计算机科学1998 Vol. 25 №. 3

超微内核嵌入式实时操作系统的设计*)

The Design of Embeded Nanokernel Real-Time Operating System

徐仑峰 熊光泽 刘锦德
(电子科技大学微机所 成都610054)

摘 要 本文通过对现代实时操作系统需求及微内核系统设计优点的分析，提出一种嵌入式实时操作系统的设计方法。它结合了操作系统微内核设计与传统设计的特点，较好地解决了实时操作系统的可伸缩性、实时性能、可移植性及编程界面标准化等问题。

关键词 实时操作系统、微内核、纳核、BSP

随着实时应用领域的扩大，实时应用覆盖的范围也不断加大，从低端到高端都有应用的需求，这给现代实时操作系统(RTOS)的研究与开发提出了新的要求与挑战，归纳起来主要有：

(1)有良好的实时确定性和实时性能。系统的实时确定性和实时性能是RTOS重要特性之一，它要求RTOS对外部和内部事件的响应时间是确定的。RTOS的实时性能依赖于系统核心的数据结构及各类算法，且操作系统的体系结构对其时间确定性及实时性能也有重要的影响。

(2)满足多种应用需要的可伸缩性。现代RTOS应能支持多种实时应用的需求，用户希望开发的RTOS可方便配置，满足各种实时应用的需求，无论它们是高端的实时分布式应用(如银行业务)，还是低端的强实时、深嵌入应用(如智能武器系统和工业控制系统等)。即要求RTOS有良好的可伸缩性、可剪裁性及可重用性。众所周知，现代计算机硬件系统基于总线结构，它可通过在系统总线上加入/移去硬件子系统来增强或减弱全系统的硬件功能。相应在操作系统设计时，采用Client/Server机制，以微内核(Microkernel)为核心、消息传送机制为软总线、各种应用服务器为软件子系统，可实现灵活的、高可伸缩的操作系统。

(3)可移植性强。硬件平台的迅猛发展，为实时应用提供丰富的运行平台(从单片机到DSP，从CISC结构到RISC结构的各类微处理器等)。硬件平台的多样化，要求开发RTOS时应考虑其有良好的可移植性，便于移植到各类硬件平台上运行，提高操作系统的可重用性。

(4)支持多处理器、多机及新型结构的硬件平台。现代实时应用复杂性的要求，决定了实时计算机系统的工作模式必然由监控系统下的单任务轮询，向单机多任务及多机多任务过渡，支持多处理器结构及未来新型结构的硬件平台也是研究与开发RTOS的重要内容。

(5)可靠性。实时应用背景特点(如军事应用、工业控制)，一般在条件恶劣的环境，要求作为系统软件的RTOS具有较强的可靠性，满足抗恶劣环境的需要。

(6)接口标准的开放性。任何产品的发展都有从百花齐放到标准化的过程，作为实时软件核心的操作系统也不例外。虽然目前国内外还未在嵌入式实时应用编程接口(API)上有统一的标准，但IEEE的实时UNIX标准POSIX对RTOS界面标准的指定有重要的参考价值。

传统嵌入式实时操作系统(ERTOS)设计采用一体化、高效大内核结构的方法，系统功能集中在内核实现，内核向应用提供高效率的系统调用及良好的系统响应时间。传统设计方法设计的ERTOS固然有其好的一面，但与现代ERTOS需求相比仍有较大差距。主要表现在操作系统的可伸缩性、可移植性、对新型结构的硬件平台的支持及接口标准的开放性上。大内核结构的系统设计对功能的扩展往往造成内核修改牵一发动全身。为满足现代实时操作系统需求，同时兼顾嵌入式应用的特点，在"九五"军事预研项目新型嵌入式实时操作系统研制中

*)国防科工委军事预研项目。徐仑峰 博士生。熊光泽 教授，博导。刘锦德 教授，博导。

我们采用基于超微内核的设计方法，研制高效、时间确定性强、可动态配置、可靠且满足部分 POSIX 标准的嵌入式实时运行环境。本文着重介绍“九五”项目新型嵌入式实时操作系统设计采用的结构和方法。

1 ERTOS 的可伸缩性设计

现代新型操作系统采用基于消息传送的微内核设计来实现系统的可伸缩性。微内核设计思想是将操作系统中不常用的系统调用移到核外服务器中实现，并且支持核外服务器的动态配置（加入/退出）。微内核与核外服务器之间的联系通过消息传送实现。微内核可使操作系统的功能扩展靠加入服务器来方便完成。

对核外服务器，当其加入系统时，操作系统在微内核中为其建立对应的 CHANNEL 对象；当应用请求该服务器服务时，调用发送后自阻的 MesgSend() 函数向服务器发送服务请求。它自陷入微内核与所请求服务的 CHANNEL 连接，并在服务器 CHANNEL 对象的接收消息队列(Receive Queue)中排队；如服务器在其 CHANNEL 中已有等待接收消息的线程，则被激活并将继承请求服务方的优先级来提供服务，同时将请求服务方的请求消息从 CHANNEL 对象的接受消息队列移到应答消息队列(Reply Queue)中；当服务器服务完成，根据应答消息队列中的消息，发送应答消息返回微内核中请求服务的应用，通知服务器工作已完成，微内核则唤醒因请求服务而挂起的应用程序，并将服务器 CHANNEL 对象中应答消息队列上相应的消息删除。请求服务、服务响应、及 CHANNEL 对象示意如下：

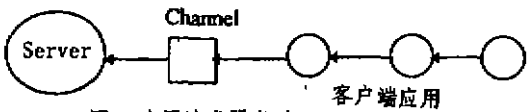

图 1 应用请求服务时

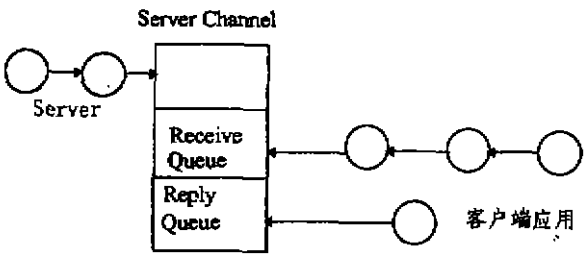

图 2 服务器 Channel 对象中的接受，应答队列

微内核的设计方法适合实时嵌入式操作系统的可伸缩性设计。

2 ERTOS 的实时性能设计

典型微内核设计的优点主要是为加强操作系统的多模块化设计。通过简化内核的结构和功能，组成微内核，在微内核中只保留最基本的功能，而将操作系统的大部分服务放在微内核外的各类服务器上实现。如 QNX 公司的 QNX4 RTOS，在其微内核内只有17个最基本的系统调用，处理 IPC、基本调度、底层中断处理等功能，而系统其它功能都在微内核外的进程管理服务器、文件服务器、设备管理服务器和网络管理服务器上实现。典型微内核结构设计对嵌入式强实时系统是不利的，几乎所有系统调用都是一个 RPC 过程，它包含应用到微内核、微内核到核外服务器、服务器工作完成返回微内核及唤醒请求应用等过程，这主要是过分强调内核功能外移到服务器所至。对 ERTOS 设计，在采用微内核结构大框架基础上，将实时应用常用到的系统调用（如实时执行体功能）仍在微内核中实现，即对微内核的成分进行调整，既保持微内核可伸缩性、支持系统多模块设计的优点，同时又提供较好的实时性能，这种成分调整不违背微内核设计的思想。

实时微内核结构的操作系统，实时性能很大程度同微核与核外服务器通信的效率有关。典型的微内核操作系统 MACH，在微内核与核外服务器间采用的是近似于标准 RPC 的消息传送，这已被证实为效率较低的方法。现国外已开发出高效的微核与服务器通信方式，其效率较 MACH 相比提高10—20倍。设计实现低开销的微核与服务器通信器是提高 ERTOS 实时性能的重点之一。ERTOS 微内核的结构如下图：

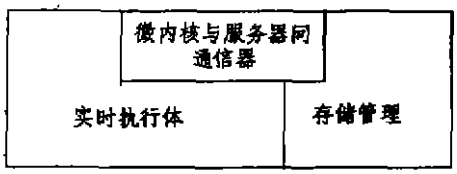

图3 微内核结构示意图

3 ERTOS 的可移植性设计

基于微内核结构的嵌入式实时操作系统，系统的可移植性主要体现在微内核的可移植上。从上述可知，微内核由实时执行体、存储管理和消息通信管理组成，分析其内部功能可分为以下三类：

·与微处理器特性有关的管理（如线程管理、中断管理、内存管理和系统初始化）；

·与BSP有关的管理(实时时钟管理、字符I/O管理);

·与硬件特性无关的管理(邮箱管理、信号量管理、异步通信管理、软时钟管理、微内核与服务器的消息通信及出错处理)。

对于线程间通信等与硬件无关的管理功能,只是对系统中的数据对象进行操作,它与操作系统的数据定义相关,而与微处理器的特性、外围电路组成无关,微内核的移植重点在另两类管理功能。

3.1 与微处理器有关的系统功能移植

·线程管理。通过引入纳核来完善微内核的可移植性(即微内核可以不经修改或很少修改就可从一处理器移植到另一类处理器上),微内核只完成管理的逻辑部分,硬件相关部分在纳核中实现。微内核线程管理功能包括线程的创建、删除与切换、挂起与恢复、优先级的设定与提取等,如功能操作与微处理器特性无关,则管理在微内核的线程对象中实现,否则需要纳核中线程对象的支持(如线程的创建、切换等就需要纳核的支持),在微内核线程对象中有一特定属性与纳核中对应线程对象关联。可见线程管理在不同微处理器上的移植只需修改纳核,保持纳核与微内核间调用接口的一致时,微内核不用做任何修改。

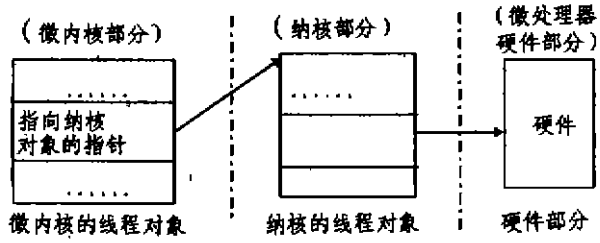

图4 线程对象示意

·中断管理。为缩短中断响应时间,当中断发生时,操作系统不进行任何干预和任务切换,直接进入中断处理程序,为防止中断嵌套时栈溢出,系统提供自动切换中断栈。在中断管理设计时,为与POSIX

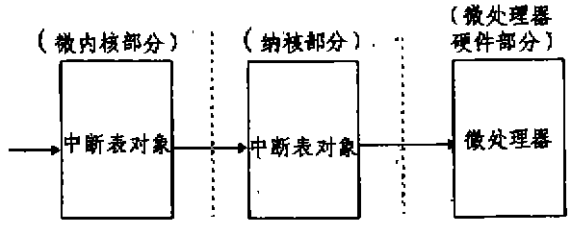

图5 中断对象示意图

界面一致并保证微内核可移植,在微内核中引入逻辑中断矢量表,通过InterruptDetach()系统调用动态为中断矢量设置对应中断处理程序,而实际则是调用纳核中中断对象相关操作来完成,纳核是与微处理特性相关的唯一层次。

·内存管理。大型操作系统如UNIX、WINDOWS NT、VMS、Solorias等都提供了多种存储管理模式,主要是利用微处理器内存管理单元(MMU)提供虚拟内存管理,实现任务间、任务与系统间的存储保护。对强实时应用,虚拟存储机制对时间确定性不利,且不是所有微处理器都有MMU功能,因此不采用虚拟存储管理。但为在应用代码与系统程序间提供适当保护,内存管理上利用微处理器段(Segment)的特点,采用分段的SPM保护模式(多数微处理器支持内存段管理模式),用户代码段与系统代码段位于不同访问优先级的段中,如系统代码段为0级,应用代码段为3级,高优先级的段代码可以访问低优先级的段,反之则报错。因为不使用虚拟内存管理,系统内线性地址即为实际物理内存地址。

系统在内存分配管理上采用动、静态分配结合的方法,对在实时应用中一直有效的数据结构,采取在系统配置时,提前在空间上预留的静态内存分配法,将系统中的系统变量、对象控制块、线程和中断处理程序的堆栈等在从低到高的内存空间中一次分配,这些空间只能被重指派而不能被释放。如一个线程被删除后,其控制块占有的内存空间并不释放,而将重新被其它新生成的线程使用。对于应用可能提出的不同大小空间申请,采用基于堆(heap)的动态内存分配法。因微处理器几乎都支持段式存储管理,且堆算法与硬件无关,所以存储管理直接在微内核实现。

3.2 板极软件支持包(BSP)的设计

对深嵌入、强实时应用,系统硬件多样性不仅表现在微处理器的差异,也可能体现在外围电路(如定时器、串/并行通信控制器等)的不同。外围硬件特性同微处理器的特性一样,也不宜在操作系统内固定不变。为此,引入硬件板极支持软件包(BSP)来支持操作系统在各类OEM板或用户自定制硬件系统板上的移植,BSP提供设备驱动程序及各种标准子程序,与BSP有关的微内核管理有定时器管理、字符I/O管理等。

·实时时钟管理。实时操作系统保留一32位数据结构,它与计数器硬件设备联系起来,为操作系统提供实时时钟,它由计数器的中断处理程序调用

ClockTick 系统调用处理。所以在操作系统的初始化中定义计数器硬件电路的初始化接口，而在 BSP 中根据接口实现不同类型外围设备的初始化程序和驱动程序。

·字符 I/O 管理。本嵌入式实时操作系统设计，没有设计设备管理服务器，但在微内核提供字符 I/O 管理。字符 I/O 管理在微内核建立一个128字符的 FIFO 输入缓冲区和输出缓冲区，允许线程和 ISR 配合起来完成字符 I/O 的操作，线程使用 IoWrite 调用将一字符放于输出缓冲区，如缓冲区满，则挂起线程，直到缓冲区有空为止；线程使用 IoRerad 调用能从输入缓冲区中得到一字符，当缓冲区为空时，挂起线程直到缓冲区有字符；IoWaitRead 调用挂起线程直到接受一指定字符。ISR 使用 IsrWrite 和 IsrRead 调用，一次传送一个字符到操作系统输入缓冲区或一次从操作系统的输出缓冲区中传出一个字符。对于 IsrWrite 和 IsrRead 都在 BSP 中实现，以满足不同输入/输出硬件设备的需要。

4 ERTOS 的应用编程接口标准化设计

在 ERTOS 的微内核中直接实现了大部分的实时及线程服务，为满足开放性的需要，在定义服务原语时，尽量向已有的国际标准或工业标准靠拢，在设计时，遵循 posix 1003. 1c(线程服务)、1003. 1/a/b(时钟服务)、1003. 1b(定时器服务)、1003. 1d draft standard(中断服务)、1003. 1b(调度管理)及1003. 1c 标准(解决优先级反转的互斥信号量)。

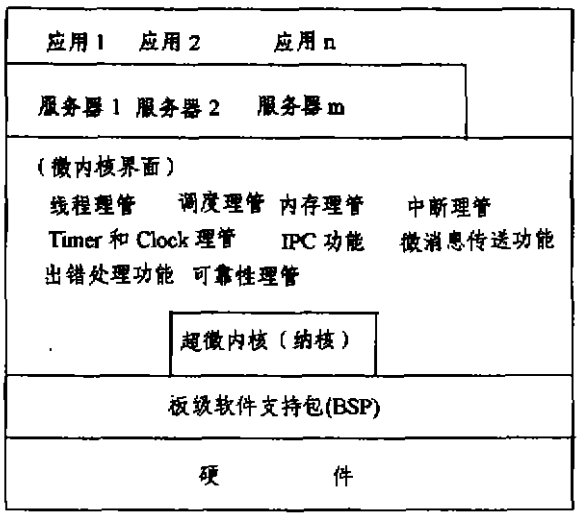

图6 ERTOS 系统结构示意图

5 "九五"基于超微内核的实时操作系统(ERTOS)结构

ERTOS 设计采用微内核结构和 Client/Server 模式，利用微内核固有的特点加强系统的可伸缩性、可剪裁性、可靠性及代码的可重用性，采用面向对象的设计技术、独立的超微内核(纳核)可移植界面及板级软件支持包(BSP)来提高 ERTOS 的可移植性，并在"八五"军事预研成果 CRTOS/386P 的基础上，加强并完善实时性能相关的算法和对象设计，ERTOS 的总体结构框架如图6所示，

ERTOS 结构从上到下有四个层次，最上一层是实时应用及 ERTOS 的核外服务器，目前服务器设计上只提供基于 TCP/IP 协议的网络管理服务器，以支持基于网络连接的松耦合多机系统；第二层是微内核界面及其可移植的纳核层，它直接在微内核内为实时应用提供线程管理、线程间低级的通信服务、微内核与服务器间的通信管理、定时器和时钟管理、中断管理、内存管理、字符 I/O 和可靠性服务，并提供与硬件特性无关的纳核层可移植界面，纳核本身与微处理器的特性有关，纳核与微内核设计是整个系统设计的核心；BSP 层提供屏蔽硬件板级特性(如微处理器的外围电路)的功能，便于 ERTOS 在不同外围电路组合的硬件平台上实现移植。

参 考 文 献

[1]孟庆余，电子数字计算机实时操作系统，国防工业出版社
[2]The Neutrino MicroKernel, http://www.qnx.com
[3]VRTX Real-Time Operating System, http://www.mci.com
[4]Michel Gien, Micro-Kernel Architecture Key to Modern Operating System Design, http://www.chorus.com
[5]Murc Guillemont, MicroKernel Design Yields Real Time in a Distributed Environment, http://www.chorus.com
[6]Michel Gien, Next Generation Operating Sysem Architecture, http://www.chorus.com
[7]CRTOS/386P 概要设计，电子科大微机所
[8]CRTOS/386P 详细设计，电子科大微机所