中国科学技术大学

硕士学位论文

一种新的实时系统内核调度算法研究设计

姓名：薛安琪

申请学位级别：硕士

专业：通信与信息系统

指导教师：屈玉贵

20100520

ABSTRACT

With the development of network technology, embedded systems and network communication technology combined with each other more and more closely. Embedded real-time operating system should not only meet requirements of real-time system but also to meet the network communication service quality. In the scheduling process of kernel, the system should not only rapid response system of real-time task, but also compatible with network transmission characteristics. The introduction of the network protocol stack based on embedded operating system has become the inevitable trend of the embedded system development.

In this paper，we made a thorough investigation and study in kernel scheduling algorithm, conducted in-depth study of UCOS-II(MicroC/OS-II) embedded real time operating system, and propose a new kernel scheduling policy. The main contents are as follows:

1） **A new kernel scheduling algorithm was proposed**

The new scheduling algorithm combined the advantages of Task Priority based scheduling algorithm and Time Slot Cycling based scheduling algorithm. The task ID was automatically managed, which provide a user-friendly interface. This algorithm effectively mitigate the defects of the original UCOS-II, which has only a few tasks, multiple tasks was not supported in the same priority level, and did not support priority inheritance, etc. Experiments showed that the efficiency of the new kernel scheduling algorithm has been significantly improved.

2） **UCOS-II's dynamic memory management strategy was improved**

The dynamic memory management strategy provided by UCOS-II is too simple to achieve high memory usage efficiency. This paper improved the dynamic memory management strategy of UCOS-II and implemented a size variable dynamic memory management strategy. Using this new strategy, the memory can automatically merge together after been released, which provides users with convenient and stable interface functions.

3） **Developed a network protocol stack for wireless meter reading system based on the new embedded kernel**

We completed the new kernel porting, driver development and the underlying network protocol stack development based on hardware of wireless meter system. The

entire protocol stack is divided into five layers, and it integrates with the traditional TCP / IP protocol.The network protocol stack implemented a network communication protocol which is compatible with IPv6, and capabilities of reliable data delivery, automatic networking and dynamic routing were achieved. After several tests, the system has good performance, fully meet the relevant national convention. In the network transmission environment, the new kernel has greatly improved the efficiency of the system.

Key words: Embedded Systems, Kernel Scheduling, Dynamic memory management strategy, Network protocol, Wireless meter reading system, IPv6 Protocol

中国科学技术大学学位论文原创性声明

本人声明所呈交的学位论文,是本人在导师指导下进行研究工作所取得的成果。除已特别加以标注和致谢的地方外，论文中不包含任何他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的贡献均已在论文中作了明确的说明。

作者签名：_____　　签字日期：_____

中国科学技术大学学位论文授权使用声明

作为申请学位的条件之一，学位论文著作权拥有者授权中国科学技术大学拥有学位论文的部分使用权，即：学校有权按有关规定向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅，可以将学位论文编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。本人提交的电子文档的内容和纸质论文的内容相一致。

保密的学位论文在解密后也遵守此规定。

□公开　□保密（____年）

作者签名：_____　　导师签名：_____

签字日期：_____　　签字日期：_____

第 1 章　绪论

1.1　嵌入式系统

随着信息化、智能化、网络化的高速发展，嵌入式系统（朱珍民， 隋雪青， 段斌编，2006）在人们的生产生活中得到越来越广泛的应用。嵌入式系统是以应用为中心，融合了计算机技术、自动化技术以及网络和通信技术，对功能、可靠性、安全性、功耗、成本都有严格的要求。

嵌入式系统是在特定的运用环境下，针对具体的需求设计的系统，因此用户可以根据需要对系统的软硬件进行裁剪，不但节约了成本，而且提高了系统的性能。嵌入式系统一般具有以下的特征（陈是知，2007）：

1）较小的系统内核：由于应用环境的限制，系统的资源比较有限，系统内核比传统操作系统内核要小很多，一般只有几 KB~几十 KB。

2）精简的系统：嵌入式系统对于系统软件和应用软件没有具体的区别，软件设计与硬件构成紧密相关，功能设计以满足需求为目标，不具备很强的扩展性，有利于成本的控制以及系统的安全。

3）较强的专用性：嵌入式系统的软硬件设计都是以应用为中心。不同的应用，软硬件构成差别很大，不具有通用性，同时软硬件结合的很紧密。

4）实时性的要求高：嵌入式系统一般对实时性的要求比较高。实时性的提高一般通过较好的内核调度算法以及软件的固化存储实现。

5）多种技术的结合体：嵌入式系统是计算机、半导体以及电子技术相结合的产物，完全是一种技术密集型创新型系统。

6）嵌入式软件设计逐渐标准化：为了提高软件开发的效率，提高实时性和可靠性，软件设计的标准化将成为必然趋势，但这些必须有多任务调度操作系统的支持，尽管软件设计可以不需要操作系统，直接基于硬件开发，但为了合理的利用资源，提高开发效率，减少重复开发，应该选用适当的实时操作系统开发平台，利用系统函数以及函数接口，有效的保证了软件的质量。

1.2　嵌入式系统构成

1.2.1　嵌入式硬件简介

如图 1.1 所示，嵌入式硬件系统一般是由嵌入式核心处理器、存储设备、输入/输出接口、人机交互接口、通信传输模块以及电源模块等组成。嵌入式系统是面向应用、面向用户的，因而它和通用的计算机系统不同，嵌入式系统的硬件是在具体的需求的基础上高效的设计，用户可以根据成本和功能需要对进行硬件裁剪（朱珍民， 隋雪青， 段斌编，2006）。

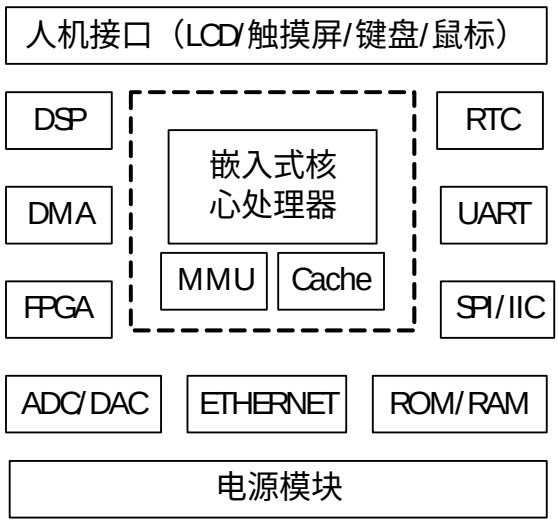

图 1.1 嵌入式系统硬件组成

1） 嵌入式核心处理器：嵌入式系统的核心，是控制、辅助系统运行的硬件单元。嵌入式处理器对功耗的要求比较高，必须满足低功耗的要求，嵌入式系统一般任务都是实时的，因此嵌入式处理器对实时任务的支持能力强，要求较短的中断响应时间。随着技术的革新，嵌入式处理器的扩展能力越来越强。嵌入式处理器可以分为以下几类：

a) EMPU（嵌入式微处理器，Embedded Microprocessor Unit）：嵌入式微处理器是在通用 CPU 的基础上，保留了与嵌入式应用相关的功能，大幅度减小系统的功耗和体积。整个系统功耗更小，对环境的适应性更强，稳定性更高，成本更低。主流的嵌入式微处理器有 ARM 系列、MIPS 系列、Motarola68000 系列、XScale 系统等。

b) MCU（嵌入式微控制器，Microcontroller Unit）：嵌入式微控制器是将整个系统集成在一块芯片上，也就是单片机系统。它是以微处理器为核

心，芯片内部将存储器、总线、定时器、数模转换器、uart 等模块集成在一起，用户可以根据自己的需求选择相应的单片机芯片。单片机系统将需要的功能模块集中在一块芯片上，整个系统的体积和功耗都大大的降低，可靠性得到了提高，但单片机系统的性能比较有限。比较主流的单片机芯片包括 8051、ATmega128 系列、MC68HC05 等。

c) EDSP（嵌入式 DSP 处理器，Embedded Digtal Signal Processer）：嵌入式 DSP 处理器主要针对连续的数据流处理以及高精度、高复杂度的运算。它对系统结构以及指令集进行了特殊的设计，使得编译效率和指令执行速度得到了提高。主流的嵌入式 DSP 芯片主要有 TMS320、DSP56000 系列等。

d) Soc（嵌入式片上系统，System on Chip）：嵌入式片上系统是将微处理器、模拟 IP 核、数字 IP 核和存储器(或片外存储控制接口)集成在单一芯片上，是一种面前客户、面向应用的标准产品。它是一种高度集成化、固件化的系统集成技术。使用 SoC 技术设计系统的核心思想，就是要把整个应用电子系统全部集成在一个芯片中。

2） 总线、通信接口：总线是指一组互联和信息传输的信号线，是系统各种功能部件之间传送信息的公共通信干线，总线结构简化了硬件设计，提高了系统性能，便于外部设备的扩充和维护。常用的总线包括：ISA 总线、PCI 总线、I2C 总线、RS-232、SPI（串行外部接口）、IEEE802.3、CAN 总线接口等。

3） 存储器模块：存储器是系统的记忆设备，用来记录系统的数据和实现系统功能的程序。存储器的选择决定了整个系统的性能和操作。嵌入式系统包括的主流存储器如下：高速缓存 Cache、静态易失性存储器（RAM、SRAM）、动态存储器（DROM）、NORFlash、NANDFlash、EEPROM、SD Card 等。

4） 其他设备：人机交互设备是人与系统传递、交换信息的接口，是系统的重要组成部分。开发者通过人机交互设备进行系统的开发调试，用户通过人机交互设备对系统进行配置，系统也是通过它把数据信息展示给用户；电源设备是整个系统正常运行的基础和动力。

1.2.2　嵌入式软件简介

如图 1.2 所示，嵌入式软件系统（Raskovic, D.，Revuri, V.，Giessel, D.，2009）一般由硬件驱动层、嵌入式实时操作系统、中间层已经应用层四部分构成。由于资源有限，因而要求软件代码紧凑、稳定，而且对实时性有很高的要求。

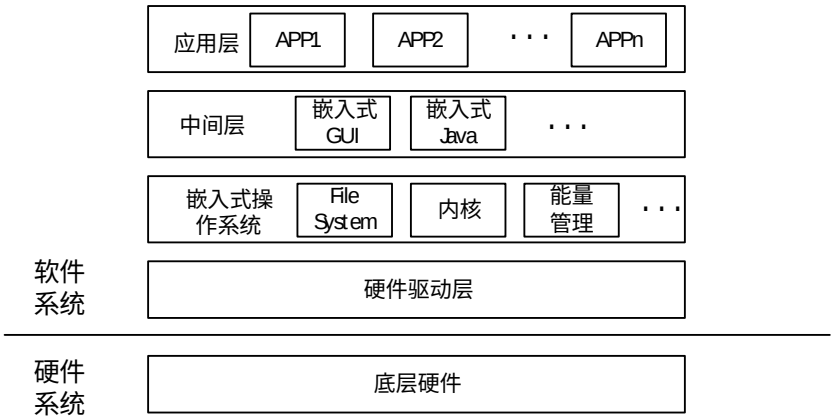

图 1.2 嵌入式软件构成

1） 硬件驱动层：主要是对底层的硬件进行抽象，为上层提供硬件操作接口，它与底层硬件系统关系密切，与具体的操作系统无关。对于嵌入式操作系统来说，底层完全透明。

2） 嵌入式实时操作系统：嵌入式操作系统是指一种能够支持实时性控制操作的操作系统。它的功能与通用的操作系统相似。嵌入式实时操作系统是嵌入式系统软件的核心部分。其具有的功能比不带操作系统的嵌入式系统强大很多。嵌入式实时操作系统主要由任务调度、任务管理、任务通信以及内存管理等组成。用户可以根据需要对文件系统、网络协议等进行添加和裁剪。比较主流的嵌入式实时操作系统有：UCOS-II、T-Kernel、QNX、uClinux、VxWorks 以及 Windows CE 等。

3） 中间层：嵌入式系统中间层主要包括一些比较复杂的中间件技术，比如用于交互界面设计的嵌入式 GUI，用于 Java 开发的嵌入式 Java，用于软件无线电台运用的嵌入式 CORBA 等。

4） 应用层：应用层是用户根据具体的需要所开发的应用程序，它们都是在嵌入式操作系统以及中间层的基础上。操作系统向用户提供接口函数，为用户屏蔽具体的底层操作。应用层主要是根据用户的具体需求开发的一系列的任务组成。这些任务协调工作实现特定的功能，而各个任务的协调是由操作系统完成。

1.3　传感器网络简介

由于网络通信技术的发展，新的需求的提出，嵌入式系统与网络技术联系的越来越紧密。传感器网络就是一种综合传感器技术、嵌入式技术、分布通信技术的综合型网络。它能够对于网络分布区域内的检测对象以及各种环境进行实时的监测、感知和采集，并对这些信息融合处理，然后把准确详细的信息发送给用户。

传感器网络（Liu Yong-Min，Jiang Xin-Hua，2009；Zhi-Ting Lin,Yu-gui Qu,Xiao-Fang Zhou，2009；ZHENG Jie，GUO Shu-jie，QU Yu-gui，2007；Qing-Hua WANG， Yu-Gui QU，2008）具有广泛的应用前景，是一种新的信息获取系统。传感器网络关键技术涉及通信组网、分布式信息融合处理、管理和同步等多个方面。与传统的网络不同，传感器网络一般以数据为中心，网络节点数目大、密度高，节点的能量有限，存储空间和计算能力也有很大的限制，网络拓扑结构复杂多变。传感器网络的这些特点给传感器技术研究提出了新的挑战：

1） 低能耗、低成本

由于传感器网络能量有限（柏荣刚，赵保华 ，屈玉贵，2009），有效节省能量才能延长网络寿命，因此在满足功能的基础上传感器网络协议及算法应尽可能简单，传统网络协议及算法都不能直接应用于传感器网络。低成本是工业生产的要求，同时传感器网络节点数量较大，人工管理和维护开销大，因此需要网络有自组织能力，能够自动配置和管理。

2） 网络拓扑复杂多变

传感器网络的外部环境复杂多变，存在着各种各样的外界因素。时时刻刻都有节点的死亡以及新节点的加入，网络拓扑不断的变化，因而就要有高效健壮的算法使得整个系统稳定的运行，同时还要考虑能耗的因素。

3） 安全性、实时性

传感器网络会运用于比较敏感的环境中，尤其用于军事。传感器网络严格的资源限制，需要低开销的协议和算法，这些也会带来严重的安全问题。符合传感器网络特性的简洁有效的安全机制是传感器网络最迫切需要的技术之一。同时实时性和能耗也是一对矛盾，要根据具体的需求选择不同的实时标准。

第 2 章　嵌入式实时操作系统

2.1　嵌入式实时操作系统

嵌入式实时操作系统是嵌入式系统的关键技术之一。实时操作系统有硬实时和软实时之分，硬实时要求操作系统设计时保证在规定的时间内必须完成操作；软实时则只要按照任务的优先级，尽可能快地完成操作即可。嵌入式系统实时的响应用户的请求，并为用户屏蔽底层操作，对于上层的开发者底层完全透明，开发者利用系统提供的接口实现所需要的功能。但并不是所有的嵌入式系统都使用嵌入式操作系统，这些系统一般直接面向硬件操作，系统的功能比较简单，对程序员的要求比较高，系统的实时性比较低，而且这种程序一般是一种无限循环的结构，一旦任务崩溃，任务队列中的其它任务也不会得到处理。而传感器网络节点资源有限，这也要求嵌入式操作系统不能过于复杂（朱珍民， 隋雪青， 段斌编，2006）。

嵌入式实时操作系统核心是多任务调度的内核（Robert Filman，2001），它以特定的调度算法调度就绪任务、管理系统的资源、任务间的通信等。内核主要有可剥夺型内核和不可剥夺型内核两种：不可剥夺型内核中断响应快、共享数据方便、上下文的切换较少，但高优先级任务的响应时间受到极大的影响，紧急任务不能得到及时的处理；大多数的实时操作系统的内核都是可剥夺型的内核，每个任务都有自己的优先级，处理器根据任务的优先级及时的处理高优先级任务，但可剥夺型内核的需要的上下文切换较多，调度时间损耗增加（Jean J.Labrosse，2002）。

2.1.1　任务状态转换

系统运行时任务一般有 3 种状态（de Pablo，D.A.L.，2009；Jeffay, K.，Donelson Smith, F.，1998）：等待态、就绪态和运行态。处于等待态的任务一般等待某个事件的完成，或者某项资源被其他任务占用，等待其他任务释放该资源，也可能由于某些原因任务自身延迟一段时间；就绪态是指任务已经获得请求的资源，已经就绪，等待系统分配处理器资源；运行态是指任务获得处理器资源，正处于运行

状态。而任务调度就是指系统按照一定的策略，动态地把处理机分配给处于就绪队列中的某一个任务，以使之执行。3 中状态转化如图 2.1 所示：

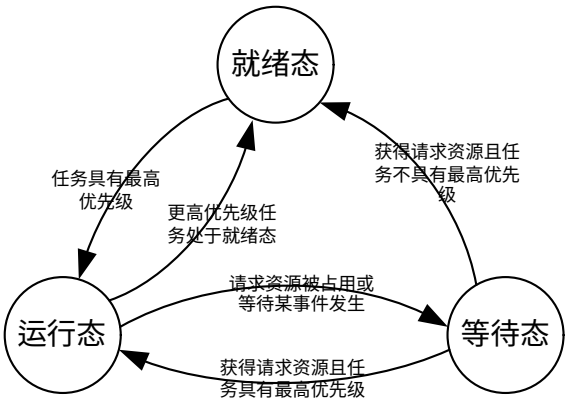

图 2.1 任务状态转化图

2.1.2　调度算法

在多任务系统中，内核根据相应的调度算法为每个任务分配处理器资源。比较常用的调度算法（Bruno Dutertre，Victoria Stavridou，2000；C．M．Krislma,King．GShin，2001）有 5 种：

1) FCFS(先来先服务，First Come First Served)算法，此算法不考虑任务的重要性以及任务周期，按照任务进入就绪队列的顺序进行先后调度，算法复杂度低，但服务质量差，系统效率低，实时性较差，容易引起用户不满；

2) SCBF(Shortest CPU Burst First，最短 CPU 运行优先调度算法)，此算法从就绪队列中选择 CPU 执行时间最短的任务，对其分配 CPU 资源，此算法有较好的调度性能，但很难准确预测任务 CPU 的执行时间；

3) 时间片轮询（胡希明，毛德操，2001）：为每个任务安排时间片进行轮询调度，同样存在实时性的问题，优先级较高的任务不能得到快速的处理

4) EDF（Earliest Deadline First）（Bimbard, F.;，George, L.，2008）：EDF 算法是一种最优的单处理器动态调度算法，此算法比较就绪队列中所有任务的到期时限，将到期时限最早的任务调度执行。EDF 是一种动态优先级调度算法，CPU 在每次调度都要遍历整个就绪队列，时间消耗较大，而且此算法没有兼顾不同任务的重要性；

5) 多级反馈队列方式：此算法是在系统中设置多个就绪队列，并赋予各队列以不同的优先权。优先级高的任务队列先被调度，但同一优先级下的任务实时性很难满足。

2.1.3　内存管理

内存管理是嵌入式实时系统的重要组成部分之一。不同的实时内核采用的内存管理方式不同，在嵌入式系统中一般不采用虚拟内存存储技术，虚拟内存的维护带来大量的资源消耗。同样由于资源的限制，嵌入式系统内存管理策略比较简单快速。

为了加强内存管理的高效和安全，嵌入式实时系统通常采用动态的内存管理策略（封斌，龚灼，扬学军，2002）。在一些功能简单，任务数较少的系统中也可以采用静态的内存分配方式，这种方式可以减少内存的分配时间以及动态分配策略带来的不确定性。静态内存管理内存使用效率很低。动态内存管理的常用方法是堆操作，系统向用户提供 Malloc()以及 Free()函数，用户根据需要对内存进行操作。

动态内存管理策略主要包括固定大小的内存管理以及可变大小的内存管理（William S，2001）。固定大小的内存管理是将内存分成大小相同若干块，每次用户内存操作都是以块为单位，这种内存管理方式不够灵活，每次分配都有部分内存浪费，产生内部碎片（内部碎片小于基本单位块的大小），而且单位块的尺寸需要根据具体的应用环境确定。可变大小的内存管理机制可以分配任意大小的内存块，这种方式比较灵活，可以动态的调整大小，但在内存的使用过程会产生外部内存碎片（大小无法确定）。固定内存管理策略如图 2.2 所示：假设系统内存大小为 64KB，系

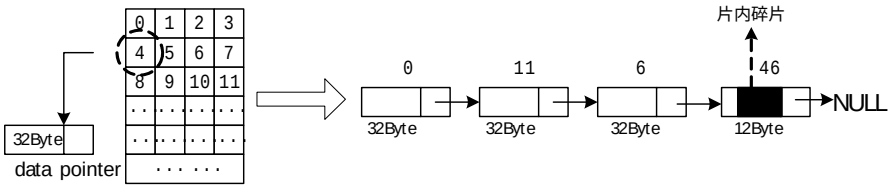

图 2.2 固定内存管理

统将内存分成若干块，每块的大小为 34Byte，其中 32Byte 用于数据存储，2byte 用与指向下一内存块，如果用户需要 108Byte 的内存空间，系统为其分配 4 个内

存块，它们通过链表连接起来，最后一块内存有 20Byte 没有使用，它就是片内碎片。可以看出固定大小动态内存分配策略不但存在片内碎片，而且还存在维护连接的开销。如图 2.3 是一种可变大小内存管理策略的实例，如图 2.3 所示：

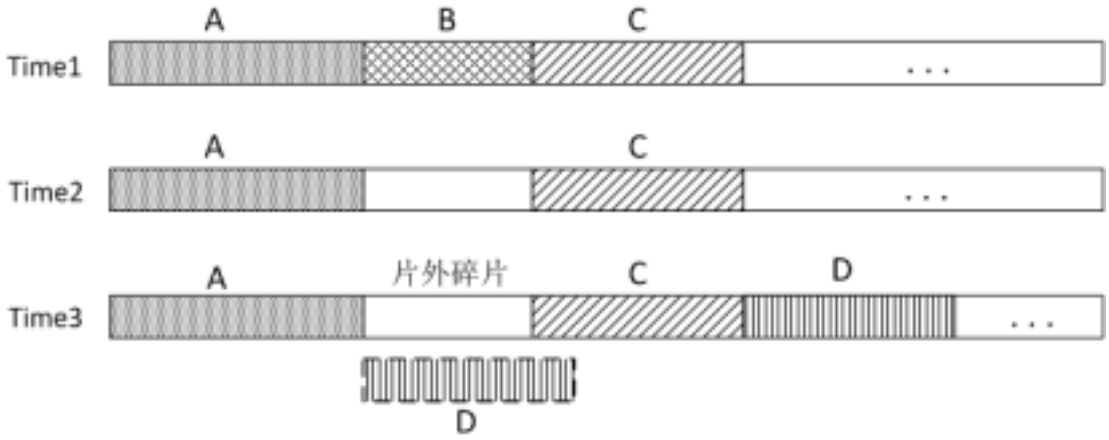

图 2.3 可变内存管理

Time1 时刻 3 块内存空间分别被 A、B、C 三个任务使用，Time2 时刻 B 任务完成（也可以是其他原因），内存空间被释放，Time3 时刻 D 任务申请内存空间，由于 B 释放的连续空间小于 D 要求的大小，那么 B 释放的内存就成为了片外碎片，片外碎片大小无法预料，如果系统存在大量的片外碎片，系统效率将大大降低，处于阻塞状态（等待态）的任务将增加。

2.2　UCOS-II 简介

UCOS-II 是一个开源的、抢占式内核的、实时操作系统。它包含了任务调度、任务管理、时间管理、内存管理、定时管理以及任务间的通信与同步（陈是知，2007）。UCOS-II 使用抢占式的内核，这使得任务的实行性得到了很大的提高。而且 UCOS-II 移植性强，系统稳定可靠，用户可以根据需要对系统进行裁剪。在嵌入式领域，尤其在中小型嵌入式实时系统方面，UCOS-II 凭借其优良的特性得到了广泛的应用。

2.2.1　UCOS-II 体系结构

UCOS-II 是一种微内核，为了加强系统的通用性，它只对嵌入式处理器以及硬件时钟进行了封装和抽象，其他的硬件抽象层没有提供，用户在使用 UCOS-II 时自己要实现需要的硬件抽象，这是一只 UCOS-II 的重要工作之一。

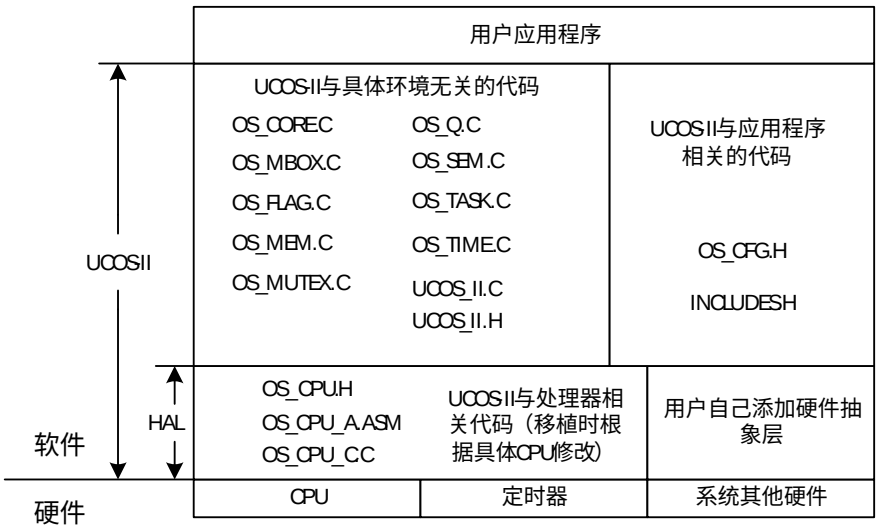

图 2.4 UCOS-II 体系机构

如图 2.4 所示，在 UCOS-II 内有部分代码与具体的环境无关，其中 OS_CORE.C 主要涉及系统的初始化，系统的启动，时钟中断处理，任务切换，任务调度处理等。OS_TASK.C 主要涉及的任务的管理：任务的建立、删除、修改、挂起、恢复等等。与任务通信相关的代码块主要有 OS_MBOX.C、OS_FLAG.C、OS_MUTEX、OS_SEM.C 以及 OS_Q 等。OS_TIME.C 主要用于时钟的管理。OS_MEM.C 主要用于内存管理。UCOS_II.H 主要是用于信号量、任务块数目的设置，以及各种结构体的定义。

UCOS-II 区分系统空间和用户空间，使得系统的安全性得到了保证。绝大部分系统服务有确定的执行时间，系统稳定而可靠。

2.2.2　UCOS-II 任务以及调度分析

任务就是独立执行的线程，是内核调度的基本单位。UCOS-II 中的任务形式上类似于 C 语言中的普通函数，内核不会调用这些函数，任务也没有返回值，因此在形式上任务定义成 void 类型。UCOS-II 是一种可剥夺型内核，在没有更高优先级任务来强度当前任务 CPU 资源时，任务将一直运行，直到任务自身放弃 CPU 的使用权。实时系统中，用户可以在系统启动时创建需要的任务，但大部分任务是通过外部事件触发的。用户完成事件功能函数（任务），外界条件满足，任务将自动运行（Jean J.Labrosse，2002）。

UCOS-II 有一种特殊的结构体——任务控制块 OS_TCB（陈是知，2007）。它用来记录任务的堆栈指针、任务的当前执行上下文、任务的优先级、任务相关

的时间控制块指针等其他一些与任务相关的属性。其中一些信息包含任务的执行代码以及任务堆栈信息。任务的存储结构如下图所示：

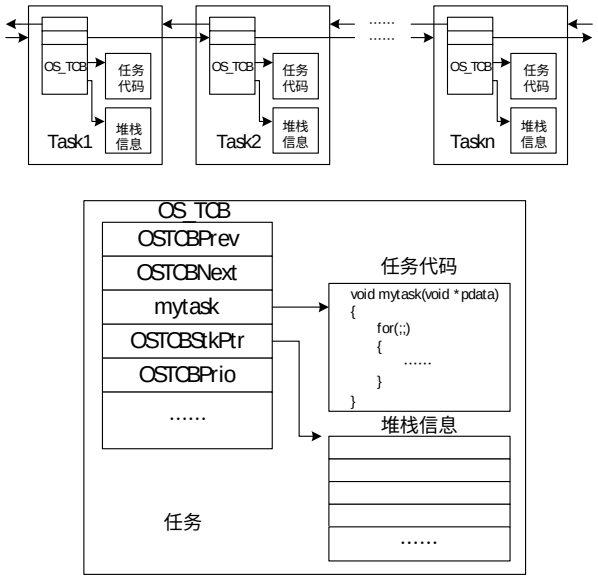

图 2.5 任务存储结构

OSTCBPrev 和 OSTCBNext 是任务控制块的双向链表指针，这对指针在 UCOS-II 内核系统中多处用到：任务创建之前，所有的任务控制块都存放在一个空任务池中 OSTCBFreeList，空任务用 OSTCBNext 连接起来；新任务创建后系统将 OSTCBFreeList 指向空任务块进行赋值填充，并把此任务添加 OSTCBList 中，最后改变 OSTCBFreeList 的值。OSTCBList 中的任务也是通过 OSTCBPrev 和 OSTCBNex 连接起来，时钟节拍函数中将通过它们遍历整个 OSTCBList 链表，修改每个任务的延迟。在任务通信模块中，也将用到这两个指针。

在 OS_TCB 数据结构中 OSTCBStkPtr、OSTCBExtPtr、OSTCBStkBottom 以及 OSTCBStkSize 是用来描述任务堆栈信息的，UCOS-II 用堆栈来保存任务执行的上下文，当发生中断时（文旭，陈兵，2008），系统将任务执行环境保存在任务控制块对应的堆栈中；当发生任务切换时，优先级较低的任务失去处理器资源，执行环境被保存，系统从高优先级任务的任务控制块中得到任务的堆栈地址，将高优先级的任务执行状态加载到处理器寄存器中。

任务控制块结构中还有一些任务相关的属性字段：OSTCBEventPtr、OSTCMsg、OSTCBFlagNode 以及 OSTCBFlagRdy 这些都是与任务通信相关的字段；OSTCBDly 表示系统延迟节拍数；OSTCBPrio 记录任务的优先级；OSTCBStat 记录任务的状态信息；OSTCBX、OSTCBY、OSTCBBitX 以及

OSTCBBitY 这四个属性是系统用来加速系统进入就绪状态，利用空间资源换取时间资源。

系统调度的对象是处于就绪状态的任务，在 UCOS-II 中用就绪表来记录任务是否处于就绪状态，就绪表中包含两个变量 OSRdyTbl[]以及 OSRdyGrp。由于 UCOS-II 系统中的每个任务对应一个优先级，每个优先级上只存在一个任务，以 UCOS-II V2.52 为例，系统最多支持 64 个不同优先级的任务，OSRdyTbl[]是一个类型为 INT8U(此类型在 OS_CPU.H 中定义)的数组。当某个优先级的任务处于就绪状态时，相应的比特位置 1。若 OSRdyTbl[]中某个元素不为 0，那么 OSRdyGrp 相应位为 1。两者的关系如下图所示：

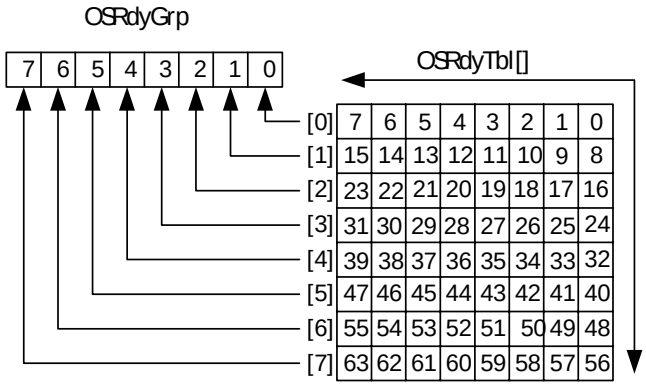

图 2.6 就绪表

当某个任务进入就绪状态或从就绪状态转为其他状态时，系统将自动维护就绪表，任务控制块中相应的属性 OSTCBX、OSTCBY 等将快速的修改就绪表，提高系统效率。任务调度还有一个特殊的数组 OSUnMapTbl[]，将就绪表通过 OSUnMap[]的映射，系统快速得到优先级最高的任务的优先级(prio)。每个任务创建时系统将其任务控制块地址存放在 OSTCBPrioTbl[]数组中，通过优先级 prio 就可以得到最高优先级任务的任务控制块。如上文所描述的，任务控制块包含了任务的堆栈指针、任务的当前执行上下文、任务的优先级等，此时系统就可以进行执行上下文的保护等堆栈操作，完成任务的切换。

UCOS-II 内核调度主要通过 OSSched()函数进行，内核在每个时钟节拍或者发生中断的时候调用 OSSched()函数，这个函数利用映射数组 OSUnMapTbl[]在就绪表中快速找到当前系统中处于优先级最高的就绪任务，然后根据其优先级由控制块优先级表 0STCBPrioTbl[]取得对应任务控制块的地址，然后由

OS_TASK_SW()程序调用 OS_CtxSW()进行上下文的切换，被调度出的任务将自身的执行环境保存在任务块堆栈中。若有中断发生，系统则转向执行中断程序，执行完毕后不一定返回中断调用处，而是由 OSIntExit()进行内核任务调度，执行当前系统就绪表中优先级最高的任务。

2.2.3　UCOS-II 内存管理机制

嵌入式实时操作系统的动态内存管理机制一般有两种：固定大小内存管理和可变大小内存管理机制。在 ANSI C 中一般都是用可变大小的内存管理机制，用户可以调用 malloc()和 free()函数动态的申请和释放内存，但是动态内存分配机制存在片外碎片的问题，片外碎片的大量存在使得系统效率大大降低。在 UCOS-II 系统中，内核对内存进行了分级管理，系统先将连续的内存空间分成若干个区域；再将每个分区分成大小相等的内存块。内存块时系统内存管理的基本单位。不同分区的内存块大小可以不一样，系统可以从不同的分区中得到大小不同的内存块。但处于同一分区内的内存块大小必须相等，内存块的数据类型必须相同[12]。

在使用动态内存之前，开发者先要创建一个内存分区，在 UCOS-II 中一般采用内存管理块 OS_MEM 对内存块进行管理分配。内存管理块记录和跟踪每个内存分区的状态。OS_MEM 机构如下所示：

```
typedef    struct {
    void          *OSMenAddr；        //内存分区指针
    void          *OSMenFreeList；    //内存控制块可用块指针
    INT32U        OSMemBlkSize；      //单位块大小
    INT32U        OSMemNBlks；        //分区内包含的单位块数目
    INT32U        OSMemNFree；        //分区内包含的可用单位块数目
} OS_MEM；
```

当 UCOS-II 初始化时会调用内存控制块的初始化函数 OS_MemInit()，系统将维持一个空内存控制块的链表 OSMemFreeList，这是一个空的内存控制块池，当用户创建内存控制块是，将从其中获得一个空的内存控制块。一个内存控制块被初始化后，其 OSMenAddr 和 OSMenFreeList 指针都指向此内存分区的首地

第 3 章　一种新的嵌入式系统内核

3.1　问题提出

在传感器网络运用环境中，嵌入式操作系统由于受到处理器资源、存储器空间以及能耗等方面的限制，要求其占用的存储空间小，调度不能过于复杂，同时还要满足实时性、安全性、并发性的要求（高富强，秦昌硕，游纪原，2009）。

UCOS-II 在设计时采用单一的基于优先级的抢先式调度算法，每个任务具有不同的优先级，有效地保证了系统实时性。在各种嵌入式操作系统中，其任务切换带来的时延很小，非常适合强实时性的任务要求。虽然 UCOS-II 内核调度算法有效的保证的实时性，但也有许多不足之处：

1) UCOS-II 内核支持的任务数量较少：最新的 UCOS-II 内核最多支持 256 个优先级，每个任务具有不同的优先级。系统最多只能同时运行 256 个任务

2) UCOS-II 内核不支持同优先级任务：一般的网络传输应用中存在大量的发送和接收任务，这些任务之间优先级的高低不是很绝对，每个任务的地位基本是相同的。这些任务实时性的要求不是很高，并有一定的周期性。

3) 基于优先级的调度算法对于优先级反转问题缺乏有效的解决方案：解决优先级反转最好的方式是优先级的继承，由于 UCOS-II 内核不支持同优先级任务，因此 UCOS-II 只能对信号量设定较高的不变的优先级，采用优先级极限的方式解决。这种方式在某些情况下效率比较低。

4) UCOS-II 内核内存管理策略效率低：UCOS-II 采用分级固定大小内存分配策略，用户只能简单的申请和撤销一个内存块，功能简单，不能完成一些较为复杂的内存操作。

本章综合常用的内核调度几种算法（李小群，赵慧斌，叶以民，2003；陈文智，谢铖，石教英，2006），基于无线传感器网络的特性，将优先级调度与基于时间片的调度结合起来。系统存在多个优先级，每个任务初始化分配一个时间片，相同的优先级任务初始时间片相同，优先级越高初始化时间片越小。优先级高的就绪队列先得到调度，结合 SCBF 算法的特性，同优先级中时间片较小的任务先得到

调度。我们对 UCOS-II 内核调度程序进行修改，在 UCOS-II 实现新的调度算法。并改进 UCOS-II 的动态内存管理策略进行改进，使其更加方面用户使用管理。

3.2　内核调度算法改进

为了满足网络服务特性和系统实时性的要求，新的调度算法将采用分层式的调度。有个整体的优先级，系统的任务是分优先级的，高优先级的紧急任务会优先获得 CPU 的资源。而同时系统也是兼顾公平的，系统允许一个优先级对应多个任务，这些任务的地位是相同的，他们是通过时间片进行调度，为了提高系统性能，系统将在同优先级上调度拥有时间片最少的任务执行。

3.2.1　内核相关数据结构的修改

在新的调度算法中，一个优先级对应多个任务，因此系统加入一个新的结构体 OS_Prio（优先级块），还要增加一个全局的优先级块池 OSPrioFreeList，它的作用与 OSTCBFreeList 类似，当某个优先级的任务创建时，将在 OSPrioFreeList 申请一个可用的 OS_Prio 结构，它的地址将被存入 OSPrioTbl[]中。每个创建好的 OS_Prio 结构将指向一个双向链表，其中包含的是此优先级下不同的任务。他们之间是通过时间片进行调度，如图 3.1 所示。

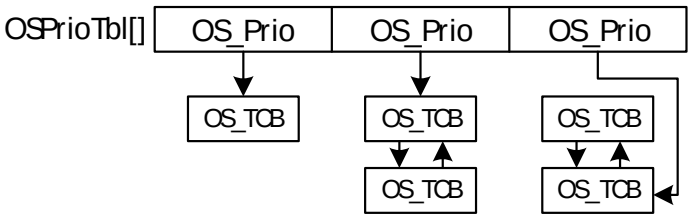

图 3.1 OS_Prio 与任务块之间的关系

新的结构体 OS_Prio 主要用来第一层的调度，调度不同的优先级，它主要由下面几个成员组成：

INT8U OSPrio：优先级块的优先级。

INT8U OSPrioStat：优先级块的状态，如果其中有任务处于就绪状态，那么这个优先级块就置为就绪。

INT8U Count：此优先级下已有任务数。

INT8U Rcount：此优先级下进入就绪态的任务数。

INT8U Tcount：此优先级就绪任务中拥有时间片的数量。

struct os_tcb *OSTCBF：指向此优先级对应的任务双向链表中需要调度的任务。

INT16U OSTCBFId：双向链表的第一个任务 Id。

INT16U OSTCBLId：双向链表的最后一个任务 Id。

INT8U OSIdGrp：和 OSIdTbl[]一起对 TCB 任务块进行 Id 分配。

INT8U OSIdTbl[]：和 OSIdGrp 一起对 TCB 任务块进行 Id 分配。

任务控制块 0S_TCB 也要做一定的修改，在其中增加三项：

INT8U TimeCt：当前任务所拥有的时间片，任务创建时根据当前优先级分配。

struct os_tcb *OSTCBN：指向双向链表的下个任务。

struct os_tcb *OSTCBP：指向双向链表的上一个任务。

3.2.2　任务 Id 的自动分配

传统操作系统中，内核为每个进程自动生成进程控制符 PID，进程控制符唯一的标示一个进程。内核可以通过 PID 对进程进行控制管理。在 UCOS-II 系统中，每个任务拥有一个优先级，每个优先级最多一个任务，因此任务的优先级就可以作为任务的 PID，系统可以通过任务优先级 prio 在数组 OSTCBPrioTbl[]得到任务的 OS_TCB(任务控制块)地址，系统通过任务控制块就可以得到任务的执行上下文、堆栈信息、事件信息等，可以对任务块进行操作。在新的调度算法中，同一优先级下有多个任务，优先级已经不能作为一个任务的唯一的标识。在这里我们用 OSTCBId 作为任务的唯一标识。OSTCBId 在 UCOS-II 中作为保留项，在此系统中将其用作同优先级不同任务的 Id 号，系统将通过 OSTCBId 对任务进行控制管理。最新的 UCOS-II 内核支持最多 256 个优先级任务，本算法对原 UCOS-II 进行裁剪，支持 8*8 个优先级，每个优先级下最多支持 8*8 个同优先级任务，最多可支持 4096 个任务。OSTCBId 由两部分构成(如图 3.2 所示)：前 8 位为任务块的优先级，后 8 位为系统自动分配的 Id 号。

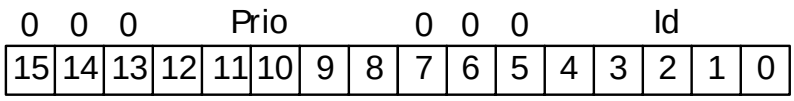

图 3.2 OSTCBId 结构

用户创建任务时不必关心具体的 Id 号，只要提供任务的优先级，而任务 Id 的维护和管理由系统自动完成的。OS_Prio 结构体中的 OSIdGrp 和 OSIdTbl[]两项属性就是用来对任务 Id 进行管理的。

OSIdGrp 和 OSIdTbl[]的功能与 OSRdyGrp 以及 OSRdyTbl[]作用类似，利用 OSIdGrp、OSIdTbl[]和 OSUnMapTbl[]能够快速的为任务动态分配 Id。初始化时 OSIdGrp 和 OSIdTbl[]所有位置 1，表示此编号的 Id 没有被使用；当某个 Id 被分配给任务块时，OSIdTbl[]相应位置 0。如果 OSIdTbl[]数组某个元素为 0，相应的 OSIdGrp 置 0，如下图所示。

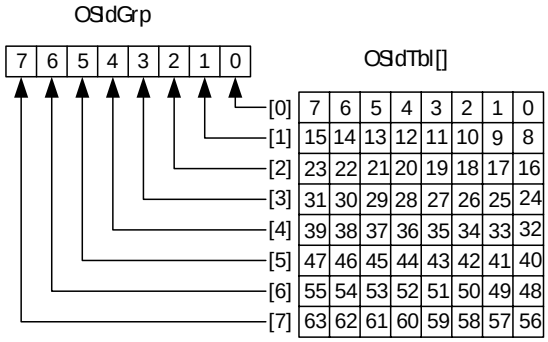

图 3.3 OSIdGrp 和 OSIdTbl[]之间的关系

下面我们分析任务创建的过程以及 Id 的分配。用户新建优先级为 Prio 的任务，系统先从 OSPrioTbl[Prio]获取其地址，如果这个值为 0，表示 OS_Prio 没有建立，那么先创建 OS_Prio 结构体，系统从 OSPrioFreeList 链表头部获得一个空的 OS_Prio，对其初始化然后将其地址存入 OSPrioTbl[Prio]中。OS_Prio 机构体进行赋值操作，OSPrio 的值为 Prio，OSIdGrp 和 OSIdTbl[]初始化时所有的位置 1，Count 值为 0。然后创建第一个任务，系统分配其 Id 为 0，此任务块的 OSTCBId 将由 Prio 及 Id 组合而成。任务创建完成，系统将 OSIdTbl[0]的第 0 位置 0，表示此 Id 已经被使用。如果有同优先级的其他任务创建，先利用 OSIdGrp 及 OSIdTbl[]到 OSUnMapTbl[]的映射获取未被使用的 Id。如果 OS_Prio 中 Count 值为 64，表示此优先级下所有 Id 都被分配结束。某个任务创建完成，系统将其

地址存入 OSTCBPrioTbl[]数组中，在任务调度时将由 OSTCBPrioTbl[OSTCBId]得到任务块地址。

3.2.3　任务调度

在每个任务创建时，系统为其分配一个时间片(TimeCt)，时间片的大小应该是一个与优先级相关常量数组 DTime[prio]，优先级越高 TimeCt 初始值越小，相同优先级任务获得的时间片相同，用户可以根据需要对常量数组 DTime[]进行配置。如果某个任务进入就绪态，那么该任务将被加入到对应 OS_Prio 指向的双向链表中。由于在基于时间片的调度算法中，同优先级下拥有时间片较少的任务应该被优先调度，因此应要遍历整个链表，这样将大大提高调度所用的时间。系统在就绪任务加入双向链表时，进行一定的处理，从而保证时间片较少的任务先得到执行。任务加入双向链表过程如图 3.4 所示：

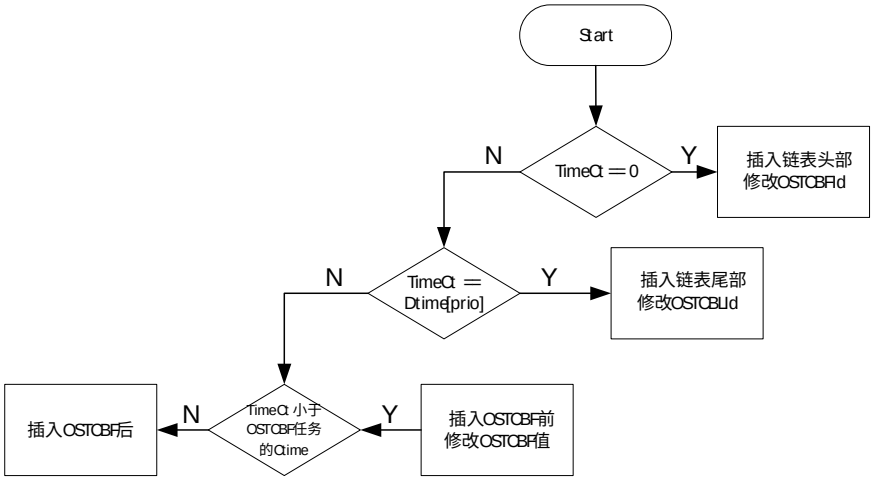

图 3.4 任务块插入

在任务加入双向链表前，查看当前任务所拥有的时间片 TimeCt。若 TimeCt 的值为 0，则将此任务插入到双向链表头部并改变任务优先级级块 OS_Prio 中 OSTCBFId 属性值；若 TimeCt 的值为此优先级对应的任务所分配的初始时间片值，则添加到双向链表的尾部并修改 OSTCBLId ;若介于两者之间，则与 OSTCBF 的 TimeCt 比较，如小于则插入当前 OSTCBF 前并将 OS_Prio 中 OSTCBF 指针指向新任务，否则就插入到 OSTCBF 的后面。

经过上面的操作整个调度算法就得到了一定程度的简化，如图 3.5 所示：

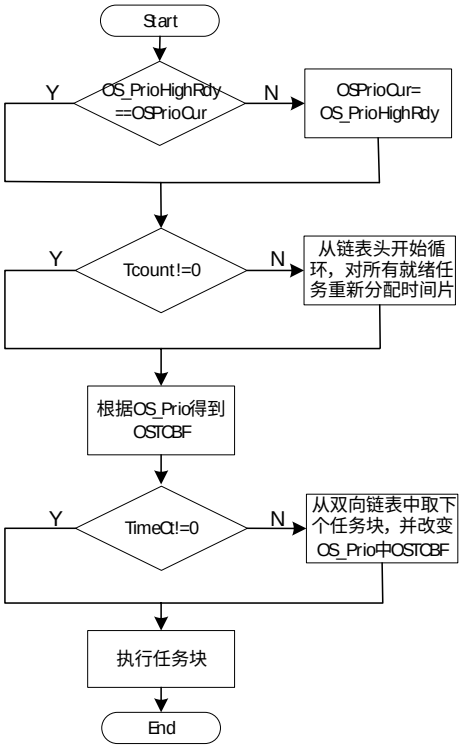

图 3.5 调度流程图

从图 3.5 可以看出，调度开始时先判断当前系统运行的优先级块是否是正在运行的任务对应的优先级块。若不是，则对 OSPrioCur 重新赋值。如果 OS_Prio 中所用的任务时间片用完，则对所有的就绪任务重新分配时间片，然后就是查看 OS_Prio 指向的调度任务是否拥有时间片，若没有就调度下个任务执行。

新的调度算法采用分层调度，结合优先级调度和 SCBF 算法优点，有效的解决了 UCOS-II 在调度中的不足，系统最多可支持 64 个不同优先级 4096 个任务。新内核同一优先级支持多个任务，有效的解决优先级反转等问题。在第 5 章中，我们将在无线抄表软硬件的基础上，将两种内核进行性能比较，从实验结果可知，新的内核调度算法大大提高了网络传输中的系统效率。

在具体的现在实现过程中，任务间的通信(信号量、邮箱等)与任务调度结构相似，因此也要做相应的修改。本系统只用到了信号量，因此只对信号量相关做了相应的改进。

3.3　内存动态管理策略改进

UCOS-II 采用分层式的内存管理策略，先将内存分成多个分区，每个分区内使用固定大小的动态内存管理策略。系统提供简单的对单位块申请和释放的函

数，向用户提供的功能过于单一，用户程序需要过多的操作，固定大小的内存分配策略会产生较多的内部碎片，而且此内存策略需要大量的空间维护内存块之间的连接关系。我们对 UCOS-II 的动态内存管理策略进行改进以提高系统效率。

本系统采用可变大小内存控制策略，用户可以根据具体的需求动态的请求内存空间任意大小的内存空间，系统自动对释放的内存空间进行回收合并。

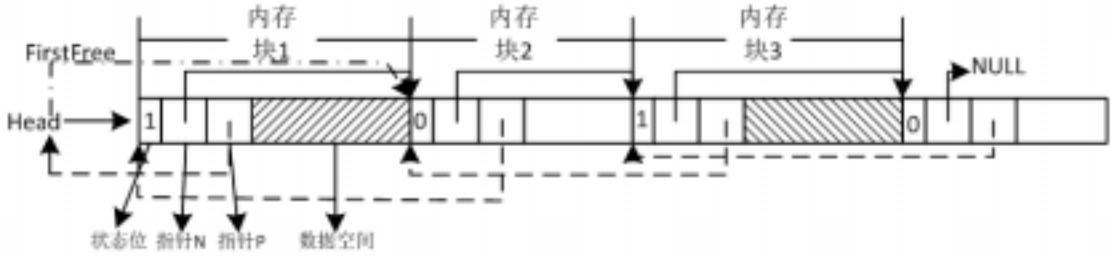

图 3.6 内存管理示意图

如图 3.6 所示，存在一个对内存进行管理的内存控制块 Head，系统用这个内存控制块记录跟踪内存使用状态。内存控制块结构如下：

```
typedef    struct {
        void    *MemAddr；          //内存块开始地址
        void    *FirstFree；        //指向内存中第一块未使用区域
        INT32U MemSize；            //内存块的总大小
        INT32U MemFree；            //当前可分配的内存大小
} Head；
```

新的内存管理策略中，用户可以申请任意大小的内存空间，系统采用最小颗粒对齐，最小颗粒为两个字节。使用的内存一般分为 2 个部分：内存管理单元以及用户数据块。用户管理单元式固定大小如图 3.6，其中包含一个比特的标志位。如果标志位为 1，表示此块内存正被使用；如果标志位为 0，表示此块内存处于空闲状态。紧跟在状态位后面的是 2 个指针，都是 2 个字节大小，因此本系统最大可管理的内存大小为 64KB。内存块的两个指针构成一个双向链表结构，可以很方便的访问邻近的内存块。第一个指针是 N 指针，它指向下个内存块；第二个指针是 P 指针，它指向上个内存块。如果某个内存块的 P 指针指向 Null，表示此后都是未使用部分。

新的内存管理策略主要提供 3 种内存操作，内存初始化 MemInit()，内存分配 MemMalloc()；以及内存的释放 MemFree()。

内存的初始化，主要是创建 Head 结构，并对其进行赋值，MemAddr 和 FirstFree 指向可分配内存的起始位置，MemSize 和 MemFree 相等，为初始化内存区域的大小。并赋值状态位为 0，P 指针指向 Head，N 指针指向 Null。

内存的分配过程中，系统通过 Head 结构体中的 FirstFree 找到第一块空闲内存块，利用 N 指针地址和自己的地址得到内存块的大小，如果不合要求，就利用此内存块的 N 指针访问下个内存块，首先判断状态位，如果是可用空间，则进行大小判断，否则之间通过 N 指针继续遍历。如果 N 指针指向 Null，表示后面都是未使用过的内存部分，系统根据用户需求分配合适的空间，如果在遍历过程中找到合适的内存块，则将此内存块分成两块，一个使用块一个空闲块，并改变相应的状态位，改变双向链表，此过程中还有可能对 Head 中的 FirstFree 重新赋值。在分配结束后还要修改 Head 中 MemFree 的值。内存分配操作将返回给用户实际的内存分配空间(本系统采用是字节对齐)。

内存的释放主要涉及到内存管理块中状态位的改变，以及空闲块的合并过程。内存块的合并要对前后的内存块的 P 指针和 N 指针进行修改，主要分成 4 中情况：

1） 释放块的左右都是空闲块：此情况下 3 块空闲块将合并，修改第三个空闲块 N 指针指向的内存块，将其 P 指针指向第一个空闲块；再修改第一空闲块，将第三个空闲块的 N 指针的值赋予第一空闲块。

2） 释放块左边是空闲块：此情况下要改变左边空闲块的 N 指针，让其指向释放块 N 指针指向的内存块，还要修改释放块的下个内存块(N 指针指向的内存块)，修改其 P 指针，让其指向左空闲块。

3） 释放块右边是空闲块：此情况与上种情况类似，修改释放块的 N 指针以及右空闲块指向的内存块的 P 指针，此种情况下可能修改 Head 结构体中 FirstFree 的值。

4） 释放块左右都是非空闲块：这个情况下，只需要改变状态位，不需要改变左右指针，但这种情况下，可能需要改变 Head 中的 FirstFree 的值，将释放块的地址与 FirstFree 比较，FirstFree 取两者之间的小值。

第 4 章　基于 IPv6 无线抄表系统

4.1　背景

为满足营销现代化和建设统一坚强智能电网的需要，确保电力用户用电信息采集数据的及时性、准确性、完整性和采集系统安全、稳定、高效的运行。非常有必要实现用电信息集中采集系统的一体化，建立一套稳固、高效的数据采集系统，实现各种集抄系统的兼容、安全、稳定运行，确保电能信息采集与预付费业务，以及每年冬夏两季用电高峰的负荷管理信息的有效性、及时性；确保对“ SG186 ”信息系统工程和智能电网建设需要的扩展服务，得到及时、满意的解决。

长期以来，电表数据抄送问题都是相关供应部门非常想解决但又得不到切实解决的问题。在行业信息化过程之中，户表数据的自动化抄送具有非常重大的意义，因为户表数据是相关行业销售过程中最原始的数据，这个数据的准确度和及时性直接影响了行业内部其它信息化水平。传统的手工抄表费时、费力，准确性和及时性得不到可靠的保障，这导致了相关营销和企业管理类软件不能获得足够详细和准确的原始数据。一般手工抄表都按月抄表，对于用户计量来说是可行的，但对于相关供应部门进行更深层次的分析和管理决策却不够。

采用智能化的数据采集系统（齐秀波，2006；徐伟，2009）可以大大减少了人工劳动，提高工作效率，减轻劳动强度，减员增效，降低了管理费用，基于 UCOS-II 的数据采集系统是电子信息领域集成电路产品的开发应用，它是由一定数量节点组成的无线自组网络，综合了传感器、嵌入式计算、分布式信息处理、微电子技术和无线通信技术。

目前，大多数无线传感器网络都不能孤立的运行，一个单独的无线传感器网络必须和一个外部网络连接在一起，以便使外部网络可以监测和控制无线传感器网络。TCP/IP 协议栈现在已经成为了大规模网络标准，因此将无线传感器网络和 TCP/IP 网络互联是非常有意义的。

根据全球 IPv4 地址近年的消耗状况，地址资源专家预测 IPv4 地址将会在 2012 年前后耗尽。为了保证未来的 Internet 中的设备都有一个唯一的地址，下一代 Internet

协议——IPv6 可以进行扩展从而得到足够多的 IP 地址。由此可以看出为了扩大无线传感器网络的应用并在无线传感器网络的研究中保持国际先进的地位，实现支持 IPv6 的无线传感器网络节点的意义重大。

4.2　硬件模块介绍

在数据采集网络中节点分为两种，一种为数据采集器，一种为数据集中器。它们在硬件组成上没有什么区别，只是数据集中器节点增加了与网关进行连接的接口(GPRS 模块)。节点的硬件结构主要由微处理器模块、存储模块、无线射频模块、电源模块和串口通信模块（表计）组成，框图如下：

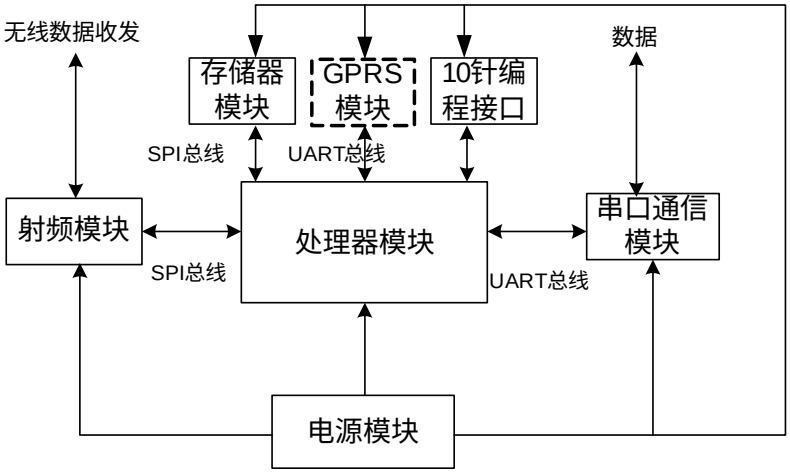

图 4.1 无线抄表系统硬件框图

本无线采集系统处理器模块采用 ATmega128L，射频模块采用 NRF905 射频芯片实现无线数据的收发，存储器模块采用ATMEL 串行 FLASH 存储器，通过 SPI 总线与微处理器进行数据传送与接收。串口通信模块采用 RS-485 总线接口芯片 MAX3430_ESA，允许多达 128 个收发器接入总线，可以使一个采集器节点连接多个智能表计。串口通信模块通过 UART 总线将数据发送给处理器，处理器模块将数据通过 SPI 总线存储到 Flash 存储器中。从射频模块收到数据通过 SPI 总线传到处理器，处理器根据内部的 EEPROM 中存储的信息执行相关的操作，可能将得到的数据存储转发，也可能先从 EEPROM 得到路由信息，再从存储器模块读取数据，通过 SPI 总线发送到射频模块，再由射频模块完成数据的发送无线。

数据集中器主要完成对网络的控制、维护、数据的汇集、存储和提交。通过 UART 总线将数据发送到 GPRS 模块，通过 GPRS 模块发送到 Internet 与网关服务

器相连。数据采集器可以设置成定时上传表计数据给数据集中器，也可以由数据集中器下达指令给数据采集器，要求数据采集器上传某一时段的表计数据。

4.2.1　电源模块

电源模块给整个硬件系统提供稳定而准确的能源，是系统能够工作做起来的先决条件。本节点需要的电压范围为 3.0vDC ~ 5vDC。电源模块图如图 4.2 所示：

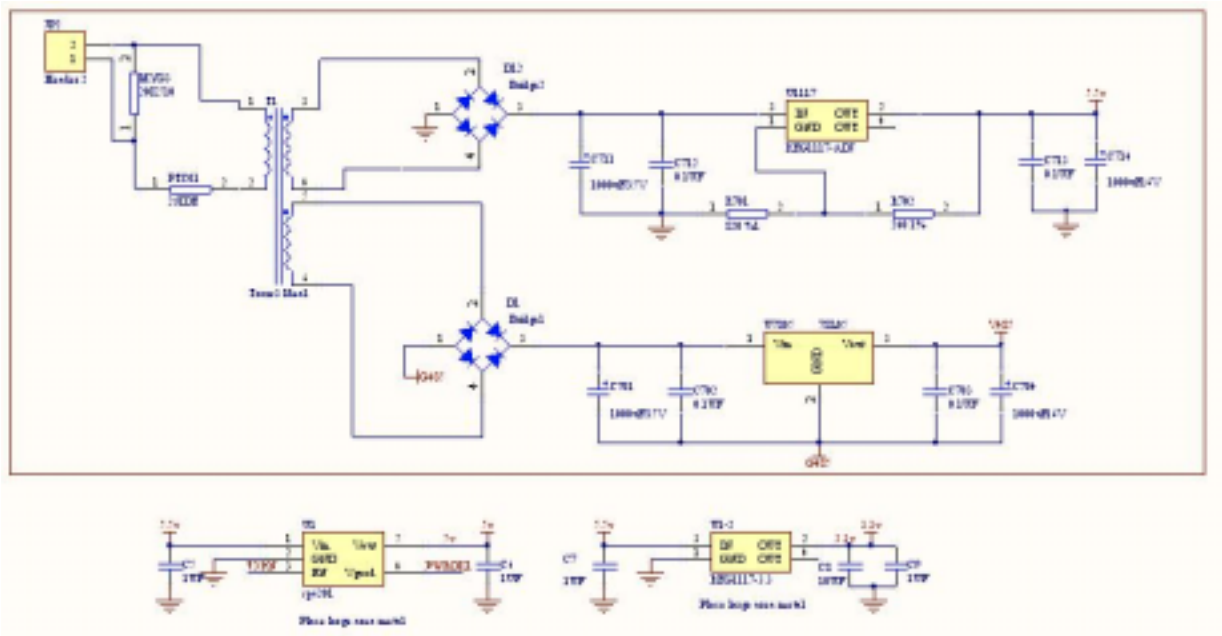

图 4.2 电源模块

系统的输入为 4 节 AA 电池，在调试阶段可用稳压源代替。第一阶电源模块电路产生 5.5v 电压以及 5v 电压，5v 电压主要给 RS485 串口通信模块供电，第二层的电源转换电路产生稳定的 5v 和 3.3v 电压。5v 电压给 RF 射频模块供电，3.3v 电压给处理器供电。

4.2.2　射频数据收发模块

射频模块主要完成数据的无线收发。主要芯片选用 nodic 公司的 nRF905 单片无线收发器（Guo Qing， Hu Jingquan，Hu Hongzhi，2007；杨剑，2007）。工作在 433/868/915MHZ 的 ISM 频段。由一个完全集成的频率调制器，一个带解调器的接收器，一个功率放大器，一个晶体震荡器和一个调节器组成。ShockBurst 工作模式的特点是自动产生前导码和 CRC ，可以很容易通过 SPI 接口进行编程配置。电流消耗很低在发射功率为-10dBm 时，发射电流为 11mA ，接收电流为 12.5mA。ShockBurst 工作模式的特点是自动产生前导码和 CRC；可以很容易通

过 SPI 接口进行编程配置；具有 POWERDOWN 模式，可以很容易实现节能。使用 nRF905 可以使无线抄表系统采用 1GHz 以下的频段，避免了干扰。射频电路模块如图 4.3 所示：

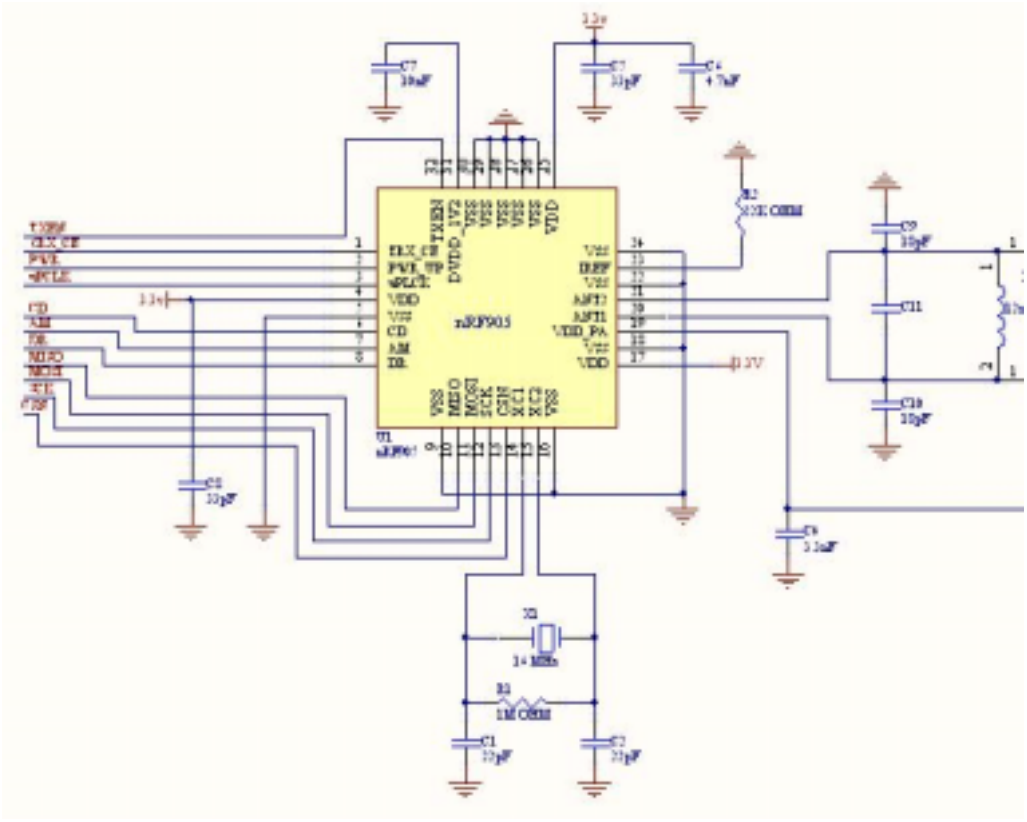

图 4.3 射频电路模块

射频模块的主要功能在于无线数据的准确发送和准确接收以及无线数据的加解密，保证数据的安全可靠传输。NRF905 芯片与 CPU 之间通过 4 线 SPI 接口进行一个字节的数据传输，另外有 6 根线作为他们之间的控制信号，以及 1 根线作为中断信号线连接到 CPU 外部中断引脚上，实现接收中断。

4.2.3　处理器模块

采用 ATmega128L（刘兰香，张秋生，2006），基于 AVR RISC 结构的 8 位低功耗 CMOS 微处理器。由于其先进的指令集以及单周期指令执行时间，ATmega128L 的数据吞吐率高达 1 MIPS/MHz，可以满足节点数据处理和传输的速度。内部有 128KB 的 FLASH 程序空间，4KB 的 RAM 空间以及 4KB 的 EEPROM 空间，4K 字节的 EEPROM，寿命为 100,000 次写/ 擦除周期，可满足节点协议栈的运行条件。JTAG 在线编程，方便调试。串行外设接口 SPI 允许

ATmega128L 和外设之间进行高速的同步数据传输,实现与射频芯片的数据传输；具有两个 USART，实现与外设的串行通信。处理器外接串行 FLASH 存储器 AT45DB041B，实现对数据信息的存储。处理器模块电路如图 4.4 所示：

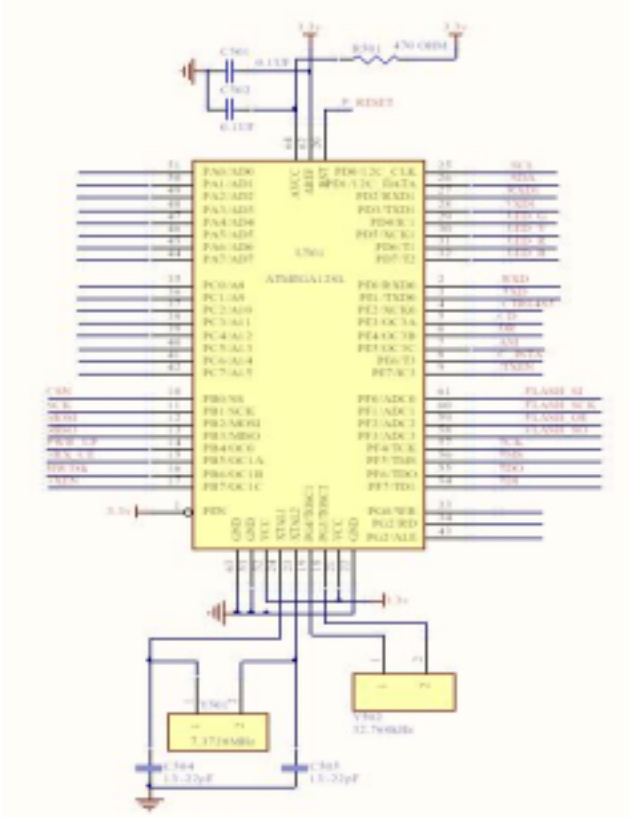

图 4.4 处理器模块电路

4.2.4　串口通信模块

采用 RS-485 总线接口芯片 MAX3430_ESA, MAX3430 是带故障保护的 RS-485 收发器，输出引脚可容忍对地±80V 的过压；;无错传送数据速率为 250kbps。MAX3430 具有 1/4 单位负载的接收器输入阻抗，允许多达 128 个收发器接入总线，可以使一个采集器节点连接多个智能表计。数据采集电路设计如图 4.5 所示：

RS-485 总线接口芯片与微处理器间采用光耦隔离电路，信号单向传输，输入端与输出端完全实现了电气隔离，抗干扰能力强，工作稳定，使用寿命长，传输效率高。

在采集器节点中，串口通信模块连接智能表计，以实现智能表计与采集器的数据通信，将表计信息上传至抄表系统中，进行传递。

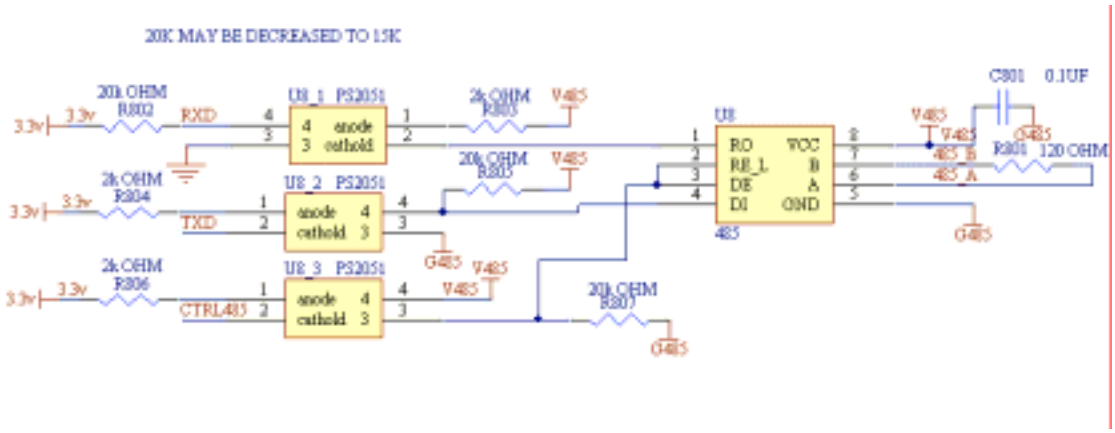

图 4.5 数据采集电路

存储器模块

存储器模块采用ATMEL 串行 FLASH 存储器 AT45DB041B，存储容量为 2Mbit，用来存储数据,通过 SPI 与微处理器进行数据传送与接收。

编程调试模块

本模块主要用于程序的下载和调试，另外本模块外加了串口线，可以作为串口用。编程调试模块电路原理图如下图 4.6 所示：

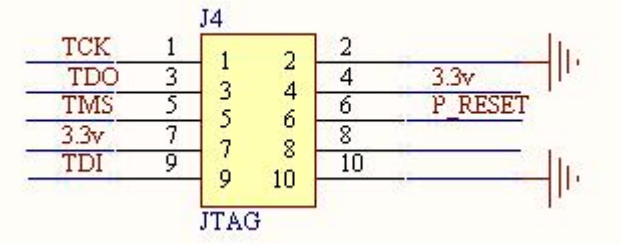

图 4.6 编程调试模块电路图

4.3　系统移植

系统移植的过程就是对与处理器相关的代码的修改、应用程序相关代码的修改、内核调度算法的改进以及硬件抽象层的实现（http://www.micrium.com）。

4.3.1　处理器相关代码修改

处理器相关代码主要包括 OS_CPU.H，OS_CPU_C.C 及 OS_CPU_A.ASM 文件。OS_CPU.H 包括与处理器 atmega 相关的常数、宏及类型的定义（Jean J.Labrosse，2002）。

```
typedef    unsigned    char    BOOLEAN;
typedef    unsigned    char    INT8U;      /* Unsigned 8 bit quantity */
typedef    signed    char       INT8S;      /* Signed 8 bit quantity */
typedef    unsigned    int       INT16U;    /* Unsigned 16 bit quantity */
typedef    signed      int      INT16S;    /* Signed 16 bit quantity */
typedef    unsigned long    INT32U;    /* Unsigned 32 bit quantity */
typedef    signed      long    INT32S;    /* Signed 32 bit quantity*/
typedef    float    FP32;       /* Single precision floating point*/

typedef unsigned char    OS_STK; /* Each stack entry is 8-bit wide*/
typedef unsigned char    OS_CPU_SR;
```

OS_CPU.H 还包含 OS_ENTER_CRITICAL()、OS_EXIT_CRITICAL()的实现，这两个函数保护临界代码，对中断进行关闭和开启。在 UCOS-II 系统中，对临界代码的保护有三种策略：第一种策略是直接执行处理器对应的关中断指令 CLI 以及开中断的指令 STI；第二种策略是关中断的时候将原中断状态保存到堆栈中，然后关中断，开启中断时将状态出栈回复中断状态；第三种策略是执行特定的函数将中断状态寄存器的内容保存到局部变量中。在系统移植过程中，我们采用第三种方式，构造两个函数 OS_CPU_SR_Save()和 OS_CPU_SR_Restore()，他们将中断寄存器的内容存放在局部变量 cpu_sr 中。由于在 ATmega 系统中处理器的堆栈是从高内存地址向低内存地址递减的，因而设置 OS_STK_GROWTH 的值为 1。实现 OS_TASK_SW()函数，此函数用来任务的切换，它是通过 OSCtxSw()实现的，OSCtxSw()函数将在文件 OS_CPU_A.S90(OS_CPU_A.ASM)文件中用汇编代码实现。

在 OS_CPU_C.C 文件中主要实现的函数是 OSTaskStkInit()，任务初始化时将调用此函数初始化任务堆栈结构，此函数模拟中断，将 ATmega 的 32 个通用寄存器以及中断寄存器 SREG、堆栈指针 SPH 和 SPL 的值保存在堆栈中。

在文件 OS_CPU_A.S90 中主要实现实现 4 个汇编函数：OSStartHighRdy()、OSCtxSw()、OSIntCtxSw()以及 OSTickISR()。OSStartHighRdy()函数用来被 OSStart()调用来用使得就绪态任务中优先级最高的任务开始运行，此函数中将设置全局变量 OSRunning 的值为 True，然后对处理器的寄存器进行赋值，从运行任务的堆栈中

加载任务的状态。OSCtxSw()用来在任务切换时对当前堆栈和最高优先级堆栈之间的切换，然后对处理器寄存器进行任务状态的加载。OSIntCtxSw()与 OSCtxSw()类似，在 ISR 中执行任务切换功能。OSTickISR()时钟节拍中断处理函数，本系统用 ATmega 的 16 号中断 TIMER0 COMP 来实现 UCOS-II 的时钟节拍。

4.3.2　应用程序相关代码修改

应用程序相关代码的修改主要涉及的文件有 OS_CFG.H 和 INCLUDE.H。OS_CFG.H 文件中主要是对系统的配置：是否使用互斥信号、是否使用信号队列、任务数目等等。系统所有的*.C 文件都会包含 INCLUDE.H 文件，而且只会包含 INCLUDE.H 头文件，这样用户不必关心程序具体需要哪些头文件，加强了系统的移植性，但增加了系统的编译时间。在 INCLUDE.H 中我们加入了 iom128.h 和 ina90.h，这两个头文件主要是 ATmega 处理器的 IO 端口配置。还在其中加入了运用程序相关的头文件定义 app.h。

4.3.3　内核代码修改

在无线抄表系统中，我们将修改原 UCOS-II 的内核调度算法以及内存分配策略。在调度算法改进中，系统加入新的结构体 OS_Prio，并对 OS_TCB 进行修改，修改任务进入就绪态的策略，对 OSCtxSw()、OSIntCtxSw()以及 OSTickISR()进行修改，实现分层形式的内核调度算法。由于在任务通信模块和任务调度有部分相似，本系统只用到了信号量，因此对信号量中的事件块 OS_ENENT 进行修改，实现分层结构。在内存动态管理策略中加入新的结构体——内存控制块，并对用户任务提供内存的动态分配 MemMalloc()和释放函数 MemFree()函数。在内存动态管理策略中还提供了动态内存的初始化函数 MemInit()。

4.3.4　硬件抽象层

硬件抽象层主要是对硬件的抽象，为上层提供透明的接口函数，上层软件可以方便高效的操作底层硬件系统。在无线抄表系统中硬件驱动层主要是串口驱动（陈是知，2007；宋宝华，2008）。协议栈的底层部分，系统直接对底层进行封装，也是硬件抽象层的一部分，我们在下节将会详细介绍。

串口驱动主要是实现 UART 的数据收发，串口的数据收发是通过缓冲区或者 IO 端口位置的读写来实现的。UART 的数据收发可以通过轮询方式以及中断方式，

轮询方式是通过检测 UART 的状态寄存器的比特位实现，这种方式下可能造成数据的丢失，因此无线抄表系统中采用中断方式来接收和发生数据。

在中断模式下，接收数据时系统每次从移位寄存器中接收一个字节的数据，并将数据送入到数据输入寄存器，同时产生一个中断。数据发送时，系统从数据输出寄存器发送一个字节数据，发送结束后产生一个中断。在 UART 的数据收发过程中一般会用到缓冲区域，用它们来缓存数据供应用程序使用。此时在数据存入缓冲区域以及应用程序调用数据时就会产生互斥，系统通过信号量完成互斥操作，只用获得信号量才能相应的存入和读取操作。在本系统中我们采用 OS_COM_BUF 结构体将收发缓冲区以及相应的信号量结合起来。

```
typedef struct {
    INT16U      NumOfRx;      //接收缓冲区字节数
    OS_EVENT    *BufRxSem;    //接收信号量
    INT8U       *BufRxIn;     //接收缓冲区数据输入指针
    INT8U       *BufRxout;    //接收缓冲区数据输出指针
    INT8U       BufRx[COM_Rx_SIZE];    //数据接收缓冲区

    INT16U      NumOfTx;      //发送缓冲区字节数
    OS_EVENT    *BufTxSem;    //发送信号量
    INT8U       *BufTxIn;     //发送缓冲区数据输入指针
    INT8U       *BufTxout;    //发送缓冲区数据输出指针
    INT8U       BufTx[COM_Tx_SIZE];    //数据发送缓冲区
  } OS_COM_BUF;
```

接收和发送缓冲缓冲区都采用环状结构，用 NumOfRx/ NumOfTx 记录缓冲区内的数据字节数，用 BufRxIn/ BufTxIn 和 BufRxout/ BufTxout 指针来控制缓冲区数据的变化。BufRxSem/ BufTxSem 是用来保证缓冲区数据正确性的信号量。BufRx[COM_Rx_SIZE]/ BufTx[COM_Tx_SIZE]是用来存放数据的缓存区。

UART 串口驱动接收模型如图 4.7 所示

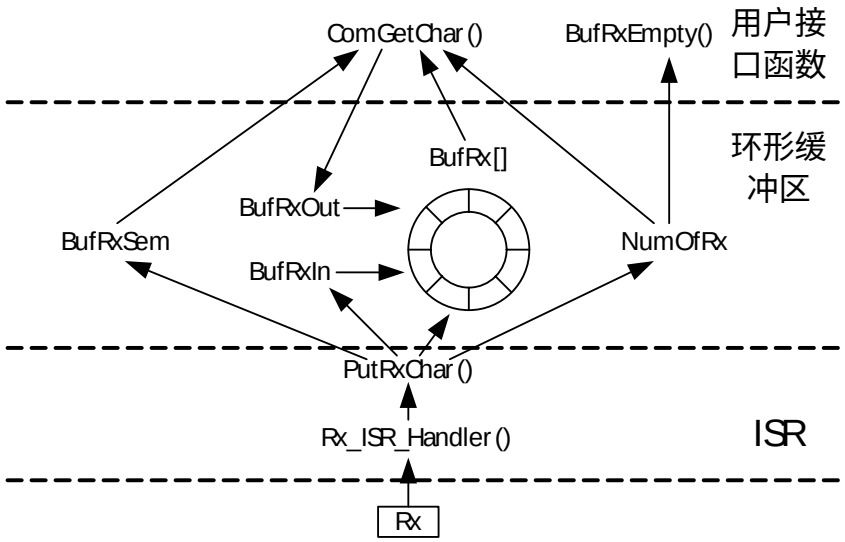

图 4.7 UART 串口驱动数据接收模型

用户任务通过调用 ComGetChar()函数从环形缓冲区中获取数据。任务等待接收信号量 BufRxSem，来判断缓冲区 BufRx[]中是否有数据，如果缓冲区中没有数据，任务就进入挂起状态，等待信号量。如果缓冲区中有数据，则函数将通过 BufRxOut 指针从缓冲区中读取一个字节的数据，并对 BufRxOut 进行循环移位，此函数还将改变 NumOfRx 的值。

UART 总线从接收移位寄存器中接收到一个字节数据，当数据送入数据输入寄存器时产生一个 UART 接收中断，系统将运行中断服务程序 Rx_ISR_Handler()，该程序将调用 PutRxChar()函数，此函数首先通过比较 NumOfRx 和 COM_Rx_SIZE 检查 BufRx[]中是否存在空闲的空间，没有则函数返回，否则将把数据输入寄存器中的存入到接受缓冲区 BufRx[]中，并循环移位 BufRIn 指针使其指向新的空闲块，同时更新 NumOfRx 的值，此函数还将释放一个信号量，通知那些挂起的任务。那些因为缓冲区没有数据而挂起的任务就会进入等待状态。

在中断服务程序结束后，内核调度最高优先级任务运行（此时调用 ComGetChar()函数的任务可能不是最高优先级任务）。如果调用 ComGetChar()函数任务时最高优先级任务，那么将会如前面所描述的从缓冲区内读取一个字节数据，并对环形缓冲区的成员变量进行修改。

在串口驱动中还提供一个辅助接口 BufRxEmpty()，此函数返回一个布尔型的变量，如果接收缓冲区为空则返回 TRUE，否则返回 FALSE。

UART 串口驱动发送模型与接收模型很类似，如图 4.8 所示：

与接收模型类似，驱动程序为用户提供两个接口函数：ComPutChar()和 BufTxEmpty()。与接收模型不同的是，串口只有在发送字节结束后才能产生中断，因此第一次发送就不能够通过中断服务程序完成。在本系统中，我们将人为的触

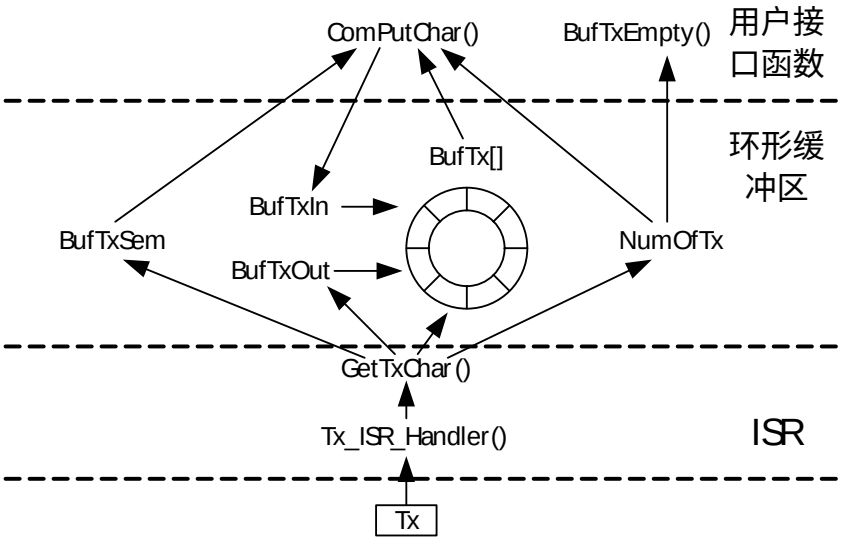

图 4.8 UART 串口驱动发送模型

发中断，然后由中断服务程序执行具体的发送操作。

在串口驱动程序中还提供了 2 个接口函数：ComInit()和 ComConfig()。ComInt()主要完成端口的初始化、缓冲区结构体 OS_COM_BUF 初始化以及信号量初始化等等。ComConfig()主要完成串口的配置，主要涉及波特率、奇偶校验位等等。

4.4　协议栈的开发

数据采集系统的网络是一个动态的网络，一个节点可能会因为某些原因退出网络运行，一个节点也可能由于工作的需要而加入到网络中，这些都会使网络的拓扑结构随时发生变化，因此网络应该具有动态拓扑组织功能。无线网络中节点通信距离有限，一般在几十到几百米范围内，节点只能与它的邻居直接通信。如果希望与其射频覆盖范围之外的节点进行通信，则需要通过中间节点进行路由，因此需要网络具有多跳路由。我们在基于新的嵌入式内核的的基础上，进行协议的详细设计。

在传统的网络系统模型中，一般采用 TCP/IP 模型。TCP/IP 模型一般分为 4 个层次：物理层和数据链路层、网络层、传输层以及应用层。本无线抄表系统实现与传统网络的融合，因此我们在 UCOS-II 的基础上实现 TCP/IP 传输模型，为了满

足下一代网络传输的需求，网络层还实现了对 IPv6 的兼容（W.Richard Stevens，2000；Jeremy Bentham，2002）。

4.4.1　协议层次图

本无线抄表系统的协议栈主要分为 5 层：物理层、数据链路层、网络层、传输层以及应用层，如图 4.9 所示。

图 4.9 协议栈层次机构

物理层主要是对无线通信相关的物理模块的抽象，它实现稳定可靠的无线收发，并为上层提供接口函数，方便上层的调用；数据链路层主要实现有效实时的无线收发，对上层的报文进行封装，实现自动重复发以及退避机制；网络层提供唯一的 IP 地址兼容 IPv6，提供可靠的点对点的数据传输，实现动态的路由；本系统采用 UDP 报文实现端到端的传输，抄表系统中报文的长度比较短，实时性的要求比较高。系统 UDP 传输层协议提高系统的性能，而系统的可靠性通过数据链路层完成；应用层根据抄表系统的规约要求为用户的程序开发提供功能接口。

4.4.2　物理层

物理层是对具体硬件的抽象，本抄表系统物理层最主要的工作是完成数据的无线收发。我们采用 NRF905 射频模块收发数据，处理器通过 SPI 总线与射频模块通信。系统在 UCOS-II 的基础上对射频模块进行抽象，为 MAC 层提供硬件操作接口。

NRF905 模块存在多种工作模式，用户可以对其动态切换。处理器可以根据上层的需要修改射频模块的接收地址和发送地址，NRF905 模块还包含了载波侦听、地址匹配、自动校验等机制，这些都需要用户发出控制命令（Nordic VLSI ASA，

2005）。处理器和射频模块的通信通过 SPI 总线完成。为上层系统屏蔽通信细节，物理层系统主要提供下面几个接口函数:

void nrf905_Config(UINT16 Channel_Bytes)

//完成对 NRF905 射频模块的设置，包括工作频率、工作模式、收发地址等。

UINT8 nrf905_SpiRead()

//从 nrf905 模块中读出一个字节的数据，存放在返回值中。

void nrf905_SpiWrite(UINT8 val)

//同过 SPI 总线将一个字节的数据 val 送到 nrf905 模块中。

void nrf905_SetRxAddr(DWORD RX_ADDR)

//设置 nrf905 射频模块的接收地址，可用 void nrf905_ReadRxA()读取接收地址。

void nrf905_SetTxAddr()

//设置 nrf905 射频模块的发送地址，可用 void nrf905_ReadTxA()读取发送地址。

void nrf905_SetData()

//处理器将 TxBuf[32]中的数据发送到射频模块，nrf905 缓冲区最多存储 32 字节的数据。

void nrf905_SendData()

//处理器发送控制命令给射频模块，使得射频模块将缓冲区的数据发送出去。

除了这些接口函数，还有其它一些射频控制接口，比如射频模块发送/接收模式转换函数，中断开启/屏蔽函数还有一些测试函数。

物理层还提供了其他一些总线抽象函数，比如 eeprom 与处理器直接通信的 IIC 总线，eeprom 中存储了节点的配置信息以及各个节点路由表信息，

void EEPROM_Write_Byte(unsigned int eep_addr, unsigned char eep_date)

//这是一个向 eeprom 写字节函数。函数首先检查写操作是否结束，并且关闭全局中断。

unsigned char EEPROM_Read_Byte(unsigned int eep_addr)

//这是一个向 eeprom 读字节函数。函数首先检查写操作是否结束，并且关闭全局中断。

Flash 存储模块与处理器之间的通信主要通过上节中提供的 uart 总线驱动，此外系统还提供一些时钟中断接口，方便用户调用。

4.4.3　数据链路层 MAC

MAC 层采用自主设计的数据链路协议，在 CSMA/CA 的基础上，实现 BC、RTS、CTS、ACK、DATA 帧，实现了自动重发机制。MAC 层最大数据载荷数据为 270 字节，系统自动分片，射频模块每次最多发送 32 字节的数据，系统将数据帧分成 9 片，每片包含 2 字节的控制信息以及 30 字节的数据信息。接收端接收到数据后，系统将自动合并数据并返回给上层调用。MAC 还实现了退避策略以及数据的重发机制。

MAC 协议中包含了多种帧格式：包括广播帧 BC、请求发送帧 RTS、允许发送帧 CTS、应答帧 ACK 以及数据帧 DATA 等。他们的帧结构如下：

RTS 帧结构：

```
typedef struct {
    UINT8    FrameType;          //MAC 帧的类型，RTS 帧为 0x01
    UINT8    Num;                //保留字，赋值为 0
    DWORD DesAddr;               //目的节点 MAC 地址
    DWORD SourAddr;              //源节点 MAC 地址
    UINT16     TxTimeNeeded;     //发送数据帧所需要的时间
    UINT8    PartNum;            //需要传输的数据块的分片数
} ZMAC_RTS;                      //RTS 帧
```

CTS 帧结构：

```
typedef struct {
    UINT8    FrameType;          //MAC 帧的类型，CTS 帧为 0x02
    UINT8    Num;                //保留字，赋值为 0
    DWORD DesAddr;               //目的节点 MAC 地址
    DWORD SourAddr;              //源节点 MAC 地址
    UINT16     TxTimeNeeded;     //发送数据帧所需要的时间
```

```
    UINT8    PartNum;                //应答需要传输的数据块的分片数
} ZMAC_CTS;                          //CTS 帧
```

RTS 帧和 CTS 帧结构基本相似，Num 为保留字，系统赋值为 0x00，PartNum 为发送端需要发送的数据帧的分片数，DesAddr 和 SourAddr 为发送节点及接收结点的 MAC 地址，FrameType 为 MAC 帧的类型，系统规定 RTS 帧为 0x01，CTS 帧为 0x02，ACK 帧为 0x03，BC 帧为 0x04，DATA 帧为 0x05。系统可以很方便的通过 MAC 帧的帧类型对不同的帧进行操作。

ACK 帧结构：

```
typedef struct {
    UINT8    FrameType;          //MAC 帧的类型，ACK 帧为 0x03
    UINT8    Num;                //保留字，赋值 0x00
    DWORD DesAddr;               //目的节点 MAC 地址
    DWORD SourAddr;              //源节点 MAC 地址
    UINT8    PartNum;            //数据块的分片数
    BOOL     PartRXed[9];        //ACK 帧中记录的每个分片的接收情况
} ZMAC_ACK;                      //ACK 帧
```

ACK 帧包含一个 BOOL 类型的数组 PartRXed[9]，系统规定每一帧最多被分成 9 片，每片为 32 个字节。ACK 帧中用 PartRXed[]来记录对分片的接收情况，分片成功接收该元素赋值为 True，否则为 False。发送端通过分析 ACK 报文，得到数据帧的发送情况，对未成功发送的分片进行重发。

BC 广播帧：

```
typedef struct {
    UINT8    FrameType;          //MAC 帧的类型，BC 帧为 0x04
    UINT8    Num;                //整个 BC 帧分成的片数
    DWORD SourAddr;              //目的节点 MAC 地址
    UINT8    PartNum;            //此块数据所处的片数
    UINT16   FrameLength;        //BC 帧的总长度
```

} ZMAC_BC;　　//BC 广播帧

BC 广播帧一般由集中器发送应用层的控制信息，也可能是有采集器发送来更新路由信息。BC 帧后面还包含着一些控制信息以及路由，因此还有一部分在载荷，如果超过 32 字节，系统将对 BC 帧进行分片发送，FrameLength 就是 BC 帧的总长度，Num 就是 BC 帧分的片数。PartNum 指此分片是整个 BC 帧的第几个片。

DATA 数据帧：

```
typedef struct {
    UINT8    FrameType;        //MAC 帧的类型，DATA 帧为 0x05
    UINT8    Num;              //此块数据所处的片数
} ZMAC_DATA;                   //数据帧
```

数据帧分片前两个字节为 MAC 头，其余的 30 字节用来保存载荷数据，一个数据帧最多分成 9 片，因此一个数据帧最大的数据量为 270 字节。数据帧结构体重第一项 FrameType 用来表示帧类型，第二项为数据分片所处的片数，此值最大值为 RTS 帧中 PartNum-1。MAC 层主要涉及的函数如下：

```
void ZMAC_ReceiveInterrupt(void);
```

//MAC 层接收到 MAC 帧后的处理函数，此函数根据 MAC 帧的 FrameType，对 MAC 进行分别处理。

```
void ZMAC_FrameConstruct(UINT8 Type, DWORD DAddr, UINT16 Length);
```

//此函数建立 MAC 帧头信息，MAC 数据载荷在 txDataBuffer[270]中

```
void ZMAC_FrameSend(UINT8 Type,UINT8 *pData);
```

//发送 MAC 帧，此函数将先调用 ZMAC_FrameConstruct()函数，然后再发送 MAC 帧。

```
BOOL ZMAC_PacketSend(UINT8 Type, DWORD DAddr, UINT8 *pData, UINT16 Length);
```

//此函数将作为接口被上层调用，函数将上层 pData 数据构成 MAC 帧，然后调用 ZMAC_FrameSend 函数发送出去。

物理层和 MAC 层协调工作，完成对数据的可靠的传输。数据的收发流程图如图 4.10、图 4.11 所示：

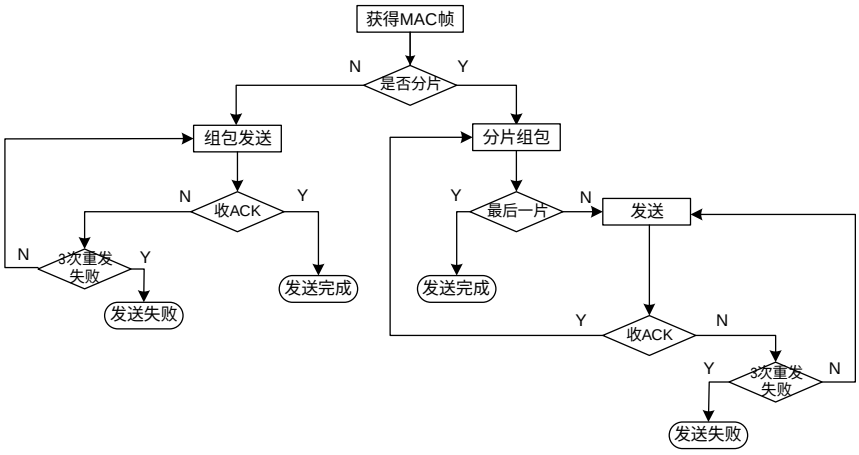

图 4.10 MAC 帧发送流程图

在 MAC 发送过程中，系统先会对 MAC 进行分片，发送端在接收到目的节点的 ACK 后会根据 ACK 帧的信息进行选择重发，如果连续 3 次都发送失败(可能是收到 3 次有问题的 ACK 帧，也可能在指定时间内没有收到应答)，则表示 MAC 帧发送失败。

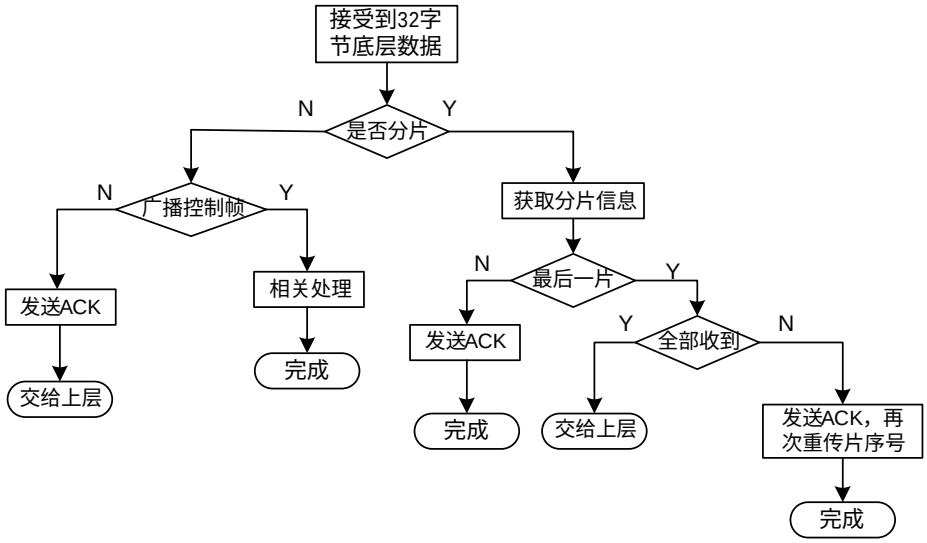

图 4.11 MAC 帧接收流程图

接收端在收到正确的 MAC 帧后，会根据 RTS 帧中的分片信息，对 DATA 帧进行组合，然后返回给上层使用。

4.4.4　网络层

在本无线抄表系统中，网络层主要包含 3 个协议:ARP/RARP 协议、IP 协议以及 ICMP 协议。为了与下一代互联网相兼容，用户可以根据具体的需要选择不同的 IP 报文头以及 ICMP 报文头结构，本系统通过不同的宏定义来选择不同的报文

头。在一般的内部网络传输过程中，系统采用裁剪的 IP 报文头，裁剪后的 IP 报文头体积小，资源的利用率比较高，此时系统采用 32bit 的 IP 地址。内部网络中的节点也可以直接与外部 Internet 通信，此时节点采用 IPv6 报文头，节点的 IP 地址为 128bit 的 IPv6 地址。

IP 报文头定义如下：

```
#ifndef IPV6     // 精简的报文
// IP 报文头部类型  14 bytes
typedef struct {
    ip_addr_t src_ip;              // 源地址
    ip_addr_t dst_ip;              // 目标地址
    UINT16        checksum;        // 整个 IP 报文的校验
    UINT16    pload_len;           // 上层报文长度，不包括 IP 扩展头部的长度
    UINT8     next_header;         // 上层报文类型
    UINT8     hop_lim;             // 跳数
} IP_HDR;
#else   // 完整的 IPv6 报文
// IPv6 报文头部类型  40 bytes
typedef struct {
    UINT32    ver:4;                 // 版本以
    UINT32    traffic_class:8;       // 优先级
    UINT32    flow_label:20;         // 流标记
    UINT16    pload_len;             // 上层报文长度，不包括 IP 扩展头部的长度
    UINT8     next_header;           // 上层报文类型
    UINT8     hop_lim;               // 跳数
    ip_addr_t src_ip;                // 源地址
    ip_addr_t dst_ip;                // 目标地址
} IP_HDR;
```

#endif

在 IP 报文头中 ip_addr_t 类型也是根据宏定义改变，在精简报文投中被定义成 32 比特，在完整的报文头中被定义成 128 比特。在精简报文头中的 checksum 变量是整个 IP 报文的校验，系统采用 CheckSum()函数实现。此函数的输入是整个 IP 报文以及报文的长度，输出为 16 个比特的校验数据。

系统中采用 32 比特的 MAC 地址,精简报文的 IP 地址为 32 位,它与节点的 MAC 地址相同，完整报文中系统的 IP 地址为 128 位，128bit 地址构成如下图所示：

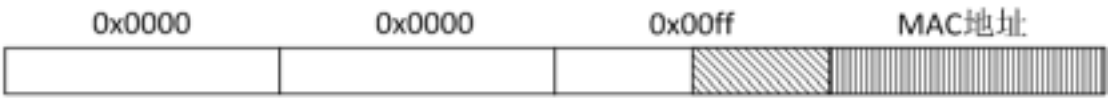

图 4.12 IPv6 地址构成

完整的 IP 报文的 IP 地址 0-31 位为节点的 MAC 地址，32-47 为全 1，其余为 0。系统使用接口函数实现 ARP/RARP 协议。

ip_addr_t　MAC2IP(mac_addr_t mac)

// 将 MAC 地址转换为 IP 地址，系统会根据 IPV6 宏来决定返回 32bit 精简 IP 地址还是 128bit 的 IPv6 地址。

mac_addr_t IP2MAC(ip_addr_t ip)

//将 IP 地址转换为 MAC 地址

另外系统还提供了两个用户比较 IP 地址以及 MAC 地址的辅助函数。

BOOL EqualMAC(mac_addr_t m1, mac_addr_t m2)

//m1 和 m2 为相同的 MAC 地址返回 True，否则返回 False。

BOOL EqualIP(ip_addr_t ip1, ip_addr_t ip2)

//p1 和 p2 为相同的 IP 地址返回 True，否则返回 False。

对与 IP 协议系统也提供了相关的接口函数，主要是 IP 报文的发送和接收。在接收处理函数中，函数会根据内部的报文类型返回给不同的协议进行处理，在发送过程中，系统提供了 2 个队列：IP 控制报文发送队列和 IP 数据报文发送队列。这两个队列用 IP 发送报文任务队列进行管理。

typedef struct{

#define TX_DATA_ITEM_CNT (1)

```
#define TX_CMD_ITEM_CNT        ((1000-sizeof(IP_TXPKT_DATA)*
                                TX_DATA_ITEM_CNT)/sizeof(IP_TXPKT_CMD))
    IP_TXPKT_CMD tx_cmd_item[TX_CMD_ITEM_CNT]; //  发送队列中至多同时存放
20 个报文
    IP_TXPKT_DATA tx_data_item[TX_DATA_ITEM_CNT]; //  发送队列中至多同时存
放 1 个报文
} IP_TXPKT_Q;          //IP 发送报文任务队列
```

系统定义 IP 数据发送报文队列的长度为 1，对于同时多个数据任务，系统采用在 UCOS-II 的基础上构建多个同优先级的数据发送任务来完成的，通过 UCOS-II 的系统调度，使得整个过程有比较好的服务质量。系统定义 IP 控制报文队列最多存在 20 个 IP 控制报文。IP 控制报文比数据报文体积上要小很多，他主要是一些 ICMP 报文以及提交给上层的拓扑信息。本系统中对 ICMP 报文进行了简化，它封装在 IP 报文中。系统提供的 IP 协议接口函数如下所示：

```
BOOL IP_RecvPacket(UINT8* ip_pkt, UINT16 length);
```

//接收 IP 报文，并对 IP 报文进行处理，函数将根据载荷中封装的报文类型信息，返回给不同的操作函数。载荷的报文类型分为 ICMP 报文和 UDP 报文。

```
BOOL IP_SendPacket(ip_addr_t* src_ip, ip_addr_t* dst_ip,
                   UINT8 nextheader, UINT8 hop_lim,
                   UINT8* payload, UINT16 pload_length,
                   UINT8 count_down,
                   UINT8 max_resend_cnt);
```

//IP 报文发送函数，对发送队列缓存数组 txDataBuffer[]进行赋值，然后将根据报文的将其分类，然后将不同的报文压入不同的发送队列。

```
BOOL IP_RfTx(UINT8* ip_pkt, UINT16 length)
```

//分别从两个 IP 报文发送队列中取出相应的报文，调用 MAC 的发送函数 ZMAC_PacketSend()进行数据的报文的发送。

实现动态路由功能是网络层的重要职责之一，本系统通过 ICMP 报文来实现节点的路由信息更新。无线抄表系统中每个节点都会将路由表存放在 eeprom 中，

系统在发送 IP 帧时，将通过 IIC 总线读取 eeprom 的路由信息进行信息的发送。系统的路由信息的建立以及拓扑结构的获得都是通过 ICMP 实现的。

每个节点都会有一个结构体保存自己的拓扑信息，此信息系统系统启动后就会加载到内存中。

```
typedef struct {
    IP_TXPKT_Q iptx_q;            // ip 报文发送的任务队列
    mac_addr_t    mac;            //  本节点的 mac 地址
    ip_addr_t    ip;              //  本节点的 ip 地址
    ip_addr_t    parent_ip;       //  节点的上一跳节点的 ip 地址
    ip_addr_t    bs_ip;           //  基站的 IP 地址
    UINT8    bs_hop;              //  距离基站的跳数
    UINT16    valid_life;         //  节点的有效生命期，指本节点是否能和基站进行通信
    UINT16    re_valid_life[MAX_NODES_CNT];    //  路由信息的条目的生命期
    ip_addr_t    invalid_bs_lst[MAX_INVALID_BS_CNT];    //  该列表中记录哪些基站不可以加入
    int    invalid_bs_lst_life[MAX_INVALID_BS_CNT];//  该列表的有效期
} LOCAL_CACHE;     //  节点在本地保存的信息
```

此结构保存在本节点的相关的拓扑信息，iptx_q 是 IP 报文发送队列，bs_ip 是基站的地址。bs_hop 是距离基站的跳数，parent_ip 是节点的上一跳节点的 ip 地址，在更新路由表是 bs_hop 以及 parent_ip 都有可能发生更新，在系统建立拓扑的时候，节点会将自己的 LOCAL_CACHE 信息发送到基站(集中器)。invalid_bs_lst[]中保存着不可加入基站的信息，节点在更新路由信息时，将根据此信息进行过滤。

ICMP 报文头结构也分为两种：

```
typedef struct {
#ifdef IPV6
    UINT8 type;                    //报文类型，ICMP 报文中存在 3 种类型
```

```
    UINT8 code;
    UINT16 checksum;            //整个 ICMP 报文的校验
#else
    UINT8    type;              //报文类型，ICMP 报文中存在 3 种类型
#endif
    UINT16        length;    //  整个 ICMP_PKT 报文的长度
}ICMP_HDR;                     //ICMP 报文头
```

本无线抄表系统中定义了 3 中 ICMP 报文类型，分别是 ICMP_ROUTER_ADV、ICMP_NI_QRY 以及 ICMP_NI_RSP。ICMP_ROUTER_ADV 报文时路由广播报文，它将发送 BC 帧广播自己的路由信息。ICMP_NI_QRY 报文是节点信息请求报文，此报文向指定的节点请求其路由信息。ICMP_NI_RSP 报文时节点信息应答报文，它一般作为 ICMP_NI_QRY 的应答报文。在 ICMP 报文的载荷中包含了节点距离基站的跳数 bs_hop、基站的 IP 地址 bs_ip、以及同步的握手时间等等。

系统提供了 ICMP 报文的收发处理函数，系统定时调用这些函数来更新自己的路由信息。

```
BOOL ICMP_SendPacket(ip_addr_t* dst_ip,
                     UINT8 hop_lim,
                     UINT8 type, ICMP_PLOAD* pload,
                     UINT16 length);
```

//ICMP 报文发送接口函数，接口调用者指定 ICMP 报文的目的地址以及跳段数限制 hop_lim。系统实现 ICMP 报文时通过对 IP 报文的封装，因此此函数将会调用 IP 报文发送函数 IP_SendPacket()。

```
BOOL ICMP_RecvPacket(ip_addr_t* src_ip, ICMP_PKT* icmp_pkt, UINT16 length);
```

//ICMP 报文接收函数，此函数将对不同类型的 ICMP 报文进行不同的处理

对于 ICMP_NI_QRY 类型的报文，节点会回复一个 ICMP_NI_RSP 报文，如果接收到的是一个 ICMP_NI_RSP 报文，节点会根据需要修改自己的本地信息。如果接收到一个 ICMP_ROUTER_ADV 报文，本节点会根据 bs_hop 信息决定是否做处理，如果

本节点是更加好的路由节点，节点回复 ICMP_NI_QRY 给源节点，通知源节点更新路由信息。

系统还提供了用于节点添加以及路由维护的处理函数：

void ICMP_OnTimer()

//此函数会定期的发送 ICMP_ROUTER_ADV 报文以及 ICMP_NI_QRY 报文，这些报文会为新的节点添加路由信息，并对整个网络的路由信息进行维护。

4.4.5　传输层

本系统传输层采用用户数据包协议 UDP(User Datagram Protocol)提供端到端的传输服务，UDP 报文资源占用比较小，传输时效比较高。系统中提供了 2 中端口应用 UDP_PORT_RPRTTOPO 以及 UDP_PORT_APP。端口 UDP_PORT_RPRTTOPO 用于上传拓扑结构，UDP_PORT_APP 用于传递应用层的报文。传输层在收发报文时，现在从本地 eeprom 中查找路由信息，然后讲接收端地址以及报文传给 IP 层，由 IP 层将报文发送出去。

路由表项如下所示：

```
typedef struct {
        ip_addr_t    dst_ip;       //  目的节点 IP
        ip_addr_t    next_hop;    //  下一跳的 IP
} ROUTE_ENTRY;                          //路由表项
```

每个路由表项中包含两个变量，一个用来存储目的节点的 IP 地址，另外一个用来存储路由过程中下一跳节点的 IP 地址。

UDP 报文头比较简短，UDP 报文头结构如下所示：

```
typedef struct {
    UINT16 src_port;                    //发送端端口号
    UINT16 dst_port;                    //接收端端口号
    UINT16 checksum;                    //整个 UDP 报文的校验
    UINT16 length;       //  整个 UDP 报文的长度
}UDP_HDR;                   //UDP 报文头
```

系统 UDP 报文主要有三个作用：第一是为上层应用层提供接口函数以及为 IP 层提供路由信息，第二个是上传整个网络的拓扑，第三个作用就是用于通知非法节点。UDP 层提供的接口函数如下：

```
BOOL FindRouteEntry(ip_addr_t* p_dst_ip,
                    UINT16 * p_index,
                    ip_addr_t* p_next_hop);
```

//函数根据 p_dst_ip 在 eeprom 中查找路由项，如果查找成功函数返回 Ture，p_index 保存它在 eeprom 中的位置，p_next_hop 中保持下一跳的 IP 地址。查找失败返回 False。

系统调用 FindRouteEntry()函数后得到了路由表项在 eeprom 中的索引，系统可以直接可以根据函数返回的 p_next_hop 进行数据的发送，也可以根据 p_index 索引调用 eeprom 物理层驱动提供的接口函数，对路由表项进行修改。

```
static void UART_RPRT_TOPO(TOPO_ENTRY* tp, UINT16 valid_life);
//此函数由应用层调用，可以向上层提供拓扑信息，方便开发者调试
void Rprt_Topo_OnTimer();
//定时的向基站提供节点的拓扑信息。此函数定期向基站发送 UDP 报文
BOOL UDP_SendPacket(ip_addr_t* dst_ip,
                    UINT8    hop_lim,
                    UINT16    dst_port,
                    UINT8*    pload,
                    UINT16    length);
```

//UDP 报文发送函数，对上层的数据进行封装，然后调用 IP 层的 IP 报文发送函数 UDP_SendPacket()。

```
BOOL UDP_RecvPacket(ip_addr_t* src_ip, UDP_PKT* p_udp_pkt, UINT16 length);
```

//UDP 报文接收处理函数，函数将根据端口号做不同的处理，如果端口为 UDP_PORT_APP，则提交到 APP 层；若端口为 UDP_PORT_RPRTTOPO，则通过 UART 驱动接口向用户提供拓扑信；若端口为 UDP_PORT_DENY，则修改 CACHRE_LOCAL 信息将此基站添加到 invalid_bs_lst[]中。

4.4.6　应用层

应用层主要是根据国家规约的具体需要，为具体的用户开发提供功能接口。应用层报文头如下所示：

```
typedef struct {
        UINT8    type;                  //应用层报文类型
        UINT16 pload_len;
} APP_HDR;                              //应用层报文头
```

系统根据具体的需求将 app 报文分为多种类型，主要包括添加设备、删除设备、获取某个时刻数据、获取当时数据、改变射频工作频率、设置节点时钟等等。App 应用层为用户提供接口函数如下：

```
static BOOL APP_SendPacket(ip_addr_t* dst_ip,
                           UINT8 hop_lim,
                           UINT8 type,
                           UINT8 *pload,
                           UINT16 pload_len);
```

//用户根据需要填写相应的报文类型，函数将用户提供的数据进行封装，然后调用传输层的报文发送函数 UDP_SendPacket()放送报文。

```
BOOL APP_RecvPacket(ip_addr_t* src_ip, APP_PKT* pkt, UINT16 length)
```

//应用层报文接收函数，系统分析报文的类型，调用不同的底层处理函数，报文得到成功处理函数返回 True，否则返回 False。

```
void Get_Device_Actual_Num();
```

//得到节点下挂只能电表的数量。此函数将在 APP_RecvPacket()调用。

```
void Set_Channel_OnTimer();
```

//定时改变射频频率，变频操作，改变底层的传输信道。

```
void Get_Certain_Record();
```

//得到固定时刻抄表数据，此函数也将被 APP_RecvPacket()函数调用。

系统在应用层为用户屏蔽了底层的通信细节，为用户提供了简单透明的接口函数，用户非常方便的在接口的基础上进行二次开发，上层只需要调用函数就完成了整个抄表系统。

4.4.7　多基站节点加入过程

在无线抄表系统中存在多个基站，这对采集器节点的加入以及路由过程带来了问题。新加入的采集器可能处于多个基站的通信范围之内，在布置新节点前，上位机将发送命令给其中的某个基站，此基站将把此集中器的 IP 信息添加到自己的采集器管理列表中，新的采集器启动后发送请求报文，附近的基站对其进行响应。采集器节点加入过程流程图如下：

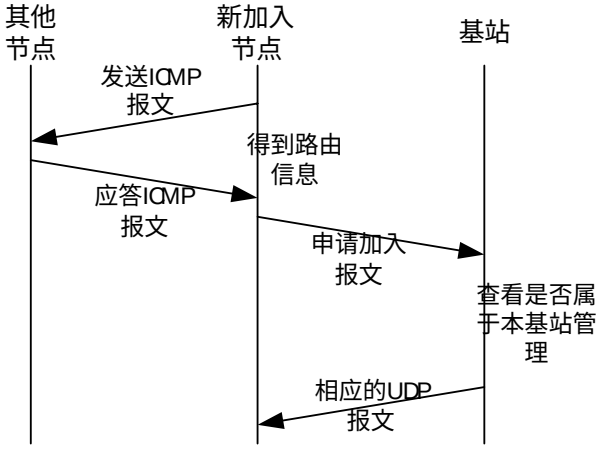

图 4.13　节点加入简单流程

新节点加入后，会发送广播报文 ICMP 请求报文，通过 3 次握手以后，其他节点会发送应答报文，应答报文中包含路由信息。节点通过分析应答信息，得到相应的基站和路由信息，新增加节点会分别通过路由向所有可达的基站发送申请加入信息。基站接收到 UDP 报文后，基站查看本地的采集器管理列表，一般调用 IsValidCollector()函数，如果此 IP 不在管理列表中，该基站根据路由信息回复端口号为 UDP_PORT_DENY 的 UDP 报文，申请加入节点根据收到的 UDP 报文将此基站信息保存在 LOCAL_CACHE 的 invalid_bs_lst[]变量中，以后的路由更新中，如果基站在 invalid_bs_lst[]中的路由信息将被过滤。

在系统的定时更新路由中，节点会发送 ICMP_ROUTER_ADV 类型的 ICMP 报文，应答节点先和更新节点进行 3 此握手，然后将自己的路由信息发送给更新节点，

第 5 章　系统测试

本章在上文软硬件平台基础上，进行无线抄表系统通信测试，并对测试结果进行分析。整个测试过程分成三个部分：1）底层协议链路通信：主要测试各种情况下底层报文的收发可靠性；2）上层协议功能性测试：本测试过程中将在小区安装节点，通过报文的收发得到整个小区的拓扑信息；3）内核性能比较：在同样的场景下，将原 UCOS-II 以及新的内核进行性能比较。

5.1　底层数据通信测试

本文的第四章详细描述了系统各层协议设计，为了保证数据的的可靠性，系统采用了多种机制：ICMP 报文的三次握手、IP 报文的数据校验、数据链路层采用的 CSMA/CA 机制以及自动重传机制等。底层数据通信的可靠是整个系统稳定高效的基础。本章我们将模拟底层各种通信环境，对底层通信的可靠性进行测试。

测试节点拓扑图如下所示：

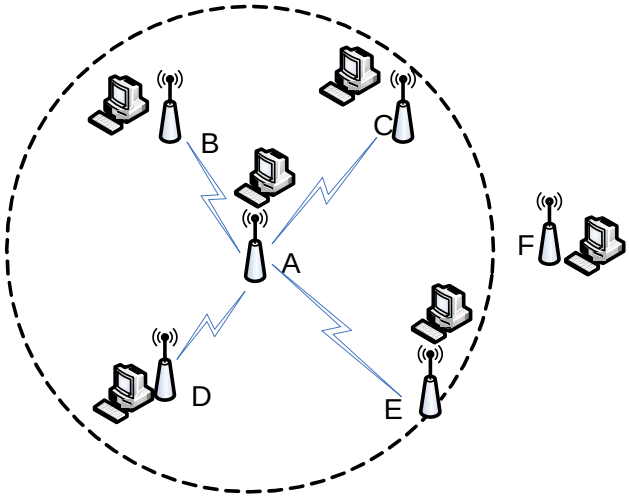

图 5.1 底层通信测试节点拓扑图

如图 5.1 所示，测试系统中存在 6 个采集节点，我们将从以下 4 个方面对底层通信进行测试：1）点对点的正常通信：A 节点发送分别给 C 和 D 节点发送 270 字节的数据，数据分成 9 片，多次收发测试数据收发成功率；2）广播分片：A 节点向周围的节点发送广播帧；3）CSMA/CA 机制测试：A 节点向 C 节点发送报文，同时设置 B 节点定时的用同信道向 D 节点发送数据；4）重传机制：节点

向 E 节点发送数据报文，设置 E 点收到报文后首先回复某分片发送失败，然后恢复 E 节点，使其正常工作。每个节点使用 UART 总线将收发情况传给上位机。

通过 24 小时的循环测试，测试结果如下：

表 5.1 底层数据测试结果表

序号	功能名称	操作方法	实验次数	成功率	结果	备注
1	点对点通信分片	某个节点发送 270 字节报文给另一个节点，要分成九片发送	12467	99.98%	PASS	
2	广播通信分片	某个节点广播 270 字节报文到周围节点，要分成九片发送	20567	99.99%	PASS	
3	CSMA/CA 机制测试	使信道处于占用状态，让节点发送数据	5908	99.83%	PASS	
4	重传机制	使接收节点丢片，让发送节点重传	9739	99.91%	PASS	

如上表所示，经过多次的重复传输，系统的底层数据传输非常可靠，完全满足国家 90%的标准。

5.2　上层协议的功能测试

在上层协议功能测试中，我们在实际的小区中安装抄表节点，系统启动后上位机会根据管理节点上传的路由信息，形成相应的网络拓扑，我们将其与实际情况作比较，来验证其正确性。

小区实际分布图如图 5.2 所示，小区内共有 10 栋居民楼，小的居民楼采用一个采集器，大的居民楼用两个采集器。图中的 9 号楼采用 2 个采集器。BS 是集中器(基站)，它收集采集器的数据，通过 GPRS 模块发送到供电中心。

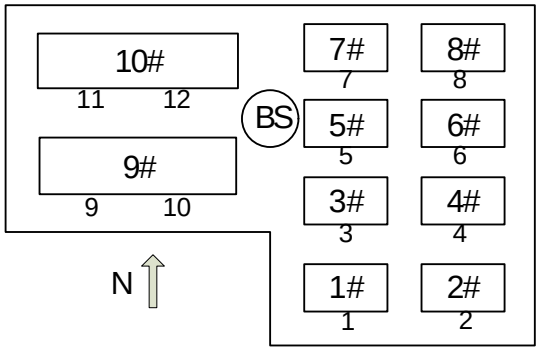

图 5.2 小区结构图

系统启动后，基站收集小区内采集器通过广播报文，获取自己的路由信息，完成组网，并把路由信息保存在本地存储器中。经过一段时间运行后，系统拓扑将趋于稳定，此时供电中心通过上位机管理软件查看小区的拓扑结构，上位机的得到的拓扑结构如下图所示：

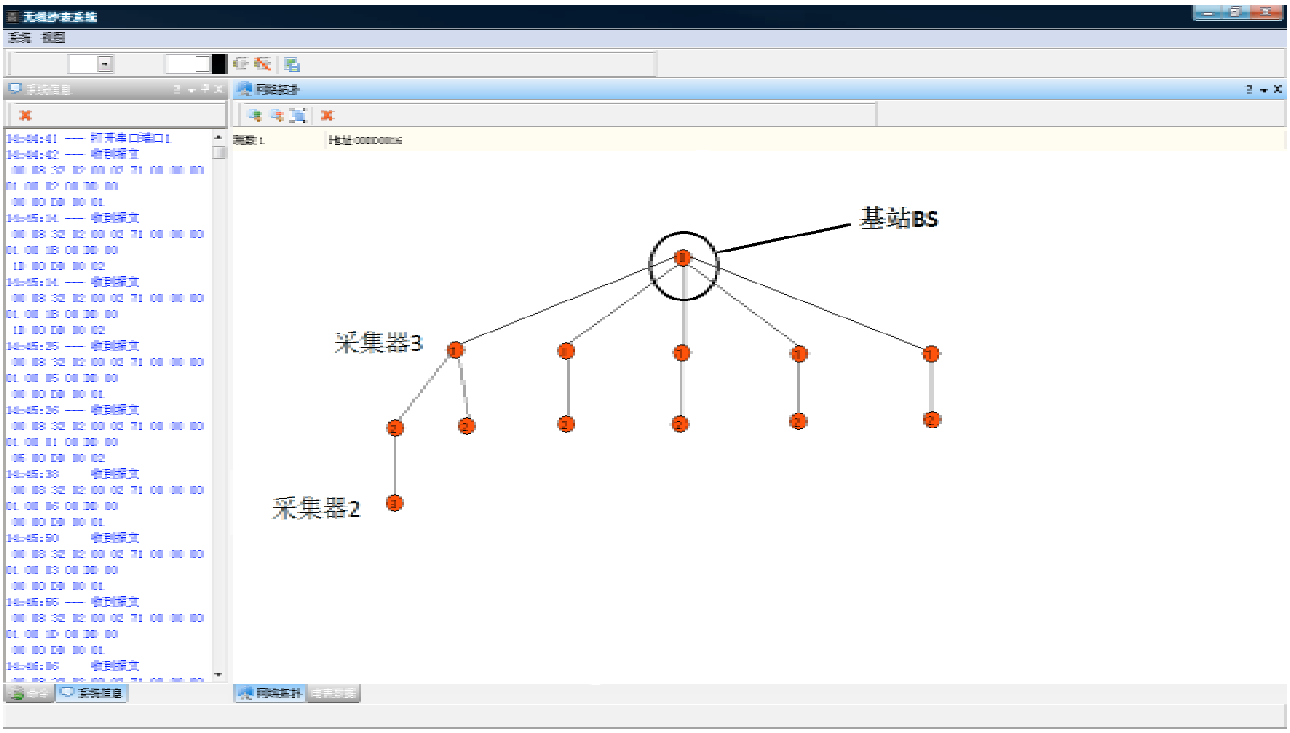

图 5.3 系统上位机得到的拓扑结构

图 5.3 是系统上位机中的树状拓扑图，分析可知，上图与小区实际情况相一致，下面我们将人为的改变小区的采集器的分布，这里我们移除编号为 1 的采集器，系统运行一段时间后，再次从上位机中得到系统的拓扑图，如图 5.4 所示

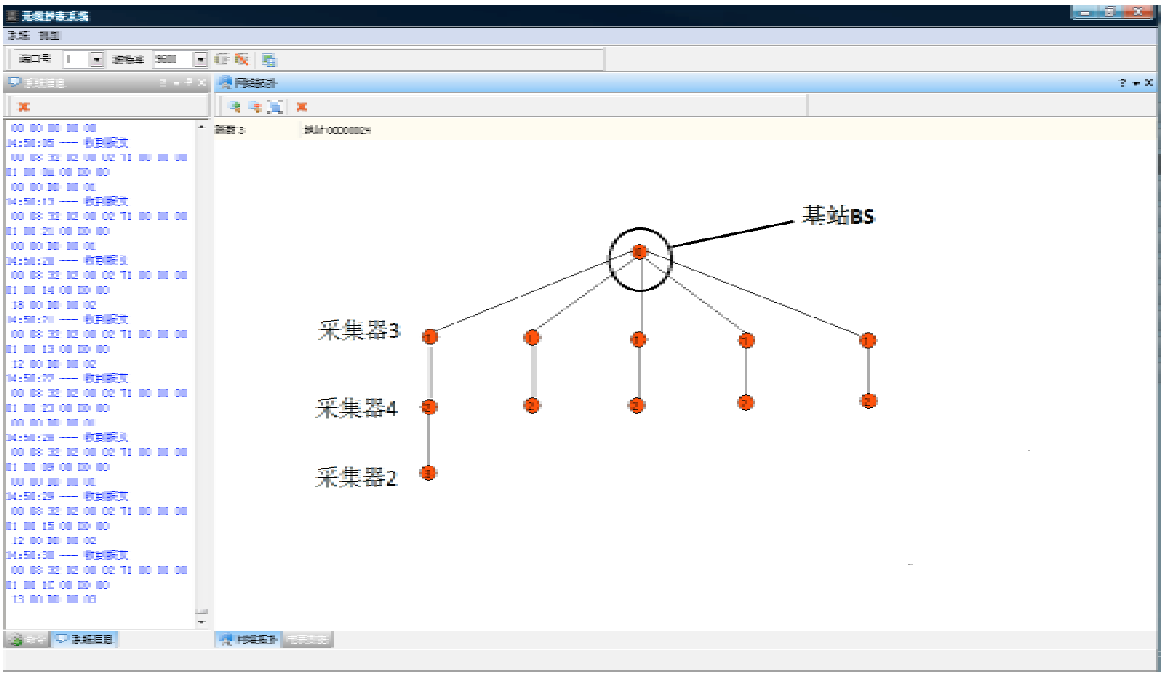

图 5.4 某个节点退出后系统拓扑

系统运行一段时候后，节点通过定时的路由维护，修改自己的路由信息，由于采集器 1 的被移除，采集器 2 收到采集器 4 的 ICMP 应答报文，修改自己的路由信息，基站得到这些路由信息，将拓扑情况发送给供电中心。

5.3　内核比较

实验中用到多个采集节点，其中一个采集节点向其余的节点发送数据分片报文，接收节点收到报文后发送 ACK 应答报文，并对接收到的数据进行处理，数据发送节点接收到 ACK 报文后发送下一片报文，直到数据发送结束。如图 5.1 所示，A 为发送节点，BCDE 为报文接收结点。我们分别用两种拥有不同内核的系统实现该过程，并在接收节点上创建统计时间任务，用于记录每个接收节点从发送数据请求到完成报文处理的时间。试验中，我们逐步改变接收节点的数量，试验结果如下表所示：

第 6 章　总结及展望

6.1　本文的研究内容和成果

嵌入式系统无疑是当前最热门最具发展前途的 IT 应用领域之一。随着信息化、智能化、网络化的高速发展，嵌入式系统在人们的生产生活中得到越来越广泛的应用。近年来网络通信技术与嵌入式技术联系越来越紧密，网络通信环境对嵌入式技术提出了新的挑战，嵌入式系统既要满足系统实时性的要求又要满足网络通信服务质量。本文对实时嵌入式操作系统进行分析，提出一种适用于网络通信环境的内核调度算法，并将新的内核应用于无线抄表系统，本文主要成果如下：

1)　在实时调度内核 UCOS-II 的基础上，对其内核调度算法进行改进，将基于优先级的调度算法与基于时间片的调度算法相结合，既满足网络通信要求又满足系统实时性的要求。

2)　改进 UCOS-II 的动态内存管理策略，提高了内存的利用效率，为用户提供了更加方便可靠的接口函数。

3)　在新内核的基础上进行了无线抄表系统的协议栈的设计，为抄表系统提供稳定、可靠、高效的链路连接，并为系统的二次开发屏蔽的底层细节，提高了开发效率。

6.2　进一步的工作

本文的重点是嵌入式实时操作系统的改进以及无线抄表系统协议栈的开发，在此基础上可以进行进一步的探索，主要有以下几个方面：

1)　在内核调度算法中，第二层调度采用时间片进行调度，同优先级下时间片较小的任务首先得到调度，然而为了减小调度时间，我们采用的是次优的方式将任务加入到就绪链表中。在不同优先级任务初始时间片的赋值也没有得到深入的研究。

参考文献

Bimbard, F.;，George, L.，EDF Feasibility Conditions with Kernel Overheads on an Event Driven OSEK System[C]，Third International Conference on Digital Object Identifier，2008， Page(s): 277 - 284.

Bruno Dutertre，Victoria Stavridou．Formal Analysis for Real-Time Scheduling[C]．Digital Avionics Systems Conferences，2000．Proceedings．DASC．The Philadelphia．PA．USA，2000，1

C．M．Krislma,King．GShin．Rea-Time Systems[J]．Taingima University Presa，McGraw-Hill，2001

de Pablo, D.A.L.，On scheduling models: An overview[C]，Computers & Industrial Engineering, 2009. CIE 2009. International Conference on 6-9 July 2009 Page(s):153 – 158

Guo Qing， Hu Jingquan，Hu Hongzhi，Design and Implementation of Testing Network for Power Line Fault Detection Based on nRF905[C]，Electronic Measurement and Instruments，2007. ICEMI '07. 8th International Conference on Digital Object Identifier，2007，Page(s): 3-513 - 3-517

http://www.micrium.com/

Jean J.Labrosse ， MicroC/OS-II The Real-Time Kernel Second Edition[M]，CMP Books 6,2002

Jeffay, K.，Donelson Smith, F.， Moorthy, A.， Anderson, J.，Proportional share scheduling of operating system services for real-time applications[C]，The 19th IEEE ，1998，Page(s): 480 - 491.

Jeremy Bentham，TCP/IP for Embedded System[J]，Real-Time Operating System for Embedded Computing，2002，12:26-29.

Liu Yong-Min，Jiang Xin-Hua，A Protocol Model for Wireless Sensor Network[C]，Wireless Communications and Trusted Computing，2009. NSWCTC '09. International Conference on Volume: 2，Page(s): 588 – 591.

Nordic VLSI ASA，nRF905 Product Specification[Z]，Norway：Nordic VLSI ASA，2005.

Qing-Hua WANG, Yu-Gui QU,An Energy-efficiency Route Protocol for MIMO-based Wireless Sensor Networks[C], The 11th Asia-Pacific Network Operations and Management Symposium (APNOMS)，2008.10，pp.449-453

Raskovic, D.， Revuri, V.， Giessel, D.;，Milenkovic, A.，Embedded web server for wireless sensor networks[C] ，System Theory，2009. SSST 2009. 41st Southeastern Symposium on Digital Object Identifier 2009 ，Page(s): 19 – 23.

Robert Filman，Embedded Internet System Come Home[J]， IEEE Internet Computing，2001, 5:52-53.

William S，Operating system internals and design princinples[M]，电子工业出版社，2001.

W.Richard Stevens，The TCP/IP of Embedded System[C]，IEEE Transaction on Embedded System，2000，12:749-751.

ZHENG Jie, GUO Shu-jie, QU Yu-gui, ZHAO Bao-hua，Energy Equalizing Routing for Fast Data Gathering in Wireless Sensor Networks[J],The Journal of China Universities of Posts and Telecommunications, 14(4):13-21,2007.

Zhi-Ting Lin,Yu-gui Qu,Xiao-Fang Zhou,Bao-hua Zhao,Analysis and design of mobile Wireless Social Model[J],Computer Communications 32(2009)927-934.

柏荣刚，赵保华 ，屈玉贵，一种节能的无线传感器网络跨层结构[J]，中国科学技术大学学报，2009 年 8 月 39 期.

陈是知，μC/OS-II 内核分析、移植与驱动程序开发[M]，人民邮电出版社，2007.

陈文智，谢铖，石教英，一个构件化嵌入式操作系统的精确控制内核[J]，计算机学报，2006 年 6 期.

封斌，龚灼，扬学军，实时操作系统保护模式下的内存管理策略[J]，华中科技大学学报，2002，30(3)：94-96.

高富强，秦昌硕，游纪原，邹恒，UC/OS-II 内核扩充时间片轮转调度算法的设计[J]，计算机应用，2009 年 4 期.

胡希明，毛德操，Linux 内核源代码情景分析(上、下)[M]，浙江大学出版社，200l.

李小群，赵慧斌，叶以民，孙玉芳，RFRTOS:基于 Linux 的实时操作系统[J]，软件学报，2003 年 7 期

刘兰香，张秋生，ATmega128 单片机应用与开发实例[M]，北京-机械工业出版社，2006.

齐秀波，电能计量集中抄表技术的发展[J]，电测与仪表，2006，43(488):48~50

宋宝华，Linux 设备驱动开发详解[M]，人民邮电出版社，2008.

文旭，陈兵，μC/OS-II 高速网络通讯中 NAPI 的设计与实现[J]，小型微型计算机系统，2008 年 2 期

徐伟，短距离无线通信与 GPRS 通信相结合的小区抄表系统设计[J]，仪表技术，2009，23(3):37~41.

杨剑，基于 nRF905 的无线射频数据采集系统的开发与实现[D]，中南大学， 2007.

朱珍民， 隋雪青， 段斌编，嵌入式实时操作系统及其应用开发[M]，北京邮电大学出版社，2006.

攻读硕士学位期间的研究成果与科研项目

已发表的论文：

1．薛安琪，屈玉贵，赵保华，一种新的实时系统内核调度算法，计算机工程，2011.6

研究项目：

1. 国家 863 计划：无线局域网的安全管理和分析系统，2007-2009
2. 产学研项目：基于无线 IPv6 的数据采集系统，2008-2010

致　谢

三年时间，转瞬即逝。在此，我要感谢每一个帮助过我的人。

首先我要感谢我的导师屈玉贵教授。2007 夏天，非常幸运的进入了屈老师的实验室开始了我的研究生生活。屈老师在我的学习和生活方面都给予了悉心的指导以及无微不至的关怀。屈老师在科学研究经验丰富、治学严谨、而且具有不拘一格的创新精神；在生活中平易近人、乐观豁达，她的科研和人生态度都深深的影响到了我，在此我由衷的感谢屈老师对我多年的教导，愿屈老师合家欢乐，健康长寿。

赵保华教授是另一位指导我成长和进步的引路人，他对待科研严谨的态度以及深厚而渊博的专业知识深深的影响到我，感谢多年来赵老师对我的关心和培养。我还要感谢实验室的刘老师和实验室其他所有的老师，感谢他们给我的研究提供了非常优越实验环境和条件。

感谢蔺智挺、王庆华、郑杰、柏荣刚、李奇越、周晓芳、黎洁。在这三年的研究生生活中，这些师兄师姐给予我极大的帮助，在这里我要感谢他们，愿他们工作顺利，生活美满。还要感谢我的同学郑越明、李翠、巩浩、曹小虎、苏宇等，在这 3 年的科研生活中，我们建立了深厚的友谊，非常感谢你们这 3 年给我的帮助，感谢 3 年你们带给我的欢乐。感谢跟我一起在科大学习生活 7 年的同学马远鹏、王晓飞、金恬，谢谢！

最后要深深感谢我的父母支持，他们给了我无私的爱，使我得以顺利完成学业。祝愿他们健康、幸福！

2010 年 5 月