计算机科学 1997Vol. 24№. 2

微内核操作系统计时模型的设计和实现

潘 清 张晓清

(国防科工委指挥技术学院 北京 101407)

摘 要 In the microkernel operating system, the new architecture also brings out new problems, such as timing problem. Through the establishment of a new timing model by introducing the concept of client thread and the extension of message passing mechanism, we provide a new timing mechanism for the microkernel operating system.

关键词 Process, Task, Thread, Microkernel, Timing model.

1. 引言

在传统的操作系统中,用户对操作系统服务的请求都是通过系统调用由操作系统内核直接完成的,操作系统能准确地计算出用户在系统态和用户态所用的时间。但在微内核操作系统下,许多服务都移出了内核,用户对操作系统服务的请求中的绝大多数都是通过消息传递给服务器,由服务器来完成,再通过消息将结果发送给用户。因此,如果采用传统的方法进行计时,操作系统为用户请求提供的服务的时间将记入服务器中而不是用户任务中,这样,操作系统中的计时收费系统就变得如同虚设,用户也很难统计出自己程序所用的处理机的时间。例如,基于微内核结构的 MACH3.0 系统中,系统计时就有上述问题。本文在传统的操作系统计时机制的基础上,通过引入新的计时模型,扩充系统的消息传递机制来解决基于微内核结构的操作系统计时问题。

2. 传统的操作系统计时模型

在传统的操作系统中,对于每个进程的处理机时间使用量,在每个进程结构中都设有处理机时间使用的统计域,系统计时一般是放在处理机时钟中

究"一文中详细阐述),目前正在进一步深入研究预测因素的离散性和并发性;同时在 OOFMD 系统的基础上研究适应瞬息万变的市场需求的生产计划快速反应系统。OOFMD 系统的结构和设计思想,对于与 GIS 信息相关的问题研究(如:区域农业病虫害预测)具有重要的参考价值。本文提到的重用思想,事件驱动思想和集成技术对于提高软件的质量和产量有重要的指导意义。

本课题的研究与本文的写作中,得益于黄德才副教授,徐守真副教授的指导,在此表示万分感谢。

参考文献

[1]James W. Hooper et al., Software Reuse: Guideline and Methods, Plenum Press, New York, 1991

[2]杨芙清、邵维忠、梅宏、面向对象的 CASE 环境青鸟 I 型系统的设计与实现。中国科学,A 辑,(5)1995

[3]吕丽民等,面向对象建模与设计,人民邮电出版社,(8)1996

[4]寿志勤等,集成环境下的预测应用网络系统的设计与实现,软件学报,(9)1995

[5]暴奉贤、陈宏立,经济预测与决策方法,暨南大学出版社,1991

[6]仲萃豪等,信息系统的开发方法及体系模型,软件学报,6(增刊),1995

[7]郭荷清等,MIS 系统的程序生成方法,软件学报,6(增刊),1995

断服务程序中，系统时钟每隔一段时间中断处理机一次来完成与时间相关的系统服务。用户进程的时间统计是其中的一部分。系统一般采用如下代码段来统计：

```
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
```

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，因为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，操作系统在完成用户请求时，实际上是在用户任务空间中执行的。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(时钟中断周期长度)加入被中断的进程中，而不管该进程是否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，而且在新的操作系统中引入了细粒度的并行执行部件—线程时，对于线程的计时也可以采用和进程相同的方法。

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 MACH3.0 中引入了基于时间戳的精确计时机制。在 MACH3.0 系统中，每个线程有两个计时器，一个用于用户态计时，一个用于系统态计时；系统中专门设立了一个系统计时器来统计中断所用的时间。这些计时器的工作方式和秒表相同，系统在任意时刻只有一个计时器在工作。当用户线程进入用户态时，切换到用户态计时器；当用户线程进入到系统态时，切换到线程的系统态计时器；当系统发生中断时，切换到系统计时器。计时操作从时钟中断服务程序移到了用户进入和退出系统，系统进入中断处理和退出中断处理处及线程切换处。

```
系统调用入口：
    统计当前态进程所用的时间
    切换到当前进程的系统态计时器
        .
        .(系统调用处理)
        .
    统计当前进程系统态所用的时间
    切换到当前进程的用户态计时器
系统调用出口：
中断入口：
    统计进入中断之前的进程所用的时间
    切换到系统的中断处理计时器
        .
        .(中断处理)
        .
    统计系统的中断处理时间
    切换到进入中断前的进程计时器
中断出口：
```

采用这种计时方法能精确地统计出每个进程所用的处理机时间，但是这种方法开销比较大，而且系统中需要一个精确的计时器。

3. 基于顾客和服务器模型的计时模型

在基于顾客和服务器模型的操作系统中(见图1)，操作系统提供的大部分服务都是由服务器来完成的，服务器具有自己独立的任务空间，微内核向服务器提供最基本的操作系统服务，如任务管理，线程调度，内存管理及任务间通讯等，服务器向用户提供传统的操作系统服务，如进程管理，文件管理，网络通讯等等。

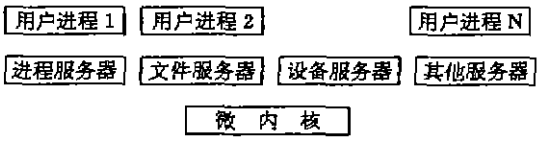

图 1　微内核结构的操作系统结构

以 MACH3.0 系统为例，用户发出的 OPEN 系统调用是通过以下方式处理的(见图 2)：

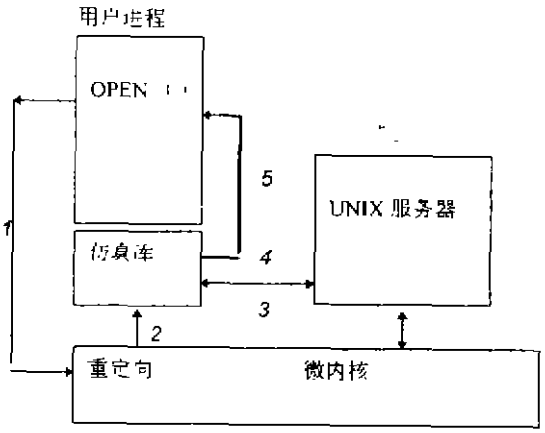

图 2　用户请求的处理过程

在MACH3.0系统中，UNIX系统服务是通过UNIX系统服务器来实现的，当用户进程调用UNIX系统调用时，和传统的UNIX系统一样，自陷进入系统，但是微内核不做任何处理，微内核将系统调用重新定向到用户进程的仿真库，在仿真库中通过消息将系统调用的参数传递给UNIX服务器，UNIX服务器通过请求微内核的服务来完成用户请求，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的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，本文引入了委托线程的概念，建立了新的用户进程计时模型。在client/server模型中，顾客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，服务器是被动的，当用户请求时，服务器为用户完成一定的服务，即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。在单服务器体系结构下(见图3)，一个服务器为多个用户线程提供服务，用户通过消息将服务请求发送给服务器，服务器中有多个线程接受并处理这些请求。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，并在系统时钟中断服务程序中增加为委托线程计时的代码：

```
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
```

当系统为服务器统计处理机时间使用量时，如果发现当前线程结构中的委托线程域不为空，则将处理机时间片也记入委托线程结构中，这样就达到了将系统为用户线程服务所用的时间记入用户线程中的目的。如果我们将服务器中服务所用的时间看成是操作系统服务所用的时间，使用如下代码则可以将服务器中所用的时间加到用户线程的系统态时间中去。

```
if(当前线程结构中有委托线程)
    增加委托线程的系统态处理机时间使用量
```


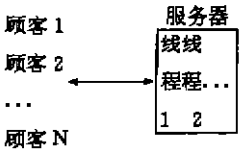

图3 单服务器结构

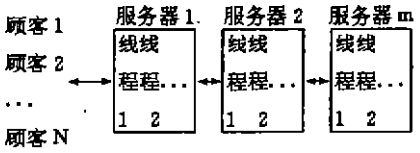

图4 多服务器结构

在多服务器体系结构下(见图4)，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，在这种多服务器情况下，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的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，必须对操作系统的消息传递机制进行扩充，使顾客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。操作系统的消息传递机制由两个部分组成，消息发送和消息接收。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偶发的通讯而导致的用户线程的计时错误。

上面介绍了两种基于传统操作系统的计时模型，一种是在分时系统中广泛使用的基于统计的计时系统。另一种是MACH3.0系统提出的基于时间戳的精确计时系统。说明了这些计时系统在微内核操作系统中计时的缺陷。作者通过在服务器中引入委托线程的概念，建立了基于微内核体系结构的操作系统的计时模型，并通过扩充系统消息传递机制在MACH3.0系统上加以实现，取得了满意的结果。新的计时模型几乎不增加任何系统开销，而且很容易可以看出新的模型是建立在传统计时模型之上的，是传统计时模型的发展。作者认为这种计时模型可以作为微内核结构操作系统计时的通用模型。

4. 实验情况

下面通过使用UNIX命令time来获取用户程序在系统中所用的时间来比较两种模型的计时情况。实验程序是一个UNIX管道读写操作，其结果如下(时间单位为秒)：

表1 两种计时机制的比较

系统	用户态时间	系统态时间	程序在系统中停留时间
MACH3.0	4.05	5.15	24.68
改进的	3.94	20.71	24.70

从上表中我们可以看到在这两种不同的计时系统中，用户程序所用的用户态时间及用户程序在系统中停留的时间几乎相同，但是，系统态时间却相差很大。可以得出结论，MACH3.0中的计时是非常不准确的，因为系统服务器为它服务所用的处理机时间都记入到服务器中去了。在改进的MACH3.0系统中，将服务器为用户服务所用的处理机时间记入用户进程中，从而使微内核操作系统的计时系统和UNIX系统计时系统语义保持一致。

```
#define  N1024
char     x[N];
main( )
{
    int fd[2],i;
    pipe(fd);
    for(i=10000;i>0;i--){
        write(fd[1],x,N);
        read(fd[0],x,N);
    }
}
```

参考文献

[1]M. J. Acceta et al. ,Mach:A New Kernel Foundation for UNIX Development, In Proc. of the Summer 1986 USENIX Conf. July 1986

[2]D. L. Black et al. , Microkernel Operating system architectures and Mach, J. Infor. Processing, Dec. 1991

[3]D. L. Black, Scheduling and Resource Management Techniques for Multiprocessors, PhD dissertation, School of Computer Science, Carnegie Mellon University, July 1990

[4]D. R. Cheriton, The V Distributed System, CACM, 31(3)1988

[5]Greg, Thiel, LOCUS operating system, a transparent system, Computer Comm. , 14(6)1991

[6]Allan Bricker et al. , Architectutal issues in microkernel based operating systems: the CHORUS experience, Same to [5]

[7]Peter D. Varhol, Small Kernels Hit It Big, BYTE, Jan. 1994

[8]Nicholas Baran, Operating Systems Now and Beyond, BYTE special Edition. OUTLOOK·1992