实时分布式操作系统中共享对象通信模型的实现

刘谋用 葛霁光
(浙江大学生物医学信息研究所 杭州 310027)

摘 要 与消息传递、远程过程调用和共享内存等进程间通信模型相比,在分布式存储多处理器上实现的共享对象模型能更自然、更高效地描述进程间的相互作用.它通过共享对象消除了各结点之间的物理边界,为系统提供了一种透明的分布式处理能力.这样,整个系统(包括硬件和软件)在逻辑上成为一个单系统.该模型已在仪器用弱实时分布式操作系统(IOWRTDOS)的全局共享对象(GSO)层上实现.文中研究了共享对象通信模型,并详细讨论了实现中的关键问题.

关键词 分布式计算系统,实时分布式操作系统,通信模型,线程,对象

中图法分类号 TP301

THE IMPLEMENTATION OF A SHARED OBJECT COMMUNICATION MODEL IN REAL-TIME DISTRIBUTED OPERATING SYSTEMS

LIU Mou-Yong and GE Ji-Guang
(*Institute of Biological and Medical Information, Zhejiang University, Hangzhou* 310027)

Abstract Compared with the interprocess communication model of message passing, remote procedure call, and shared memory, a shared object model realized on a distributed memory multiprocessor can describe the interaction of processes more naturally and efficiently. It transcends the physical boundaries between the processor nodes by making use of a few global shared objects and presents the system with a transparent distributed processing capability. Hence, the entire system, both hardware and software, can be viewed logically as a single system. This model has been implemented on the globally shared object (GSO) layer of the instrument-oriented weak real-time distributed operating system (IOWRTDOS). The shared object communication model is explored and several key issues of implementation are disussed in detail.

Key words distributed computing system, real-time distributed operating system, communication model, thread, object

1 引 言

80年代中期以来,随着微线工艺技术的不断完善和进展,Moore定律得到广泛证实和应用,低成本、高性能微处理器与先进通信技术的不断出现,计算机系统正经历着一场从中心化系统向分布式系统过渡的革命.分布式计算系统(DCS)由于具有性能价格比高、可靠性好、增量可扩充性强以及满足内在分布式应用需求等优点,自从出现以来取得了迅速的发展[1].但近年来,DCS的发展遇到了很大阻力,其中之一是缺乏良好

原稿收到日期: 1998-06-04;修改稿收到日期: 1998-12-02.刘谋用,男,1969年 1月生,博士研究生,主要研究方向为实时分布式操作系统和医疗仪器.葛霁光,男,1934年 12月生,教授,博士生导师,主要研究方向为生物医学信息处理和实时诊断等.

的软件支持环境和程序设计工具.

系统的通信机制是影响 DCS可用性与性能的一个重要因素,为了提供高效、方便的通信模型,人们付出了巨大的努力. DCS刚出现时,最直接的一种方法就是采用显式的消息发送与接收[2],但消息传递的通信方法不易被程序员使用,也不利于已有应用软件的移植和重用.为了解决消息传递的缺点,人们提出了远程过程调用(RPC)的概念[3], RPC的基本思想是允许程序像调用本地过程一样调用远程机器上的过程,由于 RPC屏蔽了客户和服务器之间直接进行的报文发送和接收,并且还具有语义清楚、简单、容易使用、通信效率高及通用性好等优点,因此已广泛地用作 DCS的基本通信技术[4].

以上两种方法的主要缺点是程序员需要过多考虑底层的通信细节而非所需解决的问题本身. 80年代中,人们又提出了共享虚拟存储系统(SVM)的概念[5],即在分布存储多计算机上实现物理上分布但逻辑上共享的存储系统. SVM提供了一种与传统的串行程序类似的共享存储器通信模型,具有编程容易、可扩展性好、系统价格低、较好的软件可移植性等特点,因此受到了越来越广泛的重视.但 SVM共享的数据粒度为页,容易引起伪共享的产生,具有较差的性能,因此人们又在寻找更高一级抽象的通信模型.

面向对象技术是一种面向数据流,并集模块化、数据抽象、信息隐蔽和消息传递等诸多优点于一体,既适合于系统分析又适合于程序设计的工程技术.它利用"对象"概念,将数据与在其上的算法逻辑操作封装在一起,通过消息传递来完成对象之间的联系.这种技术设计出来的系统具有可靠性高,可维护性好,系统部件再用性强,系统结构灵活,剪裁方便等特点,并且对象具有并行性,因此面向对象技术非常适合于 DCS的构造.共享对象模型就是希望在实时分布式操作系统中提供一种更为自然、更为高效的进程间通信模型.

2 IOWRTDOS

现代仪器系统尤其是现代化新型医疗仪器和医疗器械系统正朝着多功能、多参数、智能化、网络化方向发展[6].这种发展趋势迫切需要一种高性能的软、硬件支撑平台.由于这类系统或者计算量大,或者需要推理功能,或者有严格的时间要求,或者需要极高的可靠性,或者物理上是分布的,只有分布式并行处理系统才能满足这种要求.相应的硬件需要相应的软件来支持.操作系统是联系用户与系统硬件的桥梁,是现代仪器系统的关键技术.现有的操作系统如 DOS, Windows98和 Unix等均不能满足现代中高档仪器系统快速实时处理的需要,因此提供一种支持分布式并行计算、满足实时性能的全新仪器用操作系统有着重大的理论与应用价值,它将极大地促进现代仪器系统的发展.

我们研制的是一个以现代智能仪器系统为应用领域,以高性能的硬件结构为基础,利用先进的操作系统逻辑构造技术,借助于成熟的软件工程原理,能根据用户特定应用需求而自由定制的弱实时分布式操作系统(instrument-oriented weak real-time distributed operating system,简称 IOWRTDOS).其总体结构如图 1所示.

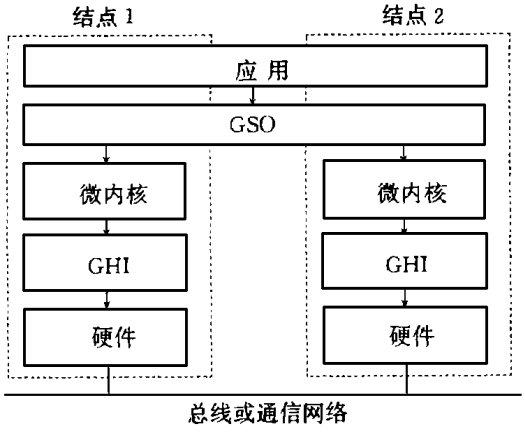

图 1 IOWRTDOS总体结构

这是一个综合了微内核模型与分层模型的混合结构,主要由 3层组成:通用硬件接口(GHI)层,微内核(microkernel)层和全局共享对象(GSO)层. GHI主要完成一些依赖于硬件的功能,如系统引导初始化、处理器调度(线程上下文切换)、中断处理、多处理支持(锁、机间中断等)和致命错误处理等.它屏蔽了硬件细节,为操作系统的其余部分提供了一个理想的机器结构.这样上层的微内核部分与机器硬件完全隔离开来,增强了系统的可移植性.微内核是整个操作系统的核心,主要完成多线程管理、内存管理、任务间通信和中断管理等基本功能.它只包括一些实现机制,尽量不包括分配策略,这样随着时间的推移,策略的改变,微内核部分相对保持稳定,增强了系统的灵活性. GSO通过任务、内存块、信号量等全局共享对象为系统提供了一种透明的分布式处理能力,也为应用程序设计者提供了一种易于使

用的、通过共享对象进行通信的分布式程序设计方法.

IOWRTDOS采用了现代计算机系统与仪器系统的诸多先进技术,如平面(flat)内存模型、基于线程的细粒度并发执行模型、完全抢先式内核、确定的中断响应时间、透明的分布式处理等.由于在总体结构上采用微内核构造技术,在系统分析与设计过程中采用面向对象技术,因此系统具有很好的模块性、可移植性和高性能价格比,并能根据用户需求而任意组合以适应于特定应用领域.

3 共享对象通信模型

3.1 线程

线程是 IOWRTDOS的主动实体,除拥有自己的堆栈、寄存器及优先级等少数私有资源外,共享其所属进程的资源.因此线程是比进程更细粒度的并发执行单位.多线程结构就是在单个进程中包含并发执行的多个线程,具有吞吐率高、处理器上下文切换快、程序设计结构清晰等优点.对多 CPU系统,每一个线程可以在不同的 CPU上运行,多线程结构是应用程序使用硬件并行性的最佳解决方案.

线程可以动态创建或删除,但不具父子关系,即系统中所有线程是平等的,独立地处理各自的请求.线程的执行场点由程序员指定,并且不能迁移.线程通过静态优先级方法与循环轮转方法相结合的完全抢先式调度算法来控制 CPU的执行.线程之间共享资源,通过互斥锁(mutex)与信号量(semaphore)等实体来同步,而通过信箱(mailbox)来进行通信.

3.2 对象

在 IOWRTDOS中,所有的实体(如内存块、信号量、信箱等)均是作为对象来管理的.每一个对象属于一个特定的类型,一个类型描述了具有相同特征的对象集合.类型同样描述了对象的数据结构与操作,用户只能通过操作来访问对象.

IOWRTDOS的对象模型模仿的是文件模型,即具有以下特点:① 读写操作前必须先打开;② 打开后返回一个 ID号,用于与打开对象有关的下一步操作,当使用完对象时,关闭对象;③ 当两个线程同时打开到一个对象 ID号时,它们共享这个对象.

面向对象技术是以消息机制为基础的,如果采用纯对象技术会影响系统性能.因此我们只采用面向对象的一部分思想,对象的管理基本上采用传统的方法,对象之间通过成员函数的调用来相互通信.

3.3 共享对象通信模型

IOWRTDOS采用单任务多线程并发执行模型,即单个应用程序可以划分为几个独立的任务,每个处理器结点上分配一个任务.但每个任务可以按照需要创建多个线程,分布在多个处理器上的多个线程相互作用,共同完成某一应用.线程之间通过共享的对象进行通信,这些对象可以是局部或全局的.局部对象只能由本地结点上的线程访问,一个结点上所有局部对象形成了一个局部对象空间.全局对象可以被任意结点上的线程访问,系统内所有的全局对象形成了一个全局对象空间.对象是局部的还是全局的由程序设计者决定,进程可以访问任意的共享对象,而不管这些线程与对象驻留在哪儿,由 IOWRTDOS完成对象的定位与管理.通过线程与这些对象的相互作用形成了一个共享对象通信模型(见图 2),为程序设计者提供了一个统一的逻辑视图.

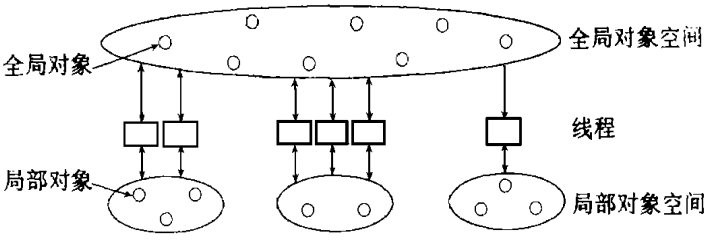

图 2 共享对象通信模型

4 共享对象通信模型的实现

仪器系统有着严格的时间要求,IOWRTDOS的一个主要目标是保证系统的实时性能.但如何为应用程

序设计者提供一个易于使用的环境也是我们的目标之一.因此我们在 IOWRTDOS的全局共享对象(GSO)层上实现了共享对象通信模型.

4.1 内部结构

GSO层主要由对象管理器(OM)、虚拟过程接口(VPI)、本地过程调用(LPC)、远程过程调用(RPC)等模块组成,它们的关系如图 3所示.

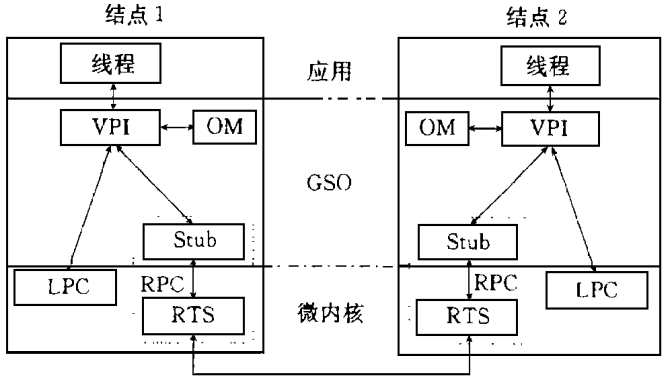

图 3 全局共享对象层内部结构

OM模块主要完成对象的创建、删除、定位等管理工作,并为系统提供名字服务.VPI主要是给本地过程调用与远程过程调用提供一个统一的接口,使得系统在逻辑上成为一个整体.RPC由两部分组成,存根(Stub)部分主要完成参数或结果的消息包装.远程传输子系统(RTS)主要是在结点之间可靠地传递消息.

4.2 对象命名与定位

IOWRTDOS中的每一个对象均有两个名字:用户提供的符号名与系统提供的唯一 ID号.通过名字服务,就可以从对象的符号名得到对象 ID号.对象 ID号包含有对象类型、结点号等信息,因此通过 ID号就能快速定位对象.IOWRTDOS采用分布式名字服务方案,在每一个结点上都维护两个表格:局部对象表和全局对象表.局部对象表包含所有在该结点上创建的对象(不论是局部的还是全局的),全局对象表包含系统中所有的全局对象.因为对象表需要频繁地访问,这种组织方法可以加快名字服务的速度,并且使系统具有一定的容错能力,但相应地也增加了额外的通信开销.为了维护数据的一致性,在全局对象创建或删除时,必须向系统广播消息,以使全局对象表保持一致.

4.3 对象访问

在 IOWRTDOS中,不管是本地对象还是远程对象,都通过统一的接口来访问.VPI的一个主要作用是为局部对象访问和远程对象访问提供一个统一的接口,使程序员根本不用关心对象的分布,即达到一种全局范围内的位置透明性.VPI引进了一定的额外开销.但为了提供更方便的系统支持环境,这一点开销还是值得的.

通过查找对象表可以得到对象的 ID号,通过 ID号便可以定位对象.若是本地对象则直接调用相应的过程,返回相应的结果;若是远程对象则调用相应的 Stub,Stub将参数和有关被引用的变量包装成一消息,通过 RTS发送到远程结点,然后挂起等待来自远程结点的应答.远程结点在接收到客户的请求后,调用相应的 Stub,将消息拆包,通过 VPI调用在该结点上的本地过程.处理完后再将结果返回给 Stub,Stub将结果包装成一消息发送回请求结点,本地线程即可继续执行.

频繁地对远程对象进行访问将极大地增加系统的通信开销.虽然对象的复制可在一定程度上降低系统的开销,但是当对象的状态需要经常性变化时,维持分布式对象一致性的开销将在很大程度上降低并行系统的潜在性能加速比.为了简单起见,IOWRTDOS中的对象是不复制的.

4.4 并发控制

并发控制就是控制远程请求的服务方式,可以采用多种策略处理到来的远程请求.一种是单线程方法,即就在通信守护线程内执行,这种方法只能以顺序方式一个一个地执行远程请求;另一种是多线程方法,即

有多个处于活动状态的通信守护线程,可以同时运行多个请求.还有一种动态线程方法,即通信守护线程根据情况对到来的远程请求创建一个新的线程来执行.

因为线程创建及上下文切换的开销是很大的,出于性能上的考虑,IOW RTDOS采用单线程执行模型.但远程对象的访问可能因竞争临界资源而阻塞,如果阻塞通信守护线程,会使得其它远程请求得不到立即响应.这个问题可以用代理(proxy)[7]机制来解决,代理是远程线程的本地表达,它与线程有类似的数据结构(只是要简单得多,只包含一些链接域、状态信息等).当远程请求服务因竞争某种临界资源而阻塞时,则申请一个代理放入等待队列,而通信守护线程立即返回以处理其它的远程请求.当阻塞条件解除时,该代理便会激活,处理余下的服务.这种代理机制既能满足性能上的要求,又具有足够的灵活性.

5 相关工作比较

基于对象的模型已经大量应用于分布式系统中,如 Amber[8], COOL[9], Orca[10]等. COOL是分布式操作系统 Chorus微内核之上的一个面向对象子系统,它的主要目标是消除粗粒度的系统对象与细粒度的语言对象之间的不匹配现象,为分布式应用提供更好的系统级支持. Orca是一种类型安全的语言,其运行系统的主要目标是隐藏硬件的物理分布性,提供有效的逻辑共享数据方法. GSO, COOL与 Orca运行系统有许多相似之处,它们之间的异同见表 1.

表 1 GSO, COOL与 Orca运行系统之间的比较

	GSO	COOL	Orca运行系统
实现层次	IOW RTDOS操作系统内部	Chorus操作系统内部	Amoeba操作系统之上
远程访问支持	RPC	SVM或 RPC	可靠的组通信
同步控制	程序员显式控制	程序员显式控制	集成于对象内部
对象状态	单拷贝	单拷贝	完全复制
负载平衡方法	无	线程迁移	数据对象迁移
程序设计环境	在传统的顺序语言中增加并行与通信原语	扩充传统的顺序语言	全新的类型安全的语言

IOW RTDOS面向的是仪器系统和控制系统应用,GSO的目标是为程序员提供一种基于共享对象进行通信的分布式程序设计模型.这些特点决定了在 GSO的设计和实现过程中始终坚持性能第一、简单实用的指导原则,因此与 COOL, Orca运行系统相比, GSO要简单得多,并能最大限度地满足系统的实时性能.

6 性能评价

众所周知,操作系统研制周期较长,国际上流行首先在通用仿真环境下测试软硬件模块功能和初型的性能,在此基础上实行整体联调.按国际上的经验,凡通过仿真性能测试后的软硬件系统联调时,修改工作量极小,这是国际上极为流行的方法.本文已通过仿真性能测试,随着仪器用实时分布式系统硬件平台研制工作进展,将逐步开展系统联调.本文在 Linux下构造了一个硬件仿真环境来进行操作系统的研制与调试. Linux是 Unix操作系统在 IBM PC兼容机上的完整实现,并且遵循 POSIX规范.它是当今微机上最好的操作系统之一,有着丰富的软件资源.我们借助 Linux先进的多任务能力和进程间通信能力构造了一个仿真分布式共享内存多处理器结构的虚拟硬件环境.它非常逼近真正的硬件,具有中断、时间分片、内存管理、多处理器支持等特性.在这种环境下调试和测试操作系统可以大大加快研制速度.

表 2 共享对象通信模型性能测试结果

操作	时间(微秒)
对象创建	31
对象删除	20
对象定位	12
本地对象访问	18
远程对象访问	120

为了评估共享对象通信模型的可行性,我们在虚拟硬件环境(硬件环境: CPU为 Pentium-MMX, 主频为 166MHz, 32M内存)下对几个主要操作的性能进行了测试,测试结果见表 2,其中本地

对象访问时间与远程对象访问时间是分别向一本地信箱和一远程信箱发送一 16字节长度的消息而得到的.

这些操作均是在仿真环境下测试的,由于大部分操作均要涉及到 Linux的系统调用,因此开销较大.但从表中可以看出,除远程对象访问外,这些操作均具有很高的性能,能满足现代仪器系统实时分布式处理的需求.

7 结束语

GSO是介于实时分布式操作系统的微内核与应用层之间的一个全局共享对象层,它通过任务、内存块、信号量等全局共享对象消除了各结点之间的物理边界,为系统提供了一种透明的分布式处理能力,也为程序设计者提供了一个统一的逻辑视图.我们在 IOWRTDOS中实现了一个共享对象通信模型.与共享内存模型相比,共享对象通信模型具有更好的可编程性、可移植性和代码的复用性,适合于各种粒度的通信形式.由于该模型是直接在操作系统微内核之上实现的,并且共享的是细粒度的对象,因而具有较高的性能.

参 考 文 献

1 Tanenbaum A S. Distributed operating system. Englewood Cliffs, NT: Prentice-Hall, 1995

2 Bal H E *et al*. Programming languages for distributed computing systems. ACM Computing Surveys, 1989, 21(3): 261~322

3 Birrell A D, Nelson B J. Implementing remote procedure calls. ACM Trans on Computer Systems, 1989, 21(3): 261~322

4 Wilbur S, Bacarisse B. Building distributed system with remote procedure call. Software Engineering Journal, 1987, SE-11(9): 148~159

5 Nitzberg B *et al*. Distributed shared memory: A survey of issues and algorithms. Computer, 1991, 24(8): 52~60

6 Kriewall T J *et al*. Computer-based medical systems. Computer, 1991, 24(3): 9~12

7 Dave A *et al*. Proxies, application interface, and distributed systems. In: Proc of the 2nd Int l Workshop on Object-Oriented in Operating Systems. California, 1992. 212~220

8 Chase J S *et al*. The Amber system: Parallel programming on a network of multiprocessors. Operating System Review, 1989, 23(5): 147~158

9 Lea R *et al*. COOL: System support for distributed programming. Communications of the ACM, 1993, 36(9): 37~46

10 Bal H E *et al*. Orca: A language for parallel programming of distributed systems. IEEE Trans on Software Engineering, 1992, 18(3): 190~205