

基于 Windows 的 DSM 系统中线程迁移的实现

郑志军,张金表,赖红武

(浙江科技学院 信息与电子工程学院,杭州 310023)

摘 要: 随着多线程 DSM 系统的发展,线程迁移已经成为 DSM 系统中一个非常重要的研究课题。在分析 Windows 操作系统的存储和控制结构的基础上,讨论了基于 Windows 操作系统的多线程 DSM 系统的基本框架,提出了一种对线程进行动态迁移的方法。并且在一个基于 Windows 操作系统工作站机群的 Smonn 系统中成功实现,表明了该方法的有效性。

关键词: 分布式共享存储系统;多线程;线程迁移

中图分类号: TP316.4

文献标识码: A

文章编号: 1671-8798(2006)02-0114-04

Implementation of Thread Migration in Windows Based DSM System

ZHENG Zhi-jun, ZHANG Jin-biao, LAI Hong-wu

(School of Information and Electronic Engineering, Zhejiang University of Science and Technology, Hangzhou 210023, China)

Abstract: With the development of multithreading DSM system, thread migration has become a very important problem and has attracted a lot of attention recently. Storage and control structure in Windows is analyzed, and a method of thread dynamic migration is presented after a discussion of the basic architecture of multithreading DSM system based on Windows. In addition, the method by implementing dynamic thread migration is proved to be effective in Smonn system which is implemented on network of Windows workstations.

Key words: distributed shared memory; multithreading; thread migration

目前,随着工作站集群系统的广泛使用,分布式共享存储系统(DSM——Distributed Shared Memory)由于结合了共享存储多处理机系统的易编程性和消息传递多计算机系统的易扩展性,已经成为高性能计算机体系结构发展的主流^[1,2]。

国内外已经实现了许多典型的 DSM 系统,其中硬件实现的 DSM 系统有 Stanford 的 DASH,SCI 的 Origin2000 等,软件实现的 DSM 系统可分为三代:第一代

系统基于单 CPU 的结点,采用顺序一致性模型,如 Ivy;第二代系统也是基于单 CPU 的结点,但采用非严格存储一致性模型,如 TreadMarks;第三代系统基于对称多处理机,采用非严格存储一致性模型,但增加了多线程机制,如 Brazos^[3]。这些系统大多基于 Unix 操作系统。近几年开始出现了基于 Windows 的 DSM 系统,如美国 Rice 大学的 Brazos 和 Maryland 大学的 CVM 系统^[4],但在这些系统中线程分配多采用人

工指定或固定分配算法,编程者负担较重,运算的性能相对较低。

笔者分析研究了多线程 DSM 系统的基本结构,针对系统中出现的线程迁移问题,提出了一种动态的线程分配、迁移方法,并在一个基于 Windows 工作站机群的 Smonn 系统中实现了线程的动态迁移。

1 多线程 DSM 系统的基本结构和线程迁移

多线程 DSM 系统的基本结构如图 1 所示。系统由若干个处理结点构成,所有的处理结点共享由 DSM 系统提供的统一存储空间,结点间的通信通过共享变量和隐式的消息传递实现。从用户角度看,一个并程序由若干个线程组成,并在运行时分布到各处理结点上。

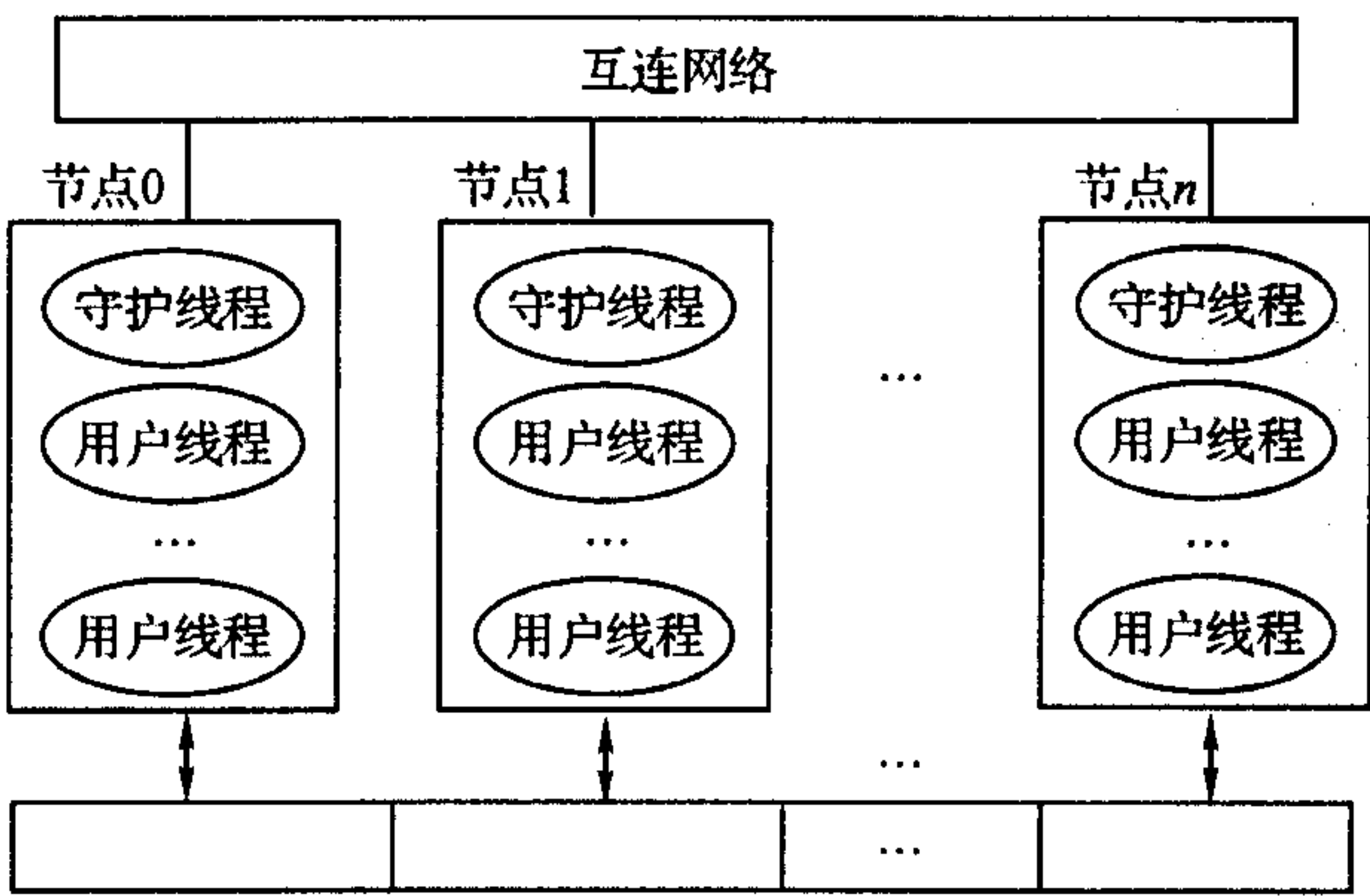


图 1 多线程 DSM 系统的基本结构

显然,在多线程 DSM 系统中,属于同一结点的线程间共享变量的开销较小,属于不同结点的线程之间共享变量的开销较大。为了最大限度地减少通信开销,提高系统性能,应该将彼此之间耦合度较高(也就是共享变量较为频繁)的线程放入同一结点,将彼此之间耦合度较低的线程放入不同结点。

由于对线程进行动态分配,能降低并行编程的难度,提高 DSM 系统的适应性和可用性,有着较大的优越性,所以本文提出了一种线程迁移方法,以支持线程的动态分配。

该动态实现方法就是在程序执行过程中定时周期性地获取性能数据,根据耦合度原则动态地将线程迁移到最佳位置。线程之间的耦合度可通过共享数据的权值来确定,共享数据的权值来自于上次线程迁移以来测量得到的数据。因此,动态实现方法使用了程序的局部性原理,即用“过去”的数据预计线程“将来”的行为。

2 多线程 DSM 系统中线程迁移的实现

在多线程 DSM 系统中,所有的应用线程分为两个层次:同一处理结点内的线程之间为紧耦合关系,它们共享物理存储空间;不同处理结点的线程之间为松耦合关系,它们通过隐式的消息传递共享逻辑存储空间。DSM 系统使用操作系统的存储保护机制来实现各结点之间存储空间的共享,使用一致性模型来保证各结点共享页拷贝的一致性。由于共享变量不由某个线程单独占有,而由整个系统维护,即使线程迁移到另外一个结点上,系统仍然能够正确地访问。因此,在 DSM 中实现与多个线程有共享关系的线程迁移,可以不考虑共享变量的一致性问题。

在操作系统中,一个线程在内存中分为程序段,数据段和堆栈三部分。程序段里放着线程的机器码和只读数据,所有线程共用进程的数据段来存放线程中的静态数据,动态数据则通过堆栈来存放,M 个线程的存储结构如图 2 所示。

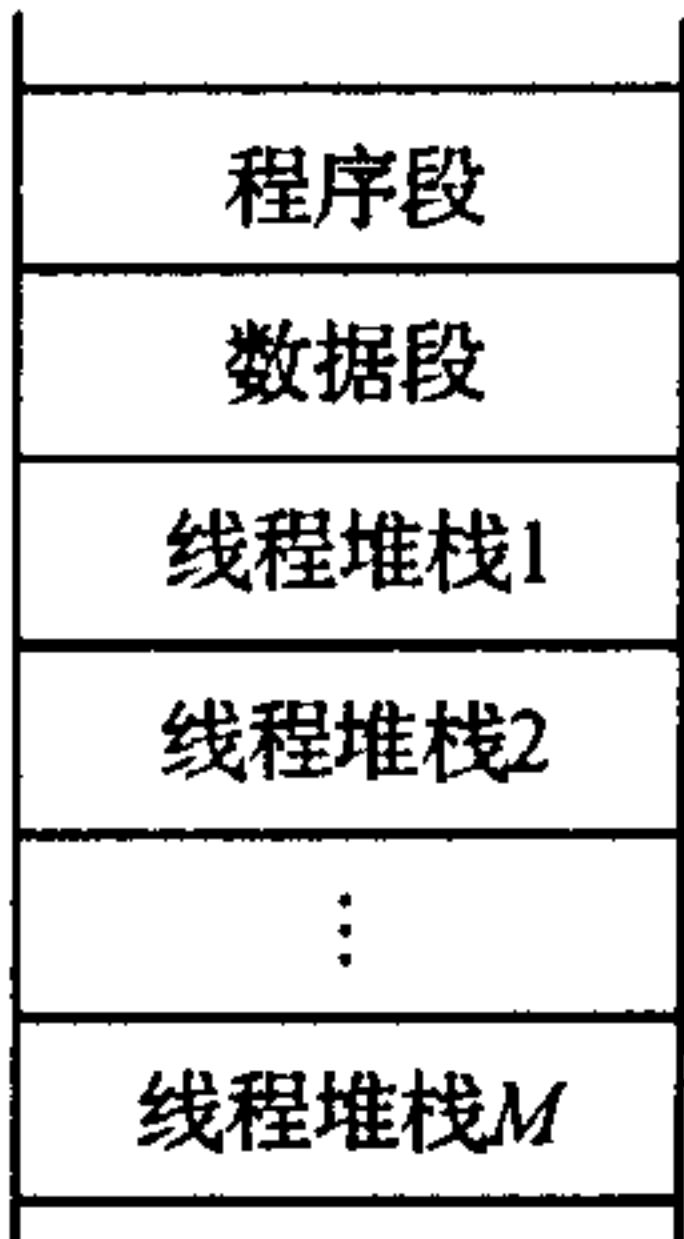


图 2 Windows 操作系统中线程的存储结构

多线程 DSM 系统在初始化时,每个结点上都有了所有线程的程序段,可以不迁移线程的程序段。另外,由于在线程迁移后,线程中的静态和全局变量将成为共享变量,迁移线程中应使用局部变量,避免使用静态和全局变量,静态和全局变量可以用共享变量代替,对于线程的数据段也不加以考虑。

每个线程都有自己的一组 CPU 寄存器,称作线程的上下文(context)。当线程最后执行时,这个 context 结构反映线程的 CPU 寄存器状态。CPU 寄存器中有一个指令指针,指出了执行线程的下一个 CPU 指令的地址,还包括一个栈指针,指出了线程栈的地址。当线程排定 CPU 时间时,系统用线程的上下文初始化 CPU 的寄存器。

因此,要动态迁移线程,只需迁移线程的堆栈信息和上下文信息。

2.1 线程迁移点的设置

为了保证迁移线程正确恢复到执行点,线程迁移点的设置应该能够准确得到程序指令计数器的值。

在 Windows 操作系统中,由于每个子函数在不同的进程的地址空间中映射的存储位置可能不同,需要得到迁移点相对于线程起始执行指令地址的偏移值。如果将迁移点设在线程函数调用的子函数内部,得到的程序指令计数器的偏移值在不同结点上的值可能是不同的,这将导致迁移的失败。因此,不能在线程函数调用的子函数内部设置迁移点,而只能在线程函数内部设置迁移点。

为了得到迁移点相对于线程起始执行指令地址的偏移值,在线程迁移点调用线程函数 `GetThreadContext` 取得线程的上下文信息,即一组 CPU 寄存器的值,其中有 CPU 指令指针的值。同时,预先取得迁移线程的起始执行指令的地址值,然后用它减去程序指令计数器的值,就得到了迁移点相对于线程起始执行指令地址的偏移值。同样地,在目标结点中取得迁移线程的起始执行指令的地址值,再加上此偏移值,就等于此线程在目标结点上要恢复执行的指令计数器的值。

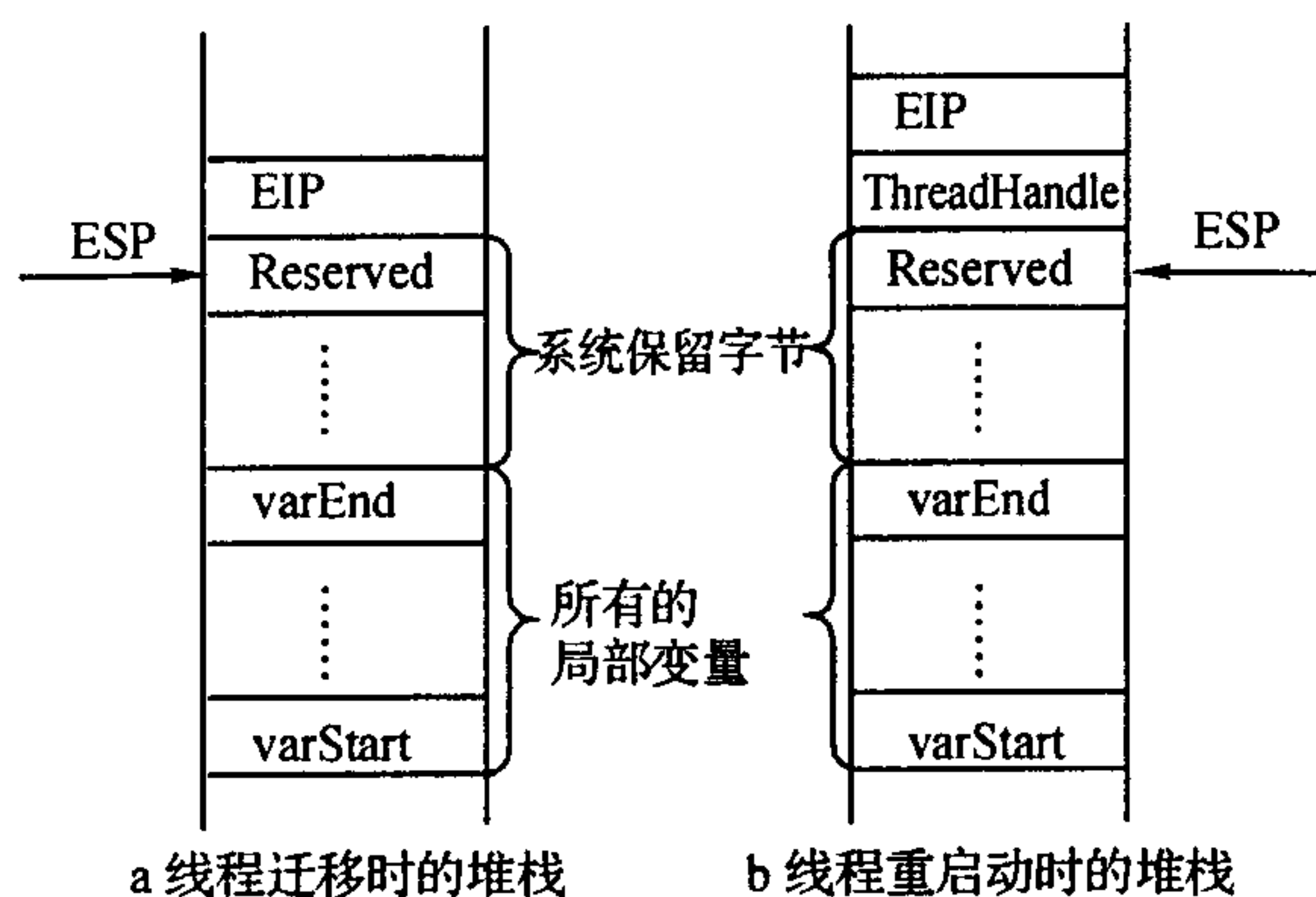
2.2 线程堆栈的迁移

Win32 操作系统在进程中创建一个线程时,同时创建了此线程的一个堆栈。系统为每个线程堆栈保留一个地址空间区域(reserve memory),并给这个保留区域提交一些物理存储(commit memory)。在目标结点重新启动迁移线程时,必须保证提交的物理存储空间足够大,能容纳下线程的所有堆栈信息。

通过引入两个系统整型变量 `varStart`、`varEnd`,可以取得所有局部变量在栈中的位置,当线程在源结点运行到迁移点时,得到的线程堆栈结构如图 3(a)所示。在目标结点重新启动线程前,将所有局部变量的值拷贝到线程的堆栈,并设置线程句柄和线程指令计数器值,恢复后的线程堆栈如图 3(b)所示。

3 线程迁移在 Smonn 系统中的控制过程

笔者在 Smonn 系统上实现了该线程动态迁移算法。Smonn(Shared Memory on Network of Workstations)系统是一个软件实现的第三代 DSM 系统,它基于 Windows 工作站机群和标准的 TCP/IP 通讯协议,采用非严格存储一致性模型^[1],提供多线程机制并支持线程在处理结点间的动态迁移。Smonn 系统对线程迁移的控制过程如下:



ThreadHandle 为线程句柄;EIP 为线程指令计数器

图 3 线程堆栈结构

(1)线程开始执行时调用宏指令 `MIGRATE_HEAD`,保存此线程的栈地址和起始指令计数器的值;

(2)源结点向目标结点发送线程迁移命令 `THREAD_MIGRATE`,并将线程号和线程的上下文环境数据,包括线程指令计数器的相对偏移值、所有的局部变量和一组 CPU 寄存器的值放在命令包中;源结点挂起线程,并等待目标结点发来的应答;

(3)目标结点接收线程迁移命令并返回应答;根据接收的线程号创建线程并挂起该线程;设置线程上下文环境后恢复执行;

(4)源结点终止迁移线程。

下面是一个线程的迁移实例:

Thread procedure

Begin

`VAR_START`; //宏定义,定义变量起始地址
变量定义;

`VAR_END`; //宏定义,定义变量结束地址

`THREAD_BEGIN`; //Smonn 系统的线程初始化

宏指令

`MIGRATE_HEAD`; //线程迁移预处理宏指令
程序代码;

`p4_migrate_thread(m)`; //线程迁移命令,将此线程迁移到第 m 个结点上

程序代码;

`THREAD_END`; //Smonn 系统的线程结束宏指令

End.

4 实例验证结果

笔者以求解 20 个城市的 TSP 问题为例,在由 4 台安装了 Windows2000 操作系统的微机构成的 DSM 环境中运行 Smonn 系统,对该线程动态分配、

迁移算法的性能进行了测试。

TSP 问题的求解过程如下:由主程序生成若干线程,每个线程完成一部分路径搜索任务。本实验的线程数目为 16。

4 台微机结点通过 100 Mb/s 的以太网相连。一般情况下,线程的堆栈不会很大,本例中线程堆栈的一个内存页为 4 kB。因此,相对于一个具体的应用,线程迁移的开销很小,可以忽略不计。实验结果见表 1。

表 1 两种线程分配算法求解 TSP 问题的实验结果

	静态分配算法	文中动态分配、迁移算法
执行时间 t/s	35	23

从表 1 可见,相对于静态算法,采用线程动态分配、迁移算法后,由于消息传递的数量大幅减少,系统的通信开销大大降低,问题求解的时间明显变短,算法的执行效率得到了很大的提高。

5 结 语

针对多线程 DSM 系统中的线程迁移问题,提出了解决方法,并在一个基于 Windows 工作站机群

(上接第 113 页)

以 MPEG-4 编解码器为例^[6]:MPEG-4 标准文档将视频定义为“具有一定语义的实体”的对象。视频内容语法结构呈现一种层次结构(图 3),从高层到低层依次是视频序列(VS)、视频对象(VO)、视频对象层(VOL)、视频对象组(GOV)和视频对象平面(VOP)。其中视频序列是由一系列具有一定时间和空间关系的视频对象构成,一个视频对象可由多个视频对象层组成,VOL 提供基于对象的分级支持。每个 VOL 由若干视频对象组构成,GOV 提供随机切入的功能。GOV 由一系列时间上相继的视频对象平面组成。VOP 是基本的编码单位,编码时 VOP 将分成 16×16 的宏块并进一步分解为 8×8 的块。

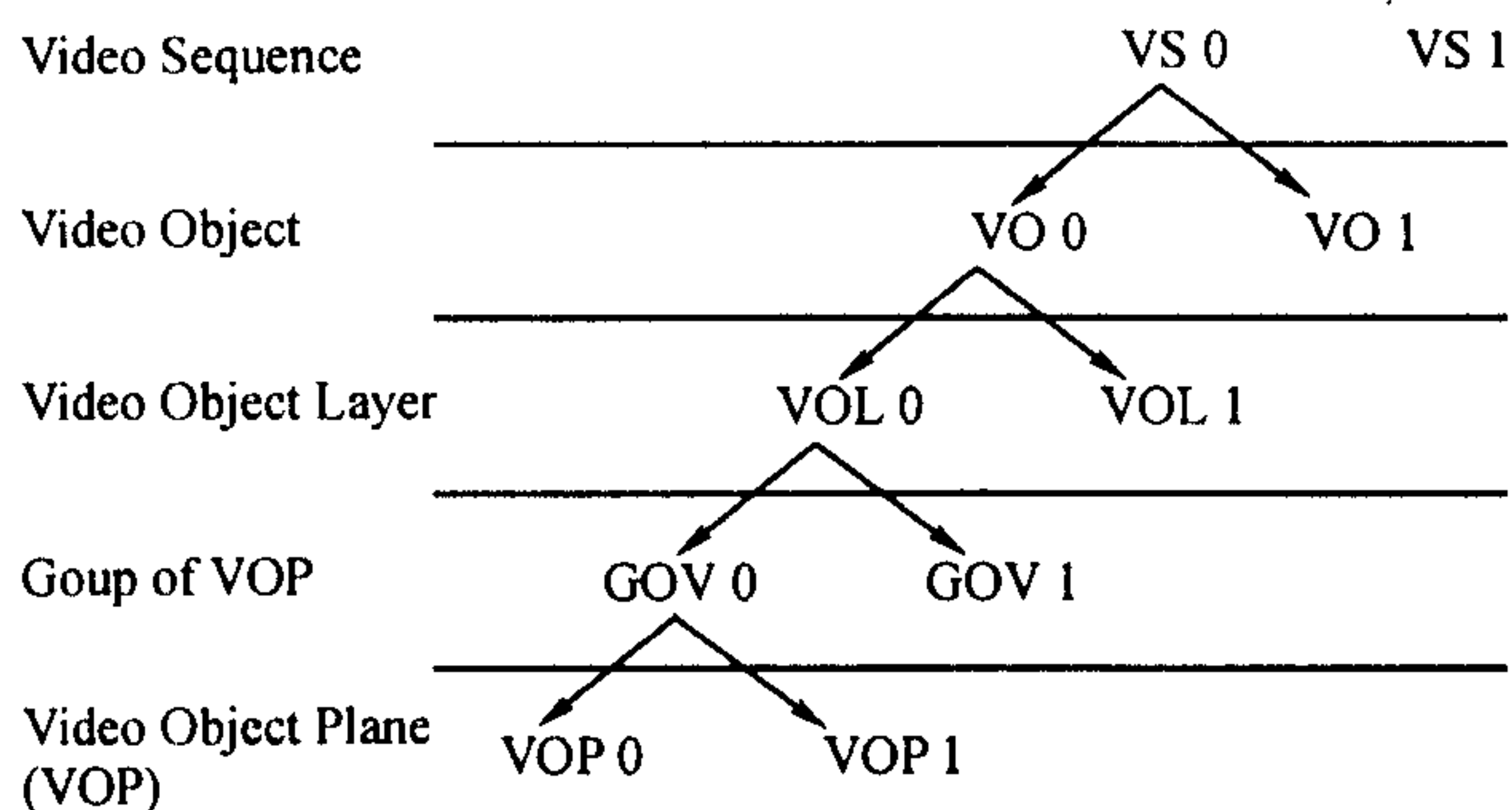


图 3 MPEG-4 视频语法结构

的 Smonn 系统中使用本方法实现了线程迁移,证实了此方法的有效性。基于 Windows 的多线程 DSM 系统因具有良好的可编程性和较高的性能价格比而正逐渐成为高性能计算的理想平台,但是在性能上与消息传递的平台,如 PVM 之间仍然存在着相当大的差距。今后进一步的工作是分析影响 DSM 系统性能的关键因素,并研究合理的线程分配。

参考文献:

- [1] TANENBAUM Andrew S. Distributed Operating Systems [M]. Cambridge:Prentice-Hall International, Inc., 1995.
- [2] ZHU W. Cluster queue structure for shared-memory multiprocessor systems[J]. The Journal of Supercomputing, 2003, 25(3):215-236.
- [3] SPEIGHT E, BENNETT J K. Brazos: A third Generation DSM Systems[C]. Proceedings of the 1997 USENIX Windows/NT Workshop, 1997:95-106.
- [4] UENG Jyh-chang, SHIEH Ce-kuen. Proteus: An efficient runtime reconfigurable distributed shared memory system[J]. Journal of Systems and Software, 2001, 56(3): 247-260.

由于 MPEG-4 协议本身以对象的形式来描述视频流的语法元素,因此在用 C 语言实现时相应的可以使用面向对象的方法来定义这些语法元素,同时将操作和数据封装在一起,如从下到上可划分为块对象、宏块对象、VOP 对象等。这样做的好处是,一方面容易与协议的描述取得一致,另一方面易对协议实现的正确性进行验证,同时使整个软件架构清晰合理,容易维护和扩展。

参考文献:

- [1] 李师贤,李文军,周晓聪. 面向对象程序设计基础[M]. 北京:高等教育出版社,1998.
- [2] 张佳骥. 从结构化分析到面向对象[J]. 无线电工程, 2005, 35(12):5-7, 59.
- [3] LINUS. linux2. 4. 32 kernel[CP/OL]. (2005-11-16) [2006-03-10]. <http://www.kernel.org>.
- [4] 毛德操,胡希明. LINUX 内核源代码情景分析[M]. 杭州:浙江大学出版社,2001.
- [5] ALESSANDRO Rubini, JONATHAN Corbet. LINUX DEVICE DRIVERS[M]. 2nd. USA:O'REILLY,2002.
- [6] 钱亚冠. 视频编码理论与 MPEG-4 的 DSP 实现[D]. 杭州:浙江大学计算机科学与技术学院,2005.