多线程技术的研究与应用

骆 斌 费翔林
(南京大学计算机软件新技术国家重点实验室 南京 210093)
(南京大学计算机科学与技术系 南京 210093)

摘 要 现代主流操作系统已经广泛采用了多线程技术.首先论述了多线程的基本概念,然后着重分析了 3种主要的多线程实现方案: 内核级线程 用户级线程和混合策略,然后介绍了多线程技术的应用.还结合面向对象数据库管理系统 NODBMS的实现,介绍了如何应用多线程技术实现多用户事务处理,并提出了一个基于多线程技术实现的对象式数据库查询优化算法,该算法取得了较好的效果.

关键词 线程,进程,用户级线程,内核级线程

中图法分类号 TP316

STUDY AND APPLICATION OF MULTITHREAD TECHNOLOGY

LUO Bin and FEI Xiang-Lin
(*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210093)
(*Department of Computer Science and Technology, Nanjing University, Nanjing* 210093)

Abstract Multithread technology has been adopted in recently developed mainstream operating systems. Firstly the basic concepts of multithread are described. Secondly, three implementation methods of multithread mechanism that is user-level threads, kernel-level threads and combined approaches are discussed. Thirdly, application of multithread programming is introduced. Through the object-oriented database management system NODBMS, how to use multithreads to realize multiple transaction processing is also introduced, and is proposed an effective query optimization algorithm of object-oriented database system based on multithread technology.

Key words thread, process, ULT, KLT

1 引 言

在传统操作系统中,进程是系统进行资源分配的单位,也是处理器调度的独立单位.进程在任一时刻只有一个执行控制流,我们将这种结构的进程称单线程进程(single threaded process).这种单线程结构的进程已不能适应当今计算机技术的迅猛发展.

从硬件技术的发展来看,并行处理和计算机网络均要求解决细粒度的并行以更好地发挥多处理器的能力.从软件技术的发展来看,系统软件和应用软件均要求人们设计出多事件并行处理的系统.但是传统进程管理方式的进程切换代价和进程通信代价均较大,从而限制了并行的粒度和并行的效率.因此要求操作系统改进进程结构,提供新的机制,于是出现了多线程(结构)进程(multiple threaded process).

目前,很多著名的操作系统都支持多线程(结构)进程,如: Solaris 2.x, Mach 2.6, OS/2, WindowNT,

原稿收到日期: 1999-07-01;修改稿收到日期: 1999-12-24.本课题得到国家自然科学基金项目(项目编号 69775019)资助. 骆斌,男,1967年生,博士,主要研究领域为数据库技术 操作系统 机器学习.费翔林,男,1941年生,教授,主要研究领域为操作系统、软件工程.

Chorus等.许多计算机公司都推出了自己的线程接口规范,如 IEEE的多线程程序设计标准 POSIX 1003.4a,可以相信多线程技术在软件开发中将会被越来越广泛地采用.

本文首先论述多线程(结构)进程中线程的概念,然后结合 3个经典实例探讨几种主流的多线程实现方法,最后讨论多线程的应用.

2 线程的概念

传统操作系统中的单线程进程由进程控制块和用户地址空间、以及管理进程执行的调用/返回行为的系统堆栈或用户堆栈构成.如果把进程的管理和执行相分离,进程作为操作系统中进行保护和资源分配的单位,允许一个进程中包含有多个可并发执行的控制流,这些控制流切换时不必通过进程调度,通信时可以直接借助于共享的内存区,这就形成了多线程进程.图 1是单线程进程和多线程进程的模型.

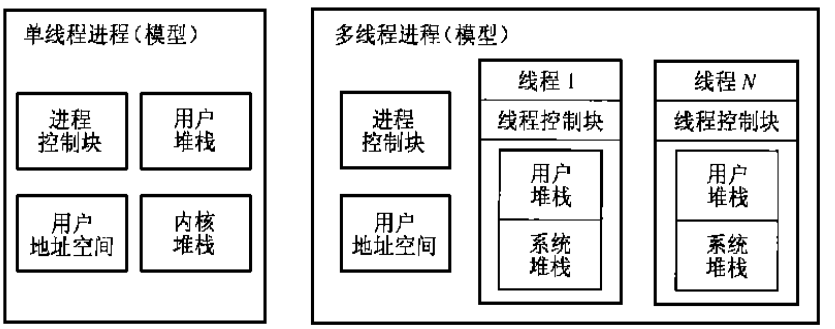

图 1 单线程进程和多线程进程的模型

在多线程环境中,仍然有与进程相关的 PCB和用户地址空间,每个线程还存在独立的堆栈,及包含线程状态信息的线程控制块.线程之间的关系较为密切,一个进程中的所有线程共享其拥有的状态和资源.它们驻留在相同的地址空间,可以存取相同的数据.

在多线程环境中,进程是操作系统中进行保护和资源分配的独立单位,它具有:① 一个虚拟地址空间,用来容纳进程的映象;② 对 CPU、进程、文件和资源等的存取保护机制.

线程则是指进程中的一条执行路径(控制流),每个进程内允许包含多个并行执行的路径,这就是多线程.线程是系统进行处理器调度的基本单位,同一个进程中的所有线程共享进程获得的主存空间和资源,线程具有:① 一个线程执行状态;② 一个受保护的线程上下文;③ 一个独立的程序指令计数器;④ 一个执行堆栈;⑤ 一个容纳局部变量的静态存储器[1].

3 线程的实现

各操作系统采用的线程实现方法各有区别,总的来说可以归结为两类方式:用户级线程(ULT)和内核级线程(KLT).也有一些系统提供了混合式策略,同时支持两种线程实现.

3.1 内核级线程(KLT)

在纯内核级线程设施中,线程管理的所有工作均由操作系统内核来做,并专门提供了一个 API,应用区不需要有线程管理的代码.任何应用都可以被设计成一个进程中的多个线程.内核要为整个进程及进程中的单个线程维护现场信息.内核的调度是在线程的基础上进行的.Windows NT和 OS/2都采用了内核级线程.下面以 NT为例来介绍内核级线程[2].

NT的进程和线程都是用对象来实现的,包括属性和封装了的若干可以执行的动作和服务.NT中的一个进程可以包含一个或多个线程,线程可被分配到不同处理器去执行,同一进程中的线程能通过公共地址空间来交换信息和存取进程的可共享资源,因此一个多线程进程可以达到并发性而不必付出像多进程并发那样多的开销.进程或线程均有内在的同步设施.

图 2给出了 NT的线程状态,准备态是指已被选中下一个在一个特定处理器上运行,如果准备态线程的优

先级足够高,则可以从正在运行的线程手中抢占该处理器.过渡态是指一个线程完成等待后准备运行,但这时资源不可用.当资源可用时,过渡态线程进入就绪态.

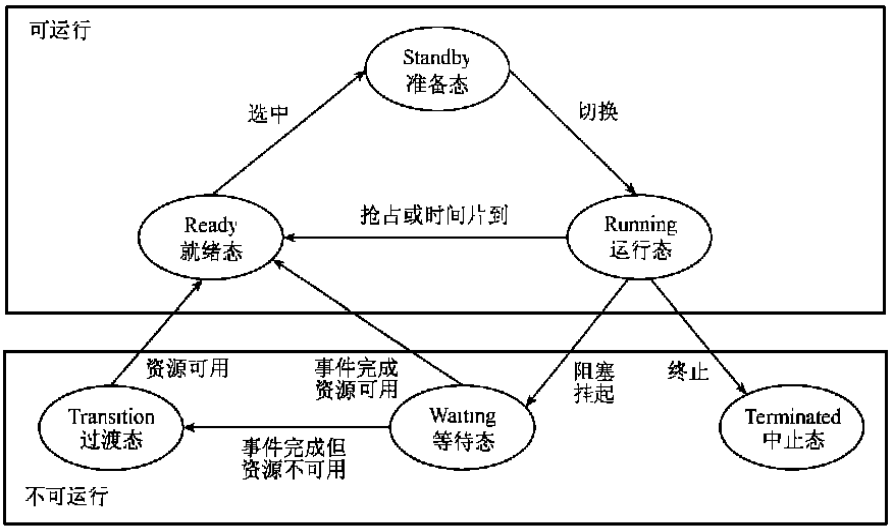

图 2　NT的线程状态

总的来说, KLT有若干优点: 首先,在多个处理器环境中内核能够同时调度同一进程中多个线程;其次,若进程中的一个线程被阻塞了,内核能调度同一进程的其它线程占有处理器;另外,操作系统内核例行程序本身也能被多线程化. KLT的主要缺点是: 在同一进程中,控制权从一个线程传送到另一个线程时需要内核态和用户态之间的模式切换,开销较大.

3.2　用户级线程 (ULT)

纯 ULT设施中,线程管理的全部工作都由应用程序来做,内核不知道线程的存在,把进程作为调度单位.用户级线程由线程库来实现,任何应用程序均需通过线程库提供的函数进行程序设计,再与线程库连接后运行来实现多线程.并且线程的调度也是由线程库实现的. ULT是语言级的线程提供机制,作为实例,下面介绍 Java语言的用户级线程机制[3].

在 Java语言的多线程系统中,"调度程序" (scheduler)用来调度线程,处理器资源是按时间片分配的,每个线程被赋予一个优先级,采用"抢占式" (preemptive)调度方式.为使低优先级线程能够有机会运行,较高优先级线程可以不时进入"睡眠" (sleep)状态.线程的优先级如果相同,将依据"先来先服务"原则调度.线程组是 Java用以管理线程的概念,每个线程均属于某一线程组,一个线程组可以包含多个线程或其他线程组,从而形成线程之间的层次关系. Java语言对多线程的支持则主要是通过类 Thread和类 Runnable来提供.而 synchronized关键字和 monitor可以用来控制线程的访问同步.

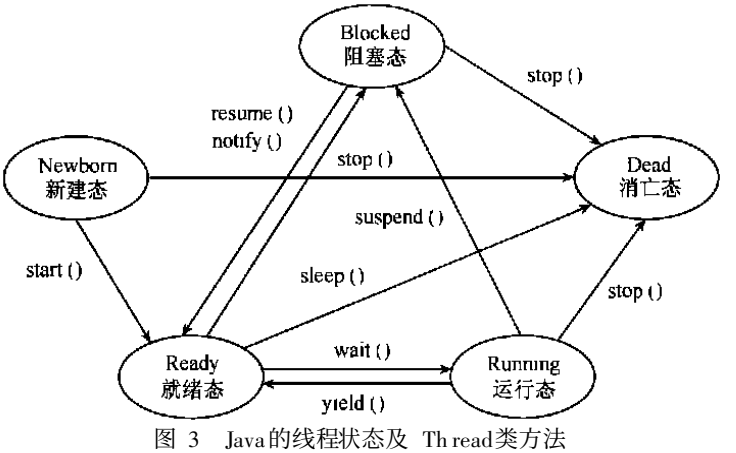

图 3　Java的线程状态及 Thread类方法

图 3给出了 Java的线程状态.线程在已被创建但尚未执行这段时间内处于新建态;线程对象可通过 start

()方法切换到就绪态;消亡态表示线程已退出运行态,并不再进入就绪队列.

总的来说, ULT有许多优点:① 线程切换的开销小;② 可按应用特定需要来调度;③ ULT能运行在任何操作系统上.但 ULT也有 2个明显的缺点:① 传统操作系统中,线程执行大多数系统调用时会阻塞整个进程;② 在纯 ULT中,多线程应用不能利用多重处理的优点.可以采用 jacketing技术来解决阻塞线程的问题,其主要思想是把阻塞式的系统调用改造成非阻塞式的,当线程执行系统调用时,首先执行 jacketing实用程序,由 jacketing程序来检查资源使用情况,以决定是否执行系统调用或传递控制权给另一个线程.

3.3 混合式策略

有些操作系统同时提供了 ULT和 KLT,线程创建是完全在用户空间做的,单个应用的多个 ULT映射成一些 KLT.程序员可按应用和处理器数调整 KLT数目,以达到较好的并行效果.

Solaris便是这样一个混合系统[4,5].其线程实现分为 2个层次:用户层在用户线程库中实现;核心层在操作系统内核中实现. Solaris采用了 4个分离的与线程有关的概念:

(1) 进程(process):通常的 UNIX进程,它包含用户的地址空间和 PCB.

(2) 用户级线程(ULT):通过线程库在用户地址空间中实现.用户可以在每个进程中创建多个 ULT,而不必占用核心资源.一个进程中的多个 ULT共享用户空间, ULT的切换仅是数据结构的切换,其时间开销远低于两个 KLT间的切换时间.

(3) 轻量级进程(LWP): LWP可看做是 ULT和 KLT之间的映射.一个进程可以设计为一个或多个 LWP,每个 LWP支持一个或多个 ULT, LWP与 ULT一样共享进程的资源. KLT和 LWP是一一对应的, LWP是核心调度单位,它可以在多个处理器上并行执行.

(4) 内核线程(KLT):它们是能被调度和指派到系统处理器上去运行的基本实体.

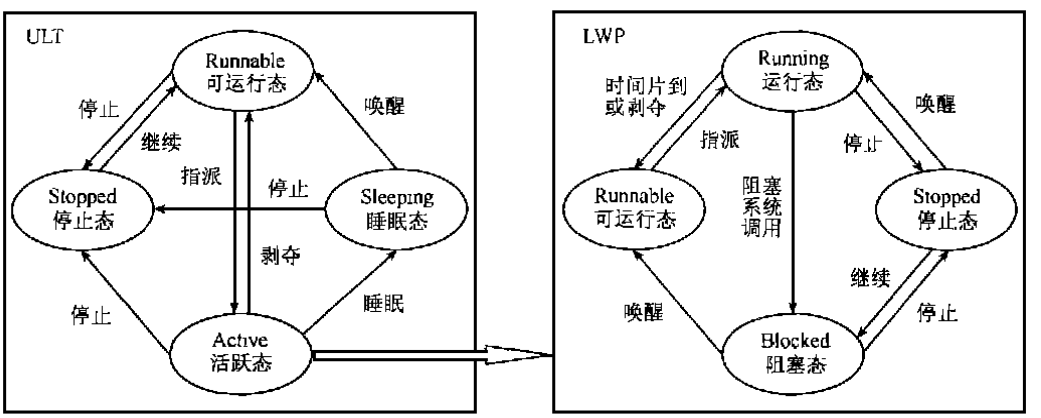

图 4 Solaris的线程状态

图 4简单说明了 ULT和 LWP的状态.用户级线程可能处于 4个状态之一:可运行、活跃、睡眠和停止.处在活跃态的 ULT目前分配到 LWP上,并且在对应内核线程执行时,它被执行.图 4还显示了 LWP的状态转换图,可以把它看做是处于活跃态的 ULT的详尽描述,一个活跃态的 ULT只能捆绑到一个 LWP上,只有当 LWP处于运行态时,一个处于活跃态的 ULT才真正执行.当活跃态的 ULT执行一个阻塞的系统调用时, LWP进入阻塞态,这个 ULT将继续处于活跃态并继续与相应的 LWP绑定,直到线程库显式地做出变动.

混合式策略包括 2个层次的多线程接口, ULT接口的切换代价较小,适用于解决逻辑并行性问题;而 KLT接口则适用于那些需要在多个处理器进行并行计算的问题.如果设计得当的话,混合应用两种接口可以带来更大的灵活性和优越性.如:当应用不需要并发运行时,可以利用单线程进程结构;当程序设计时要表示并发而不真正需要多线程并行执行时,可以建立多个 ULT,所有 ULT对应于一个 LWP,由单个内核线程支撑;当应用需要全面并行时,可以建立多个 ULT,每个 ULT对应于一个 LWP,从而使内核并行机制完全对应用可见.

4 线程的应用

并发多线程程序设计使得系统性能获得很大提高,具体表现在:① 快速线程切换;② 减少(系统)管理开销;③ (线程)通信易于实现;④ 并行程度提高;⑤ 节省内存空间.

目前多线程技术在应用软件和系统软件的实现中得到了广泛的使用,如操作系统中的并行文件操作;数据库中的多用户事务处理;窗口系统中的多个相关子窗口操作;实时系统中多事件的同时响应;网络中多个客户共享网络服务器等等.下面给出了一些线程应用的例子:

(1) 加快执行速度.一个多线程进程在计算一批数据的同时,读入设备(网络、终端、打印机、硬盘)上的下一批数据,而这分别由两个线程实现.

(2) C/S应用.局域网上服务器处理多个用户文件(任务)请求时,创建多个线程,若该服务器是多 CPU的,则同一进程中的多线程可以同时运行在不同 CPU上.

(3) 前台和后台工作.如在一个电子表格软件中,一个线程执行显示菜单和读入用户输入,同时,另一个线程执行用户命令和修改电子表格.

(4) 多事件处理.每当用户要求执行一个动作时,就建立一个独立线程来完成这项动作.当用户要求有多个动作时,就由多个线程来实现.

(5) 多窗口系统.当用户要求建立多窗口系统时,用多个线程分别实现主从窗口.

5 多线程技术在面向对象数据库管理系统中的应用

下面以我们在 Solaris上开发的面向对象数据库管理系统 NODBMS为例,给出在数据库管理系统的实现中应用多线程的实例.数据库管理系统首先要解决的是多用户并发问题,NODBMS服务器进程初始化时首先创建 3个线程:连接处理线程、锁管理线程和死锁监测线程.当客户端应用请求建立一个连接时,连接处理线程专门创建一个新的用户服务线程来处理来自客户端的 OSQL语句.

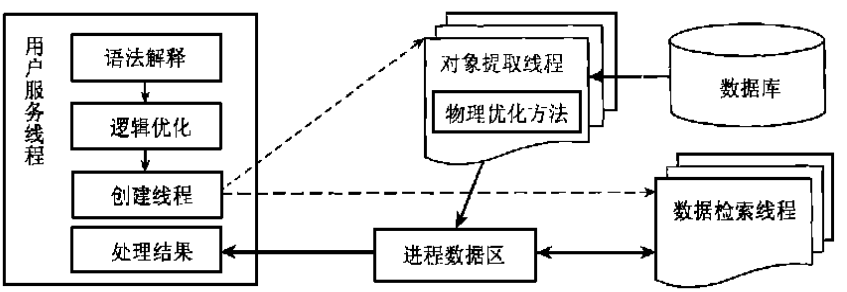

图 5 一种基于多线程技术实现的面向对象数据库管理系统查询优化方案

查询是数据库管理系统最为繁重的任务,NODBMS采用了多线程技术来实现查询优化,提高查询效率.如图 5所示,用户服务线程在接收到一个 SELECT语句时,它首先进行语法解释和逻辑查询优化,将其转换成多个可并行执行的原子查询,再根据具体情况动态地创建多个线程.每个原子查询可以对应于一个数据检索线程和多个对象提取线程,而一个对象提取线程用于提取一个类中的对象,它可以同时为多个数据检索线程服务.这样,一个复杂的查询就可以分解成多个可以并行执行的子任务,这些线程直接通过共享的数据区通信,再通过合适的线程同步机制,就可以快速有效协同地完成查询.具体的算法简述如下.

用户服务线程:

① 语法解释;

② 逻辑查询优化;

③ 取原子查询;

④ 分析原子查询涉及的对象;

⑤ 创建该原子查询对应的数据检索线程;

⑥ 若不存在对应该类对象的对象提取线程,则创建对象提取线程,分配对象缓冲区并把它和数据检索线程绑定;

否则找到相应的对象提取线程,并把它和数据检索线程绑定;

⑦ 若还存在原子查询,GOTO步骤③;

⑧ 启动所有对象提取线程运行;

⑨ 等待直到所有数据检索线程运行结束;

⑩ 执行基于对象标识符(OID)集合的条件归并;

⑪ 读取并处理结果数据.

对象提取线程:

① 等待启动;

② 若存在物理查询优化方法,则提取相应对象页面,否则提取全部页面;

③ 进一步形成对象缓冲区;

④ 若对象缓冲区满或已经完成全部对象提取,则激活与它绑定的所有数据检索线程;

⑤ 若已经完成全部对象提取,则结束本线程;

否则等待直到所有数据检索线程对该对象缓冲区加工结束,GOTO步骤②.

数据检索线程:

① 等待对象缓冲区;

② 进行检索获得满足条件的对象标识符;

③ 若未完成全部对象提取,则向相应的对象提取线程汇报已经结束一次数据检索,GOTO步骤①;

④ 进行基于路径表达式的反向对象标识符归并;

⑤ 向用户服务线程汇报已经结束一个数据检索,并结束本线程.

为了验证多线程技术的有效性,我们把使用多线程技术实现的数据库管理系统和使用传统技术实现的数据库管理系统进行了对比.实验数据库包括 12个类,其中:学生类(抽象类)、研究生类(抽象类)、本科生类、双学位类和硕士生类构成多继承关系,课程类(抽象类)、本科课程类和研究生课程类构成普通继承关系,教师类和导师类构成普通继承关系,这 10个类和选课类、系科类构成多种复合关系(包括复合环关系).基于该数据库,测试程序连续单条执行 OSQL语句,包括连续插入 116个实例并继续执行 96条用于查询、更新、删除和特例化的 OSQL语句.在具有两粒处理器的 SUN工作站上,同时运行 3个同样的测试程序,使用传统技术实现的数据库管理系统在处理时共花费 20分 16秒,使用多线程技术实现的数据库管理系统在处理时则花费 2分 56秒,效率提高近 7倍.即使在单个 CPU的环境下,多线程解决方案的效率也成倍地优于传统的解决方案.

6 结束语

多线程技术可以广泛地应用于解决各种逻辑并发性问题和物理并行性问题,大大提高了系统处理的效率.

参 考 文 献

1 William S. Operating Systems: Internals and Design Principles 3rd. Englewood Cliffs, NJ: Prentice-Hall, 1998

2 Pham T, Garg P. Multithreaded Programming with Windows NT. Englewood Cliffs, NJ: Prentice-Hall, 1996

3 徐永森,陈俊良. Java应用程序设计和开发环境. 南京:南京大学出版社, 1998
(Xu Yongsen, Chen Junliang. Application Programming and Development Environment of Java (in Chinese). Nanjing: Nanjing University Press, 1998)

4 Powell M L, Kleimen S R *et al*. Sun OS multi-thread architecture. In: Proc 1991 USENIX Winter Conference, Dallas, TX, 1991. 1~14

5 Lewis B, Berg D. Threads Primer. Englewood Cliffs, NJ: Prentice-Hall, 1996