实现数据存储、数据计算和资源管理的分离*

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](上海大学 计算机工程与科学学院,上海 200072)

Separating Data Storage, Data Computation and Resource Management

LIU Fu-Yan[1,2+], YOU Jin-Yuan[1]

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

[2](School of Computer Engineering and Science, Shanghai University, Shanghai 200072, China)

+ Corresponding author: Phn: +86-21-52541638, Fax: +86-21-52541638, E-mail: lfy428@hotmail.com, http://www.sjtu.edu.cn

Received 2003-07-06; Accepted 2004-02-16

Liu FY, You JY. Separating data storage, data computation and resource management. *Journal of Software*, 2004,15(6):850~857.

http://www.jos.org.cn/1000-9825/15/850.htm

Abstract: It's impossible for traditional operating systems to separate the abstracts for data storage (process virtual address space), for data computation (thread), and for resource management (process itself). This paper first analyzes the problems due to not able to separat these three abstracts. On the base of the analysis, the idea that the three abstracts should be separated is proposed, and then, the operating systems based on virtual address spaces on files (OS-BVASF) is developed. Then, OS-BVASF's architecture model thread migration and instruction accessing file that implement separation of the three abstracts is investigated. Finally, its implementation, test, and performance evaluation are discussed. The work in this paper shows it is feasible to separate data storage, data computation, and resource management in operating systems.

Key words: operating system; abstract; resource management; address space; thread; process

摘 要: 在传统操作系统中,数据存储的抽象(进程虚拟地址空间)、数据计算的抽象(线程)和资源管理的抽象(进程)是不可分离的.首先分析了在操作系统中由于 3 类抽象不可分离而存在的问题,根据分析提出了数据存储抽象、数据计算抽象和资源管理抽象互相分离的思想,进而根据这一思想提出了虚拟地址空间基于文件操作系统,分析该操作系统的体系结构模型,研究了实现 3 类抽象分离的线程迁移技术和指令对文件寻址技术,最后讨

* Supported by the National Natural Science Foundation of China under Grant No.60173033 (国家自然科学基金); the Youth Technology Research Foundation of Shanxi Province under Grant No.20021016 (山西省青年科技研究基金); the Defense Science and Technology Key Laboratory Foundation under Grant No.51484030301JW0301 (总装备部国防科技重点实验室基金); the Research & Development Item on the University Science & Technology of Shanxi Province of China under Grant No.MZ20030902 (山西高校科技研究开发项目); the Important Science and Technology Key Item of Shanghai Science and Technology Bureau under Grant No.02DZ15013 (上海市科委重大科技攻关项目)

作者简介: 刘福岩(1965－),男,河北滦南人,博士,副教授,主要研究领域为操作系统,分布移动计算,中间件,网格,计算机辅助设计与制造;尤晋元(1939－),男,教授,博士生导师,主要研究领域为操作系统,分布对象计算.

论了系统的实现、测试和性能评价.此项研究说明了在操作系统中实现数据存储、数据计算和资源管理的分离是可行的.

关键词: 操作系统;抽象;资源管理;地址空间;线程;进程

中图法分类号: TP311 **文献标识码**: A

操作系统的功能就是构造一定的抽象并提供操作这些抽象的功能调用,构造的抽象至少包括 4 类:数据存储的抽象、数据计算的抽象、资源管理的抽象和输入输出的抽象.在传统操作系统中,数据存储的抽象是属于某个进程的进程虚拟地址空间;数据计算的抽象是属于某个进程的线程;资源管理的抽象则是进程本身,操作系统通过进程抽象实施资源的管理和分配;输入输出的抽象则是不属于任何进程的文件.在这 4 类抽象中,除文件以外,其他 3 类抽象是不可分离的:

(1) 线程和进程不可分离:线程只能使用所属进程的资源,进程的资源也只能被属于该进程的线程所访问.

(2) 线程和进程虚拟地址空间不可分离:属于某个进程的虚拟地址空间只能被属于相同进程的线程所访问,属于某个进程的线程也只能访问该进程虚拟地址空间中的数据.

(3) 进程虚拟地址空间和进程本身不可分离:不属于某个进程的虚拟地址空间和没有虚拟地址空间的进程都是不可能存在的.

由于这 3 类抽象(数据存储-进程虚拟地址空间、数据计算-线程、资源管理-进程本身)是不可分离的,使得当属于某个进程的线程访问其他进程提供的服务时,存在以下问题:

(1) 由于线程和进程不可分离,当客户线程调用服务器执行的功能时,不能直接进入服务器进程中执行,而只能通过客户/服务器模型和消息传递、信号量、管道、信箱等通信机制,由服务器进程中的某个线程作为代理,代替客户线程执行.客户线程请求服务器执行一次功能,需要执行多次信息传递(请求和应答)、线程切换、线程睡眠唤醒等操作.与调用库函数或操作系统内核中的功能调用相比,调用服务器中的功能性能要低得多.这是微内核结构操作系统性能不高的原因之一[1].

(2) 由于线程和进程虚拟地址空间不可分离,服务器线程不能访问客户虚拟地址空间中的数据,客户线程需要通过消息传递、管道、信箱等向服务器线程传送调用参数,服务器线程需要向客户线程返回执行结果.如果参数传递的数据量很大,无疑会降低系统的性能.这也是微内核结构操作系统性能不高的的原因之一[1].

(3) 进程虚拟地址空间和进程本身不可分离引起的问题包括以下几个方面:

- 数据永久性问题:由于虚拟地址空间和进程本身不可分离,而进程是有生命期的,虚拟地址空间和进程具有相同的生命期.由内存和外存的对换区构造的虚拟地址空间丧失了对换区具有的永久性.
- 复杂数据结构的永久性存储问题:高级语言中指针类型的数据本质上就是进程虚拟地址空间地址,即使永久性地保存了指针类型的数据,一旦进程终止,指针也就成为毫无意义的数据,因此包含指针的复杂数据结构在文件中不能直接保存,需要把它转换成不包含指针的数据才能永久保存.
- 复杂数据结构的共享问题:指针是属于某个进程虚拟地址空间的,在一个进程虚拟地址空间中定义的指针类型的数据,在另一个进程虚拟地址空间中是毫无意义的.传统操作系统虽然实现了共享存储区、存储映射文件等共享技术,但是仅仅实现了存储区域的共享,不能共享包含指针类型的复杂数据结构.

以上分析说明,传统操作系统构造抽象的方式还有可以改进之处,作为一种探索,我们提出了一种新型的操作系统体系结构.该系统按下列方式构造抽象:

(1) 操作系统仅仅构造数据计算的抽象(线程)、资源管理的抽象(进程)和输入输出的抽象(文件),把数据存储的抽象(进程虚拟地址空间)的功能合并到输入输出的抽象(文件)之中,就像传统操作系统中的存储映射文件技术一样,指令对文件直接寻址,线程直接在文件上运行.

(2) 数据计算的抽象(线程)、资源管理的抽象(进程)和输入输出的抽象(文件)三者之间不存在隶属关系,三者是分离的.文件和其他两个抽象(线程、进程)的分离无须解释;通过内核提供的线程迁移功能调用,线程可以从一个进程进入另一个进程,从而实现线程和进程的分离.

由于该操作系统把进程虚拟地址空间和文件统一起来,应用程序直接在文件上运行,线程访问的虚拟地址

空间就是文件空间,我们称其为虚拟地址空间基于文件操作系统(operating systems basing virtual address spaces on files,简称 OS-BVASOF).图 1 是 OS-BVASOF 构造的各种抽象及其相互关系.OS-BVASOF 可以解决由于 3 类抽象不可分离而引起的问题:

(1) 由于实现了线程和进程的分离,线程通过线程迁移直接进入服务器,调用服务器提供的功能,避免了信息传递、线程切换、线程睡眠唤醒等操作.

(2) 由于把进程虚拟地址空间和文件统一起来,所有线程直接在文件上运行,线程不再局限于仅仅能够访问某一个进程虚拟地址空间,而是可以访问所有文件中的数据,也就避免了在客户/服务器之间拷贝调用参数和执行结果引起的降低系统性能的问题.

(3) 由于所有线程在文件上运行,高级语言中指针类型的数据本质上不再是进程虚拟地址空间地址而是文件地址,指针类型的数据结构不再局限于某一个线程或进程,既可以在文件中保存,也可以在各个进程之间共享,因此不存在复杂数据结构的永久性存储问题和共享问题.

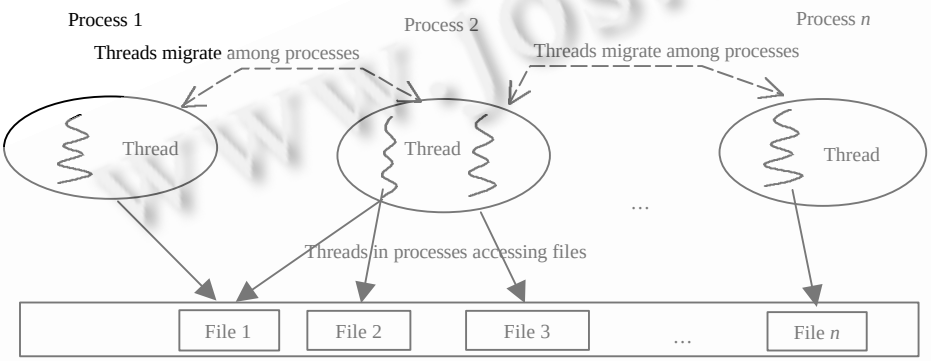

Fig.1 Abstracts constructed by OS-BVASOF as well as their the relations

图 1 OS-BVASOF 构造的各种抽象及其相互关系

1 虚拟地址空间基于文件操作系统(OS-BVASOF)的体系结构

1.1 OS-BVASOF的层次结构

OS-BVASOF 体系结构如图 2 所示.图 2 的中间一层为操作系统内核,内核实现了线程迁移、信号量操作、处理机管理和内存管理;阴影部分为硬件抽象层提供的功能,通过硬件抽象层使内核独立于硬件平台;在内核之上是各种服务器.任何应用程序在系统中都以进程的形式作为服务器存在,其他应用程序通过线程迁移可以被调用其功能.

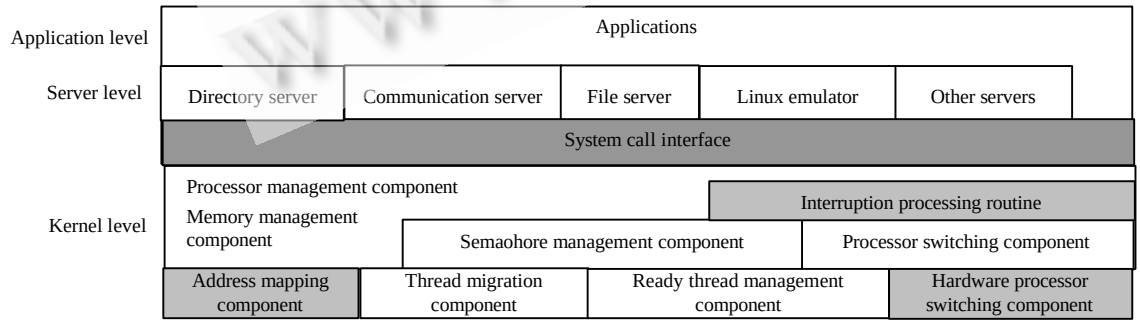

Fig.2 Architecture of operating systems basing virtual address space on files (OS-BVASOF)

图 2 虚地址空间操作系统(OS-BVASOF)的体系结构

1.2 内核的功能

OS-BVASOF 内核中仅仅实现以下功能:线程迁移和返回,信号量及 P,V 操作,处理机管理和内存管理.通过线程迁移和线程返回,可以实现线程在进程之间的迁移;通过信号量的 P,V 操作,可以实现线程之间的同步、互斥等相互控制;处理机管理(例如创建终止进程、创建终止线程、设置线程优先级等)和内存管理(例如创建终止存储域、打开关闭文件、建立地址影射等)也在内核中实现;文件系统既可以由核外的服务器实现,也可以由设备驱动程序在核内实现.

1.3 文件服务器、通信服务器和应用程序

在 OS-BVASOF 中,所有核外运行的程序,都是以进程的形式存在和运行的.进程都被看成是虚拟文件服务器,可以被其他应用程序调用的程序模块,以及实现对应用程序划分和保护的模块单位.线程通过线程迁移,进入另一个进程调用其提供的功能,执行完毕后可以通过迁移返回调用并重新返回至原来的进程中继续运行.

1.4 Linux仿真器的功能

就像传统微内核结构的操作系统一样,在 OS-BVASOF 中通过 Linux 仿真器实现对 Linux 软件的兼容.Linux 仿真器也是以进程的形式存在和运行的,并通过线程迁移被其他应用程序调用.

2 实现数据计算和资源管理的分离——线程迁移技术

2.1 线程迁移的概念

在 OS-BVASOF 中,线程迁移是改变线程当前所属的进程,实现线程的执行流程从一个进程转移到另一个进程,实现线程从一个进程中调用另一个进程功能的机制.线程迁移是内核的主要功能之一,通过线程迁移调用,一个线程可以从一个进程迁移到另一个进程,从而改变了线程所属的进程,也改变了线程的执行流程和执行的资源环境.OS-BVASOF 通过线程迁移,而不是通过客户/服务器模型及通信机制实现控制流程在不同应用程序之间的切换,通过线程迁移技术实现了数据计算(线程)和资源管理(进程)的分离.

2.2 线程迁移的相关技术

许多操作系统都认识到了客户/服务器模型引起的系统性能下降问题,并采用相应的技术解决该问题.线程迁移就是受到这些系统的启发而提出来的:

- 在 Spring OS 操作系统中,支持 Door 的概念,通过 Door 线程可以从一个 domain 直接进入另一个 domain,甚至可以实现线程在异构系统之间的迁移[2].线程迁移的提出就是受到了 Spring OS 操作系统中 Door 的启发.
- 在 Grasshopper 操作系统中[3],构造了 Loci 抽象(类似于传统操作系统中的线程),Loci 可以在不同的 Container(类似于进程虚拟地址空间)之间迁移.Grasshopper 操作系统甚至实现了 Loci 的永久性.在 Grasshopper 操作系统中,Loci 抽象启发我们实现了数据计算和资源管理的分离,其实现技术即为线程迁移技术.
- Aegis 操作系统采用了外核的思想,通过异常传递(exception forwarding)和向上调用(upcall),实现控制流程在不同应用程序之间的转移[4,5].在 OS-BVASOF 中,通过线程迁移处理异常就是受了 Aegis 操作系统启发而设计的.

2.3 线程初始执行点

线程初始执行点就是当线程通过线程迁移进入目标进程时,在目标进程中开始执行的第 1 条指令的地址.就像在传统操作系统中进入内核时只能从系统功能调用入口点开始执行一样,在 OS-BVASOF 中,当执行线程迁移调用时,线程从目标进程取出线程初始执行点和初始执行信息,只能从线程初始执行点开始运行,而不能从其他的位置开始执行.通过在初始执行点处安排一定的检测程序,可以防止迁移进来的线程在进程中执行非法操作,保证系统的安全性.

2.4 线程迁移的返回

迁移到某个进程的线程,可以通过线程返回调用返回到原来的进程中继续运行.在执行流程上,线程迁移类似于传统操作系统中的功能调用,二者的不同是:传统操作系统中的功能调用是线程对操作系统内核中功能的调用,而线程迁移调用则是线程对核外进程中功能的调用,OS-BVASOF 通过线程迁移机制把传统操作系统核内的功能放在核外由进程实现,避免了传统微内核结构操作系统中的消息传递和客户/服务器模型对性能的影响.

3 实现数据存储和数据计算与资源管理的分离——指令对文件直接寻址技术

实现数据存储和数据计算与资源管理分离的方法是指令对文件直接寻址技术,在实现机制上,指令对文件直接寻址与传统操作系统中的存储映射文件[6]是很相似的.

在计算机系统中,指令是对虚拟地址空间直接寻址的,例如,在保护模式下奔腾汇编语言 MOV DS:[2000H],AX 表示把 AX 寄存器中的内容送入虚拟地址 DS 段中偏移量为 2000H 的地址单元中去,这个 2000H 单元可能对应物理内存的某一地址单元,也可能对应外部存储设备某一物理位置.在后一种情况下,该指令的执行将引起存储器访问失效异常,由异常处理程序把数据由外部存储设备调入内存,然后修改段表和页表,重新启动指令执行.

虚拟地址空间至物理空间和外部存储设备的对应关系是由操作系统维护的,操作系统可以根据需要定义虚拟地址空间的涵义.如果把虚拟地址空间定义为文件空间,把映射关系定义为文件至物理空间和外部存储设备的映射,即可实现指令对文件的直接寻址.当指令对某一段寻址时,就好象指令对文件直接寻址一样,对应用程序而言没有了进程虚拟地址空间的概念,程序不是在进程虚拟地址空间中运行,而是在文件空间中运行,也就实现了虚拟地址空间基于文件.

由于只有文件的概念而没有了进程虚拟地址空间的概念,数据存储的功能完全由文件实现,而文件不属于任何线程或进程,因此实现了数据存储(文件)和数据计算(线程)与资源管理(进程)的分离.

4 OS-BVASOF 的安全性

安全性是操作系统的一项重要内容,在 OS-BVASOF 中,采用 ameba 操作系统相似的稀疏 Capability[7]实施对各种对象进行命名和保护,保证系统的安全性.下面以文件为例,介绍 OS-BVASOF 通过稀疏 Capability 实施保护的机制.

4.1 文件命名方案

如图 3 所示,文件 Capability 由 5 部分组成:网络标识、处理机标识、进程标识、文件标识、操作权限和校验.网络标识指示该 Capability 对应的文件存在于哪个网络结点,处理机标识指示该 Capability 对应的文件存在于网络结点的哪个处理机结点,进程标识指示该 Capability 对应的文件由哪个文件服务器维护,文件标识指示该 Capability 对应的文件是文件服务器维护的哪一个对象,操作权限指示拥有该 Capability 的线程可以操作该文件的权限,校验用于验证该 Capability 是否是一个合法的 Capability.

Network ID	Processor ID	Process ID	File ID	Right	Verification

Fig.3 Components of a file capability

图 3 文件 Capability 的组成

文件 Capability 是一个全局统一的网络标识,当线程调用文件服务器时,根据网络标识和处理机标识判断被调用的目的进程是在本地处理机还是远程处理机,如果是本地处理机,则根据进程标识直接调用相应的文件服务器;如果是远程处理机,则调用本地通信服务器,本地通信服务器把调用请求传输到远程处理机上的通信服务器,由远程通信服务器作代理,根据进程标识对指定的文件服务器执行调用,再通过通信服务器返回调用的执行结果.

4.2 文件保护

操作系统仅仅负责对网络标识、处理机标识和进程标识的解释,Capability 中剩余部分的解释由文件服务器负责,一般划分成如图 3 所示的文件标识、操作权限和校验 3 部分.当线程调用某个文件服务器时,根据文件标识确定该 Capability 对应的对象以及拥有该 Capability 的线程具有的权限,把校验信息和进程内部保存的信息进行比较,判断该 Capability 是否合法,如果该 Capability 合法,则执行相应的操作;如果非法,则拒绝执行,从而实现了文件操作的保护.

文件的校验信息是保密存储的,只有合法用户才能得到某个文件的校验信息,因此只要校验信息的位数足够长,即可保证文件的安全性.

5 OS-BVASOF 的实现、测试和性能评价

在 PC 平台上利用 Linux 环境下的 C 语言(gcc)已经实现了一个完整的 OS-BVASOF 内核,并利用 Linux 环境下的 lilo 实现了系统的安装.开发环境为 Redhat Linux 7.1,内核版本 linux-2.4.2,硬件平台为 Pentium-IV CPU,主频 1.8G,主存容量 512M,硬盘容量 40G.

5.1 线程迁移的性能测试

在相同的平台上,分别测试了控制流程在进程之间迁移需要的时间(在 OS-BVASOF 中通过线程迁移,在 Linux 操作系统中通过管道和消息缓冲区),测试结果见表 1.与 Linux 操作系统相比,OS-BVASOF 实现控制流程在不同的进程之间迁移的效率要高得多.

Table 1 Compare of executing time for OS-BVASOF and Linux to migration control between process (μs)

表 1 OS-BVASOF 和 Linux 实现控制流程在进程间迁移执行时间的比较 (μs)

Executing time length for control flow migration among process in OS-BVASOF			
User mode→ User mode	User mode→ Kernel mode	Kernel mode→ User mode	Kernel mode→ Kernel mode
3.2	2.9	3.3	3.4
Executing time length for control flow migration among process in Linux			
Through message buffer		Through pipe	
6.3		6.4	

在 OS-BVASOF 中,线程既可以运行在核心态,也可以运行在用户态,当源进程和目标进程 CPU 所处的状态不同时,线程迁移的执行时间是不同的,因此,表 1 中 OS-BVASOF 的测试数据有 4 项.

5.2 指令对文件直接寻址的性能测试

在实现技术上,虚拟地址空间基于文件和传统操作系统中的存储映射文件[6]很相似.为了评价 OS-BVASOF 中存储管理的性能,测试了 Linux 操作系统中存储映射文件性能,并与虚拟地址空间基于文件的性能进行了比较.测试方法为:

- 在 Linux 环境下,把文件/dev/zero 映射到进程的地址空间中,通过重复访问映射区域测试 1 次页故障的处理时间.文件/dev/zero 是一个支持存储映射文件的字符设备文件,从该设备中读出的数据全部为 0.
- 在 OS-BVASOF 环境下,打开一个文件,通过重复访问文件数据测试一次页故障的处理时间.为了具有可比性,该文件中的数据也全部为 0.既测试了通过内存管理器中的驱动程序读写文件数据,也测试了通过核外的文件系统进程读写文件数据.

测试结果为:Linux 操作系统中一次页故障的平均处理时间为 19.8μs.OS-BVASOF 中的测试结果见表 2.在 OS-BVASOF 中,每次处理存储访问失效异常,内存管理器可以同时为多个页框建立地址映射,建立页框的数目由线程指定.表 2 中左边一列为 1 次处理存储访问失效时,内存管理器建立地址映射的页框数.

从表 2 可以看出,如果由内核中的设备驱动程序实现文件的读写,1 次页故障的处理时间比 Linux 操作系统中的执行时间(19.8ms)要短,如果由核外的文件系统实现文件的读写,则比 Linux 操作系统中的执行时间要长.

Table 2 Time length for processing a page fault when reading or writing files (Execute both reading and writing) (μs)

表 2 读写 zero 文件时一次页故障的处理时间(同时对页框执行了读写) (μs)

Number of page frames that has built address mapping	Time length for reading and writing file data by device drivers in memory manager	Time length for reading and writing file data by file system process
1	15.7	37.6
2	13.3	29.6
4	12.2	28.8
8	11.6	28.7
16	11.5	27.8
32	11.4	27.7
64	11.4	27.7

5.3 应用程序测试

实现数据存储(文件)、数据计算(线程)和资源管理(进程)分离的技术是线程迁移和指令对文件直接寻址,上面是对这两项技术的性能测试.计算机系统性能的关键不是操作系统采用的技术的性能,而是在操作系统之上运行的应用程序的性能,因此测试了一个典型的应用:高斯消元法求解线性方程组.两个平台下运行的测试程序是完全相同的,编译连接的工具和参数也都一样.测试结果见表 3.

从表 3 可以看出,在 Linux 操作系统和 OS-BVASOF 操作系统上,高斯消元法求解线性方程组执行时间相差不多.这是可以理解的,OS-BVASOF 操作系统虽然与 Linux 操作系统具有不同的体系结构,二者采用的实现技术却是相同的.

Table 3 Time length for sloving linear algebraic equations by Gaussian elimination (μs)

表 3 高斯消元法求解线性方程组执行时间 (μs)

Element number	Linux operating system	OS-BVASOF operating system
4	3.48	3.53
8	4.64	4.06
16	7.54	6.69
32	18.56	17.98
64	106.72	117.74
128	854.34	904.80
256	7 028.44	7 151.40
512	57 605.60	56 831.30
1 024	46 788.90	453 803.02

以上仅仅是对 OS-BVASOF 的简单测试.操作系统的性能问题是很复杂的,关于 OS-BVASOF 的更加全面的测试和性能分析与评价,可参见文献[8]的第 5 章"系统的安装、测试和性能分析",其中包含 OS-BVASOF 性能更加全面的测试、分析和评价.

6 结束语

通过取消消息传递和客户/服务器模型,而代之以线程迁移,通过采用指令对文件直接寻址,取消进程虚拟地址空间,使进程直接在文件上运行,虚拟地址空间基于文件操作系统(OS-BVASOF)实现了数据存储(文件)、数据计算(线程)和资源管理(进程)的分离.与传统操作系统相比,OS-BVASOF 可以解决由于 3 类抽象不可分离而引起的问题,实现和测试说明实现 3 类抽象分离是可行的,且具有相应的优点.

References:

[1] Wang SY, Guo FS, Zang TY. The impact of the structure of microkernel operating systems on their performance. Journal of Computer Research and Development, 1999,36(1):57~61 (in Chinese with English abstract).

[2] Mitchell JG, Gibbons JJ. An overview of the spring system: Digest of Papers. In: Spring COMPCON' 94 (Cat. No.94CH3414-0). Los Alamitos: IEEE Computer Society Press, 1994. 122~131.

[3] Dearle A, di Bona R, Farrow J, Henskens F, Lindström A, Rosenberg J, Vaughan F. Grasshopper: An orthogonally persistent operating system. Computing Systems, 1994,7(3):289~312.

[4] Engler DR, Kaashoek MF, O' Toole Jr, J. The exokernel operating system architecture for application-level resource management. In: Proc. of the 15th ACM Symp. on Operating Systems Principles. 1995. 251~266.

[5] Kaashoek MF, Engler DR, Ganger GR, Briceo HM, Hunt R, Mazières D, Pinckney T, Grimm R, Jannotti J, Mackenzie K. Application performance and flexibility on exokernel systems. In: Proc. of the 16th ACM Symp. on Operating Systems Principles (SOSP' 97). 1997. 52~65.

[6] Liao HB, Qu GZ, Wang YP, Trans. Unix Internals—The New Frontiers. Beijing: Tsinghua University Press, 1999. 382~384 (in Chinese).

[7] Tanenbaum AS. Distributed Operating System. Upper Saddle River: Prentice-Hall, Inc., 1995. 384~388.

[8] Liu FY. The study of operating systems basing virtual address spaces on files [Ph.D. Thesis]. Shanghai: Shanghai Jiaotong University, 2002. 85~103 (in Chinese with English abstract).

附中文参考文献:

[1] 王世铀,郭福顺,臧天仪.微核心操作系统的结构对性能的影响.计算机研究与发展,1999,36(1):57~61.

[6] 聊鸿斌,曲广之,王元鹏译.UNIX 高级教程:系统技术内幕.北京:清华大学出版社,1999.382~384.

[8] 刘福岩.虚地址空间基于文件操作系统体系结构探索研究[博士学位论文].上海:上海交通大学,2002.85~103.

****