文章编号:1671-4598(2015)01-0320-04　　中图分类号:TP311　　文献标识码:A

事件驱动的无线传感器网络嵌入式操作系统研究

石文铎，陈俊英，包天悦，万江文
(北京航空航天大学 仪器科学与光电工程学院，北京　100191)

摘要：相比于一般的嵌入式系统，无线传感器网络节点对操作系统的能量利用率、实时性和多任务并发等方面提出了更高的要求，针对这些特点，提出了基于事件驱动的无线传感器网络嵌入式实时操作系统；采用分层结构的设计思想，构建内存管理、事件管理和任务管理等模块化的操作系统组件，利用内存控制块动态链表，实现简易高效的内存管理；基于事件驱动和任务优先级，实现系统低功耗和抢占式的任务调度；实验结果表明，系统功耗低，响应速度快，实时性好。

关键词：无线传感器网络；嵌入式操作系统；内存管理；任务调度

An Embedded Operating System Research for Wireless Sensor Networks Based on Event-driven

Shi Wenduo, Chen Junying, Bao Tianyue, Wan Jiangwen
(School of Instrumentation Science and Opto-electronics Engineering,
Beijing University of Aeronautics and Astronautics, Beijing　100191, China)

Abstract: Compared with the general embedded system, the operating system for wireless sensor networks is more constrained in the aspects of energy efficiency, real-time performance and multitasking. Being aimed at these features, a real-time operating system based on event-driven is proposed for wireless sensor networks application. A thought of hierarchical structure is adopted to build the modular system components which contain memory management, event management and task management. Simple and effective memory management is achieved using the memory control block dynamic list. The low power dissipation and preemptive task scheduling is implemented based on event-driven and task priorities. The experimental result shows that the system has low power consumption and good real-time quality.

Keywords: wireless sensor networks; embedded operating system; memory management; task scheduling

0　引言

无线传感器网络（wireless sensor networks，WSN）节点是一种典型的嵌入式系统，具有多任务并发、能量敏感和硬件资源受限等特点[1-4]。嵌入式操作系统如 Linux，QNX，μC/OS[7]，VxWorks，WindowsCE 等，提供大量的系统服务，但其较大存储容量需求和能量开销导致其不适合在 WSN 节点上应用。面向 WSN 节点的嵌入式操作系统有 MANTIS OS[5] 和 TinyOS[6] 等，其中 TinyOS 以其组件的模块化、低功耗、并发型等优点成为目前应用最为广泛的 WSN 节点操作系统之一，然而 TinyOS 单线程的任务调度机制不能很好地支持实时性要求较高的应用。为此，提出了事件驱动的嵌入式实时操作系统(event-driven real time operating system，ERTOS)，基于事件驱动机制，实现系统的低功耗；采用抢占式的任务调度策略，保证系统实时性。

1　ERTOS 结构设计

ERTOS 采用分层结构的设计思想，在内核中，将事件驱动和任务执行分离，采取抢占式的多线程调度模式，同时提供有效的内存管理。ERTOS 的层次结构如图 1 所示。

收稿日期:2014-05-13；　**修回日期**:2014-06-30；
基金项目:国家自然科学基金项目资助(61371135)。
作者简介:石文铎(1990-)，男，辽宁人，硕士研究生，主要从事嵌入式系统方向的研究。
万江文(1963-)，男，湖北人，教授，博士生导师，主要从事无线传感器网络技术方向的研究。

抽象层将 WSN 节点硬件映射到上层模块中，隐藏硬件细节，提供与硬件的接口函数，提高系统移植效率；核心层包括事件管理，任务管理以及内存管理，这一层采用 C 语言编写，独立于硬件系统平台，与硬件相关的操作全部通过调用硬件抽象层提供的接口函数完成。最上层是 WSN 节点的嵌入式应用程序。

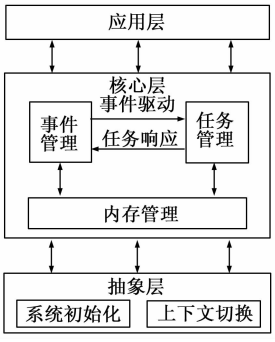

图 1　ERTOS 系统层次结构

2　ERTOS 模块设计与实现

2.1　抽象层

抽象层由系统初始化模块和上下文切换模块组成，直接与 WSN 节点的硬件关联，分层设计中，抽象层与其他层次相互独立，有效提高了系统的移植效率。在不同的 MCU 平台下，只需修改抽象层的相关代码，即可实现整个系统的完全移植。

2.1.1　系统初始化模块

基于不同的硬件平台，系统初始化首先完成嵌入式微处理器和外部设备的初始化，设置核心寄存器和控制寄存器，更改工作模式和局部总线模式，将 WSN 节点设置为系统所需的工作状态，这部分是纯硬件的初始化过程，在移植不同的微处理器时，仅需修改硬件平台相关的初始化函数。在系统初始化后立即完成各类数据结构和软件参数的初始化，为 ERTOS 和应用程序创建必要的运行环境，并准备把系统控制权交给 ERTOS 内核。

2.1.2　上下文切换

应用程序中的每一个任务都拥有各自的运行上下文，上下文内容被保存在各自任务堆栈和任务控制块中。任务上下文包括以下内容：

1）当前程序计数器（program counter，PC）；

2）CPU 通用寄存器；

3）任务堆栈指针（stack pointer，SP）；

4）内核控制结构和任务优先级。

任务上下文的内容涉及微处理器的各类寄存器，在进行上下文切换时不可避免要使用微处理对应的汇编语言，将上下文切换函数封装起来通过 C 语言调用，系统移植时，仅需使用微处理器对应的汇编语言填充封装的上下文切换函数 switch_to ()。

2.2　核心层

内核结构在设计上要满足结构合理、执行效率高和高内聚低耦合原则。ERTOS 采用微内核的设计思想，设计的内核仅包含事件管理、任务管理和内存管理模块。ERTOS 核心层的结构如图 2 所示。

ERTOS 根据进入 WSN 节点的事件流，将与事件对应的任务置为就绪状态，并挂载到任务就绪哈希表中，任务调度模块根据优先级对就绪任务进行调度，并使获得运行机会的任务开始执行。内存管理为事件系统和任务系统提供必要内存支持。

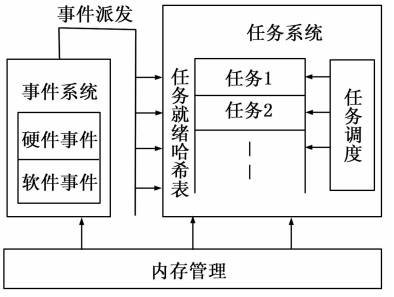

图 2　ERTOS 核心层结构

2.2.1　内存管理

存储资源是系统运行的基础，同时也是 WSN 节点最受约束的资源之一。为有效利用节点的有限存储资源并降低系统能耗，ERTOS 在设计上不区分内核地址空间和用户地址空间，所有任务和内核都处于单一的地址空间中，在任务被调度时，能免去地址空间切换的能量消耗，在动态内存分配方面，仅提供单一的内存分配和回收函数，不对内存碎片进行整理，以降低系统能量消耗。与动态内存分配相关的数据结构如图 3 所示。

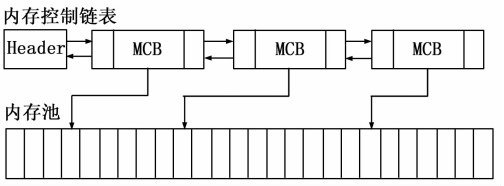

图 3　ERTOS 动态内存结构

内存管理由内存控制块链表和内存池组成，使用无符号字符型数组 MemPool 表示物理地址连续的内存池空间，使用内存控制块（memory control block，MCB）描述分配内存在内存池中的位置偏移量和大小，内存池 MemPool 和内存控制块 MCB 的数据结构定义如下：

```
unsigned char MemPool [POOLSIZE];
struct MCB
{
  struct MCB * pre;
  struct MCB * next;
  int offset;
  int size;
};
```

MemPool 是一块物理地址连续的内存空间，pre 指向内存控制块链表的前一个元素，next 指向内存控制块链表中的后一个元素，offset 表示分配内存在内存池中的位置偏移量，size 表示分配内存的大小。

用户程序提出动态内存申请时，内存管理模块首先遍历内存控制块链表，通过内存控制块链表上相邻两个内存控制块的偏移量和大小可以计算出两块地址间空闲内存的大小，如果空闲内存大小足够容纳用户程序提出的内存申请，则返回空闲内存首地址，直至内存控制块链表的末尾，如果内存池中没有足够的内存空间，则返回空指针，内存分配失败，任务挂起，等待内存重新分配。

用户程序提出内存回收申请时，仅需将描述内存的内存控制块从内存控制块链表中删除，被释放的内存被回收至内存池，等待其他用户程序的申请和利用。

2.2.2　事件管理

ERTOS 中，事件主要有两个来源：一个是软件事件，来源于任务间的通信和同步；另一个是硬件事件，来源于硬件和外部设备的中断请求，所有事件均由事件派发器处理。ERTOS 中事件的产生、映射和派发流程如图 4 所示。

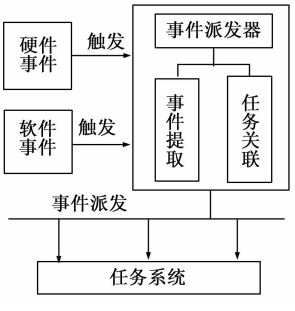

图 4　事件管理流图

当任务间通过事件进行任务间通信和同步时，软件事件被触发，源任务通过系统调用产生一个软件事件，并向事件派发

器提交一个标明目标任务标识符的事件请求，最终由事件派发器将事件派发到目的任务。

当外部设备发出中断请求时，硬件事件被触发，系统转入中断服务程序（interrupt service routine，ISR）做必要处理，在 ISR 返回之前，事件派发器提取中断请求对应的事件，并将该事件派发到目的任务。

ERTOS 中，使用结构体 struct Event 来表示事件基本属性和事件行为之间的关系。

```
struct Event
{
  int ID _ Event;
int ID _ Task;
  int ( * post) (struct TCB * task, unsigned char id _ event);
};
```

其中，ID _ Event 为事件标识符（Identification，ID），ID _ Task 为目标任务 ID，ptr _ post 为事件派发器函数指针，ERTOS 本身提供事件派发算法，同时，用户可以关联自己设计的事件派发器到 ptr _ post 指针，如此设计，有利于满足个性化需求，增强了事件管理的可扩展性。

2.2.3 任务管理

ERTOS 中，每个任务对应的处理程序以函数的形式给出，任务处理函数的形式定义如下：

```
void Task (void)
{
…
}
```

内核以任务控制块（task control block，TCB）的形式对任务进行管理，任务控制块数据结构如下：

```
struct TCB
{
  int ID;
  int prio;
  void ( * ptr _ task) (void);
  char * sp;
  struct TCB * next;
};
```

其中，ID 为任务的 ID 号，prio 为任务的优先级，ptr _ task 为任务处理程序的入口地址，sp 为任务堆栈指针，next 是任务就绪哈希表中指向同优先级 TCB 的指针。任务管理器的数据结构如图 5 所示。

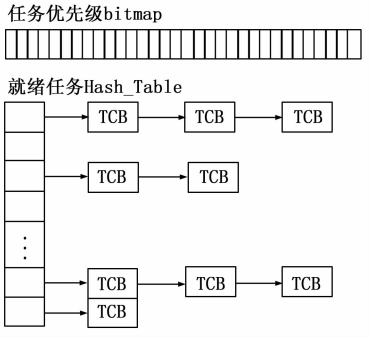

图 5 任务管理器数据结构

任务优先级 bitmap 是一个 32 位的无符号整型全局变量，ERTOS 中共有 32 个任务优先级，每个任务对应一个任务优先级，任务优先级的绝对值越小，任务优先级的级别越高，0 对应最高优先级，31 对应最低优先级，每一个任务优先级对应 bitmap 中的一个 bit，0 表示该优先级没有就绪任务，1 表示该优先级存在至少一个就绪任务。就绪任务 Hash _ Table 包括 32 个入口，每个入口对应一个任务优先级，所有处于就绪状态的任务 TCB 根据任务优先级被挂载到 Hash _ Table 中，处于同一优先级的就绪任务通过单链表链接起来。

任务挂载和事件派发密不可分，事件派发器将某事件派发到对应任务时，事件派发器首先提取目的任务的任务优先级，通过系统调用将 bitmap 中与该任务优先级对应的位置 1，并将对应的任务控制块挂载到就绪任务 Hash _ Table 中，等待任务调度器调度并执行任务。

ERTOS 是基于优先级的可抢占操作系统，系统总是选择具有最高优先级的任务来优先执行，并且高优先级任务可以抢占低优先级任务的执行机会。启动任务调度后，通过系统调用读取 bitmap 提取当前运行状态下就绪任务的最高优先级 Current _ High _ Priority，并通过 Current _ High _ Priority 在就绪任务 Hash _ Table 中获得任务 TCB，最后调用抽象层提供的 switch _ to ()，完成任务切换。

2.3 应用层

ERTOS 中，事件和任务对应的源程序有特殊的结构。事件源程序结构描述为：

```
void Event (void)
{
  …
  post _ task (struct TCB * task);
  schedule ();
}
```

事件完成相关处理后，通过 post _ task () 函数将与事件对应的任务挂载到就绪任务哈希表中，然后调用 schedule () 函数，完成任务调度，这时，如果刚刚挂载的任务在当前运行状态下拥有最高优先级，则该任务可立即被执行。任务源程序结构描述为：

```
void Task (void)
{
  …
  schedule ();
}
```

进入任务处理函数后，开启相关硬件模块，当前任务执行完毕后，关闭相关硬件模块，并调用 schedule () 函数启动任务调度，如果就绪任务哈希表中还存在其他就绪任务，则根据调度策略切换至其他任务，如果就绪任务哈希表为空，系统进入休眠，实现系统的低功耗。

3 性能评估

3.1 功耗分析

ERTOS 是基于事件驱动的操作系统，与典型的实时嵌入式操作系统 μC/OS 相比，ERTOS 具有低功耗的特点。

当 μC/OS 处于空闲状态时，系统全速执行空闲任务，并且所有相关外设模块都处于工作状态，具有很高的能量消耗。ERTOS 中任务都由事件触发，当任务被执行时，与任务相关的外设处于工作状态，而与任务无关的外设均处于关闭状态，

当 ERTOS 处于空闲状态时，系统会根据需要直接进入休眠状态。无线传感器网络节点在多数时间处于空闲状态，因此，ERTOS 能够有效降低系统功耗。

3.2　实时性分析

ERTOS 是抢占式的嵌入式实时操作系统，高优先级任务可以优先执行，并且优先级较高的任务可以抢占优先级较低的任务执行的机会，抢占式调度和单线程调度的实时性比较如图 6～7 所示。

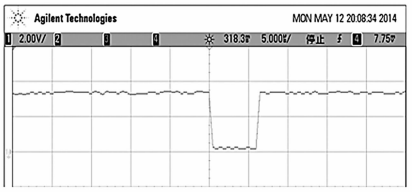

图6 抢占式调度实时性分析

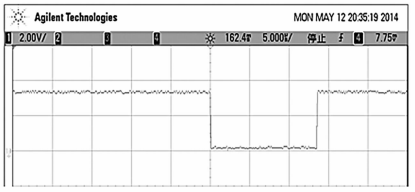

图7 单线程调度实时性分析

分别创建任务 A 和任务 B，任务 A 处于低优先级，任务 B 处于高优先级。图 6 中，下降沿表示任务 A 开始执行，上升沿表示任务 B 开始执行。系统启动后，由系统挂载任务 A，并在任务 A 中挂载任务 B。图 6（a）采用抢占式的实时调度，在任务 A 中挂载任务 B 后，任务 B 由于拥有更高的优先级而立即被调度执行，图 6（b）采用单线程的调度机制，高优先级任务 B 必须等待低优先级任务 A 先执行完，才能被执行。图 6（b）中，任务 B 的执行比图 6（a）中推迟约 10 μs，并且，随着任务 A 复杂度的增加，任务 B 将进一步被推迟。实验表明，ERTOS 的抢占式调度机制，有助于提高系统的实时性。

4　应用实例

将 ERTOS 应用于以 STM32F103 为硬件核心的移动无线传感器网络节点中，承担节点中各应用程序的调度任务，移动无线传感器网络节点被安装到一个由电机驱动的可移动小车上，软件部分主要包括 GPS 数据解析任务、通信任务、传感数据处理任务和电机控制任务，GPS 数据解析任务和传感数据处理任务通过软件事件触发通信任务。测试结果表明，ERTOS 操作系统功耗低、实时性好且运行稳定。

5　结论

基于事件驱动的嵌入式操作系统 ERTOS，采用分层结构的思想设计了抽象层、核心层和应用层，提高了系统的可移植性；采用内存控制块链表，实现了简易高效的内存管理；基于事件驱动和任务优先级，实现了可抢占的任务调度。

将 ERTOS 应用到移动无线传感器网络节点上，系统运行稳定，实时性好，功耗低。

参考文献：

[1] 张　昊，崔永俊，沈三民，等. 基于 CAN 总线的传感器网络设计 [J]. 计算机测量与控制，2013，21（11）：3103－3105.

[2] 周海鹰，左德承，李　剑，等. 基于混杂模式的无线传感器网络操作系统的研究与设计 [J]. 高技术通讯，2013，23（2）：130－138.

[3] Buttazzo G C，Bertogna M，Yao G. Limited preemptive scheduling for real-time systems. a survey [J]. Industrial Informatics，IEEE Transactions on，2013，9（1）：3－15..

[4] Farooq M O，Kunz T. Operating systems for wireless sensor networks：A survey [J]. Sensors，2011，11（6）：5900－5930.

[5] Bhatti S，Carlson J，Dai H，et al. MANTIS OS：An embedded multithreaded operating system for wireless micro sensor platforms [J]. Mobile Networks and Applications，2005，10（4）：563－579.

[6] Levis P，Madden S，Polastre J，et al. TinyOS：An operating system for sensor networks [M]. Ambient intelligence. Springer Berlin Heidelberg，2005：115－148.

[7] Labrosse J J. MicroC/OS-II：The Real Time Kernel [M]. CRC Press，2002.

[8] Shan-shan M，Jian-sheng Q. Energy Balanced Non-Uniform Distribution Node Scheduling Algorithm for Wireless Sensor Networks [J]. Applied Mathematics & Information Sciences，2014，8（4）.

[9] Comer D. Operating System Design：The Xinu Approach，Linksys Version [M]. CRC Press，2011.

[10] Cormen T H，Leiserson C E，Rivest R L，et al. Introduction to algorithms [M]. Cambridge：MIT press，2009.

（上接第 319 页）

[2] 张培仁，杜洪亮. CAN 现场总线监控系统原理和应用设计 [M]. 合肥：中国科学技术大学出版社，2011.

[3] 李文燕，郭　涛，徐香菊. MEMS 高量程微加速度计温度补偿的设计 [J]. 计算机测量与控制，2012，20（10）：2857－2859.

[4] 王　祁. 传感器信息处理及应用 [M]. 北京：科学出版社，2012.

[5] 牛跃听，周立功，方　丹. CAN 总线嵌入式开发——从入门到实战 [M]. 北京：北京航空航天大学出版社，2012.

[6] 杨春杰，王曙光. CAN 总线技术 [M]. 北京：北京航空航天大学出版社，2010.

[7] 郭　涛，鲍爱达. 一种压阻式硅微复合量程加速度计 [J]. 弹箭与制导学报，2010，30（6）：186－189.

[8] 王世清，姜　彤，侯占民. 单晶硅压力传感器温度漂移的补偿方法 [J]. 传感器与微系统，2006，25（7）：33－35.

[9] 孙艳梅. 压阻式压力传感器温度补偿方法研究 [D]. 齐齐哈尔：齐齐哈尔大学，2012.

[10] 刘　鹏. 压阻式压力传感器温度补偿方法实现的研究 [D]. 天津：天津大学，2010.

[11] 张培仁. CAN 总线设计及分布式控制 [M]. 北京：清华大学出版社，2012.