小 型 微 型 计 算 机 系 统
Journal of Chinese Computer Systems

2012 年 6 月 第 6 期
Vol. 33 No. 6 2012

一种符合 POSIX 规范的 FreeRTOS 模拟器的设计与实现

宋洪治[1,3],武 杰[2,3],张 杰[2,3]

[1](中国科学技术大学 物理系,合肥 230026)

[2](中国科学技术大学 近代物理系,合肥 230026)

[3](中国科学院 核探测技术与核电子学重点实验室,合肥 230026)

E-mail: dldz@mail.ustc.edu.cn

摘 要: 目前主流的网络仿真工具大都停留在对算法和网络协议的仿真. 虽然它们具有良好的扩展性,但是仿真结果和强烈依赖于实际应用的传感器网络仍存在一定的距离. 为了提高传感器网络仿真的完整性、逼真度和代码的可重用性,需要模拟实际运行的节点. 因此,本文设计并实现了可以和网络仿真器共同运行的 FreeRTOS 模拟器. 模拟器使用线程模拟任务,信号模拟硬件中断. 此方法也同样适用于其它轻量级多任务 OS 模拟器的实现.

关 键 词: 传感器网络; FreeRTOS; 模拟器

中图分类号: TP331 **文献标识码:** A **文 章 编 号:** 1000-1220(2012)06-1273-05

Design and Implementation of a POSIX Compliant FreeRTOS Simulator

SONG Hong-zhi[1,3], WU Jie[2,3], ZHANG Jie[2,3]

[1](*Department of Physics, University of Science and Technology of China, Hefei* 230026, *China*)

[2](*Department of Modern Physics, University of Science and Technology of China, Hefei* 230026, *China*)

[3](*Key Laboratory of Technologies of Particle Detection & Electronics, Chinese Academy of Sciences, Hefei* 230026, *China*)

Abstract: Most of the mainstream network simulators only simulate algorithms and network protocols. Though they are scalable, their simulations can not reflect real behaviors of sensor networks that depend on specific applications to some extent. To improve simulation completeness, fidelity and code reusability, real applications that run on network nodes should be simulated. So, a FreeRTOS simulator is designed and implemented. Tasks and hardware interrupts are simulated by threads and signals respectively. And the design method is also suitable for implementing other light weight OS simulators.

Key words: sensor network; FreeRTOS; simulator

1 引 言

传感器网络在近几年内发展迅速,被广泛应用于生产、生活、国防、科学研究等领域. 一般无线传感器网络包含几十到几千个节点[1]. 而用于陆上地震油气勘探的传感器网络,可以在 30km^2 的范围内包含 30000 个传感器,整个系统的数据传输速率达到 4.5Gbps[2],并且还要求数据以近实时的速率处理[3]. 因此,从实际应用的需求以及网络管理的角度考虑,传感节点必须具有自组织网络以及高效管理自身的能力. 但是单个节点的电源供给、运算能力以及存储能力有限,为了满足实际需求,就必须研究适合系统特点的通讯协议和网络管理方法.

然而,在研究阶段构造实际的大规模传感器网络造价昂贵,在真实的网络上进行重复实验很难与外部环境达到一致. 所以,构造系统的仿真模型,并在模型上进行实验成为传感器网络系统分析、研究的有效手段[4].

传感器网络的系统架构、运行的操作系统、使用的网络协议,以及网络内的分布数据处理方式都与实际应用紧密相关. 而且,传感器网络负责感知的实际物理世界也将一定的不确定性引入运行的网络. 所以,对于传感器网络,需要对实际的应用进行全面的研究,包括使用的操作系统以及协议栈.

1.1 网络仿真工具应满足的特性

对于一个好的传感器网络仿真工具,需要具备 4 个关键特性[5]:

扩展性: 网络仿真器必须能够在不同的配置下仿真数千节点的网络.

完整性: 网络仿真器必须能尽可能多地仿真系统之间的交互,在不同的层次尽可能精确地仿真系统的行为. 虽然单纯对算法和网络协议的仿真是研究的有效手段,但是传感器网络的本质决定了必须要仿真具体的应用.

逼真度: 网络仿真器必须能够细粒度地模拟网络行为. 节点之间通讯细节的模拟对于网络的评估和测试起着非常重要的作用. 而且,网络仿真器必须能够仿真未曾预期的行为,而不仅仅是预先估计的可能行为.

复用性: 在网络仿真器中运行的算法协议的仿真验证代码可以不加修改或很少修改地运行在实际的节点上. 因为,有

收稿日期: 2011-01-25 基金项目: 国家科技重大专项项目(2011ZX05008-005)资助. 作者简介: 宋洪治,男,1983 年生,博士研究生,研究方向为嵌入式系统设计; 武 杰,男,1975 年生,副教授,研究方向为高速数据采集与传输; 张 杰,男,1985 年生,博士研究生,研究方向为高速数据采集与传输.

时虽然算法可以工作,但实际的具体实现却表现很差.

目前,满足上述特性的仿真工具只看到用于 TinyOS[6] 仿真的 TOSSIM[5],但是它只能提供基于 TinyOS 的无线传感器网络仿真,无法进行其它类型的网络仿真. 其它的网络仿真工具,如 NS-2[7] 作为学术界网络研究的事实标准,虽然提供了众多的网络协议与实际系统的模型,但是仅满足扩展性的需求. 为了利用现有的网络仿真工具,并且满足传感器网络仿真的完整性、逼真度和复用性需求,就需要将网络节点中实际运行的 RTOS 与网络仿真器结合. 利用网络仿真器模拟网络的信道,物理层与 MAC 层,利用 RTOS 模拟器运行实际的网络协议. 并且,对于具有层次结构的传感器网络,可以将不同层次运行不同 RTOS 的节点同时仿真. 而且,采用这种方法还可以方便地验证 RTOS 中实现的网络仿真器中已存在的协议.

考虑到主流的网络仿真的工具,如 NS-2、NS-3[8] 以及 OMNet + +[9] 均提供源代码,有易于用户扩展功能,而且均可以运行于 POSIX[10] 兼容的操作系统. 所以,RTOS 模拟器也应该运行在相同的环境下,即 RTOS 模拟器应该是 POSIX 兼容的.

1.2 相关工作

目前,看到的被移植到 POSIX 环境下,提供模拟器的开源的 RTOS 有 Trampoline[11] 和 FreeOSEK[12]. 虽然这两个操作系统本身都支持抢占式的任务调度,但是它们的 POSIX 模拟器只在单一进程内模拟了多个任务[13],这就带来一些限制:

1) 需要使用显式的函数调用保存调用该函数时任务的运行环境,任务的切换也需要使用显式的函数调用,不能在任意时刻切换.

2) 某一任务不能调用会引起阻塞的系统函数,也不能出现没有调用 RTOS 系统函数的死循环,否则会导致 RTOS 无法切换任务;

3) 由于只有一个线程,所以调试时无法方便地观察每个任务的运行情况;

4) 模拟的任务的栈空间无法自动增长,仿真时可能会遇到栈空间溢出的情况.

由于上述限制,导致它们的 POSIX 版本非常适用于非抢占式多任务,但是不适用于抢占式多任务,并且模拟的完整性和逼真度也有限. 虽然有人贡献了 FreeRTOS[14] 在 Linux 环境下运行的代码,也使用了线程模拟任务,信号模拟硬件中断. 但是,在 Linux 系统的信号处理函数中使用了非异步信号安全的函数,这导致其运行非常依赖于底层函数库和主机操作系统的版本. 而且,在模拟的外设的中断到来之后,也没有抢占调度的判定. 另外,在实际使用中发现这个实现本身也存在一定的问题,经常产生错误的任务切换,无法用于传感器网络的仿真.

本文采用 FreeRTOS 6.0.4 版设计并实现了符合 POSIX 规范的 FreeRTOS 模拟器,与已有的网络仿真器结合,可满足传感器网络仿真的扩展性、完整性、逼真度与复用性的需求.

2 FreeRTOS 模拟器的设计与实现

FreeRTOS 作为一个轻量级的实时操作系统,提供任务管理、时间管理、信号量、具有优先级继承的互斥、消息队列、内存管理和追踪功能等,其调度策略可配置为抢占或非抢占模式,可满足传感器节点的需要. 而且作为一个设计良好的系统,为了保证系统的灵活性和扩展性,FreeRTOS 的所有功能都建立在体系结构相关的底层功能之上,包括任务相关的功能和中断处理相关的功能. 为了满足传感器网络仿真的完整性和逼真度,必须尽可能满足在任务运行的任何时刻都可以实现切换. 因此,对于任务的模拟就不能使用单进程模拟多线程的方法,而是使用了 POSIX 线程库. 考虑到硬件中断具有异步特性,所以对硬件中断的模拟使用了主机系统的信号.

2.1 基本策略

由于 FreeRTOS 假设系统只有一个处理器,所以,所有的体系结构相关的基本操作都必须按照时间顺序原子地完成.

然而,对于 POSIX 多线程程序,在多核处理器上运行时可能会有多个线程同时运行,而且没有办法原子地操作一个进程内所有线程的信号屏蔽. 所以,为了保证操作的原子性就需要一定的同步机制. 由于采用了主机系统的信号来模拟硬件中断,所以在主机系统的信号处理函数内可能包含需要原子完成的基本操作. 但是,POSIX 线程库提供的多线程同步的方法并不保证是异步信号安全的,这个特性与具体的实现紧密相关. 而且 POSIX 指定的异步信号安全函数也极为有限. 虽然可在主机系统的信号处理函数内使用非异步信号安全的函数,但是,使用时必须保证信号处理函数内出现的非异步信号安全函数不是被当前信号处理函数打断的那个函数. 而且,对于不同的操作系统,一些函数的实现也很难说清是操作系统内核提供的,还是函数库提供的. 所以,很难保证可以在信号处理函数内安全地使用非异步信号安全函数. 因此,为了保证基本操作的顺序性与原子性,使用了一个管理线程负责处理基本操作,任务线程或信号处理函数向管理线程发送基本操作的请求. 模拟硬件中断的信号只在管理线程响应.

管理线程和任务线程以及信号处理函数之间的通讯通过请求队列完成. 任务线程和信号处理函数将请求发送到请求队列,管理线程读取并处理请求队列中的请求. 由于请求队列要被任务线程和信号处理函数写入,而信号的到来是异步的,所以可能在任务线程正在写入请求队列时,管理线程收到信号,并在信号处理函数中发送请求到请求队列. 因此,请求队列的写入操作必须是原子的. 然而,POSIX 线程库提供的同步方法并不保证是异步信号安全的,操作 POSIX 信号量的方法中也只有增加信号量的方法是异步信号安全的. 因此,无法使用这些同步方法提供对请求队列的原子写入. 为了克服原子写入的问题,采用了管道实现请求队列,因为使用管道可以保证在单次写入操作小于 PIPE_BUF[10] 字节时,其操作时原子的,而且,对于常用的 POSIX 兼容的操作系统,PIPE_BUF 最小值为 512[15],可以满足需求.

发送到请求队列中的请求可以分为两类: 基本请求和管理请求. 基本请求是任务线程发送的,用于完成 FreeRTOS 体系结构相关的基本操作的请求. 在任务线程发送基本请求之后,将等待管理线程的回应,保证基本请求之后的任务代码运行在正确的环境中. 管理请求是用于 FreeRTOS 模拟器内部

的请求,用于主机系统信号模拟的硬件中断相关的操作.

2.2　中断的模拟

因为信号具有异步的特点,并且 POSIX 支持信号通知的异步 I/O,便于硬件外设的模拟,所以使用信号模拟硬件中断. 为了满足操作的顺序性与原子性,以及异步信号安全的限制,信号模拟的硬件中断的处理被分成了顶半部和底半部. 顶半部在信号处理函数中运行,底半部是与模拟的硬件相关的函数,在管理线程中被调用.

在模拟硬件中断的信号到来时,如中断处于使能态,则顶半部将合适的底半部请求及一个底半部管理请求送入请求队列. 在管理线程中,底半部请求先被处理,之后底半部管理请求被处理. 由于可能有多个信号被同时挂起,所以,底半部管理请求维护了一个底半部管理请求嵌套计数. 当发出底半部管理请求时,计数加 1,执行底半部管理请求时,计数减 1,只有当这个计数为 0 时,底半部管理请求才会检查模拟的中断是否要求任务切换,并作相应的操作. 这样做减少了不必要的任务切换操作. 而且这个行为和运行在实际硬件上的系统的行为一致,最大可能地满足了模拟器的完整性和逼真度的要求.

如果当前中断处于禁止态,则将底半部操作送入延迟队列. 在管理线程中,当中断被使能时,延迟队列中的底半部请求及一个底半部管理请求被送入请求队列. 保证了模拟器的行为和运行在实际硬件上的系统的行为一致.

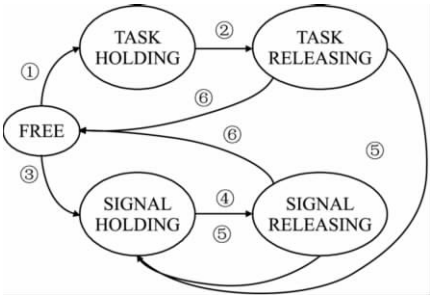
2.3　中断请求和任务请求的同步

无论是模拟的系统时钟中断,还是硬件外设的中断,在 FreeRTOS 工作在抢占模式下时,在中断结束之后,都可能造成任务调度. 所以,在请求队列中,在中断的底半部请求之后一定不能出现任务线程的请求. 否则,可能出现中断的底半部要求切换任务,在任务切换之后,管理线程却在处理已经挂起的任务线程的请求的情况. 所以,对于请求队列的写入,需要信号处理函数和任务线程之间达成同步.

同样,受到异步信号安全的限制,不能使用已有函数库提供的同步方法实现请求队列写入的同步. 为了达到同步,引入了请求队列的状态,并使用了 GCC[16] 的内建原子操作扩展[17] 原子地检测并改变请求队列的状态. 因为操作系统体系结构相关代码和编译器也紧密相关,而且,GCC 也可以运行在很多平台之上,所以,这种做法并无不妥.

请求队列的各个状态及转换关系及条件如图 1 所示. FREE 态表示请求队列为空,并且当前没有正在被写入的请求. TASK HOLDING 态表示任务线程正在写入请求. TASK RELEASING 态表示任务线程已经完成请求的写入. SIGNAL HOLDING 态表示当前信号处理函数正在写入底半部请求,或请求队列中还存在未处理的底半部请求. SIGNAL RELEASING 态表示请求队列中的底半部请求已经处理完毕,并且当前没有信号处理函数写入底半部请求.

对于任务线程,当发送基本请求时,若请求队列的状态为 SIGNAL HOLDING、SIGNAL RELEASING 或 TASK RELEASING 时,则主动放弃处理器,让管理线程继续处理请求队列. 对于信号处理函数,当发送底半部请求时,若请求队列的状态为 SIGNAL HOLDING,则仍然将底半部请求和底半部管理请求送入请求队列. 若状态为 TASK HOLDING 或 TASK RELEASING,则将底半部请求送到延迟队列. 这样,就保证了在请求队列中未被处理的底半部请求之后不会出现任务线程的请求.



① 任务线程准备向请求队列发送请求
② 任务线程已经将请求发送到请求队列
③ 信号处理函数在中断使能时向求求队列添加底半部请求
④ 在底半部管理请求被处理并且其嵌套数为 0 时
⑤ 在管理线程中,中断被使能,并且延迟队列不空时
⑥ 在管理线程中,当中断被禁止,或延迟队列为空时

图 1　请求队列状态转换及条件

Fig. 1　Request queue status and transition conditions

2.4　管理线程的实现

FreeRTOS 模拟器管理线程的主循环如图 2 所示. 模拟硬件中断的信号在读取请求队列时没有被屏蔽,此时,管理线程可以响应这些信号. 在管理线程取出一个请求之后便将这些信号屏蔽,以保证所有的基本请求都被原子地处理,并保护如延迟队列这类在管理线程和信号处理函数中都会操作的数据结构. 在实际硬件上运行的 FreeRTOS 的这些基本操作也都是原子的,所以,模拟器中的上述处理也符合实际情况.

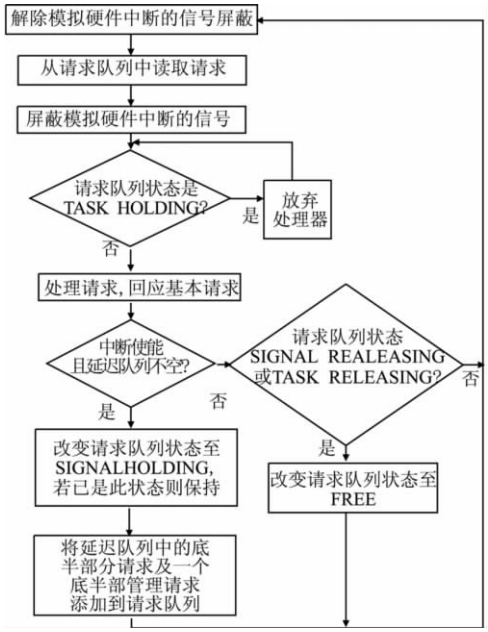


图 2　管理线程的主循环

Fig. 2　Main loop of management thread

当任务线程向请求队列发送基本请求时,可能在请求发

送完毕,任务线程置请求队列状态为 TASK RELEASING 前,或由于主机系统产生了任务调度,或由于在多核处理器上管理线程和任务线程同时运行,管理线程看到的请求队列状态为 TASK HOLDING. 所以,在处理请求之前,需要检查当前请求队列的状态,若为 TASK HOLDING,则主动放弃处理器,让任务线程完成对请求队列状态改变. 否则,可能会出现这样一种情况: 当前基本请求要求当前任务放弃处理器,并且存在更高级别的就绪任务,管理线程又刚好在任务线程置请求队列状态为 TASK RELEASING 前将其挂起,则在将来某一时刻重新切换到这个任务时,请求队列状态会被无条件地设置为 TASK RELEASING,造成错误.

在处理完当前请求之后,如果中断处于使能态,并且延迟队列不空,则将延迟队列中的所有底半部请求及一个底半部管理请求送入请求队列. 请求队列的状态将按照 2.3 节的描述变化.

2.5 任务管理

FreeRTOS 的任务调度相关逻辑是在体系结构无关部分中完成的,调度器使用体系结构相关部分完成具体的调度动作. 体系结构相关部分任务相关的操作主要包括: 创建任务、结束任务和切换任务.

每一个任务在模拟器中使用如下数据结构表示:

```
struct xThreadState {
portLONG lIndex;                    //模拟器中的任务索引
pthread_t hThread;                  //模拟任务的线程 ID
xTaskHandle hTask;          //FreeRTOS 任务控制块指针
//模拟器跟踪的任务状态
sig_atomic_t volatile xStatus;
//任务的临界区嵌套深度
unsigned volatile portBASE_TYPE uxCriticalNesting;
//用于继续任务的信号量
RBSem_t xResumeSem;
ITC_t xITC;                 //用于任务线程和管理线程通讯
};
```

为简化管理,模拟器使用定义好的数组存储每一个任务的状态,lIndex 即为任务在这个数组中的索引. 模拟 FreeRTOS 任务 hTask 的线程 ID 为 hThread. 为了保证任务切换的顺序性与原子性,模拟器内部单独维护了任务的状态 xStatus. 由于 FreeRTOS 提供的 API 可能在临界区内部主动放弃处理器,产生任务切换,所以需要维护每一个任务的临界区嵌套深度 uxCriticalNesting. 在切换任务时,根据每个任务的临界区嵌套深度设置系统的中断使能. 在任务被挂起后,挂起的任务等待 xResumeSem 信号量,直到任务被重新激活. 由于 xResumeSem 信号量要在信号处理函数中使用,所以受到异步信号安全的限制,这个信号量使用管道实现. 任务通过管道实现的线程间通讯对象 xITC 和管理线程通信.

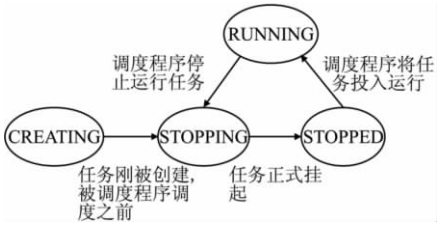


图 3 模拟器的任务状态

Fig. 3 Task status of the simulator

模拟器的任务状态如图 3 所示. CREATING 态表示任务正在被建立; STOPPING 态表示任务正在被停止; STOPPED 态表示任务已经被停止; RUNNING 态表示任务正在运行.

2.5.1 任务的挂起与激活

为了满足模拟的完整性与逼真度的要求,任务必须能够在任意点被挂起,即任务的挂起必须是异步的. 因此,采用了主机系统的信号挂起任务线程. 任务的挂起与激活如图 4 所示,图中虚线表示不同线程之间的相互作用.

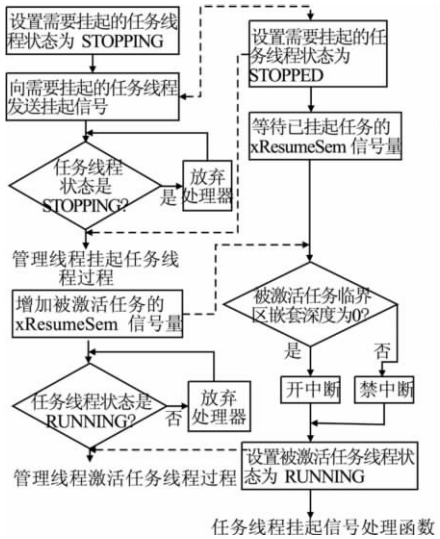


图 4 任务的挂起与激活

Fig. 4 Task suspension and resumption

在需要将任务挂起时,为了保证操作的顺序性,管理线程首先设置任务线程的状态为 STOPPING,然后向任务线程发送挂起信号,并等待任务线程进入信号处理函数. 任务线程进入挂起信号的信号处理函数,置任务线程的状态为 STOPPED. 此时,管理线程完成了任务挂起的操作,而任务线程在信号处理函数中等待 xResumeSem 信号量.

在需要激活任务时,管理线程首先增加 xResumeSem 信号量. 为了保证操作的顺序性与原子性,管理线程等待任务线程进入运行状态. 任务线程的信号处理函数在信号量等待完毕之后,检查被激活任务的临界区嵌套深度. 如果为 0,则在当前激活任务运行环境内使能中断,否则,禁止中断. 最后,任务线程的信号处理函数设置激活任务的状态为 RUNNING,并退出,回到任务线程继续运行. 此时,管理线程见到任务线程状态的改变,从而完成了任务激活的操作.

2.5.2 任务的创建与销毁

创建任务时,管理线程首先设置新建任务的状态为 CREATING. 这个状态仅仅表明当前任务正在被创建,主要用于调试. 之后,按照 FreeRTOS 创建任务时提供的参数创建模拟任务的线程. 为了保证基本请求处理的原子性,管理线程将等待任务进入 STOPPED 状态. 新建的线程首先设置本线程的信

号屏蔽. 除了必须响应的信号 SIGSTOP、SIGKILL、SIGFPE、SIGILL、SIGSEGV 和 SIGBUS 外,任务线程只需要响应挂起信号. 除了用于模拟硬件中断的信号,对于其它的信号可以根据需要有选择地响应. 在初始化必要的资源之后,为了让任务线程可以在任何地方退出,设置其取消状态为异步取消[10]. 并在真正运行任务代码前将自己挂起. 在挂起信号处理函数内任务状态会被置为 STOPPED. 此时,任务的创建完成,FreeRTOS 调度此任务时,任务开始运行.

销毁任务时,管理线程首先检查被销毁的任务是否为当前运行任务. 如果是,则产生一次任务调度,激活其它挂起的任务. 此时,欲销毁的任务已经不是当前运行任务了,管理线程取消被销毁任务线程的运行,并等待任务线程结束. 当主机系统结束了任务线程后,便完成了任务销毁的过程.

3　测　试

FreeRTOS 模拟器的测试使用了 Debian 5.0 Linux 发行版,内核版本 2.6.26-2-686,处理器 AMD Athlon(tm) II X2 245. 测试时,运行有 FreeRTOS 提供的算数运算、消息队列、信号量、互斥、动态任务创建/销毁相关的测试任务. 同时,还运行有使用了 POSIX 异步 I/O 模拟的网络外设的 echo 客户端任务,与主机的 echo 服务端程序通信. 包括 FreeRTOS 的 idle 任务,运行时,FreeRTOS 创建的任务数最少为 35 个,最多 39 个. 整个测试持续了 9 个小时,共产生 12163642 次模拟的硬件中断,发生 2204523 次任务切换,echo 客户端发送了 7850245 条消息. FreeRTOS 模拟器只运行 idle 任务时,消耗 128KB 内存. 每个新增测试任务消耗 8 ~ 12KB 内存. 模拟器对内存小的需求量,决定了其具有良好的扩展性. 若仿真传感器网络时,每个节点上运行的 FreeRTOS 创建了 4 个任务,则仿真 1000 个节点只需要消耗不到 200MB 内存,一般的 PC 都可以满足仿真的资源需求.

4　结　论

目前主流的网络仿真器都具有良好的扩展性,但是由于仿真的大都是算法和网络协议,所以在完整性与逼真度上与实际的实现有一定的差别. 而在代码的复用性上,存在巨大的缺陷. 为了满足传感器网络应用的需要,又要利用现有网络仿真器的优点,本文设计并实现了符合 POSIX 规范的 FreeRTOS 模拟器,与网络仿真器结合使用,可以克服单纯使用网络仿真器的缺陷. 而且,使用线程模拟任务,信号模拟硬件中断的方法,也适用于其它轻量级多任务 OS 模拟器的实现.

在实现了 FreeRTOS 模拟器后,下一步的工作是让 FreeRTOS 模拟器运行于 NS-3 的节点之上,利用 NS-3 模拟传感器网络的信道、物理层和 MAC 层,真正的网络协议直接在 FreeRTOS 上实现.

References:

[1] Jennifer Yick, Biswanath Mukherjee, Dipak Ghosal. Wireless sensor network survey [J]. Computer Networks, 2008, 52(12): 2292-2330.

[2] Stefano Savazzi, Umberto Spagnolin. Synchronous ultra-wide band wireless sensors networks for oil and gas exploration [C]. IEEE Symposium on Computers and Communications, 2009: 907-912.

[3] Mihai Beffa, Doug Crice, Roy Kligfield. Very high speed ordered mesh network of seismic sensors for oil and gas exploration [C]. IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems, 2007: 1-5.

[4] Li Chuan-wen, Gu Yu, Li Fang-fang, et al. SnSim: study and implementation of a wireless sensor network simulator [J]. Journal of Chinese Computer Systems, 2010, 31(6): 1025-1029.

[5] Philip Levis, Nelson Lee, Matt Welsh, et al. TOSSIM: accurate and scalable simulation of entire TinyOS applications [C]. Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, 2003: 126-137.

[6] Philip Levis, Sam Madden, Joseph Polastre, et al. TinyOS: an operating system for sensor networks [C]. In: Ambient Intelligence, Part II. New York: Springer-Verlag, 2005: 115-148.

[7] Information Sciences Institute, University of Southern California. The NS-2 network simulator [EB/OL]. http://www.isi.edu/nsnam/ns/, 2011-01.

[8] Tom Henderson. The NS-3 network simulator [EB/OL]. http://www.nsnam.org/, 2011-01.

[9] OMNeT++ Community. OMNet++ network simulation framework [EB/OL]. http://www.omnetpp.org/, 2011-01.

[10] Standard for Information Technology-Portable Operating System Interface (POSIX®) Base Specifications [S]. 2008, 7.

[11] Jean-Luc Béchennec, Mikaël Briday, Sébastien Faucou, et al. Trampoline an open source implementation of the OSEK/VDX RTOS specification [C]. IEEE Conference on Emerging Technologies and Factory Automation, 2006: 62-69.

[12] FreeOSEK-scalable RTOS for embedded systems [EB/OL]. http://opensek.sourceforge.net/, 2010.

[13] Ralf S Engelschall. Portable multithreading: the signal stack trick for user-space thread creation [C]. Proceedings of the Annual Conference on USENIX Annual Technical Conference, 2000: 239-250.

[14] Real time engineers Ltd. the FreeRTOS project [EB/OL]. http://www.freertos.org/, 2011-01.

[15] Marc J Rochkind. Advanced UNIX programming (2nd Edition) [M]. Boston: Addison-Wesley Professional, 2004.

[16] Free software foundation. GCC, the GNU compiler collection [EB/OL]. http://gcc.gnu.org/, 2011, 1.

[17] Free software foundation. atomic builtins-using the GNU compiler collection (GCC) [EB/OL]. http://gcc.gnu.org/onlinedocs/gcc-4.1.2/gcc/Atomic-Builtins.html, 2011-01.

附中文参考文献:

[4] 李传文,谷　峪,李芳芳,等. WSN SnSim: 一种无线传感网络仿真平台的研究与实现 [J]. 小型微型计算机系统, 2010, 31(6): 1025-1029.