维普资讯 http://www.cqvip.com



# 基于 Pfair 的分布式实时调度策略 Linux 下实现

张辉宜,赵海军,周秀丽
(安徽工业大学 计算机学院,安徽 马鞍山 243002)

**摘 要**:任务调度策略是嵌入式分布式实时系统关键问题之一,以 Pfair 公平调度为代表的全局调度技术是当前研究的热点,调度方法要在实际中得到应用,需要与具体的操作系统相结合。分析了分布式实时系统的调度理论,比较研究了几种 Pfair 算法,通过修改 Linux 内核的数据结构和调度函数,初步实现了 Pfair 的 $PD^2$ 算法,实验证明达到了预期的实验结果。

**关键词**:分布式实时系统;任务调度;$PD^2$ 算法
**中图分类号**:TP311 **文献标识码**:A **文章编号**:1673-629X(2008)02-0031-03

# Based on Pfair Implementing Distributed Real-Time Scheduling in Linux Kernel

ZHANG Hui-yi, ZHAO Hai-jun, ZHOU Xiu-li
(School of Computer, Anhui University of Technology, Ma'anshan 243002, China)

**Abstract**: Task scheduling is one of the key problems in distributed real-time system, Pfair global scheduling is studied by many researches. If a scheduling approach would be applied in the real world, it must be implemented in an operating system. For this aim, distributed real-time system scheduling theory is discussed, primarily implement the $PD^2$ algorithm of distributed real-time scheduling based on Pfair in Linux kernel.

**Key words**: distributed real-time system; task scheduling; $PD^2$ algorithm

## 0 引 言

在信息家电、医疗仪器、智能汽车、工业控制、通讯设备等各个领域,嵌入式系统无处不在。随着网络技术的飞速发展,在嵌入式应用中,往往包含许多设备,若干个嵌入式系统组成一个分布式系统进行协调工作可以扩展其功能,提高其性能及可靠性,实现远程访问与远程监控,降低接入成本,提高系统的防卫能力,因此嵌入式领域迫切需要分布式。

分布式的嵌入式系统(DES, Distributed Embedded System)除了具有分布式系统的特征外,还具有嵌入式的实时性特征,DES 的主要要求是分布式系统的实时性。在 DES 的软件部分,操作系统是实施分布式嵌入式技术的关键,而其采用的任务调度策略则是影响 DES 系统的重中之重。

GPL(General Public License)协议下的 Linux 由于软件和驱动程序支持丰富、内核精简、可裁减等优秀特性,在嵌入式操作系统领域广泛使用。嵌入式 Linux 由于其源代码开放,较好地支持 POSIX (Portable Operating System Interface Extend)标准,成为研究操作系统的理想平台。

## 1 调度理论

### 1.1 可调度条件

定义 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"的几率。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。

实时系统调度的关键问题是,能否满足所有任务的时间约束。假定:一个实时分布式系统,在 $N$ 个处理器上运行 $m$ 个周期性的任务,$C_\tau$ 是任务所需要的处理器时间,$P$ 是任务 $\tau$ 的执行周期。如果按动态优先权进行调度,离结束期最近的任务具有最高优先权。那么,在满足:

**收稿日期**:2007-05-18
**基金项目**:安徽省自然科学研究项目(2006KJ064B)
**作者简介**:张辉宜(1963-),男,四川富顺人,副教授,主要研究领域为计算机控制与仿真、嵌入式系统开发及应用。

维普资讯 http://www.cqvip.com



$$\mu = \sum_{\tau}^{m} \frac{C_\tau}{P_\tau} \leqslant N \quad (1)$$

时,就认为该任务集是可调度的[1]。

### 1.2 调度层次

分布式系统的调度层次分两级:局部调度和全局调度。局部调度将任务分配给特定处理器,每个处理器维护本地任务等待队列,调度是局部化的。全局调度将所有等待调度的任务组成全局等待队列,等待处理器调度,处理器允许调度任务时,选择优先级最高的任务执行,调度是全局化的。

### 1.3 动态调度和静态调度

分布式实时调度策略按特征可以分为动态调度和静态调度[2]。动态调度策略是在运行期间调度,当检测一事件时,动态抢占式算法立即决定是运行与这个事件相连的任务还是继续运行当前的任务;而对于动态非抢占式算法,它仅知道有另一个任务可以运行,在当前任务结束后,它才在就绪的任务中选择一个来执行。静态调度是在任务集运行之前就产生一个静态的调度表,在运行时总是按照调度表决定从就绪任务队列中选择哪个任务来运行,这类调度算法假设系统中实时任务的特性是全部已知的。它的可调度性分析是脱机地进行。这类调度算法适合于问题需求确定,并且运行中不会有较大变化的情况。

动态调度较适合于事件触发的系统,静态调度较适合于时间触发的系统。而分布式嵌入实时系统大多是一种确定性系统,它的工况是可预知的,对给定的事件的响应,需要运行哪些任务,这些任务的执行顺序,它们之间的相互关系,都是预知的,很多实际的系统都采用静态调度。理论上,静态调度算法可以穷举所有的调度方案以找到最优方案,但这一搜索时间随任务数成指数增长,是一个 NP 难问题,所以有效的静态调度策略一般基于启发式算法的次优方案。

## 2 Pfair(Proportionate fair)调度技术

Baruah 提出的按比例公平使用资源的调度方法 Pfair 是一种基于同步时钟的,适用带权 RR(Round - Robin)调度技术,是一种典型的全局调度技术。

### 2.1 基本调度策略

令 $\tau$ 表示在 $M$ 个处理器上被调度的 $N$ 个任务的集合,$T \in \tau$,且每个任务分配合理的权值 $T.w \in (0, 1]$,表示在单个处理器上执行的几率。处理器的执行时间被分成各个时间片,在每个时间片,一个处理器最多可以处理一个任务,一个任务最多可以分配给一个处理器。任务迁移是允许的。理想情况下,在任何一个长 $L$ 的时间间隙,一个任务 $T$ 有 $[0,t)$ 处理器时间单元。$A(T,t)$ 表示在时间间隙 $[0,t)$ 分配各任务 $T$ 的时间份额,记 $\mathrm{lag}(T,t) = T.w * t - A(T,t)$,当且仅当 $\forall T,t :: |\mathrm{lag}(T,t)| < 1$,这个调度才是公平调度的[3]。

Pfair 有 PF[3],PD[4] 和 PD$^2$[5] 三种基本调度算法,这三种调度算法都是将子任务时限作为调度优先级,当子任务时限相同时,这三种算法确定任务优先级的方式不同。PD$^2$ 是在 PD 基础上发展来的,引入了组时限的概念,当子任务时限相同时,根据任务组时限确定优先级,如果组时限也相同,则认为这两个任务调度优先级相同,处理器可任意选择一个任务调度。

### 2.2 扩展算法

近年来,Srinivasan 和 Anderson 等人在 Pfair 的理论基础上,先后提出了 ERfairness(Early - release fairness),ISfairness(Intra - sporadic fairness),GISfairness(generalized intra - sporadic fairness)[5]等调度算法,其中比较重要的是 ERfairness,它允许子任务在其运行窗口之前释放,即如果两个子任务是同一任务的子任务,当一个子任务运行结束后,允许另一个子任务运行;ISfairness 扩展了 ERfairness,它允许窗口右移,但相邻的两个窗口必须保证 Pfairness,GISfairness 又在 ISfairness 上作了扩展,它允许子任务可以被忽略。

## 3 Linux 内核中的实现

Linux 标准内核中加进了 POSIX 1003.b 实时扩展部分,引进了实时进程概念,允许把某个进程定义为实时进程,并为实时进程提供了两种实时调度策略:SCHED_FIFO 和 SCHED_RR[6]。在 2.6 版的内核中,开始全面强化了 SMP 的可扩展性,每个处理器有自己的锁和可执行队列,但由于分布式系统的复杂性还未实现分布式的实时调度。

### 3.1 数据结构和算法

定义几个队列:

* DES_Running 存储正在运行的分布式任务

* DES_SchNow 存储在当前时间片非重复调度的全局任务

* DES_SchNext 存储在下一时间片非重复调度的全局任务

* DES_ReschNext 存储在下一时间片重复调度的全局任务

* DES_Elig 存储在下一时间片有资格调度的全局任务

* DES_Incoming 新加入的在下一时间片还没资格被调用的全局任务

在各个队列中全局任务按优先级排序,优先级高

维普资讯 http://www.cqvip.com



的任务首先有资格被调用。

初始化函数 DES_ Initialize()在各个调度程序执行前首先被调用,首先所有在 0 时刻的全局任务被放进 DES_ Eligible 队列,然后决定如何调度

```
DES_ Initialize(){
    DES_ Elig = DES_ Elig∪DES_ Incoming[0];
    while (|DES_ Elig | > 0∧| DES_ SchNext | < M)
      DES_ SchNext = DES_ SchNext∪{Extract - Max(DES_
Elig)};
}
```

DES_ SelectTask()函数选择在 $t+1$ 时间片的任务放在那个队列。如果 $T$ 属于当前调度任务队列,则进入 ReschedNext 队列;否则进 SchedNext 队列。

```
DES_ SelectTask(T){
      if (T ∈ St)
        DES_ ReschNext = DES_ ReschNext∪{T};
      else
        DES_ SchNext = DES_ SchNext∪{T};
}
```

DES_ Schedule()函数是实现该算法的重要函数,它被处理器 P 调用,通过修改 DES_ SchedNow, DES_ SchedNext, DES_ ReschedNow, DES_ ReschedNext 队列,使新任务进入 DES_ Eligible 队列,然后选在处理器 P 执行的任务,并记录。该任务根据它继承的 子任务的优先级更新。选择在 $t+1$ 时间片运行的任务,最后更新 DES_ SchedCount。该算法实现了 $O(1)$ 时间复杂度、$O(N)$ 空间复杂度[7,8]。

```
DES_ Schedule(p)
{
   if(SchedCount = M){
       t = t + 1;
       DES_ Elig = DES_ Elig∪ Incoming[t+1];
       Swap(DES_ SchNow, DES_ SchNext);
       Swap(DES_ ReschNow, DES_ ReschNext);
     }
   if(Running[p]∈DES_ ReschNow){
       T = Running[p];
       ReschedNow = ReschedNow / {T};
   }
   else if(|DES_ SchNow| > 0)
       T = Extract - Min(DES_ SchNow);
   else{
       T = ⊥;
     }
   Running[p] = T;
   if(T ! = ⊥){
       UpdatePrio (T);
       if (T. elig≤t+1)
         DES_ Elig = DES_ Elig∪ {T};
       else
         Incoming[T. elig] = Incoming[T. elig]∪ {T};
     }
   if (|DES_ Elig | > 0)
       SelectTask(Extract - Max(DES_ Elig));
   SchedCount = (SchedCount %M) + 1;
}
```

### 3.2 内核的修改

修改 Linux 内核内容繁多,下面只对重要部分作介绍。

·修改扩充 task_ struct 结构。

task_ struct 结构是 Linux 进程管理的重要数据结构之一,为支持文中上述的算法需要增加 Global_ task 结构,定义如下:

```
struct Global_ task
{
  pid_ t pid;
  unsigned int deadline_ jiffies;
  unsigned int deadline_ tsc;
  struct task_ struct * task;
  struct DES_ Running;
  struct DES_ SchNow;
  struct DES_ SchNext;
  struct DES_ ReschNext;
  struct DES_ Elig;
  struct DES_ Incoming ;
}
```

pid 为进程描述符中的 pid,主要用来标识该进程,deadline_ jiffies 和 deadline_ tsc,主要用来标识实时进程的截止期,task 指向实时进程的进程描述符,该结构主要是对标准 Linux 内核中进程描述符实时属性的补充,并用来将实时进程链接起来。

选择此结构的好处是,对内核的修改比较小,内核的进程相关的系统调用基本不用进行修改,只需增加一些实时进程相关的系统调用,原始的调度程序修改也比较小。

·扩充全局调度策略。

在调度策略中增加 DES_ Schedule()钩子函数,该函数采用上述基于启发式的 $PD^2$ 算法,负责处理全局实时任务调度,完成全局任务处理机的分配。

### 3.3 结　论

对 Linux 中的任务调度策略以及项相关的内容的修改扩充后编译,并在基于 ARM 9 的嵌入式系统上进行了调试,上述分布式实时调度策略运行良好,产生了策略中应有的效果。

(下转第 37 页)

维普资讯 http://www.cqvip.com



节点,若有则执行④;否则,执行⑤。

④比较这两个元素的父节点是否也是语义等价,若等价,则将 $e$ 的子元素、属性添加到该元素的对应位置;否则,执行⑤。

⑤生成对应的元素,并将 $e$.label 改成 new$E$ 中对应的 name;将 $e$ 具有的属性取出,并在 new$A$ 中找到对应的名字;将 $e$ 在 $R$ 中的对应关系添加到 intXML 中。

⑥继续遍历 dim,若集合 $E$ 已经为空,则执行⑦;否则,执行③。

⑦继续遍历 DIM 集合,若已经为空,则执行⑧;否则,执行②。

⑧返回 intXML 文档。

### 3.4 XML 集成文档生成

利用 JDOM API 接口分析每个数据源对应的 XML 文档,并根据元素、属性名称与全局模式中的元素和属性的对应关系进行名称转化,然后将对应的数据添加到 intXML 文档中。

## 4 结　论

数据集成是解决"信息孤岛"问题的根本方法。文中提出了一种基于本体的异构数据源集成方法,将不同数据源转换成对应的 XML 数据,然后通过语义聚类、全局模式生成等步骤实现异构数据源的集成。当异构数据源中具有语义等价的信息时,文中提出的方法可以进行集成,若各个异构数据源没有语义等价的信息时,又可以对这些数据进行简单的重组,使它们都存放到一个 XML 文档中,这是因为在全局模式生成的第一步设置了与各个数据源都无关的 root 根节点,因此该方法还是可行的。但是,由于该方法中使用的 WordNet 本体库只能识别拼写正确的英文单词,所以在数据的命名方面还需要改进。

**参考文献:**


[1] Xu Xiangzhong. Knowledge-based Intelligent Query Processing System[C]//In: ICYCS 2001. [s.l.]:[s.n.],2001:1048-1051.

[2] Molina H G. Database System Implementation[M]. Englewood Cliffs,NJ:Prentice Hall, Inc,2000:420-452.

[3] Gruber T R. A translation approach to portable ontology specifications[R]. Stanford: Knowledge System Laboratory, Stanford University,1993.

[4] 邓志鸿,唐世渭,张　铭,等 .Ontology 研究综述[J].北京大学学报:自然科学版,2002,38(5):730-738.

[5] Fellbaum C, Miller G. WordNet: an electronic laxical database [M]. [s.l.]: The MIT Press, 1998.

[6] 吴永春.XML 数据存储方法研究及应用[J].计算机技术与发展,2006,16(2):139-141.

[7] 丁月华,杨　敏.基于 xml 的异构数据源集成与交换的实现[J].计算机应用与软件,2006,23(10):134-136.

[8] 李选如,何洁月.语义集成:本体映射方法研究[J].计算机技术与发展,2007,17(2):121-124.


(上接第 33 页)

## 4 结束语

全局 Pfair 公平调度是目前分布式实时系统中较理想的调度算法之一,$PD^2$ 调度算法因为其卓越的性能在越来越多的实时任务调度中被采用。文中基于 Linux 内核实现 $PD^2$ 调度算法是一种有益的尝试,目前国内实现的还较少,由于分布式系统和 Linux 内核本身的复杂性,作为一种解决方案还需要大量工作要做。

**参考文献:**


[1] 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.

[2] Tanenbaum A S, Van Steen M. Distributed Systems Principles and Paradigms [M]. Beijing: Tsinghua University Press, 2002.

[3] Baruah S, Cohen N, Plaxton C G, et al. Proportionate progress: A notion of fairness in resource allocation[J]. Algorithmica,1996(15):600-625.

[4] Baruah S, Gehrke J, Plaxton C G. Fast scheduling of periodic tasks on multiple resources[C]//In Proc. of the 9th Int'l Parallel Processing Symp. Washington: IEEE Computer Society,1995:280-288.

[5] Anderson J, Srinivasan A. Mixed Pfair/ERfair scheduling of asynchronous periodic tasks[C]//In Proc. of the 13th Euromicro Conf. on Real-time Systems. North Carolina: University of North Carolina,2001:76-85.

[6] Bovet D, Cesati M. Understanding the Linux Kernel[M]. 3rd edition. [s.l.]: O'Reilly Publishers, 2005.

[7] Holman P, Anderson J. Adapting Pfair scheduling for symmetric multiprocessors [J]. Journal of Embedded Computing, 2005,1(4):543-564.

[8] Holman P, Anderson J H. Implementing Pfairness on a symmetric multiprocessor[C]//Real-Time and Embedded Technology and Applications Symposium, 2004. Proceedings RTAS 2004 10th IEEE. [s.l.]:[s.n.],2004:544-553.