基于无线网络控制器的嵌入式操作系统支撑层设计与实现 *

何先波[1,2], 宋 钰[3]

(1. 西华师范大学 计算机学院,四川 南充 637002; 2. 中兴通讯股份有限公司 成都研究所,成都 610041; 3. 四川理工学院 网络管理中心,四川 自贡 643000)

摘 要: 针对无线网络控制器的应用特点给出了一种操作系统支撑层方案,并对其中的调度、定时器、内存管理、I/O 驱动和任务间通信等封装机制及实现进行了深入的分析与研究。该方案为无线网络控制器中的 ATM 传输网络子系统、无线信令子系统、数据库子系统和操作维护子系统的软件提供了一个统一的分布式编程平台和运行平台。通过无线网络控制器产品的实际软件开发实践证明,该方案具有较高的应用价值。

关键词: 嵌入式系统; 操作系统支撑层; 无线网络控制器

中图分类号: TP316,TP393 **文献标志码**: A **文章编号**: 1001-3695(2010)05-1826-04

doi:10.3969/j.issn.1001-3695.2010.05.062

Design and implementation of support layer of embedded operating system based on radio network controller

HE Xian-bo[1,2], SONG Yu[3]

(1. *School of Computer, China West Normal University, Nanchong Sichuan* 637002, *China*; 2. *Chengdu Institute of Zhongxing Telecommunication Equipment Corporation, Chengdu* 610041, *China*; 3. *Network Administration Center, Sichuan University of Science & Engineering, Zigong Sichuan* 643000, *China*)

Abstract: This paper suggested a schema of the operating system support layer oriented to radio network controller (RNC), and deeply studied and analyzed its scheduler, timer, memory management, I/O driver and internal process communication (IPC) encapsulation mechanism. This schema supplied a uniform distributed programming and running platform to ATM transform network subsystem, radio signal sub system, database subsystem and operation and maintenance subsystem. The software development practice of communication product ZXW10-RNC proves that this schema has high application value.

Key words: embedded operating system; operating system support layer; radio network controller

嵌入式软件的可移植性和稳定性一直是嵌入式软件开发研究的重点领域之一。由于嵌入式系统通常具有面向特定应用“量身定制”的特点,在实际的嵌入式软件开发中便出现不同类型的嵌入式软件产品可能使用不同的嵌入式操作系统的现象,从而造成嵌入式软件较差的可移植性和可重用性。另外,对特定领域带普适性的经过充分测试验证的功能模块应统一集成,避免重复开发。这些问题可通过在嵌入式操作系统之上增加一支撑层来有效解决,支撑层可看做操作系统功能面向特定应用领域软件开发的再延伸。对于无线网络控制器(radio network controller,RNC),其操作系统支撑层(operating system support layer,OSSL)试图给 RNC 应用软件提供一个一致的编程平台,在 RNC 系统中的位置如图 1 所示。

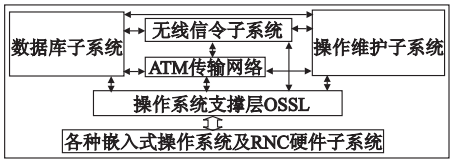

图1 OSSL在RNC中的位置

1 RNC 操作支撑子系统总体方案

本文提出的 RNC 操作系统支撑层的总体方案框架如图 2 所示。

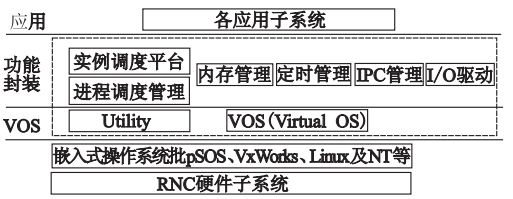

图2 RNC OSSL框架

1.1 VOS(virtual operating system)层

与操作系统为了隔离具体的硬件环境而设计出 HAL(hardware abstract layer)层类似,支撑层中 VOS 层的目的是屏蔽底层不同操作系统之间接口差异,从而为上层提供一致的与底层具体操作系统无关的接口。支撑层其他部分或应用程序只需调用 VOS 层提供的标准功能接口,VOS 层识别该接口调用后,直接或间接地路由到下层具体嵌入式操作系统提供的相

收稿日期: 2009-10-10; **修回日期**: 2009-11-18 **基金项目**: 四川省教育厅重点项目(08ZA015); 国家“863”计划资助项目(2002AA1Z2306); 西华师范大学博士启动基金资助项目(08B078); 西华师范大学校基金资助项目(08A020)

作者简介:何先波(1971-),男,四川苍溪人,副教授,博士,主要研究方向为嵌入式软件开发环境、通信网络(hexianbo_sctc@163.com).

应接口调用。由于应用程序对具体的嵌入式操作系统的依赖减少,从而便于上层应用软件在不同操作系统之间移植。提供基于 Windows NT 版本的操作支撑子系统可以大大提高应用开发调试的效率,应用一般先通过 NT 版本的调试,然后加载到嵌入式操作系统及单板上调试。

VOS 层主要实现信号量操作、消息队列操作、调度控制、内存管理、时钟管理、I/O 驱动等的基本接口 API,给上层提供平台无关的系统功能调用。

1.2　二次调度封装机制

对通信领域的应用,相互通信的任务可能位于同一 CPU 之内,也可能位于同一背板的 CPU 之间或通过网络连接远程终端之间,其通信方式既有同步的,也有异布的。而几乎所有嵌入式操作系统都提供的消息通信机制无疑是统一这些通信方式较理想的选择。同时,对于通信领域中类似交换设备多呼叫处理之类的应用,如果每个用户呼叫都使用操作系统级的任务进行处理,将会造成系统资源的极大消耗,从而导致系统性能的下降甚至瘫痪。为了有效解决上述问题,在支撑层中对调度管理采用了适合通信领域应用的二级调度方式(相对于以消息驱动为主,基于操作系统任务调度机制实现的调度方式称为一级调度)。

所谓二级调度是指在每个操作系统级任务上承载多个执行单元(本文称为次任务),任务调度由嵌入式操作系统完成,而承载于任务中的次任务调度由其承载任务进行再次调度,承载任务也称为调度任务,因此 OSSL 中通常存在两类任务,即调度任务和特殊任务。特殊任务包括进程间通信控制任务、驱动任务、定时任务、监控任务等,与调度任务不同,特殊任务不承载次任务调度功能。特殊任务与调度任务由嵌入式操作系统完成调度。OSSL 次任务、实例的调度机制如图 3 所示。

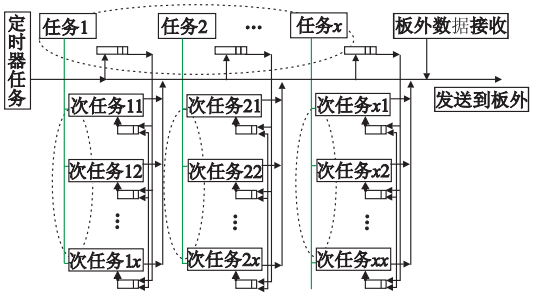

图3　OSSL二级调度机制

次任务是 OSSL 二级调度的实体,每个次任务有自己单独的栈,次任务调度由消息驱动,得到调度的次任务称为实例。OSSL 方案中的次任务分为两种,即单实例次任务和多实例次任务。单实例次任务中只有一个缺省的特殊实例,其入口函数即为整个次任务的执行入口点;多实例次任务除了缺省的特殊实例外,还有多个其他普通实例,普通实例拥有自己的数据区和共同的入口函数。

参与 OSSL 调度的每个次任务都有自己的消息队列,调度任务负责从自己的消息队列中接收消息,并派发给目的次任务所在调度任务消息队列;目的次任务所在的调度任务获得运行资格后,将控制权交给有消息待处理的次任务,次任务在消息激励下得到处理,最后进行状态跃迁,将控制权交还给调度任务。如果该次任务消息队列没有任何消息,则被挂到阻塞队列尾,否则挂到就绪队列尾,并从就绪队列头摘取新的次任务,从而保证承载任务中每个次任务都有平等的运行机会。图 4 描述了次任务及实例的调度流程。

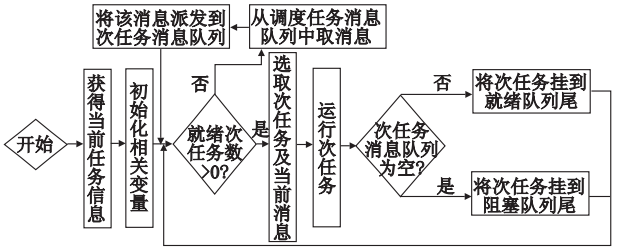

图4　二级调度中调度任务的功能流程

1.3　内存管理封装机制

针对通信领域的应用中内存申请往往呈现各种相同数据结构(即相同大小内存块)的大量申请现象,支撑层采用了如图 5 所示的块内存管理组织结构。块内存管理的基本思想为:预先向操作系统申请一片大的内存区,再将该内存区分为若干内存池,同一内存池中内存块的大小相同。分配内存时,以固定大小块方式分配给申请者,从而提高分配效率。

内存池由若干尺寸相同的内存块组成,为了有效地管理和监控内存的使用情况,每一内存池使用一内存池控制头存储其管理统计信息,该控制头记录该池对应的内存块大小,内存块数量、空闲块数目、块分配峰值、分配与释放失败计数、重复释放计数及指向每个内存块控制头的指针数组等信息。

内存池中每一内存块用一内存块控制头来描述和控制,该控制头包括申请进程号、所属内存池以及用于调试用的相应信息(如申请/释放操作所在的文件名、行号及时间等)。

每一内存区也对应有相应的统计信息(如申请的累计字节数、申请失败次数、重复释放次数等)。

同时,内存封装机制中采用“加墙方式”的内存保护监测机制。其操作方式是在要保护的内存的头部或尾部(一般为尾部,因为在实际操作时对地址增加方向的操作概率要大一些,当然也可在头部和尾部同时进行)多添加一个或几个字节,并将这个字节内容置为应用中一般不会出现的特殊数值。如果在内存释放归还时发现这些字节内容已改变,则说明对该块内存的操作已经越界,从而在释放时实现越界检测。

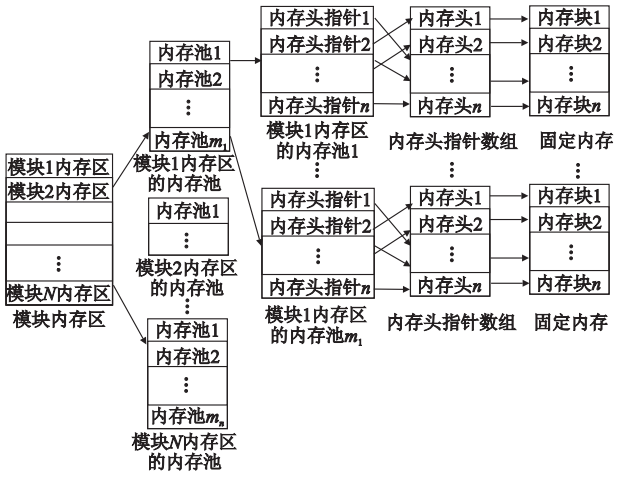

图5　块内存管理结构图

1.4　定时管理机制

OSSL 层中,定时管理的核心算法采用了循环多队列算法,其原理如图 6 所示。在图 6 中,采用若干个(图 6 中为 L)一定长度的数组和循环指针变量组成一个循环计时队列,循环指针变量以循环的方式逐一指向各数组元素,循环指针变量也称做

TICK 游标,这样,构成一单循环队列(循环队列长度为 L),而该单循环队列的每个节点又是一个单向的有序队列。

当上层应用程序申请定时器时,首先根据待设定定时器的长度找到循环队列中相应的节点元素的位置(位置的获取通过循环指针当前值 + 待设定定时器的时长对循环队列长度取模运算),然后再根据待设定定时器的循环因子(待设定定时器时长与循环队列长度 L 相除所得的商)插入到循环队列该节点上的单向队列中。

在每个计时周期到来时,TICK 游标向前走一位,查询该处的链表队列是否为空,如果为空,则不处理;如不为空,则判断该处队列中定时器循环因子值是否为 0,若为 0,则进行相应的报时操作,并删除该定时器。如果是循环定时器,则从原链表删除,计算新位置,再次插入循环队列对应元素的新链表,不删除该定时器描述数组元素;若循环因子值不为 0,则将循环因子值减 1,然后判断下一个定时器。

当上层应用程序不再需要定时器时,从循环队列中删除该定时器。

支撑层定时管理模块就是基于上述的循环队列算法实现的。其主要包括定时器管理、日期/时间管理、信息查询等功能。各功能分别对上层应用提供函数调用接口实现。定时器资源定义为定时器结构数组。

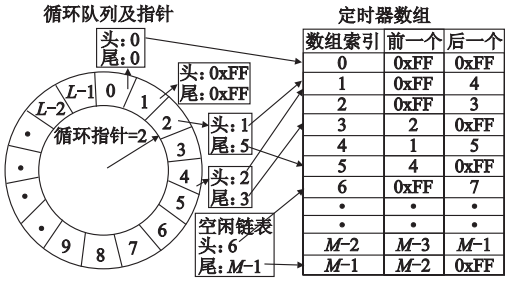

图6　循环多队列定时器

1.5　I/O 驱动封装机制

I/O 系统设计机制如图 7 所示。使用驱动任务的方式实现高效的驱动程序,各个驱动模块可以方便地挂入、卸载,从而增加了系统的可定制性和灵活性,也使得底层驱动代码实现了共享,简化了软件模块。对所有 I/O 操作仅提供 IoInit( )、IoOpen( )、IoClose( )、IoRead( )、IoWrite( )、IoCntrl 六个 API 调用接口支持。同时,对中断的实现中注意 DPC(delayed procedure call)与 ISR 的分工合作,从而确保 I/O 驱动的高效,DPC 机制采用驱动任务的方式实现。

1.6　IPC 设计说明

RNC 往往设计成一个分布式的处理环境。OSSL 的 IPC(internal process communication)需要为上层应用提供一个分布式的开发平台,屏蔽物理位置的差异。IPC 采用基于 TCP 的进程间通信,主备间通信采用 HDLC 链路。

RNC 的每个 CPU 有一个 100 兆以太网口用于板间通信,构成一个内部以太网。每块单板可以从背板读取板位信息(机架号、机框号、槽位号、CPU 号),并根据物理板位信息生成 IP 地址。所以,物理板位到 IP 地址之间是一种确定的一一对应关系。

板位信息对应用而言是透明的,用户只需要指定进程的标志 PID(process ID),而无须知道这个进程所处的位置。IPC 需要根据 PID 定位单板位置,并生成 IP 地址,进行建链传输。

板间通信具体包括处理机之间的 TCP/IP 通信、UDP/IP 通信、主备之间的 HDLC 点对点通信。整个 RNC 局域网关系如图 8 所示。

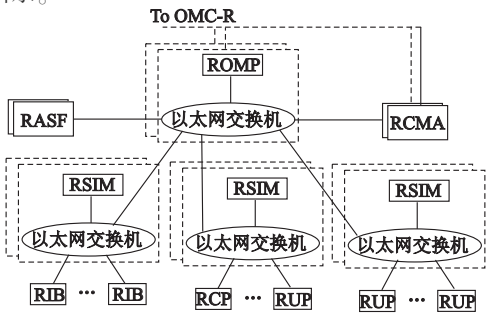

图8　RNC以太网拓扑图

IPC 主要采用 TCP 传输,以利用 TCP 的高可靠性保证板间通信的准确可靠。某些特殊场合采用 UDP 传输,以减少不必要的 TCP 链路维护的开销。主备间通信提供专门的调用接口。

TCP 链路管理是 IPC 的重点。采用 TCP 推荐的 client/server 结构,链路管理的设计基于以下考虑:

a)每个处理器的地位在以太网上是平等的,没有客户机和服务器的区分,既可以发起连接又可以接收连接请求,所有的连接均动态建立。

b)对每个处理器能够建立的最大连接数目进行控制。由于每建立一个连接都要耗费一定的系统,如果连接数目过多,处理器的性能会受到一定的影响。

c)连接作为一种全局资源,需要统一管理。设计了一个叫连接表的数据结构对连接进行管理。

d)强化连接管理中的异常处理。TCP 是一种可靠传输协议,因此使用 TCP 进行通信时的主要异常是在链路的建立、保持、拆除上。

单板间建链 TCP 连接采用 client/server 结构。对一个单板而言,有两种 TCP 连接,一种称做发送连接(由该单板作为 client 主动建链);一种称做接收连接(由该单板作为 server 被动接收连接请求)。发送连接用于对外发送消息,接收连接用于接收外来消息,这样确保板间通信的双方地位完全是对等的,也避免了建链中的互锁。

由于采用动态建链的方式,在多任务系统中存在并发建链的问题。解决的方法是对建链操作进行临界保护,以达到防止重复建链的目的。

图 9 简单地描述了两种连接(发送连接、接收连接)的建立流程。一条连接对 client 方来说是发送连接,对 server 方来说是接收连接。

由于 RNC 中单板数量可能达到一个很大的数目(如 100 个),IPC 中的 TCP 链路管理需要很大开销,通过分级建立连接的方式来控制连接数目。

另外,进程间通信的最小单位是消息,而通信链路是流 Socket,所以需要考虑链路上的消息定界、缓存。某些特定应用可能发送很大的一个消息,超出 TCP 的处理能力时,需要考虑拆包打包。同时需要考虑消息的路由转发和主备倒换时的链路保护。

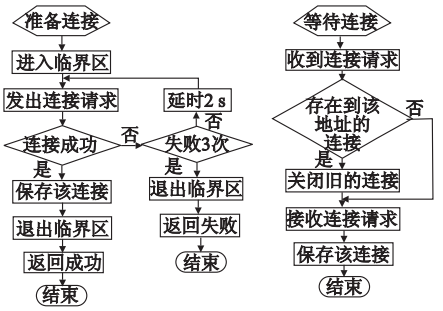

图9　发送与接收连接建立流程

2　实验测试数据与分析

2.1　块内存管理封装机制

基于 VxWorks 商用嵌入式操作系统支撑层块内存申请和释放性能测试结果如表 1 所示。从表 1 中的测试数据可以看出,支撑层的内存管理操作相对于直接底层操作系统接口的调用,时间性能上略有降低,但支撑层中增加了内存操作的统计、保护和调试等措施;在面向通信领域的软实时应用中,相比较而言,由此产生的时间性能降低是值得的。

表 1　固定大小的内存申请和释放性能测试　　ms

	Vxworks get/free	OSS VOS_get/VOS_free
5 000 个 64 Byte 的内存	9.945/10.408	11/14
2 000 个 8 KB 的内存	4.481/4.640	4/5.5
1 000 个 64 Byte 的内存	1/1	2/3
500 个 8 KB 的内存	0/1	1/1

2.2　二级调度

基于 VxWorks 的一级调度(无次任务的调度)和二级调度切换性能测试结果如表 2 和 3 所示。

从性能测试数据对比上看,对于一级调度,任务的切换时间在 2 ~ 4 μs 之间,且与任务数目有关,任务采用发送消息的方式引起的任务切换时间为 28 ~ 36 μs;对于二级调度,次任务消息引起的次任务切换时间在 7 ~ 11 μs 之间。对于消息传递所引起的次任务和任务切换,二级消息调度方法明显优于一级消息调度。

3　结束语

本文针对无线网络控制器(RNC)的应用特点提出了一种操作系统支撑层方案,并对其中的调度、定时器、内存管理、I/O 驱动及任务间通信(IPC)等模块给出了更高效和稳定的封装机制。该方案为无线网络控制器中的 ATM 传输网络子系统、无线信令子系统、数据库子系统和操作维护子系统的软件提供了一个统一的分布式编程平台和运行平台。通过无线网络控制器产品的实际软件开发实践证明,该方案具有较高的应用价值。

表 2　一级调度切换时间　　μs

测试序号	任务个数	总切换次数	切换时间(包括信号量)	实际切换时间(减去信号量)
1	2	4 000 000	2.498 4	2.000 2
2	4	4 000 000	2.500 0	2.001 8
3	8	8 000 000	2.500 2	2.002 0
4	16	8 000 000	2.500 0	2.001 8
5	32	8 000 000	2.746 9	2.248 7
6	64	8 000 000	3.999 8	2.501 6
7	80	8 000 000	4.314 4	3.822 4
8	100	8 000 000	4.498 5	4.000 3
9	160	8 000 000	4.499 0	4.000 8
10	200	8 000 000	4.500 2	4.002 0

表 3　二级调度测试结果表

切换类型	测试序号	次任务个数	切换次数	切换时间/μs	
不同任务	1	2	1 000 000	28.35	
	2	4	1 000 000	30.54	
	3	8	1 000 000	31.00	
	4	16	1 000 000	32.81	
	5	32	1 000 000	36.04	
	6	63	1 000 000	36.03	
同一任务	1	1	8 000 000	7.3	性能提高倍数
	2	2	8 000 000	7.5	3.78
	3	4	8 000 000	10.0	3.05
	4	8	8 000 000	10.7	2.90
	5	16	8 000 000	10.8	3.04
	6	32	8 000 000	11.0	3.28
	7	64	8 000 000	11.2	3.22

参考文献:

[1] 何先波,钟乐海,芦冬昕. 嵌入式操作系统支撑层的设计与实现[J]. 计算机应用,2003,23(5):89-91.

[2] 何先波,李志蜀,芦东昕,等. 一种面向通信领域的高效嵌入式操作系统进程队列模型[J]. 哈尔滨工程大学学报, 2006,27(2):263-267.

[3] 刘飞,罗克露,黄烨明,等. 通信领域嵌入式系统调度管理的设计与实现[J]. 计算机应用,2004,24(7):186-188.

[4] 刘飞, 芦东昕, 缪敬. 面向通信领域通用内存管理单元的算法和实现[J]. 计算机工程, 2003,29(22):80-82,105.

[5] 何先波,张芝萍,徐立峰,等. 一种嵌入式实时操作系统中内存分配的方法,中国:CN1722106[P]. 2004.

[6] 钱静,芦东昕,谢鑫,等. 嵌入式软件虚拟内存管理技术的研究和实现[J]. 计算机应用, 2005,25(2):180-182.

(上接第 1825 页)

[2] ERRADI A, MAHESHWARI P. WsBus: QoS-aware middleware for reliable Web services interactions[C]// Proc of IEEE International Conference on eTechnology, E-Commerce and E-Service. 2005.

[3] SHETH A, CARDOSO J, MILLER J, *et al*. QoS for service-oriented middleware [C]// Proc of the 8th International Conference on Information Systems Analysis and Synthesis. 2002.

[4] ALWAGAIT E, GHANDEHARIZADEH S. DeW: a dependable Web services framework[C]// Proc of the 14th International Workshop on Research Issues on Data Engineering Web Services for E-Commerce and E-Government Applications . 2004.

[5] GUO Hui-peng, HUAI Jin-peng, LI Huan, *et al*. ANGEL: optimal configuration for high available service composition[C]// Proc of ICWS. 2007.

[6] HUANG Gang, LIU Xuan-zhe, HONG Mei. SOAR: towards dependable service-oriented architecture via reflective middleware[J]. International Journal of Simulation and Process Modelling, 2007, 3(1/2):55-65.

[7] GUO Hui-peng, HUAI Jin-peng, LI Yang, *et al*. KAF: Kalman filter based adaptive maintenance for dependability of composite services [C]// Proc of CAiSE. 2008.

[8] 张皓. 基于 Web 服务资源框架的在线实验服务设计与实现[D]. 武汉:华中科技大学,2006.