从多地址空间到单地址空间再到无地址空间*

刘福岩[1,2], 尤晋元[1]

[1](上海交通大学 计算机科学与工程系,上海 200030);

[2](华北工学院 计算机科学与技术系,山西 太原 030051)

E-mail: lfy428@263.net

http://www.cs.sjtu.edu.cn; www.ncit.edu.cn

摘要: 在分析单地址空间操作系统的优点及其所存在问题的基础上,提出了无地址空间操作系统的思想,并介绍了一个无地址空间操作系统原型.在无地址空间操作系统中,没有进程虚拟空间的概念,指令直接对文件寻址,进程直接在文件上运行.与单地址空间操作系统相比,无地址空间操作系统不仅具有单地址空间操作系统的优点,而且能够避免单地址空间操作系统中存在的不足.

关键词: 操作系统;地址空间;存储管理;单地址空间;文件系统

中图法分类号: TP316 **文献标识码**: A

在传统的操作系统中,每个进程都在属于自己的虚拟空间中运行.操作系统通过为每个进程构造各自独立的虚拟空间,很自然地实现了进程数据隔离,从而实现了对进程数据的保护.但是同时,多个虚拟空间的存在,不仅限制了进程之间对复杂数据结构的共享,而且增加了进程切换的代价.通过使所有进程运行于同一个虚拟空间中,可以避免由于存在多个虚拟空间而引起的问题,这就是单地址空间操作系统(single address space operating system,简称 SASOS)[1~5]的思想.与传统的操作系统相比,单地址空间操作系统无疑具有许多优点,而且它也是当前操作系统领域的研究热点之一[6],但是,单地址空间操作系统本身也存在一些问题.在本文中,我们在单地址空间操作系统的基础上提出了无地址空间操作系统.我们认为,在操作系统中取消进程的虚拟空间,使进程直接在文件上运行,不仅具有单地址空间操作系统的优点,而且还可以解决单地址空间操作系统中存在的问题.

本文首先介绍了单地址空间操作系统及其存在的问题,然后提出了无地址空间操作系统及其实现原理,并介绍了一个无地址空间操作系统原型,最后分析了无地址空间操作系统和单地址空间操作系统相比所具有的优点.

1 单地址空间操作系统及其优缺点

当前,计算机正进入 64 位机时代.64 位处理机允许进程访问多达 2^{64} 字节的虚拟空间,2^{64} 字节的存储容量不仅可以"存储"一个进程在其生命期中访问的所有数据,甚至在一个由几百台计算机组成的分布式系统中,所有存储介质上的数据(既包括内存,也包括外存)也可以"存储"在一个 2^{64} 字节的虚拟空间中.2^{64} 字节的虚拟空间的容量,使得操作系统不再需要为每个进程构造各自独立

* 收稿日期: 1999-05-24; 修改日期: 1999-12-08

基金项目: 上海市科技发展基金资助项目(985115035;995115014)

作者简介: 刘福岩(1966—),男,河北滦南人,博士生,讲师,主要研究领域为计算机网络,系统软件,分布式系统;尤晋元(1939—),男,江苏常州人,教授,博士生导师,主要研究领域为操作系统,分布和移动计算.

的地址空间,它能使所有进程运行在同一个虚拟地址空间中,不同的进程在虚拟空间中处于不同的区域,这就是单地址空间操作系统的思想[1~5].

1.1 单地址空间操作系统的优点

(1) 便于在进程之间共享复杂数据结构

高级语言中指针的含义是指进程虚拟空间地址.在单地址空间操作系统中,指针的含义为系统虚拟空间地址,该虚拟空间在系统启动时创建,在系统关机时消亡,并在下次系统启动时根据关机时的状态重建.一个指针类型的数据并不限定于某一特定的进程虚拟空间,也不会随着进程消亡而失去意义,甚至系统关机也不会失去意义.因此,指针类型的数据,既可以在文件中保存,也可以在各个进程之间共享,从而可以实现在进程之间共享像图、树等包含指针类型数据的复杂数据结构[1~4].

(2) 便于处理永久性数据

2^{64}字节的虚拟空间容量,使得单地址空间操作系统能够把系统中所有的文件映射到系统虚存中.在进程创建文件的同时创建了该文件在虚存中所对应的映射区域,只要文件存在,文件在系统虚存中对应的映射区域就一直存在,如果进程具有访问该映射区域的权限,那么进程即可像访问内存数据一样访问文件中的数据.因此,在单地址空间操作系统中,文件仅仅是系统虚空间中永久性的一段存储区域而已,除此以外,和其他存储区域没有任何区别,从而对进程而言,所有数据都在系统虚空间中,既不需要读写文件,也不需要存储映射文件[1,3~5].

(3) 进程切换的代价较低

在单地址空间操作系统中,所有进程运行在一个虚拟空间中.执行进程切换不需要执行虚拟空间的切换.TLB 的内容不需要刷新,即使按虚拟地址访问 Cache,也不需要刷新 Cache.因此,执行进程切换的代价较低[2,5].

(4) 进程通信的代价较低

在单地址空间操作系统中,所有进程运行在一个虚拟空间中,执行进程通信不需要在不同虚拟空间之间拷贝数据,因此,进程通信的代价较低[5].

1.2 单地址空间操作系统中存在的问题

(1) UNIX 仿真器的实现问题和复杂数据结构的复制问题

在 UNIX 系统中,fork()功能调用根据父进程图像复制出的子进程图像.在单地址空间操作系统中,不同进程的数据在系统虚空间中处于不同的地址区间,不可能在相同的地址复制出子进程图像,因此,难于实现 UNIX 仿真器[1,2].

对复杂数据结构的复制也存在类似的问题.如果对象中包含着指向对象内部数据项的指针,那么在复制对象时,不仅需要在系统虚拟空间中复制对象数据,还要对新对象中的指针进行相应的调整,使指针指向新对象内部的数据项.因此,复制对象时需要了解对象内部的结构,使得复制对象实现起来很复杂,从软件工程的角度分析也很不合理.

(2) 系统虚拟空间的分配和回收问题

在单地址空间操作系统中,系统中的所有数据(既包括进程在内存中创建的临时性数据,也包括外存上的文件)全部映射到系统虚拟空间中.虽然系统虚拟空间具有2^{64}字节的存储容量,但考虑到系统中所有数据全部映射到了系统虚拟空间中,它们在虚空间中不能相互冲突,因此,系统虚空间的分配和回收算法是很复杂的[2].

(3) 存储保护问题

在单地址空间操作系统中,存储保护是基于页的.每次缺页,系统都要进行安全性检查.执行进程切换时也需要对地址映射数据结构进行修改.与基于段的存储保护相比,基于页的存储保护机制不仅比较复杂,效率也较低[7,8].

(4) 需要CPU支持64位虚地址,传统的32位处理机上无法实现单地址空间操作系统

传统的32位处理机虚拟空间的"容量"只有$2^{32}=4$G字节,不可能把系统中所有数据"存储"在一个惟一的系统虚拟空间中,因此无法实现单地址空间操作系统.

2 无地址空间操作系统及其实现

基于单地址空间操作系统的优点及其存在的问题,我们提出了无地址空间操作系统.该操作系统借鉴了单地址空间操作系统的思想,通过下面3点改进了单地址空间操作系统.该操作系统不仅具有单地址空间操作系统的优点,而且还能够解决单地址空间操作系统中存在的问题.

(1) 存储管理由页式虚拟存储管理改进为段页式虚拟存储管理,因此,系统虚拟空间是两维的,包括段和段内偏移量.

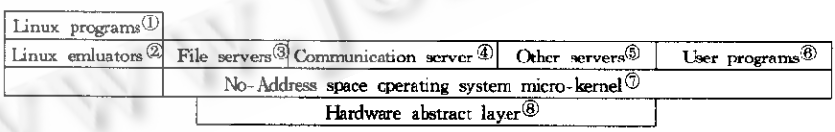
(2) 系统中的每个段就是系统打开的一个文件,文件数据在外存上的存储由文件服务器管理.进程打开文件时,系统在内存中建立一个段,关闭文件时清除对应的段.当进程访问段中数据时,操作系统通知文件服务器把数据读入到内存中的文件缓冲区中;当内存资源比较紧张或者进程关闭文件时,把文件缓冲区中的数据反写回文件中.

(3) 操作系统把进程的虚拟地址直接映射到内存的文件缓冲区内,从而对进程而言,好象进程直接在文件上运行,感觉不到进程虚拟空间的概念,因此,我们称之为无地址空间操作系统.

在无地址空间操作系统中,认为进程使用的临时性数据是一种特殊的文件,该文件在打开时创建,关闭时删除.因此,系统不支持临时性数据,只支持永久性数据——文件.

2.1 无地址空间操作系统的总体结构

我们的目标是实现一个结构上如图1所示的无地址空间操作系统,该操作系统的结构是基于微内核的.在微内核之下通过硬件抽象层使系统独立于硬件平台;在微内核之上通过实现Linux仿真器实现对UNIX软件的兼容.用户可以在微内核之上开发用户程序,也可以在Linux环境下开发和使用Linux程序.



①Linux程序,②Linux仿真器,③文件服务器,④通信服务器,⑤其他服务器,⑥用户程序,⑦无地址空间操作系统微内核,⑧硬件抽象层.

Fig. 1 The architecture of no-address space operating system
图1 无地址空间操作系统的总体结构

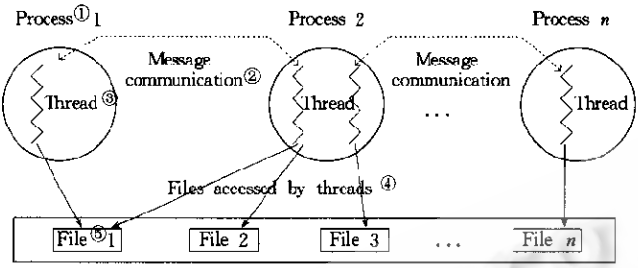
2.2 无地址空间操作系统微内核的结构

我们实现了一个无地址空间操作系统微内核的原型.该原型为用户构造了4种抽象,提供了4类,共11条系统功能调用.各个抽象之间的关系如图2所示.

2.2.1 微内核支持的抽象

(1) 进程.进程是微内核为了对系统中的资源进行有效的管理和控制而构造的抽象,可以认为

进程就是资源的持有者.系统通过为进程分配和回收资源实施对资源的控制,防止某些进程使用过多的系统资源.为进程分配的资源包括:线程、消息、文件和内存.



①进程,②消息通信,③线程,④线程访问的文件,⑤文件.

Fig. 2 The abstrcts constructed by micro-kernel

图2 微内核的结构

(2) 线程.线程是程序的执行者.它从文件中读取指令,解释执行实现对文件数据的操作.

(3) 文件.文件是系统中所有信息的惟一载体.线程执行的程序和访问的数据都存在于被打开的文件中.系统利用 capability 机制对打开的文件实施保护.只要拥有文件的 capability,任意线程都可以访问任意文件中的数据,即使属于其他进程的文件也可以访问.

(4) 消息.消息是实现线程间通信的一种抽象.通过消息通信可以实现线程之间的相互控制,也可以在线程之间交换少量的数据.消息是定长的,如果在线程之间需要交换大量数据,可以通过共享文件实现.

2.2.2 微内核提供的系统功能调用

(1) 进程和线程管理类功能调用,包括创建进程、终止进程、创建线程、终止线程和分配资源.

(2) 通信类功能调用,包括发送消息和接收消息.

(3) 文件类功能调用,包括打开文件、关闭文件.

(4) 输入/输出类功能调用,包括以下两个功能调用:

① 访问文件:当线程执行时,如果产生缺页异常现象,则异常处理程序调用该功能调用,在内存中为读写文件分配存储页面,同时,向文件服务器发读写文件请求消息,通知文件服务器执行读写操作,如果有必要使线程睡眠.线程也可以直接调用该功能调用,实现对文件数据的预读.

② 读写完成:文件服务器执行完读写操作后执行该调用,通知内核读写完成,修改页表,唤醒线程.

2.2.3 微内核的工作过程

无地址空间操作系统的实现原理基于传统的段页式虚拟存储管理.工作过程可以简单地描述为以下几点:

(1) 操作系统采用段页式虚拟存储管理,显然,这是很容易实现的.现在许多 CPU 芯片都支持段页式虚拟存储.因此,指令的地址码由两部分组成:段和段内偏移量.

(2) 系统的一个段不再是进程虚拟空间的一个组成部分,而是记录系统打开文件的一个数据结构,对应于外存上被打开文件中的一段区域.在系统段表中保存着所有打开文件的信息,其中包括该段所对应的外存上的文件,在文件中的位置、段长、权限,该段对应的页表等.

(3) 进程所属线程执行打开文件功能调用,操作系统执行打开文件操作,在内存建立该文件对应的段表和页表,把段号作为被打开文件的标识数返回给线程.

(4) 线程通过文件地址访问文件中的数据.当线程执行指令时,根据地址码中的段号,也就是

通过被打开文件的标识数,找到相应文件在内存中所对应的页表,根据地址码中的偏移量,也就是文件偏移量找到对应的页表项.

(5) 如果该页表项指示该页在内存中,则成功执行地址映射,若执行指令成功,线程继续执行.

(6) 如果该页表项指示该页不在内存中,地址转换机构产生缺页异常现象,则由操作系统中异常处理程序调用访问文件功能调用,挂起相应的线程,向文件服务器发读请求消息,请求读入该页对应文件中的数据.

(7) 文件服务器执行完读操作后调用读写完成系统功能调用,修改页表,唤醒线程,重新启动指令执行.

(8) 当内存资源比较紧张时,操作系统向文件服务器发写请求消息,通知文件服务器,把内存中的数据反写回文件中,释放占用的内存供其他进程使用.

(9) 最后关闭文件,通知文件服务器把内存中的数据反写回文件中,释放内存及相应的段表和页表.

2.3 微内核的实现和测试

开发一套完整的操作系统软件是一项巨大的软件工程.我们当前的研究仅仅集中于设计出一个合理的无地址空间操作系统微内核.该内核通过硬件抽象层独立于具体的硬件平台,通过微内核上的Linux仿真器实现对Linux软件的兼容.为此,我们用C++语言实现了一个无地址空间操作系统微内核原型,并通过仿真硬件抽象层对该内核进行了测试.测试方案及结果如下:

(1) 单线程顺序访问文件测试.首先执行打开文件系统功能调用,然后通过仿真硬件程序触发访问文件功能调用,从文件头向文件尾顺序读写文件,观察文件服务器执行读写操作的序列.测试结果如下:开始执行时,文件服务器以内存页面大小为单位,顺序读入文件中的数据;当读入的数据达到内存容量时,读操作和写操作交替进行;如果线程挂起,文件服务器读写文件的操作随之暂停;当关闭文件时,文件服务器执行连续的写文件操作后随之停止读写.测试结果表明,微内核在单线程顺序访问文件时工作正常.长时间多次重复测试未见异常.

(2) 多线程顺序访问文件测试.首先执行打开文件系统功能调用,然后创建多个线程,每个线程通过仿真硬件程序并发地调用访问文件功能调用,从文件头向文件尾顺序读写文件.观察文件服务器执行读写操作的序列.测试结果如下:开始执行时,文件服务器以内存页面大小为单位,顺序读入文件中的数据;当读入的数据达到内存容量时,读操作和写操作同时出现.与(1)不同的是,读操作和写操作不再交替进行,读写文件的请求者也不是惟一的一个线程,而是随机出现各个线程.关闭文件时,文件服务器执行连续的写文件操作后随之停止读写.测试结果表明,微内核在多线程顺序访问文件时工作正常.长时间多次重复测试未见异常.

(3) 多线程顺序和逆序访问文件测试.把线程分成两组,一组从文件头向文件尾顺序访问文件,一组从文件尾向文件头逆序访问文件.测试结果如下:开始执行时,文件服务器以内存页面大小为单位,读入文件中的数据;当读入的数据达到内存容量时,读操作和写操作同时出现.与(2)不同的是,文件服务器上出现两个读写序列,一个从文件头向文件尾,一个从文件尾向文件头,两个序列在中间交汇后继续向前推进.显然,系统中出现了两个工作集,一个从文件头向文件尾移动,一个从文件尾向文件头移动.测试结果表明,微内核在多线程顺序和逆序同时访问文件时工作正常.长时间多次重复测试未见异常,多组线程测试也未见异常.

(4) 多线程乱序访问文件测试.各个线程按正态分布访问文件,观察文件服务器执行读写操作的序列:当线程访问的文件数据集中于一个或几个区域时,系统出现稳定的工作集,读写文件操作

明显减少;当线程访问的文件数据比较分散时,会出现大量随机的读写文件操作.长时间多次重复测试未见异常.

我们仅仅完成了微内核的原型,还没有在具体的硬件平台上实现,但以上测试至少说明了我们实现的微内核原型设计是合理的,运行也是可靠的.

2.4 性能测试

Hermann Hartig 在文献[9]中对以下 3 个系统进行了性能测试,并分析了测试结果.

(1) 单一大内核 Linux;

(2) 在 L4 微内核上实现的 Linux 仿真器;

(3) 在 Mach 微内核上实现的 Linux 仿真器.

为了评价无地址空间操作系统微内核的性能,我们也实现了一个 Linux 仿真器.该仿真器非常简单,仅仅实现了几条系统功能调用.我们开发该仿真器的目的仅仅是为了对微内核进行性能测试,以便和文献[9]中的测试结果进行比较,从而评价微内核的性能.我们选择实现了 5 条系统功能调用:第 1 条取进程标识数(getpid),用于测试系统功能调用的执行时间(getpid 的执行非常简单);第 2 条向空文件中写数据,用于测试读写文件的执行时间(不考虑具体设备的执行时间);第 3 条存储影射文件,用于测试存储管理的执行时间;第 4 条创建管道(pip),用于测试通过消息传递模拟实现管道的执行时间;第 5 条创建空进程,用于测试进程管理的执行时间.

为了使测试结果具有可比性,我们也在 133M Pentium 上进行了测试,并和文献[9]中的测试结果进行比较.文献[9]中的测试结果和我们的测试结果同时列入表 1 中.

Table 1 The executing time of system calls (platform: 133M Pentium) (ms)

表 1 系统功能调用执行时间的测试结果(硬件平台:133M Pentium) (微秒)

System call①	Linux	L4 Linux	Mach Linux	NASOS Linux
Getpid	1.69	4.55	111.9	7.83
Write to null file	2.74	6.67	124.1	8.36
Mmap (4KB)	25.2	35.0	439.6	37.5
Mmap (8MB)	53.7	54.0	561.9	37.5
Pip	31.0	62.3	721.6	43.2
Create null process	983.0	2561	3572	1357

①系统调用.

从表 1 可以看出,无地址空间操作系统微内核上的 Linux 仿真器系统功能调用的执行时间远远小于 Mach Linux,接近于 L4 Linux 的执行时间,大于单一大内核 Linux 仿真器系统功能调用的执行时间.这说明无地址空间操作系统微内核在技术上是可行的,但仍需进一步完善和提高.

3 无地址空间操作系统的优点

3.1 具有单地址空间操作系统的优点

在内部实现上,无地址空间操作系统就是段页式的单地址空间操作系统,因此具有单地址空间操作系统所具有的优点.

3.2 可以解决单地址空间操作系统存在的问题

(1) UNIX 仿真器的实现问题和复杂数据结构的复制问题

在 UNIX 系统中,进程虚拟空间是一个一维页式的虚拟空间.在无地址空间操作系统中实现 UNIX仿真器时,可以使每个 UNIX 进程图像对应于一个文件,通过复制父进程图像对应的文件即

可实现 fork()系统功能调用.子进程图像在虚空间中不会和父进程图像相冲突,因此可以解决 UNIX 仿真器的实现问题.复杂数据结构的复制问题也可以通过复制文件来解决.

(2) 系统虚拟空间的分配和回收问题

在无地址空间操作系统中只有文件,而没有虚拟空间的概念,也就不需要对系统虚拟空间执行分配和回收,因此,这个问题在无地址空间操作系统中不存在.

(3) 存储保护问题

无地址空间操作系统采用段页式虚拟存储管理,在段一级而不是在页一级实施存储保护,每次进程对段寄存器赋值,系统执行安全性检查.而在单地址空间操作系统中,采用基于页的存储保护,每次缺页,系统即执行安全性检查.与基于页的存储保护相比,基于段的存储保护算法和数据结构都比较简单,执行算法的代价也较低.

(4) 无地址空间操作系统不要求必须在 64 位处理机上运行,在传统的 32 位处理机上也可以运行.

4 结束语

我们在单地址空间操作系统的基础上,提出了无地址空间操作系统,并实现了一个无地址空间操作系统的原型.与单地址空间操作系统相比,无地址空间操作系统不仅具有单地址空间操作系统的优点,而且能够解决单地址空间操作系统中存在的问题,同时,还具有一些其他优点.我们认为,无地址空间操作系统是一种比较合理的操作系统模型,值得采用和推广.

References:

[1] Wilkinson, T., Murray, K., Russell, S., *et al*. Single address space operating systems. Technical Report, UNSW-CSE-TR-9504, Sydney: University of New Sourth Wales, 1995.

[2] Chase, J., Levy, H., Baker-Harvey, M., *et al*. How to use a 64-bit virtual address space. Technical Report, 92-03-02, Department of Computer Science and Engineering, University of Washington, 1992.

[3] Chase, J., Levy, H., Tiwary, A. Using virtual addresses as object references. In: Proceedings of the 2nd International Workshop on Object Orientation in Operating Systems. Los Alamitos, CA: IEEE Computer Society Press, 1992. 245~248.

[4] Chase, J., Feeley, M., Levy, H. Some issues for single address space systems. In: Proceedings of the 4th IEEE Workshop on Workstation Operating Systems. Los Alamitos, CA: IEEE Computer Society Press, 1993. 150~154.

[5] Murray, K., Saulsbury, A., Stiemerling, T., *et al*. Osmon: design and implementation of an object-orientated 64-bit single address space microkernel. In: Proceedings of the 2nd USENIX Symposium on Microkernels and Other Kernel Architectures. Berkeley, CA: USENIX Association, 1993.

[6] Fan, Jian-ping, Li, Guo-jie. Overview of multiprocessor operating systems. Computer Research and Development, 1995,32(1):1~5 (in Chinese).

[7] Elphinstone, K., Russell, S., Heiser, G. Issues in implementing virtual memory. Technical Report, UNSW-CSE-TR-9411, Sydney: University of New Sourth Wales, 1994.

[8] Koldinger, E., Chase, J., Eggers, S. Architectural support for single address space operating systems. In: Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V). New York: Association for Computer Machinery, 1992.

[9] Hartig, H., Hohmuth, M., Liedtke, J., *et al*. The performance of μ-kernel based system. Operating System Review, 1997,31(5):66~77.

附中文参考文献:

[6] 樊建平,李国杰.并行操作系统的现状和发展趋势.计算机研究与发展,1995,32(1):1~5.

From Multi-Address Space to Single Address-Space and to No-Address Space*

LIU Fu-yan[1,2], YOU Jin-yuan[1]

[1](*Department of Computer Science and Engineering, Shanghai Jiaotong University, Shanghai* 200030, *China*);

[2](*Department of Computer Science and Technology, North China Institute of Technology, Taiyuan* 030051, *China*)

E-mail: lfy428@263.net

http://www.cs.sjtu.edu.cn; www.ncit.edu.cn

Abstract: In this paper, based on the discussion of SASOS (single-address space operating system), as well as its advantages and disadvantages, the concept and principle of NASOS (no-address space operating system) is put forword and a NASOS prototype is discussed. In the NASOS, there is no process virtual address space, instructions access files directly and processes run on files. Compared with SASOS, not only does NASOS have the advantages of SASOS, but also avoids its disadvantages.

Key words: operating system; address space; memory management; single address space; file system

* Received May 24, 1999; accepted December 8, 1999

Supported by the Sci-Tech Development Foundation of Shanghai of China under Grant Nos. 985115035, 995115014