第32卷 第19期 计 算 机 工 程 2006年10月
Vol.32 № 19 Computer Engineering October 2006
·软件技术与数据库· 文章编号：1000—3428(2006)19—0085—03 文献标识码：A 中图分类号：TP393

TinyOS内核调度机制及改进策略

陈喜贞[1]，王书茂[1]，徐勇军[2]
(1. 中国农业大学工学院机电系，北京 100083；2. 中国科学院计算技术研究所信息网络室，北京 100080)

摘 要：分析了具有代表性的无线传感器网络操作系统 TinyOS 的调度机制并指出其不足。在此基础上提出了改进方案并实现了基于优先级的调度策略。从模拟仿真及在实际系统 GAINS 节点中应用的结果可知，该改进方法能很好地改善无线传感器网络通信性能。
关键词：无线传感器网络；TinyOS；调度策略；优先级

Schedule Mechanism and Its Improving Methods of TinyOS

CHEN Xizhen[1], WANG Shumao[1], XU Yongjun[2]
(1. Department of Mechanical and Electronic Engineering, College of Engineering, China Agricultural University, Beijing 100083;
2. Network Lab, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080)

【Abstract】 This paper analyses the schedule mechanism of TinyOS, which is a representative operating system of WSN, and also points out some shortages of TinyOS. Then some improving methods are presented accompanied with the realization of prioritized task schedule scheme. The simulation result and application result on the actual WSN system-GAINS indicate that the priority schedule of TinyOS can improve communication performance significantly.
【Key words】 Wireless sensor networks; TinyOS; Schedule strategy; Priority

1 概述

随着微机电技术、传感器技术、嵌入式计算技术和无线通信技术的飞速发展和相互融合，具有感知、计算和通信能力的微型传感器节点开始出现。由这些节点构成的无线传感器网络(WSN)，是集信息采集、信息处理、信息传输于一体的综合智能信息系统，具有十分广阔的应用前景，引起了人们极大的关注[1]。

当前，对 WSN 的研究主要集中在通信协议上，约占研究点的 35%；其次是能耗管理，约占 16%；再次是定位算法、体系结构设计和可靠性研究，三者共占研究点的 24%，而对于系统软件尤其是操作系统的研究相对较少。然而，WSN 的操作系统(WSNOS)是 WSN 系统的基本软件环境，是众多 WSN 应用软件开发的基础，它的高效性、灵活性和实时性直接影响到系统的性能，因此很有必要从 WSN 操作系统的角度着手解决 WSN 的一些典型问题，例如可靠性和低功耗等。

在WSNOS的研究上，国外许多大学、研究机构或商业组织都参与其中。到目前为止，已经开始出现一些面向WSN的操作系统。按内核调度策略可分为两类：一类是面向强实时应用的抢占式操作系统，如MANTIS OS、SNOW-OS、Contiki[2]等；另一类是面向弱实时应用的非抢占操作系统，典型的有TinyOS、BTnode OS、Nut/OS、EYES OS[3]等。为了有效利用节点上的能量，延长电池的使用寿命，其中大多数WSNOS采用基于事件的编程模型，如TinyOS、BTnode OS、EYES OS、Contiki、Nut/OS等。此外，为了提高系统灵活性，增强系统功能，各个WSNOS在内核的基础上都作了不同程度的扩展，如BTnode OS支持蓝牙和GSM，Nut/OS采用动态堆栈分配并支持许多网络组件如TCP/IP、ARP、ICMP，MANTIS OS提供了一个可选的动态内存分配，Contiki支持动态下载程序。其它用于WSN的操作系统还有MicroC/OS、DCOS、Topsy v3、SenOS、TelOS、AvrX、Scout、OwlOS等，但它们仍然不能满足不断发展的无线传感器网络的应用要求，新的面向特定应用的WSNOS仍然不断被提出。

在现有的众多WSNOS中，使用最广泛的当属加州大学伯利克分校依托Smartdust项目开发出来的TinyOS。它是一个开源的轻量级嵌入式操作系统，其特点是体积小、结构高度模块化、基于组件的架构方式、低功耗等，这使得它能够突破传感器节点各种苛刻的限制，可快速实现各种应用，非常适合 WSN的特点和应用需求，因而被广泛应用于WSN中，并成为很多系统的参考设计。据统计，世界上有超过 500 家公司或研究机构在使用TinyOS进行学术研究或商业开 发[4]。TinyOS是典型的WSN操作系统，本文就其内核机制进行讨论，并提出了相应的改进方案，实验结果证明了其有效性。

2 TinyOS 内核调度机制分析

TinyOS提供任务加事件的两级调度。任务一般用于对时间要求不高的应用，它实际上是一种延迟计算机制。任务之间互相平等，没有优先级之分，所以任务的调度采用简单的FIFO。任务间互不抢占，而事件可抢占。即任务一旦运行，就必须执行至结束，当任务主动放弃CPU使用权时才能运行下一个任务，所以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[5]。

为了减少中断服务程序的运行时间，提高中断响应的快

基金项目：国家自然科学基金资助重点项目(90207002)；北京市科技基金资助重点项目(H020120120130)
作者简介：陈喜贞(1980－)，男，硕士生，主研方向：嵌入式系统设计，计算机测控技术；王书茂，教授、博导；徐勇军，博士生
收稿日期：2005-12-26 **E-mail：**chenye_cau@163.com

速性，TinyOS 把一些不需要在中断服务程序中立即执行的代码以函数的形式封装成任务，在中断服务程序中将任务函数地址放入任务队列，退出中断服务程序后由内核调度执行。内核使用一个循环队列来维护任务列表。在默认情况下，任务列表大小为 8。内核根据任务进入队列的先后顺序依次调度执行。TinyOS 调度模型有以下特点：

(1)任务单线程运行到结束，只分配单个任务栈，这对内存受限的系统很有利；

(2)没有进程管理的概念，对任务按简单的 FIFO 算法进行调度。对资源采取预先分配，且目前这个队列里最多只能有 7 个待运行的任务；

(3)FIFO 的任务调度策略是电源敏感的。当任务队列为空时，处理器休眠，随后由外部事件唤醒 CPU 进行任务调度；

(4) 两级的调度结构可以实现优先执行少量同事件相关的处理，同时打断长时间运行的任务；

(5)基于事件的调度策略，只需少量空间就可获得较好的并发性，并允许独立的组件共享单个执行上下文。和事件相关的任务集合可以很快被处理，不允许阻塞，有高度并发性。

3 TinyOS 内核调度机制的不足及改进策略

尽管 TinyOS 被广泛使用，并且得到了相当的认可，但这并不意味着 TinyOS 能够适用于 WSN 的所有应用场景。事实上，在某些场合下，TinyOS 并不能工作得很好，而可能出现过载，导致任务丢失、通信吞吐量下降等。

3.1 TinyOS 中的过载现象

无线传感器网络中，节点典型的 3 个任务为：接收待转发的路由数据包，将接收到的数据包转发出去，处理本地的传感数据并将其发送出去。节点上任务的多少取决于节点处理数据的方式，如果节点只是直接把原始数据发往基站，则任务大多数是通信路由任务；如果节点在本地采集数据并处理后才往基站发送，则本地处理任务比较多。当节点上待处理的任务超过其处理能力时，就会发生过载。对于前一种情况，如果节点上发送数据的频率过高或者网络节点密度过大导致通信任务过多时，就可能发生过载；对于后者，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[6]。

另外，当节点上中断发生频率很高，导致 CPU 除了进行中断处理外不执行其它任何任务时也会出现过载。当系统处理任务的速率低于任务发生的频率时，任务队列(当前只能存放 7 个任务)很快就满了，则会导致任务的丢失。对于本地的传感采集速率，可以人为地以软件的方式进行调节控制，例如降低节点采样频率；但对于通信路由任务的发生，则不太容易人为干涉。这时如果发生过载，则直接导致通信数据包吞吐量的下降。如图 1 所示，本地任务发生频率为 8Hz，随着该任务运行时间的增加，通信的吞吐量在下降。

发生这种现象的主要原因是数据包的发送和接收受制于本地任务。当本地任务发生频率过高时，任务队列很快就满了，这时发送或接收任务可能丢失，从而导致数据包丢失；另外，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。

此外，以下几种场合，TinyOS 的调度策略也可能导致出现问题：

(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；对于数据包的收发，就会影响波特率；

(2)本地任务发生频率过高，任务队列很快被本地任务填满，其它任务就可能丢失；此外，如果本地任务过多(例如多个通道同时进行采集，则本地任务数量多)，也会影响通信的正常进行；

(3)任务队列中某个任务如果意外出现阻塞或异常，会影响后续任务的运行，甚至导致系统瘫痪。

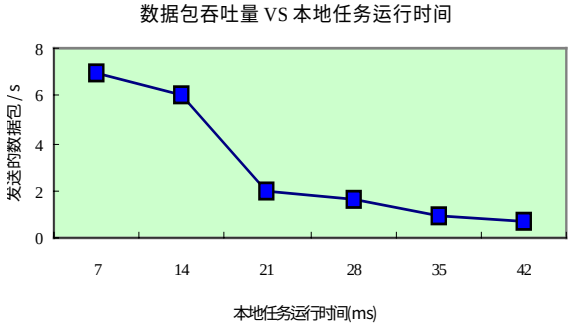

图 1 吞吐量 VS 本地任务运行时间

3.2 对 TinyOS 内核调度策略的改进

从前文所述可知，TinyOS 的 FIFO 简单调度策略在某些场合会导致过载、任务丢失、数据包吞吐量降低等诸多负面问题。另外，TinyOS 只搭建了基本的调度框架，只实现了简单的软实时，而无法满足实际应用的硬实时要求，这对嵌入式系统的可靠性会产生影响。同时，由于采用单任务内核，吞吐量和处理器利用率并不高，因此有可能需要设计多任务系统。

为保证系统的实时性，多采用基于优先级的可抢占式任务调度策略。依赖于应用需求，出现了许多基于优先级的多任务调度算法研究。把 TinyOS 扩展成多任务的调度，有利于提高了系统的响应速度。下面介绍几种典型的改进策略：

(1)基于优先级的任务调度[6]。每个任务根据其重要程度不同，被赋予一定的优先级。这种调度策略总是让处于就绪态的、优先级最高的任务先运行，但何时高优先级任务掌握 CPU的使用权，由所用内核决定。例如，可以把数据包收发和加解密任务设置成高优先级，当它们就绪时，插入在任务队列头部。当任务队列状态为“满”时，低优先级任务被丢弃。

(2)基于时限的任务调度(EDF)。每个实时任务都有一个截止时间，任务的优先级根据任务的截止时间确定。任务的绝对截止时间越近，任务的优先级就越高；任务的绝对截止时间越远，则优先级越低。当有新的任务处于就绪态时，任务的优先级可能需要进行调整。所以，这实质上是一种动态调度机制。

(3)基于时限的优先级调度，它是上述两种方法的结合。高优先级任务先运行，当任务的优先级相同时，由任务的时限来决定哪个任务先执行。

在实际应用系统设计中，应该根据具体情况来分析比较，选择合适的调度策略。

4 基于优先级的调度策略的实现、验证及分析

前面说过，本地处理任务较多时，数据包接收可能被延迟从而导致通信性能下降。这在以下场合最常见：数据在本地节点汇聚，经过各种处理后再往基站发送。如果数据包接收处理任务放在任务队列队尾，那么只有等到前面的任务都执行完后，才能处理数据接收任务，这样必然影响数据包的吞吐量。为此，如果数据接收处理的任务入队列时，将其放在队列的队首，即下一个即将执行的任务，则可能会改善通

信状况。

实现方法是将原来简单的 FIFO 调度算法改为基于优先级的 FIFO 调度。具体来说，就是给每个任务分配一个优先级，如果没有特别指明，默认为 0。不同于实时操作系统 uCOSII，这里的优先级参数数值越大，任务的优先级越高，且任务之间可以有相同的优先级。对任务入队列函数稍作修改，即增加一个优先级参数。从队列尾到队列头，任务的优先级依次上升。如果高优先级的任务入队列前，队列已满，则这时队列尾的低优先级任务将丢失。

为了进一步验证这种调度算法的改进效果，做了两个小实验。实验平台是中科院计算所 WSN 课题组自主研发的 GAINS 节点，编译器使用 AVR-GCC，烧写器采用通用的 JTAG 接口。

4.1 实验 1 提高数据包发送的吞吐量

实验建立过程：有两个节点，一个负责发送数据包，另一个负责接收数据包。发送节点的发送频率为 8Hz，同时它还有一个周期性的本地任务，其发生频率也为 8Hz。接收节点对所收到的数据包进行计数。实验中改变本地任务的运行时间，由图 2 可看出，采用基于优先级的调度策略后，发送节点的吞吐量大大提高。

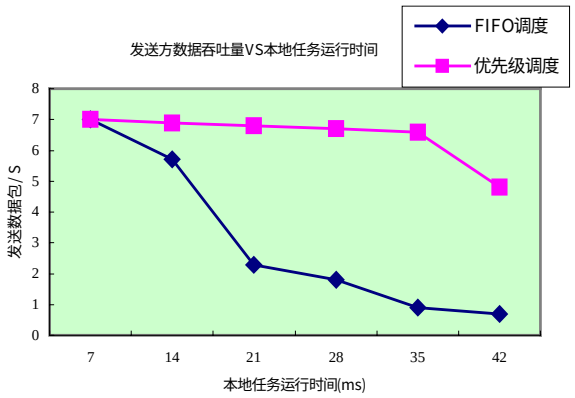

图 2 发送吞吐量 VS 本地任务运行时间

4.2 实验 2 提高数据包接收的吞吐量

任务建立过程如下：两个节点，一个负责发送数据包，另一个负责接收数据包。发送节点的发送频率为 8Hz，接收方除了数据包的接收外，还有一个周期性的本地任务，其发生频率也为 8Hz。

实验中改变接收节点本地任务的运行时间，可看出采用基于优先级的调度策略后接收吞吐量大大提高，如图 3 所示。

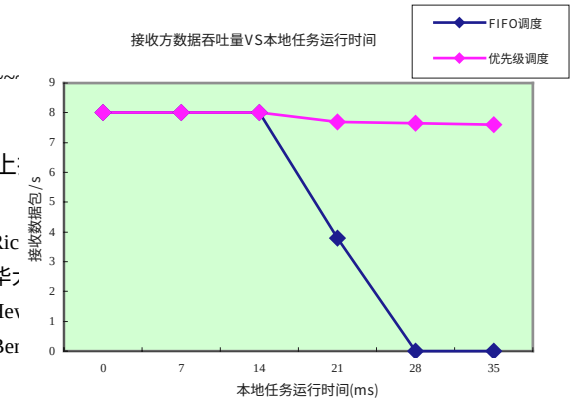

图 3 接收吞吐量 VS 本地任务运行时间

由上述两个实验可知，引入优先级调度机制后，通信的吞吐量显著改善。

5 结论

内核调度在无线传感器网络操作系统中有很实际的研究意义，高效的调度策略能够提高处理器响应速度，从而提高系统的实时性和可靠性；此外，它还能有效降低处理器功耗。所以，有必要进行基于 TinyOS 调度算法的扩展研究。本文首先对 TinyOS 内核调度机制进行分析并指出其不足，在此基础上提出了可能的改进方案并实现了基于优先级的调度策略，最后对其进行实验验证及分析。从实验结果可知，这种改进方法能很好地改善通信性能。

参考文献

1 李建中, 李金宝, 石胜飞. 传感器网络及其数据管理的概念、问题与进展[J]. 软件学报, 2003, 14(10): 1717-1725.

2 Adam D, Thiemo V. Contiki-A Lightweight and Flexible Operating System for Tiny Networked Sensors[C]. Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks, 2004.

3 EYES. EYES Project Webpage[EB/OL]. http://eyes.eu.org.

4 Mads B D. Connection Oriented Sensor Networks[D]. Department of Computer Science, Faculty of Science University of Copenhagen Denmark, 2004.

5 Tiny Microthreading Operating System: TinyOS[Z]. http://tinyos.millennium.berkeley.edu, 2004.

6 Venkita S, Huang Huangming, Seema D, et al. Priority Scheduling in TinyOS: A Case Study[R]. Washington University, TR: WUCSE-2003-74.

（上

1 Ric 华

2 Hev Ber

3 TPC Benchmark™ C Standard Specification (Revision 5.3)[Z]. Transaction Processing Performance Council, 2004.

4 Joseph C J. 毛 选, 魏海萍, 孙思云译. Oracle9i 性能调整学习指南[M]. 北京: 电子工业出版社, 2003.

5 Donald K B. 袁勤勇译. Oracle STATSPACK 高性能调整技术[M]. 北京: 机械工业出版社, 2002.