第34卷 第10期 计 算 机 工 程 2008年5月
Vol.34 No.10 Computer Engineering May 2008
·网络与通信· 文章编号：1000—3428(2008)10—0129—02 文献标识码：A 中图分类号：TP393

TinyOS中DSA调度策略的研究

周 艳
（辽东学院信息技术分院，丹东 118003）

摘 要：针对TinyOS任务调度采用非剥夺的先来先服务调度策略，而产生的系统紧急任务不能及时得到响应及节点吞吐量下降情况，该文提出一种新的可抢占时限短作业调度策略——DSA。在绝对时限前执行硬实时任务，满足了系统对实时任务的响应要求，提高处理器的响应速度，对软实时任务实行短作业优先调度策略，提高系统的吞吐量。在TinyOS上测试表明，DSA策略在不影响TinyOS原有性能的情况下，改进了传感器网络承担实时性任务的运行效果。

关键词：时限；可抢占；DSA调度策略；TinyOS操作系统

Research on DSA Scheduling Strategy in TinyOS

ZHOU Yan
(Information Technology College, Liaodong University, Dandong 118003)

【Abstract】TinyOS task scheduling is based on first-come-first-served non-preempting strategy, which is not able to give emergency tasks quick response, and throughput of nodes is lower. To address this issue, this paper proposes a new preempting algorithm, Deadline Short Algorithm(DSA): Through executing hard real-time tasks within an absolute deadline time limit, it meets system requirement for the real-time response, and improves processor's response speed. Through executing soft real-time tasks based on shortest-job-first scheduling priority strategy, system throughput is increased. Test result indicates that with more nodes of wireless sensor network, though energy consumption has a little bit increase, it provides a high real-time performance and high throughput, avoiding network congestion.

【Key words】earliest deadline; preemptive scheduling; DSA scheduling strategy; TinyOS

1 概述

当前，对无线传感器的研究主要集中在通信协议、能耗管理、定位算法与体系结构设计和可靠性研究，分别约占35%, 16%, 24%。TinyOS是基于事件驱动的嵌入式网络传感器操作系统，其目标是用最少的硬件支持网络传感器的并发密集型操作[1]。TinyOS在任务调度上采用非剥夺的先来先服务(First-Come-First-Served, FCFS)调度策略，这样的设计有利于减少系统对存储空间的需求[2]。然而，系统不能对实时性很强的网络任务作出及时的响应，并且可能出现过载[3]，导致任务丢失、通信吞吐量下降等情况的发生。文献[4]提出双环调度策略，提高了系统的响应速度。文献[5]提出在TinyOS中实现基于时限(deadline)的优先级调度，有利于提高WSN系统的实时性。文献[6]提出了一种任务优先级调度算法来相对提高过载节点的吞吐量以解决本地节点包过载的问题。本文提出一种基于时限和短任务优先的DSA算法，提高系统对紧急任务响应速度，同时考虑了系统的吞吐量。

2 TinyOS任务调度策略的分析

TinyOS 提供任务和事件的两级调度。任务之间互相平等，没有优先级之分，任务的调度采用简单的FCFS策略。任务间互不抢占，TinyOS实际上是一种不可剥夺型内核。内核主要负责管理各个任务，并决定何时执行哪个任务。调度任务单线程运行到结束，对任务按简单的FIFO队列进行调度。对资源采取预先分配，调度任务单线程运行到结束，对任务按简单的FIFO队列进行调度。对资源采取预先分配，任务由TOS_post函数提交进入队列，TinyOS的任务队列结构如图1所示。

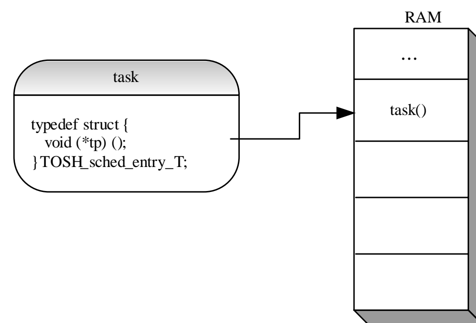


图1 TinyOS任务结构

尽管TinyOS被广泛使用，并且得到了相当的认可，但在以下几种场合，TinyOS的调度策略也可能导致出现问题[7]：

(1)某些任务(例如安全应用的加解密任务)执行时间很长，这时如果某些实时任务在该任务之后才进入任务队列，就会影响实时性；如对于数据包的收发，就会影响波特率。

(2)如果本地待处理的数据量过大或者本地任务发生频率过高，也会导致过载的发生。

(3)如果本地任务运行时间过长，则发送或接收数据包的任务要等待较长时间才能得到处理，从而降低通信速率。

TinyOS的FIFO简单调度策略在某些场合会导致如过载、

基金项目：国家级火炬计划基金资助项目(2002EB010154)；辽东学院自然科学基金资助项目(2007-Y06)

作者简介：周 艳(1968－)，女，副教授、在职博士研究生，主研方向：普适计算，传感器网络

收稿日期：2007-05-30 **E-mail：**zhy03@126.com

任务丢失、数据包吞吐量降低等情况的发生。另外，TinyOS 只搭建了基本的调度框架，它只实现了软实时[8]，而无法满足硬实时[9]，这对嵌入式系统的可靠性会产生影响[10]。

3 DSA 调度策略实现

为了改善系统对紧急任务的响应，根据任务重要性的不同，DSA 算法把任务分为硬实时任务和软实时任务两种。硬实时任务具有硬时间约束，其截止期的错过将会导致较严重的后果。软实时任务则具有软时间约束，其截止期的错过在某种程度下是可以被接受的，不会引起非常严重的后果。

3.1 任务数据结构的修改

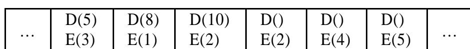
任务的数据结构是操作系统中使用频率最高的数据结构，它的作用就是内核通过此数据结构控制和管理此任务。新的定义如下：

```
typedef struct {
    void (*tp) ();
    uint16_t deadline;
    uint16_t etime;
    uint16_t runtime;
    uint16_t SP;
bool preempted;
} TOSH_sched_entry_T;
```

其中，tp 字段是任务函数入口地址，任务的运行就是通过运行该入口地址的函数来完成的。deadline 字段是新增加的字段，该字段不为空，表示该任务是硬实时任务，具有时限要求；该字段为空，表示该任务是软实时任务，没有时限要求。相对于当前，距离用户要求的任务的绝对时限还有的时钟滴答数，是个相对的时限。etime 字段是新增加的字段，表示用户估计的任务的运行时间，单位也是时钟滴答。SP 字段是本文加上的字段，表示任务当前的堆栈指针位置，主要用于发生中断时恢复现场。preempted 字段是新增加的字段，表示当前任务是否被中断，若为 1 则表示该任务被中断，再次运行前需要进行恢复现场操作。

3.2 任务提交函数的实现

TOS_edf_post()函数的作用就是将任务提交到任务队列中，如果成功放入队列就返回 TRUE，否则返回 FALSE。TOS_edf_post()函数的算法如下：判断新任务的 deadline 值，不为空，按照 deadline 值由小到大插入到队列行头部相应位置，如果 deadline 值为空，按照 etime 值由小到大插入到队列尾部相应位置上，队列最后排序如图 2 所示。所有时限 deadline 值不为空的任务排列在所有 deadline 值为空的任务前面；所有 deadline 值不为空的任务按照 deadline 值由小到大排序；所有 deadline 值为空任务按照 etime 由小到大排列；当有新任务提交时，首先判断 deadline 是否为空，然后任务队列按照 deadline 或者 etime 的值插入到任务队列相应位置上。



…	D(5) E(3)	D(8) E(1)	D(10) E(2)	D() E(2)	D() E(4)	D() E(5)	…

图 2 DSA 任务队列

TOS_edf_post()函数的数据结构为：

```
bool TOS_edf_post (void ( * tp )( ), uint16_t deadline, uint8_t etime, uint16_t runtime ) _attribute_ (spontaneous)
```

其中，_attribute_ (spontaneous)为 nesC 语法，意味着该函数将编译为全局函数，可以在其他模块中直接调用运行。任务提交函数需要以下参数：任务处理的函数指针，任务的绝对时限，任务的运行时间。修改之后的参数中增加了 deadline, etime。修改之后的任务提交函数进行 DSA 优先级调度的任务。

3.3 DSA 调度策略

任务调度函数 TOSH_run_edf_task()修改之后的任务中增加了 deadline、etime 等参数，每次调度定时器中断时，更新任务的剩余时限，选择任务队列头的任务，比较它与当前任务的剩余时限即比较两者的优先级，进行 DSA 可剥夺式调度，当任务队列头的剩余时限高于当前任务的剩余时限时，即任务队列头的任务优先级高于当前任务的优先级，进行抢占，保持现场，切换任务运行的堆栈，改变被中断的任务的中断状态位，判断队列头高优先级任务的中断状态位，如果未被中断过，直接运行这个任务的处理函数。若该任务被中断过，需要先对这个任务恢复现场，然后运行任务的处理函数，直到下一次中断的到达。

修改后的 DSA 调度算法在任务提交时，同时提交任务的其他属性，如运行时间和任务时限等。任务的优先级根据其任务时限和运行时间来确定。每个新任务到达时，在任务队列中按照优先级排序，优先级高的任务排在任务队列头，处理器处理完当前任务后从任务队列头取出最高级的任务运行，每个任务动态更新任务时限和动态调整任务优先级。

4 性能仿真

对 DSA 算法的评估，运用加州大学提供的在 PC 机上运行的 TOSSIM 仿真器进行仿真试验，TOSSIM 支持基于 TinyOS 应用程序模拟运行，并在 TOSSIM 的基础上加入能量度量模块做成 Power-TOSSIM 模拟器，用于观察传感器网络运行的能耗情况。仿真区域设定为 100 m×100 m 的平面区域，分别采用 DSA 和 FCFS 算法，系统吞吐量百分比和能量消耗百分比分别如图 3 和图 4 所示，其中，η=DSA 算法吞吐量/FCFS 算法吞吐量；β= DSA 算法能耗/FCFS 算法能耗。

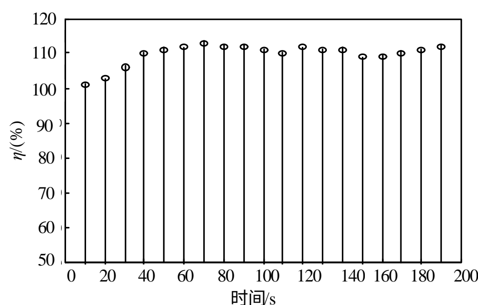


图 3 系统 DSA 和 FCFS 吞吐量百分比

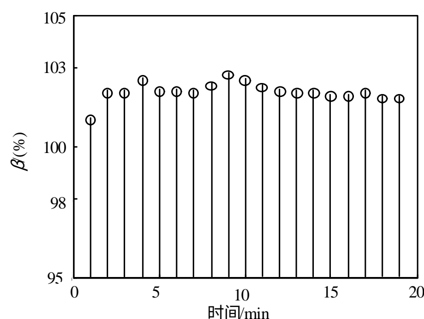


图 4 系统 DSA 和 FCFS 耗能百分比

从图 3 中可以看出，采用 DSA 算法，系统总的吞吐量提高了 10%左右，因为对于软实时任务采用短作业优先调度策略，单位时间内提高了系统的吞吐量。图 4 显示，随着时间

(下转第 143 页)