第29卷 第10期 计 算 机 工 程 2003年6月
Vol.29 № 10 Computer Engineering June 2003
· 软件技术与数据库 · 文章编号：1000—3428(2003)10 —0089—03 文献标识码：A 中图分类号： TP316

实时嵌入式多处理器操作系统实现框架

艾定军，徐家祥，蒋祥刚，张 激
(华东计算技术研究所，上海 200233)

摘 要：描述了针对实时嵌入式应用的多处理器操作系统的实现框架，简单介绍了常用的多处理器系统所采用的架构，提出了另外一种多处理器系统的实现架构，并讨论了与之相关的通信、全局资源管理、调度、cache一致性和异构性的问题。

关键词：多处理器；对称多处理；非对称多处理；共享内存；消息传递；多处理器通信接口层

Implementation Framework of Multiprocessor Operating System for Real-time Embedded System

AI Dingjun，XU Jiaxiang，JIANG Xianggang，ZHAN Ji
(East China Research Institute of Computer Technology,Shanghai 200233)

【**Abstract**】The paper describes the framework of multiprocessor operating system for real-time embedded application, and describes a usual framework of multiprocessor system. It provides another implementation framework of multiprocessor system,and discusses some related problems, such as communication,global resources manager,scheduling,cache coherent, and heterogeneous processor.

【**Key words**】Multiprocessor；SMP；AMP；Shared-memory；Message communication；MPCI

传统的实时嵌入式系统是基于单片机的，20世纪70年代末微处理器的出现使得汽车、家电、工业机器、通信装置等产品可以通过内嵌实时电子装置来获得更佳的使用性能。从80年代早期开始，实时嵌入式系统的程序员开始用商业级的操作系统编写实时嵌入式应用软件。随着实时嵌入式应用领域及其覆盖范围的不断扩大，从低端到高端各种不同的应用需求的增多，实时多任务操作系统(RTOS)逐渐发展成熟，它包含了许多传统操作系统的特征，包括任务管理、任务间通信、同步与互斥、中断支持、内存管理等功能。目前，RTOS逐步成为目前国际嵌入式系统的主流，同时实时嵌入式计算机系统也开始支持多机多任务。

1 多处理器系统常用架构

1.1 SMP(Symmetric Multiprocessing)

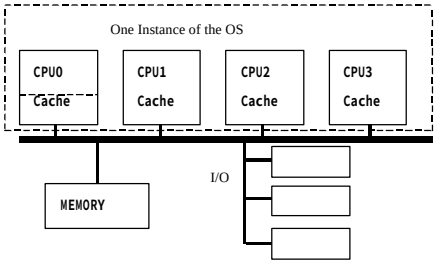

图1 对称多处理器架构图

对称多处理，是指在一个计算机上汇集了一组处理器，各CPU之间共享内存子系统以及总线结构。在这种架构中， 多个处理器运行操作系统的单一复本，并共享内存和一台计算机的其他资源。所有的处理器都可以平等地访问内存、 I/O和外部中断。系统资源被所有CPU共享，工作负载能够均匀地分配到所有可用处理器之上，系统将任务队列对称地分布于多个CPU之上，从而极大地提高了整个系统的数据处理能力，其结构如图1。在目前大多数SMP系统中，中央处理器是通过共享总线来存取数据的，因此随着加入的CPU数目的增加，将会导致总线瓶颈问题，为了能够使SMP技术支持更多的中央处理器，需要投入大量的资金和时间对处理器、处理器的总线和主板进行设计。

1.2 AMP(Asymmetric Multiprocessing)

非对称多处理器系统由主处理器和从处理器组成，主处理器能运行操作系统，是系统的核心和灵魂，从处理器完成由用户定义的指定功能。在这种架构中，资源和任务由不同的微处理器进行管理，与操作系统核心有关的操作由主处理器完成，包括系统调用、I/O控制、任务调度等，而从处理器则完成指定的用户任务。非对称多处理的每个处理器的工作负载不一样，有可能出现一个处理器负载很重，而同时另外一个处理器空闲，不像对称多处理器系统能进行负载平衡。而且整个系统的容错性很差，如果一个处理器发生故障则整个系统都无法正常工作。目前这种架构的多处理器系统已经很少见了。

2 实时嵌入式的多处理器系统

2.1 实现框架

2.1.1 系统总体框架

为提高整个系统的灵活配置性并提高系统的可靠性，我们在实现实时嵌入式多处理器系统时，采用共享内存与消息传递两种通信方式混合的架构。只有单个操作系统运行在所有的节点上，但并不是说所有的节点共享一个内核代码和数据结构的复本，而是每个节点运行自己的内核代码和全局数据结构的复本，这些全局的数据结构必须与底层的内核实现保持一致。图2显示了整个系统的框架。处理器0与处理器1之间可以通过共享内存的方式进行通信，而处理器1与处理器2之间可以通过消息传递的方式进行通信。在这种系统中，每个处理器都能够访问所有的内存，但是，访问本地内存明显要比访问远程内存快得多。

作者简介：艾定军(1977－)，男，硕士生，主研实时嵌入式操作系统、实时中间件；徐家祥，硕士生；蒋祥刚，工程师；张 激，研究员
收稿日期：2002-06-11

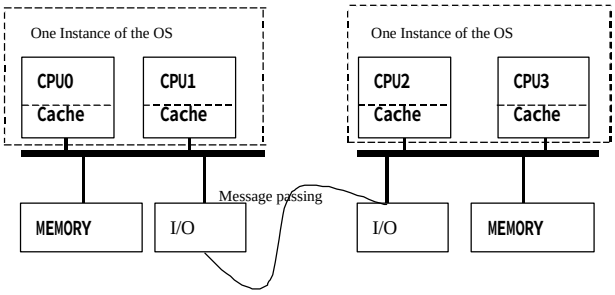

图2 本文多处理器架构图

2.1.2 **操作系统视图**

为支持多处理器的架构，必须对实时嵌入式操作系统作一定的修改。对于消息、信号、信号量、事件、任务、区域和分区等模块，都必须添加相应的支持多处理器系统的功能，目前大多数实时嵌入式操作系统都支持多处理器的架构，也都提供了各个模块对多处理器的支持。我们在操作系统与硬件之间加入MPCI层(Multiprocessor Communication Interface Layer)，与多处理器相关的任务、消息、事件、信号量、信号、区域和分区等组件。在进行与多处理器相关的处理时，调用MPCI所提供的功能进行通信。MPCI管理着一些数据包，并且在各个节点之间发送和接收这些数据包。主要有以下几个接口MPCI_Initialization、Get_Packet、Return_Packet、Send_Packet、Receive_Packet。而具体的实现方式，可以选择基于共享内存和基于消息传递的方式。在需要进行通信时，系统将根据用户多处理器通信接口层提供的入口点，调用相应的程序，发送和接收数据。而用户感觉到的是两个节点的MPCI层在进行通信，完全屏蔽了下层的实现方式与硬件连接。整个操作系统的视图如图3。

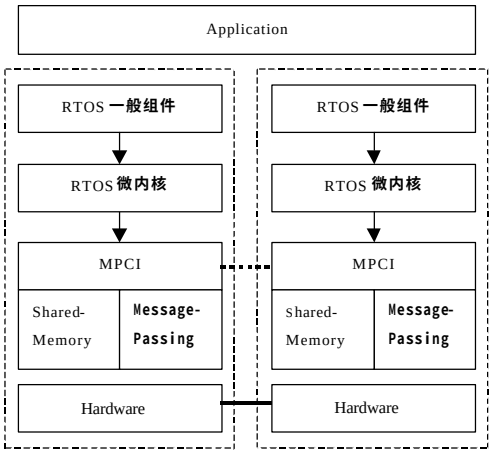

图3 操作系统视图

2.2 节点间通信方式

2.2.1 **基于共享内存的访问方式**

在这种访问下，处理器通过共享的存储器进行通信。MPCI 能简单地将数据包放在共享的内存空间。此时设计 MPCI 时，主要需要考虑对共享内存访问的互斥性和如何将数据包通知其它节点。

共享内存的互斥性问题是指多个处理器共同访问内存中的数据时，并不能同时去读写数据，可以通过在共享的内存中提供一把“锁”来解决。为满足实时系统的要求，需要着重解决如何将处理器锁定共享数据结构的时间减到最小。

对于将数据包通知给其它节点的问题，通过产生一个异步信号signal的方式，通知接收节点有数据到达，然后接收节点调用MPCI层提供的Receive_Packet来接收数据包。

2.2.2 **基于消息传递的访问方式**

在这种访问方式中，处理器通过连接介质(比如局域网)来进行通信，MPCI 跨过通信连接发送数据包到目的节点。此时主要考虑的是连接介质的选择和数据包到达通知。

网络通信连接的介质有很大的变化，对MPCI层有重要的影响。例如,通信连接的带宽对于最大的MPCI 通过量有明显的影响。采用光纤作为连接介质的ATM(Asynchronous Transfer Mode)网络延迟小、传输质量有保障，速率可高达上千兆。

对于数据包到达通知，如果硬件支持,则数据包到达的时候，会在目标节点上产生一个中断。否则,实时时钟ISR (Interrupt Service Routine) 能检查数据包的到达，然后调用MPCI层提供的功能来完成数据的接收。

2.3 全局资源管理

我们采用面向对象的管理机制，将任务、信号、信号量、消息等都看成一个对象。每个对象都拥有一个32位的对象ID，由3部分组成：ID的高六位确定对象类型，中间十位确定对象所在节点，若只有一个CPU时，则节点始终为1(0用于广播)，低十六位用来确定本节点中的同类对象。而且对于每一个对象，都有一个对象控制块，在对象控制块中，有一个属性(GLOBAL)是用来设置该对象是否为全局的。在每个节点上，都配置有两张表，本地对象表和全局对象表。

(1)本地对象与本地对象表

若GLOBAL属性为FALSE，则该对象为本地对象，它只能被本节点所访问。本地对象表维护着由本节点创建的所有对象的信息，而不论其属性是本地的还是全局的，每个节点上的本地对象表的内容是不相同的。

(2)全局对象与全局对象表

若GLOBAL属性为TRUE，则该对象是全局对象，全局对象可以被所有的结点访问，当然必须有一定的限制，我们可以修改远程对象的属性，但不能够删除远程对象。每个节点上的全局对象表必须保持完全一致，每个节点创建或删除该节点所创建的全局对象都必须明确地通知给其他所有节点。特别要说明的是，每个节点创建的全局对象不仅出现在该节点的本地对象表中，也出现在其全局对象表中。

(3) 代理

当进行远程操作的时候，比如信号量的获取或者消息队列的接收的时候，需要采用代理。它驻留在远端节点上，代表由于远程操作必须阻塞的本地任务。因为如果要阻塞的任务是本地的，可以直接修改其任务控制块，以表明它阻塞在一消息队列或信号量上。然而，任务控制块只可能和任务本身驻留在同一个节点上，所以，远程节点必须有一个代理来代表该任务，直到被重新调度。

2.4 任务调度策略

2.4.1 **一般调度策略**

实时操作系统中的任务调度策略目前使用最广的主要可分为两种:静态表驱动方式是指在系统运行前，根据各任务的实时要求用手工生成任务的运行时间表；优先级抢先式调度方式则与通用操作系统中采用的基于优先级的调度方式基本类似，我们采用了以下几种调度算法：

(1)抢占调度(Preemption)：如果有更高优先级的任务准备就绪，并且当前任务是可抢占的，则当前任务将被更高优先级的任务抢占；

(2)时间片(Timeslicing)或 轮转(Round-Robin)调度：对于具有相同优先级的任务，并且任务的时间片设置是可用的，则每个任务运行轮流相同的一段时间片；

(3)手工轮转调度(Manual Round-Robin)：这种调度方式也是针

对具有相同优先级的任务，由调用Task_wake_after或Yield_processor等系统函数时产生，当前任务立即放弃处理器，并将该任务放到具有相同优先级的任务链的末端。

而对于周期性任务，可以采用两种常见算法：

(1)单调率算法RMS(Rate Monotonic Scheduling Algorithm)：高频率任务分配高优先级，低频率任务分配低优先级，该算法事先为每个进程分配一个与事件发生频率成正比的优先级；

(2)最早期限优先算法EDF(Earliest-Deadline-First Algorithm)：该算法以任务截止期限作为优先级，截止期限越早任务优先级越高，当任务创建后，被加到就绪队列中，该队列按照截止期限排序。

2.4.2 处理器亲近调度策略

对于每个节点上任务的调度，我们加入了处理器亲近调度策略。在多处理器系统中，一个任务可以在任何一个处理器上运行，然而任务在各种情况下会被中断，被中断的任务在恢复执行的时候，如果选择恢复在另外一个处理器上执行，就会导致它失去原有的处理器cache数据。访问cache数据只需要几个纳秒(nanosecond)，而访问主存需要大约50纳秒。这时处理器运行的速度处在访问主存的级别上，直到任务运行了足够的时间，任务运行所需要的数据重新充满该处理器的cache为止。为解决这个问题，我们记录下最后执行这个任务的处理器并维持这种关系，在恢复执行被中断的任务时，尽量恢复在最后执行这个任务的处理器上执行，这就是处理器亲近调度策略。

2.5 cache一致性问题

在多处理器系统中，各CPU通过cache访问内存数据时，要求系统必须经常保持内存中的数据与cache中的数据一致，若cache的内容更新了，内存中的内容也应该相应更新，否则就会影响系统数据的一致性。 对于实时嵌入式应用来说，可以将它分为低端应用和高端应用。低端应用可能本身不支持cache，也就不存在cache的一致性问题。对于高端应用来说，可以通过确保在任何时候，共享数据的缓冲复本不存在于两个或两个以上的缓存中。可以通过互斥机制使得共享数据的访问串行化，在互斥量被释放的时候，清洗并且使缓冲的全局数据结构无效。

2.6 系统的异构性问题

在一个多处理器系统中，异构问题是必须面临的问题，此时设计MPCI层时需要作出特殊的考虑。首先，不同的处理器类型使用的不同数据表示，即组成一个数据实体的字节次序。Little endian处理器把最低有效字节放在最小的地址；而Big endian处理器把最高有效字节放在最小的地址。Little endian 处理器和big endian 处理器之间共享数据结构需要转换成公共的格式。可以将所有的数据看成32位的无符号的数据来处理，并且数据包在本地使用本地的endian格式，endian的转换可以由发送该数据包的MPCI层来处理或者由接收方来处理。

其次，要考虑数据结构元素的对齐，某些处理器允许数据元素在任何地址边界开始，而有些处理器强加地址边界限制。通常的限制是数据元素必须在偶地址或在一个长字边界上开始。我们在实现的时候，限制所有的数据包必须在一个4字节的边界上开始。

3 结束语

多处理器系统是计算机界比较关注的问题，实时嵌入式应用的迅猛发展，使得实时嵌入式的操作系统的开发越发显得必要，而支持多处理器系统的实时嵌入式操作系统必然具有广阔的前景。

参考文献

1 Bach M J.陈葆钰译.UNIX操作系统设计.北京:机械工业出版社,2000

2 Cellular Multiprocessing and Uniform Memory Access.Unisys Corporation,1999

3 Stallings W.Operating System Internals and Design Principles [third Edition].北京清华大学出版社,1998

4 RTEMS C User's Guide for RTEMS 4.5.0 [Edition 1].On-line Applications Research Corporation,2000

☆☆

(上接第37页)

了；(2)MAAC将拥塞反应限制在更少的端点。拥塞解除得快是因为邻近的路由器立即减少了输入负载，使得路由器C能够处理当前的数据包并减少队列长度。邻近的路由器会限制第一批看到路由器C发生拥塞的端点发出的数据包的数量。而在传统的系统中，第一批看到拥塞的端点并不会立即减少它们的发送率，从而导致其他的端点丢包。并减少它们的发送率。造成减少数据包发送率的端点比前者多得多。

这样，由于采用了MAAC的系统中减少发送率的端点较少，因此整个系统发生的摆动就小，这样有助于改善整个系统的吞吐量。

5 结束语

主动网络通过为网络节点赋予计算处理能力而使得网络体系结构发生了深刻的变化。利用这种可计算性，将移动代理应用到网络的拥塞控制中，提出了一种基于移动代理的拥塞控制机制MACC，解决了过去无法实时控制网络内部节点上的行为，大多数拥塞控制技术只能在端点系统上寻求解决方案的问题。可以广泛用于各种网络应用中，比如视频传输的拥塞控制等。

参考文献

1 Tennenhouse D L,Wetherall D.Towards an Active Network Architecture[M].In Proc. of Multimedia Computing and Networking 96, San Jose,CA,1996-01

2 Calvert K L.Architecture Framework for Active Network.http://www. dcs.uky.edu/~calvert/arch-docs.html,1999

3 Calvert K L,Bhattacharjee S,Zegura E W,et al.Directions in Active Networks.IEEE Communication Magazine,1998,36(1):72-78

4 Wetherall D J,Tennenhouse D L.The Active IP Option. Proceedings of the 7th ACM SIGOPS European Workshop,Connemara,Ireland,1996 - 09

5 陆月明,钱德沛,徐 斌等.Softnet——一个基于移动代理的主动网络.计算机学报,2001,24(11)

6 Bradshaw J M.An Introduction to Software Agents.,In:Bradshaw J M (Editor).Software Agents,Chapter 1,AAAAI Press/The MIT Press,1997

7 Jansen W,Mell P,Karygiannis T,et al.Applying Mobile Agents to Intrusion Detection and Response.IST Interim Report(IR)-6416,1999-10:2,5-7,10-14

8 朱向华,万 燕,孙永强移动代理系统的安全机制计算机工程,2001,27(1):137