操作系统微内核技术研究*

潘 清 张晓清

(国防科工委指挥技术学院 北京 101407)

摘要 文章介绍了作者在过去 5 年中在微内核技术上所做的工作,给出了 3 个算法.① 通过将任务调度和线程调度算法相结合的方法,来解决单纯以线程为单位的调度系统的效率和公平性问题;② 一个改进的写时拷贝算法,它结合写时拷贝算法和访问时拷贝算法的优点,来解决写时拷贝算法在 I386 体系结构上的适应性问题;③ 提出了一个微内核操作系统计时模型,它解决了传统计时算法在微内核系统中计时不准确的问题.

关键词 微内核,调度,存储管理,计时模型.

中图法分类号 TP316

由于集成电路、计算机网络、分布式处理、多机并行处理、容错等技术的迅速发展,面向单处理机,采用内核不可抢占技术的 Unix 操作系统已经很难适应硬件技术的发展.为了适应以上技术的发展,Unix 操作系统的内核越做越大,越做越复杂,完全丧失了其初始设计目标:系统短小精悍,容易理解.卡内基梅隆大学在美国国防部、国家科学基金的资助下,于 1986 年推出了一个基于微内核结构的操作系统 Mach.[1]随后,斯坦福大学等研究机构纷纷发表了他们在这个领域所做的工作.[2]各个大公司纷纷推出了基于微内核结构的操作系统.[3~5]微内核技术已成为新一代操作系统体系结构的研究热点.

基于微内核结构的操作系统和传统操作系统相比,具有以下特点:① 内核精巧.通常内核只由任务管理、虚存管理和进程间通信 3 个部分组成.传统操作系统内核中的许多部分都被移出内核,采取服务器方式实现;② 面向多处理机和分布式系统.基于微内核的操作系统,在内核中引入了多处理机调度和管理机制,并引入了细粒度并发机制——线程,使得多个处理机可以在同一个任务中并行地执行;③ 基于客户/服务器体系结构.在微内核结构的操作系统中,任务间通信机制——消息机制是系统的基础,操作系统的各种功能都以服务器方式实现,向用户提供服务.用户对服务器的请求是以消息传递的方式传给服务器的.

"八五"期间,我们在国家"八五"攻关项目的支持下,对操作系统微内核技术进行了深入研究,在微内核系统调度技术、存储管理技术、计时模型,微内核系统扩展技术及微内核操作系统原型系统构造方面取得了一些研究成果.本文将介绍这些研究成果.

1 微内核系统调度技术

与传统的操作系统内核相比,微内核调度系统中最突出的特征是增加了处理机和处理机集及线程的管理,并且向用户提供了灵活的手段来控制自己的程序在处理机上的运行.这样,微内核系统就能很好地支持多处理机体系结构.同时,线程为用户提供了细粒度的并行处理机制,使得同一个用户任务中的不同线程可以同时在多个处理机上运行.

与进程相比,线程中所带的资源很少,因此,创建线程和撤消线程的开销就比进程小.线程也称为"轻进程".在系统调度中,线程的切换开销也比进程少,但是不同任务中的线程切换会引起任务的切换,在这种情况下,线程和进程的调度开销就变成一样了.为了优化系统效率,减少由于线程切换而引起的任务切换,在调度算法中加入了以下代码:

```
IF (所选中的线程和当前运行的线程属于同一个任务)
THEN 不做任务切换;
ELSE 进行任务切换操作;
```

显然,这种方法在某种情况下会对系统性能有所帮助,但是这种方法在很大程度上属于一种"被动的",或者说是

* 本文研究得到国家"八五"重点科技攻关项目基金和"九五"国防预研项目基金资助.作者**潘清**,1964 年生,副教授,主要研究领域为操作系统,软件工程.**张晓清**,女,1963 年生,讲师,主要研究领域为操作系统.

本文通讯联系人:潘清,北京 101407,北京怀柔 3380 信箱 97 号

本文 1996-11-05 收到原稿,1997-07-07 收到修改稿

一种"碰运气"的方法.另外,单纯以线程为主的调度算法对用户任务有失公平性,以线程为主的调度算法是完全参照传统操作系统中的调度算法设计而成的.当线程投入运行时,系统为它分配固定大小的时间片,系统中线程按时间片轮转.这样,就产生了公平性问题:如果一个任务中有两个线程,那么,从理论上讲,它将比只用一个线程实现的任务多获得近1倍的处理机时间.在传统的进程调度系统中,一个用户可以通过创建多个进程来获得更多的处理机调度机会,但是,它是建立在增加了创建进程和进程间通讯的系统开销代价的基础上的.相比之下,创建线程的开销非常小,同一任务间的线程之间通讯开销也很小.

为了解决上述问题,我们提出并实现了一种将传统的任务和新的线程调度机制相结合的方法:以任务为单位分配时间片(这样可以保证调度的公平性),在线程调度时,当一个线程不是由于任务时间片用完的原因而放弃处理机时,只要系统中没有高优先级线程,就从本任务中选取线程,从而使得由线程切换而引起的任务切换操作开销达到最小.从目前的发展来看,用户任务的并行粒度越来越小,即用户任务中的线程越来越多,而每个线程所执行的操作会越来越少.因此,使用线程+任务的方法可以有效地减少单纯的以线程为主的系统调度所引起的系统开销.

2 微内核虚拟存储管理技术

微内核虚拟存储管理系统引入了存储对象(Memory Object)的概念,将物理内存看成外部存储对象的(如磁盘)高速缓存(Cache),实现了虚拟存储器写时拷贝(Copy on Write)技术,引入了lazy evaluation技术,定义了虚拟存储器和硬件存储管理机制的接口(Pmap),实现了与机器无关的虚拟存储系统.

虚拟存储器写时拷贝算法是微内核虚拟存储管理系统的核心算法.它的引入使得虚拟存储器管理的效率大大提高了一步.但是,它的实现依赖于硬件存储管理机制的页面保护机制,对于一个具有写时拷贝共享属性的存储区,其页面保护被设置成写保护.多个用户可以以共享的方式对它进行读操作,但是,当用户试图对这块区域进行写操作时,将产生写保护故障,页面故障管理程序将为用户进程复制物理页面,从而达到写时拷贝的目的.

在I386体系结构下,只有用户态页面允许写保护,在其他机器状态下,硬件存取机制将绕过页面保护机制,直接对页面进行写操作.在这种状态下,写时拷贝算法将失效.而在微内核体系结构中,可能有各种状态下的服务器,如在内核态下运行的服务器.为了解决这个问题,我们引入了写时拷贝和访问时拷贝(Copy on Reference)相结合的算法.即在用户态上使用写时拷贝算法,在其他状态下使用访问时拷贝算法来替换写时拷贝算法,以解决写时拷贝算法失效的问题.访问时拷贝算法的实现依赖于页面保护机制的缺页机制.这样,在其他状态下,在设置页面保护时将写保护改成缺页即可.

新的方法在效率上比写时拷贝算法低,但是比完全拷贝的方法高出许多,特别是与lazy evaluation技术相配合时效率会更高.由于微内核提供的写时拷贝算法是对用户透明的,即对于用户编写的任何状态下的服务器都将使用写时拷贝算法.因此,在I386体系结构下,在非用户态上运行的用户服务器有可能出错,新的算法解决了这个问题.

3 微内核计时模型

在传统操作系统中,为统计出每个进程的处理机时间使用量,在每个进程结构中都设有统计进程使用处理机时间的单元.系统计时一般是放在处理机时钟中断服务程序中.系统一般采用如下代码段来进行用户进程的时间统计.

```
IF (当前进程处于用户态)
    增加当前进程的用户态处理机时间使用量
ELSE
    增加当前进程的系统态处理机时间使用量
```

由于在传统的操作系统中,操作系统提供的服务完全由操作系统内核来完成,用户通过系统调用进入内核来取得服务.因此,采用上述方法能比较准确地统计出用户所用的处理机时间.但是,这种计时方法是一种比较粗糙的计时方法.每次时钟中断时,它就将一个固定的时间片(时钟中断周期)加入被中断的进程中,而不管该进程是否完全使用了这些处理机时间.由于这种方法实现起来非常简单,系统开销很小,几乎所有的操作系统都采用了这种方法.在新的操作系统中引入了细粒度的并行执行部件——线程,对于线程的计时也采用了和进程相同的方法.为了取得精确的处理机时间统计精度,一些新型操作系统引入了新的计时机制,如MACH 3.0中引入了基于时间戳的精确计时机制.

在微内核体系结构下,传统的操作系统功能是通过服务器的方式来实现的.服务器和用户任务一样,也作为一个进程运行.当用户进程调用操作系统服务时,微内核通过消息将系统服务的参数传递给操作系统服务器,由操作系统服务器来完成用户请求,并将结果通过消息传递给用户进程.这样,如果采用传统的方法来进行进程的处理机时间统计,就会将操作系统为用户提供服务所用的处理机时间记入服务器中,而不是用户进程中.

为了解决这个问题,我们引入了委托线程的概念,建立了新的用户进程计时模型.在客户/服务器模型中,用户通过消息请求服务器的服务,服务器接收用户的消息完成用户的请求,再通过消息将结果传给用户.在这种体系结构下,可以看成用户将自己的一部分工作委托给服务器完成,服务器是在为委托线程服务.当用户线程向服务器发出请求时,将用户线程标识传递给服务器,当服务器中的某个线程处理这个请求时,将用户线程标识记入服务器线程结构中的委托线程域中.在系统时钟中断服务程序中增加为委托线程计时的代码,就可以将操作系统服务器为用户进程服务的时间计算到用户进程中.

```
IF (当前线程结构中有委托线程)
  IF (当前线程处于用户态)
    增加委托线程的用户态处理机时间使用量
  ELSE
      增加委托线程的系统态处理机时间使用量
```

在多服务器体系结构下,一个用户请求往往需要多个服务器的协同服务,如一个文件读操作,需要文件服务器的服务,如果文件服务器发现数据存放在磁盘中,它就需要请求设备服务器的服务,设备服务器实际上是在为用户线程服务.因此,在多服务器情况下,当一个服务器向另一个服务器发出请求时,必须将自己的委托线程标识号传递给目标服务器.这样,操作系统为一个线程提供所有服务所使用的处理机时间都将计算到用户线程中去.

为了完成以上功能,必须对微内核的消息传递机制进行扩充,使用户在请求服务时能将线程的标识传递给服务器,服务器在接收消息时能接收到委托线程标识,所有这些操作必须对用户透明.微内核的消息传递机制由消息发送和消息接收两部分组成.通过在这两个原语中加入以下逻辑来实现委托线程标识的发送和接收.

```
SEND:
    IF (当前线程结构中有委托线程标识)
      将委托线程标识传递出去
    ELSE
        将当前线程的标识传递出去
RECEIVE:
    IF (当前线程是服务器)
      将委托线程号放入服务器线程结构
```

在发送原语中,可以将委托线程标识从一个服务器传递到另一个服务器.在接收逻辑中,通过增加服务器标识的判断可以避免非服务器线程之间的偶发通讯而导致的用户线程的计时错误.

4　结　论

微内核技术是当今操作系统发展的最新成果.在体系结构方面,它采用了面向对象技术来描述操作系统内核对象,提出并实现了基于客户服务器体系结构的操作系统.在算法方面,提出了许多高效新颖的算法,如线程及处理机调度算法、写时拷贝算法、与硬件无关的存储管理算法以及精确计时算法等等.在国产微内核操作系统 COSIX2.0的研制过程中,通过对国外微内核技术的消化和研究,提出并实现了一些新的算法和模型,改进了系统的性能,提高了系统的可靠性,做到了有所继承,有所创新.目前,我们正在进行基于微内核的 JAVA 虚拟机、支持服务质量(Quality of Services)的调度系统及微内核热重启(Hot Restart)技术的研究.以上内容是我们一部分研究工作的总结.

参考文献

1　Black David L *et al*. Microkernel operating system architecture and mach. In: Proceedings of the USENIX Workshop on Microkernels and Other Kernel Architecture. Seattle, Washington, 1992. 11～30

2　Cheriton David R *et al*. A caching model of operating system kernel functionality. In: Proceedings of the 1st USENIX Symposium on Operating Systems Design and Implementation. Monterey, California, 1994. 179～193

3　Custer Helen. Inside Windows NT. Redmond Wash.: Microsoft Press, 1993

4　Graham Hamiton, Panos Kougiouris. The spring nucleus: a microkernel for objects. Technical Report, Sun Microsystems Laboratories, Inc., 1993

5　Rozier M *et al*. Overview of the chorus distributed operating system. In: Proceedings of the USENIX Workshop on Micro-kernels and Other Kernel Architectures. Seattle, Washington, 1992. 39～69

Some Researches on Operating System Microkernel Technology

PAN Qing ZHANG Xiao-qing

(*Institute of Command Technology Commission of Science Technology and Industry for National Defence Beijing* 101407)

Abstract Some of the research work which the authors have done on microkernel in the last 5 years are described in this paper. Three new algorithms have been proposed. The first algorithm combines task schedule and thread schedule algorithms together to improve the efficiency of the thread schedule system, it also provides a more fairness scheduling algorithm for microkernel operating system. The second algorithm is an enhanced copy-on-write algorithm, it combines copy-on-write and copy-on-reference algorithms together to make it more suitable on 1386 architecture. The last algorithm is a new timing algorithm for microkernel operating system, it solves the inaccuracy timing problem which is caused by traditional timing algorithm running under the microkernel architecture.

Key words Microkernel, schedule, virtual memory management, timing model.