} = [J_0, J_1, \cdots, J_{n+m-1}]$;

$$\boldsymbol{J}_k = \begin{bmatrix} j_{k,0} \\ j_{k,1} \\ \vdots \\ j_{k,n+m-1} \end{bmatrix}, k = 0, 1, \cdots, n+m-1 \tag{3}$$

$$\boldsymbol{J} = \begin{bmatrix} j_{0,0} & j_{1,0} & \cdots & j_{n+m-1,0} \\ j_{0,1} & j_{1,1} & \cdots & j_{n+m-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ j_{0,n+m-1} & j_{1,n+m-1} & \cdots & j_{n+m-1,n+m-1} \end{bmatrix} \tag{4}$$

在时刻 0 时的初始判据因子,$\xi_k = D_k - C_k$;为了适应硬件逻辑的实现把判据因子 ξ_k 改造为 $\xi'_k = \xi_k \cdot \tilde{\omega}$;$\tilde{\omega}$ 为根据具体系统的时钟主频、时间片大小、系统数据总线宽度等参数所制定的一个参考因素. 把判据改造成更加适合硬件逻辑存储与处理,通常 $\tilde{\omega} = 1/\partial t$.

$$j_{x,y} = \begin{cases} 0, \xi_x \leqslant \xi_y \\ 1, \xi_x > \xi_y \end{cases}, x, y, k = 0, 1, \cdots, n+m-1 \tag{5}$$

从而由(4)与(5)得到 $\boldsymbol{J}$ 为一个 0/1 矩阵:

$$\boldsymbol{J} = \begin{bmatrix} 0 & j_{1,0} & \cdots & j_{n+m-1,0} \\ j_{0,1} & 0 & \cdots & j_{n+m-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ j_{0,n+m-1} & j_{1,n+m-1} & \cdots & 0 \end{bmatrix} \tag{6}$$

从 $\boldsymbol{J}$ 中得到 Ω 的优先级队列向量为 $p = \{|J_0||J_1|\cdots|J_{n+m-1}|\}$,与序对集合 O 中的元素 $o_g < k, |J_k| >$. 由此在一个 ∂t 内 DR-EDF 算法结束,硬件线程调度控制器可以按照 P 的内容调度各个线程.

在下一个 ∂t 内则重新构造 $\boldsymbol{J}$,公式(5)变为:

$$j_{x,y} = \begin{cases} 0, (\xi_x - q_x) \leqslant (\xi_y - q_y) \\ 1, (\xi_x - q_x) > (\xi_y - q_y) \end{cases} \tag{7}$$

这个 DR-EDF 线程调度策略不一定是线程集可调度性最大化,但其一定是最实用的.

4　DR-EDF 算法硬件逻辑映射原理与设计实例

4.1　DR-EDF 算法硬件逻辑映射原理

DR-EDF 算法的构造虽然可以用软件编程实现,但是它更适合采用硬件逻辑实现. $\boldsymbol{J}$ 判据矩阵就是为电子所能表示的 2 进制原则所设计的. 在 FPGA 内部实现一个矩阵的方法可以最大程度的节约逻辑资源,达到并行输入输出的能力,以此把判断线程集的判据 $\boldsymbol{J}$ 的计算所消耗的时间缩减到一个时钟周期内.

DR-EDF 算法硬件逻辑映射包括:时间片 ∂t 定时器、ξ_k 判据寄存器组、比较控制器组、$\boldsymbol{J}$ 矩阵并行存储器. 时间片 ∂t 的范围受到时间片定时器计数寄存器宽度限制,但是 ∂t 会存在两个特殊值:全"1"为指定系统中最大时间片的值、全"0"表示线程将以一个指令为粒度进行调度.

ξ'_k 判据寄存器组的初始值通过离线计算获得,而且必须为此嵌入式系统中所有可能存在的线程计算出它们的基因 ξ_k,并在嵌入式系统开始之前配置到一个固定的存储区域当中,当某个线程进入到就绪队列参与调度时,把对应的基因 ξ'_k 调入到判据寄存器组对应位置中,而对于未进入到就绪队列的线程则把对应的判据寄存器设为全"1",这样的设计不需要动态的分配寄存器单元,可以与比较控制器组采用直连方式,减少 DR-EDF 算法所需的时钟周期.

比较控制器组是 DR-EDF 算法中最消耗硬件逻辑资源的单元,所有的判据寄存器采用的是全连通比较网络,因此在一个时钟周期内就可以通过此网络得到比较结果,并且并行的输入到 $\boldsymbol{J}$ 矩阵并行存储器中.

对于 $\boldsymbol{J}$ 矩阵并行存储器,它是一个 $(n+m)$ 的方型

0/1 矩阵, 全连通比较得出的结果并行存储到这个矩阵当中. 可以按$(n+m)$宽度的字直接读取此矩阵内容. 但实际上并不需要对 $P=\{|J_0||J_1|\cdots|J_{n+m-1}|\}$所有信息进行掌握, 只要获得在这个时间片需要运行的线程 ID 即可, 因此把矩阵 $\boldsymbol{J}$ 的每一列J_k 进行按位逻辑或运算:

$$p_k=j_{k,0}\vee j_{k,1}\vee j_{k,2}\cdots\vee j_{k,n+m-1} \quad (8)$$

运算结果p_k 为 0 的那一列所代表的线程 Ω_k 为这个时间片需要切换的到运行状态的线程. 这样改进不仅所用的逻辑门比采用按列取模后再比较节省了很多, 信号通过时间也大大缩短.

4.2 采用 DR-EDF 算法实现的硬件多线程调度设计实例

这种内部集成了硬件线程调度控制器的处理器内核采用的是 8 位 RISC 结构, 最大可以支持 8 个周期硬件线程与 8 个非周期硬件线程的调度控制. 时间片寄存器的宽度为 8 位, 同时设置了多个硬件线程控制、中断与线程绑定、判据基因等特殊功能寄存器. 这些寄存器都被统一到片内 RAM 地址中, 用户可以用访问普通 RAM 的指令对其进行读写操作, 以达到操控硬件线程调度控制器的作用.

为了实现硬件线程的调度, 程序存储空间与数据存储空间需要进行相应的改造设计. 此款处理器的寻址范围为 1MB, 设置了 15 个程序段寄存器(CS0~CS14), 可通过设置这些程序段寄存器的值, 动态地为各个线程在 1MB 程序存储空间内以 1KB 为单位划分程序存储空间, 用户对于每个线程的操作都是从逻辑地址 0 开始的, 不必考虑实际的存储位置. 程序存储空间划分如图 1 所示:

图 1 硬件多线程处理器程序空间结构

对于数据空间的划分就更复杂一些, 在处理器中每个硬件线程都拥有自己的私有数据存储空间. 并且 16 个线程还共同拥有一个公共数据缓冲存储区, 用于线程之间的通讯. 其中, 私有的存储空间不允许其他硬件线程访问, 因此对于每个硬件线程这个私有的存储空间是"安全"的. 而对于公共数据缓冲存储区, 任何线程任何时间都可以进行访问, 这个缓冲区也可以被称作线程通讯缓冲区. 特殊功能寄存器区包括了所有的用户可以控制多线程调度的内容. 用户与处理器之间的交互都是通过这个区域的数据传递进行的. 用户以读写特殊功能寄存器的手段达到外围电路控制、线程控制、中断控制等操作.

本文的这款硬件多线程调度处理器是根据嵌入式系统中面向设备控制的具体应用进行设计的. 实际的应用中一般周期性线程的实时性要求比较低, 而非周期线程几乎都是由中断事件所引起的, 实时性要求比较高. 因此, 设计这款处理器采取的线程调度策略是在假定非周期线程的优先级总是高于周期线程的优先级, 并且把非周期线程与中断源动态绑定, 由中断事件去驱动非周期线程进入到就绪队列当中. 而对于周期性线程只是采取时间片轮询的调度策略, 一旦有非周期线程进入就绪队列, 周期线程即让出 CPU, 所以 DR-EDF 算法只在非周期线程调度中起作用. 处理器中集成的硬件线程调度控制器的逻辑电路结构如图 2 所示:

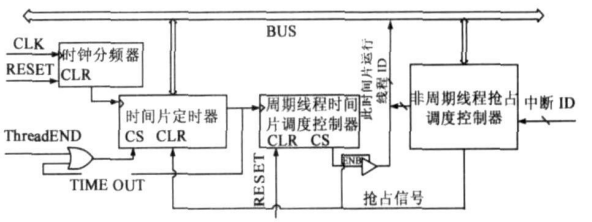

图 2 硬件多线程调度控制器逻辑电路结构

从图 2 中可以看到, 利用时钟分频器把主时钟频率分频后的时钟信号, 为周期线程时间片调度控制器提供时钟驱动信号. 当时间片基准寄存器改变, 或者计数溢出和一个线程执行完毕后, 重载时间片计数寄存器的初值. 复位或者存在非周期硬件线程进入到运行状态时, 时间片计数寄存器清零.

由于支持的最大周期硬件线程数为 8, 每当时间片定时器输出的溢出信号有效, 硬件线程计数器做 3 位数加 1 运算, 用以实现硬件线程 ID 的轮询.

非周期线程抢占调度控制器内部则根据 DR-EDF 算法动态的调整处于就绪队列中的线程调度. 如果非周期线程调度控制器输出抢占信号, 那么周期线程调度控制的输出则被变为高阻态.

在非周期线程调度控制器中 $\boldsymbol{J}$ 是一个 8×8 的方型 0/1 矩阵, 使用可以按位(bit)访问的 8 字节并行 RAM 实现. 全连通比较器组共有 64 个比较器, 每个比较器的输出结果并行的存储到 $\boldsymbol{J}$ 矩阵 RAM 中. 与此同时设置了 8 个按位与逻辑操作组, 产生的结果输入到一个运行线程标志寄存器中, 每一位对应一个非周期线程, 把值为 0 的位所对应的线程 ID 号作为调度控制器的输出.

5 DR-EDF 算法性能评价

在实时系统中, 性能的评价不能仅仅依赖于线程集调度的错失率. 所谓错失率是指系统中未被调度的线程占参与调度线程总数的百分率. 以往的 EDF 算法性能比较仅仅以错失率为评判依据, 对所有线程的其他权重因素考虑的不够全面. 下面是一个实际工程中

线程集调度实例,分别以 EDF 算法和 DR-EDF 算法对其进行调度,所使用 FPGA 芯片为 Xilinx 公司 XC3S2000,处理器使用资源约 67%,其中 DR-EDF 硬件线程调度控制器使用芯片资源约 1%,如表 1 所示. 此嵌入式实时系统主频为 8MHz,时间片寄存器为 8 位,且时间片定时器的时钟信号为主频的 8 分频信号,所以此试验所能采用的时间片,为 10μS,表中的 $q_k = 0$ 表示线程 Ω_k 在这个时间片刚刚加入到就绪队列,$q_k > 0$ 则线程 Ω_k 已经在就绪队列中经历了 q_k 个时间片的时间.

表 1　EDF 与 DR-EDF 调度算法性能分析

线程 Ω_k	D_k (ms)	C_k (ms)	q_k	EDF 调度判据 $D_k - \partial t \cdot p_k$ (ms)	ξ_k	EDF 优先级	DR-EDF 优先级	满足调度 EDF	满足调度 DR-EDF
Ω_0	0.5	0.045	0	0.5	45	0	2	是	否
Ω_1	4.2	0.135	11	4.09	395	4	4	是	是
Ω_2	3.6	0.577	31	3.09	271	3	3	是	是
Ω_3	0.769	0.75	0	0.769	1	1	0	否	是
Ω_4	15	3.588	102	13.98	1039	6	5	是	是
Ω_5	5.9	2.038	370	2.2	16	2	1	是	是
Ω_6	12	0.0085	117	10.83	1082	5	6	是	是

表 1 为测试结果,EDF 算法与 DR-EDF 算法均有一个任务无法完成,即对线程集的调度错失率都是 14.3%,根据实际测试可知,EDF 算法对 Ω_3 无法满足调度,DR-EDF 算法对 Ω_0 无法满足调度. 但是对于 Ω_3 所代表的是紧急事件处理线程,实时系统应该优先对其进行调度处理.

因此,DR-EDF 在没有增加 EDF 算法复杂度的情况下,为紧急事件线程的优先调度增加了权重,在基本不影响整个线程调度错失率的情况下,提高了对紧急事件线程调度的成功率,从而保证了这个实时系统的工程可应用性能.

6　结论

在嵌入式实时系统中,由于用软件编程所实现的线程调度在线程切换的过程中存在着大量 CPU 资源消耗. 而且在小型嵌入式系统存储资源非常有限的情况下,无法使用内核在几 K 甚至几十 K 字节的线程调度程序. 因而,本文另辟蹊径由硬件逻辑实现原本由软件编程的线程调度功能. 用硬件实现的线程调度控制,避免了线程切换过程中对 CPU 的消耗,而且真正实现了存储空间零消耗.

本文在以往线程调度模型的基础上建立了一个硬件线程调度模型,并根据实时系统中的工程应用实际需求,把 EDF 算法改进为 DR-EDF 算法,使之更加适合实际应用的真实环境.

参考文献:

[1] 温涛,王济勇,王晓霞,等. 一个面向嵌入式系统实时性能优化的抢占模型[J]. 通信学报,2005,9(26):129-134.
Wen Tao, Wang Ji-yong, Wang Xiao-xia, et al. Model of preemptive embedded systems for optimizing real-time performance[J]. Journal on communications, 2005, 9(26): 129-134. (in Chinese)

[2] Reinder J B, Elisabeth S, Wim V. Best-case response times and jitter analysis of real-time tasks[J]. Journal of Scheduling. 2004, 7(2): 133-147.

[3] 陈英革,王小英,赵海,尹震宇. 任务调度过程中就绪队列的优化研究[J]. 系统仿真学报,2006,4(18):877-882.
CHEN Ying-ge, WANG Xiao-ying, ZHAO Hai, YIN Zhen-yu. Ready queue optimization research in task scheduling[J]. Journal of system simulation, 2006, 4(18): 877-882. (in Chinese)

[4] Lehoczky J, Lui S, Ye D. The rate monotonic scheduling algorithm: exact characterization and average case behavior[A]. IEEE 9th Real-Time Systems Symposium[C]. Santa Monica: IEEE Communication Society, 1989: 166-171.

[5] Liu C L, Layland J W. Scheduling algorithms for multiprogramming in a hard-real-time environment[J]. Journal of the ACM, 1973, 20(1): 46-61.

[6] Audsley N, Burns A, Richardson M. Applying new scheduling theory to static priority preemptive scheduling[J]. Software Engineering Journal, 1993, 8(5): 284-292.

[7] Katcher D I. Engineering and analysis of real-time operating system[D]. Ph. D Dissertation, Dept of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, August 1994.

[8] Seto D B, Lehoczky J, Liu S. Task Period selection and schedulability in real-time systems[A]. IEEE Real-time Systems Symposium[C]. Madrid: IEEE Communication Society, 1998: 188-199.

作者简介:

尹震宇　男,1979 年生于辽宁沈阳,东北大学博士研究生,IEEE 计算机学会会员. 主要研究方向:嵌入式系统、无线传感器网络.
E-mail: congmy@126.com

赵　海　男,1959 年生于辽宁沈阳,东北大学教授、博士生导师,辽宁省嵌入式技术重点实验室主任. 主要研究方向:嵌入式系统、无线传感器网络、普适计算、信息融合.