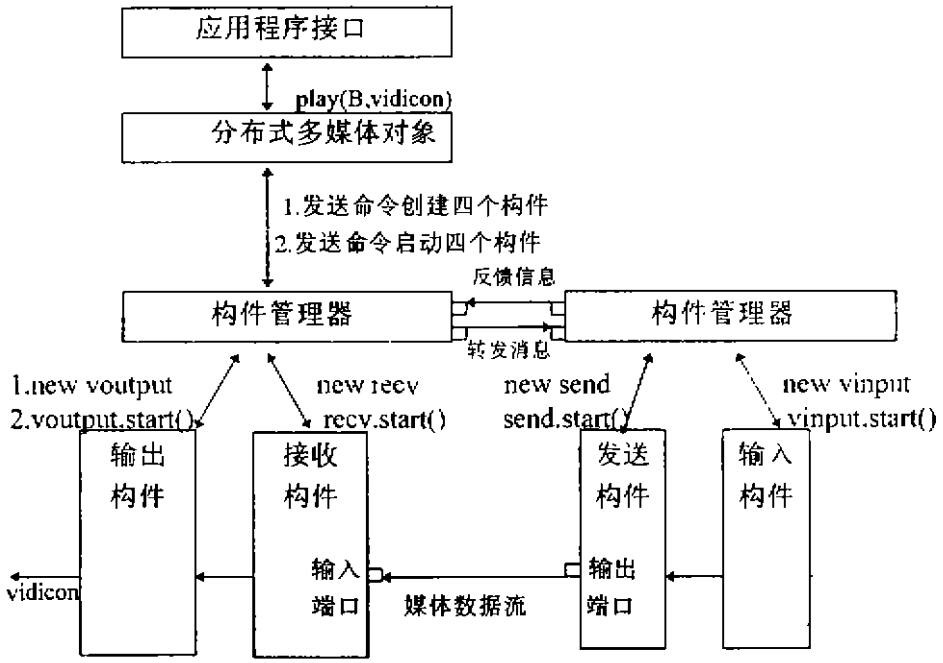

图4 DMOMS 工作流程

进行同步和 QoS 的协调工作。整个工程流程如图4所示。

参 考 文 献

[1] Giacomo Bucci et al. ,Sharing Multimedia Data Over a Client-Server Network,IEEE MultiMedia Fall 1994

[2] H. W. Peter Beadle,Experiments in Multipoint Multimedia Telecommunication,IEEE Multimedia Summer 1995

[3] Vahya Y. Al-Salqan,Carl K. Chang,Temporal Relations and Synchronization Agents,IEEE Multimedia 1996

[4] Tobias Helbig,Kurt Rothermel,An Architecture for a Distributed Stream Synchronization Service,Interactive Distributed Multimedia Systems and Services,Springer-Verlag Berlin heidelberg 1996

[5] Stuart Wray et al. , Networked Multimedia: The Medusa Environment, IEEE MultiMedia Winter 1994

（上接第113页）

表1

Machine	Processor	Spec rating	MIPS
DEC2100	R2000	8.7 Specint89	~11
DEC3100	R3000	11.8 Specint89	~15
DEC5000/125	R3000	16.1 Specint92	~25

结论 从以上种种迹象表明，外展式核心能够安全、有效地复合硬件资源，在低级的外展式核心界面上，可以通过操作系统库完成传统操作系统所做的一些抽象。我们可得出以下的结论：①外展式核心可以被有效地创建。②低级的硬件资源可以被安全地复合，并提供给应用使用。③传统操作系统完成的一些抽象可以在应用级有效地实现。④应用可以根据具体情况创建自己的抽象。我们相信外展式核心这样一种体系结构有其发展的天地，也具有发展前途。然而，外展式核心是一个新型的系统，还存在着许多不足，有许多问题值得我们去研究，去探索。

参 考 文 献

[1] Dawson R. Engler et al. , Exokernel: An Operating System Architecture for Application-level Resource Management,M. I. T. Laboratory for Computer Science

[2] Dawson R. Engler,M. Frans Kaashoek,Exterminate all Operating System Abstractions,Same to[1]

[3] Dawson R. Engler et al. ,The Operating System Kernel as a Secure Programmable Machine,Same to[1]

[4] M. Frans Kaashoek et al. , Application Performance and Flexibility on Exokernel Systems,Same to[1]

[5] Dawson R. Engler et al. , AVM: Application-level Virtual Memory,Same to[1]

计算机科学1998Vol. 25 №. 5

操作系统领域的一场革命—外展式核心

A revolution in operating system area—Exokernel

俞 晓 夏卫民

（国防科大计算机系 长沙410073）

摘 要 This paper simply introduces Exokernel with a way of summarization. Exokernel is an operating system, it is a new development technique for operating system today. Some strategy and technology which Exokernel used was introduced.

关键词 Operating system, Exokernel, Securely binding

一、引言

近些年来，尤其是94年以来，随着微软的日益强大，因特网的广泛使用，计算机事业呈现出一派蒸蒸日上、欣欣向荣的景象。然而，您是否注意到了，计算机迅猛发展的领域往往是计算机元器件、各种应用软件，而对计算机性能起关键作用的系统软件，特别是操作系统，产品的更替相对来说就缓慢得多。如计算机界巨头 Microsoft 公司早就说要推出新型操作系统，但至今却不见踪影。由此可见，设计，开发一个操作系统是何等的艰难、不易，我们又常常听到这样的抱怨："现在芯片速度很快，但应用程序的性能却未必能与芯片速度发展成正比"。究其原因是焦点集中在连结应用与硬件的桥梁—操作系统的身上。欣喜的是，许多有识之士已充分认识到这个问题，因此在操作系统领域掀起了一场革命，这就是改变操作系统的构造方式、创建一个与现今流行的操作系统完全不同的版本—外展式核心，英文名为"Exokernel"。本文围绕着外展式核心，对比传统的操作系统，从其结构、性能及所采用的相关技术等方面做一较为详尽的介绍。

二、变革之源由

近二十年来，操作系统同计算机的别的领域一样，自身也在不断地发展，竭力使之适应计算机硬件发展，尽可能多地将硬件的效能提交给应用，让用户更方便，快捷地使用计算机。于是，先后涌现出了 UNIX SVR, UNIXWARE, MACH, OS/2 及 WINDOWS 等许多著名的操作系统。这些操作系统都采用了自认为是高效的方法将硬件资源抽象化。希望所有应用都能很好地在操作系统所提供的同一界面上开发出实用、性能好的应用程序。显而易见，这是一种唐吉诃德式的想法，各种应用千变万化，要求所有应用都在一样的界面上开发，不可能所有应用都能获得很好的结果，充其量只是一部分而且仅是一小部分。这就如同人类穿衣，一个尺码显然是不可能觉得合身的。由此可见，不难解释为什么有些应用程序在某个系统上可达到很好的结果，而另一些在同样的系统上却非常糟糕的缘由了。究其原因，我们认为这完全是由于操作系统这一概念所至。操作系统的一个标准定义就是能安全地复合与抽象物理资源。我们认为将操作系统作为硬件的抽象这种观点是不充分的，也是错误的。从直觉上说，没有一个操作系统抽象可以完全适应所有的应用，要用一种方法抽象一种资源使其适应于所有的应用并用一种方式实现其抽象使其有效地适应不同的需求是完全不可能的、也是不现实的，这在实践中已得到诸多证实。下面我们具体谈谈这种观点对应用程序造成的危害。

·可靠性差：由于要抽象硬件资源、这就需要在核心中放置大量复杂的、多线程等性质的代码，象动态存储分配和管理，核心数据结构和代码的页操作等都极大地降低了系统可靠性。

·适应性差：一般的操作系统大而复杂，就拿较

俞 晓 讲师，主要研究方向是并行处理操作系统。夏卫民 研究员，主要研究方向是并行处理操作系统、并行计算技术。

小的微内核操作系统 MACH3.0来说，代码行也有10兆以上，更不用说普通的 UNIX 系统。将大而复杂的软件进行改造是很困难的。因此，要适应一个新需求、需要对操作系统进行修改也许要花许多时间，此外，由于所有的应用都依赖操作系统，改变就不是一个局部工作，牵一发而动全局。最后，由于操作系统的复杂性，使得这种改变只有操作系统设计者才能完成新的改变，这也是为什么操作系统的适应性差的原因。它所产生的负面影响就是即使提出了许多好的主意却无法实现。

·性能低下：操作系统的抽象常常是超出寻常，不合常理的。因为它们要为所有的应用提供一个统一的界面，并且所有这些应用都必须使用同样的操作系统抽象。这样，不需要这种界面的应用就必须付出许多不必要的开销，在这种情况下，就会产生许多垃圾和磁盘碎片。除此以外、简单地使用一个界面花费巨大，因为要花费许多时间从千万种选择中挑选一个或几个能够适合大多数应用的算法或策略，而且只有极少数的操作系统可以有意义地消耗主存、缓存、TLB 空间来用于完成有用的工作。另外，任何操作系统实现都要做一定的折衷：是否使用翻转页表，是否对频繁的读写操作进行优化，是否需要 copy-on-write 机制…，可是这些折衷并不是对所有的应用都是有利的，而且会使操作系统的设计者得不偿失。如果操作系统不对硬件进行抽象，这种情况是可以避免的。

·灵活性差：如果操作系统能让应用自己完成硬件抽象，那么，前面所述的三个不足均可避免。可是现在却不存在这样的可能性，应用能做的最好的事是在已存在的操作系统提供的界面上模拟一个应用所希望的界面，但这样的模拟一般都是很笨掘、复杂而代价高。例如应用不能直接访问硬盘，数据库就必须在文件上进行模拟。这样的事例举不胜举，并且还有上涨的趋势。

三、解决方法：外展式核心结构

上述的种种不利于操作系统发展的因素并非坚不可摧，如果能对症下药，就能药到病除。前面我们已经提到了，产生这一切的根源则是操作系统的定义错误。我们提出的解决方法是很直接，也很简单的：操作系统应该用一种安全的方式只是输出物理资源，而不应该是一个完美的、机器无关的提供应用的一个界面。换句话说就是操作系统只是简单地将硬件资源通过软件的方式表示给应用，而不考虑对它们的资源的管理。资源的管理交与应用去完成。

我们提出了一个操作系统的结构模式就是外展式核心，具体表现就是：不对硬件资源进行抽象，操作系统压低提交给应用的界面。一个外展式核心的功能就是用一种安全的方式分配、清除和复合物理资源，外展式核心只对应用输出如下物理资源：物理存储器、CPU、磁盘、DMA 通道、I/O 设备、TLB 表、处理上下文标识和中断/自陷事件。外展式核心通过原语的形式将这些硬件资源提交给应用，应用再经过操作系统库对这些资源进行管理，在其上开发软件。

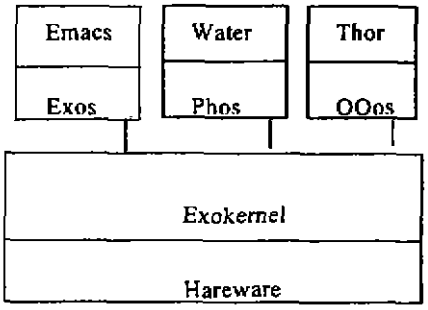

图1

图1是一个带有外展式核心的系统，此系统一般由四部分组成：第一部分是应用层，包含普通的应用程序，第二部分是操作系统库，它对硬件资源进行管理。第三部分是外展式核心层，它安全地将硬件资源以软件方式输出，交与上层使用。第四部分则是硬件资源，如一般的 I/O 设备、CPU、中断等。

从图1我们可以看到，这种系统结构同传统的操作系统的结构不一样，每一种应用带有自己对应的操作系统库，而每一个操作系统库都有一套与别的操作系统库可以完全不同的资源管理策略和方法。这样一来，每个应用就可以根据自己的实际情况选择一个或者重新构造一个操作系统库。由于每个具体的操作系统库都是为不同的应用本身定做的，那势必可以在一个特定的操作系统库上开发出高效的应用程序，

四、技术思想和策略

由于要将资源管理移到应用层，设计一个外展式核心的最大挑战就是要给操作系统库以最大的管理资源权利，但又要避免应用程序之间由于争夺资源而发生冲突。在某一个操作系统库中发生的程序设计错误不应该影响到别的操作系统库。为了获得

以上的目标，外展式核心通过一个低级界面将管理与保护分离。为此，外展式核心必须完成这样三个任务：(1)跟踪资源的所有者权利；(2)每个资源都应有哨兵守卫；(3)能够对资源进行回收。为了完成这三个任务，外展式核心采用了三个新技术：(1)使用安全性捆绑技术。操作系统库能安全地将硬件资源捆绑到某个应用程序上去。(2)回收可知(visible revocation)，允许操作系统库了解资源的回收情况。(3)失败协议(abort protocol)，一个外展式核心可以主动地将某个已捆绑到一个操作系统库的资源强行回收，终止该资源与操作系统库的联系。

下面我们详细地解释外展式核心的设计原理及采用的技术。

4.1 设计原理

一个外展式核心指定操作系统库用于请求、释放及使用硬件资源的界面。为了给操作系统库提供最大限度的控制资源的权利，外展式核心提出了下列设计思想：

·安全地暴露硬件：外展式核心的一个宗旨就是核心能够安全地提供低级的原语，操作系统库可以尽可能直接地使用这些原语来访问硬件资源。因此，一个外展式核心应该能够尽最大的努力来输出所有的特权指令、DMA 效率及机器资源。被输出的资源包含物理存储器、CPU、硬盘、TLB(translation lookaside buffer)及处理上下文标识，另外，象中断、异常及交叉域调用这些无形的东西都属于机器资源。为了给应用提供更好的灵活性，现有的硬件资源应该被更好地分割，象 TLB 的个数、TLB 的格式及当前 TLB 的集合对于操作系统库来说都应该是可见的，并且是可替换的。外展式核心还应该对操作系统库输出特权指令，使得操作系统库可以实现传统操作系统实现的高级抽象，如进程和地址空间。每个被输出的操作都被包裹在一个系统调用中，每个系统调用都需要检查该系统调用所涉及资源的所有者的身份。

·暴露分配：外展式核心应该允许操作系统库请求某个特定的物理资源，例如，如果一个操作系统库能够请求某个特定的物理页，它就可以减少工作集中的 cache 冲突的发生。操作系统库应该事先知道每个分配决定。

·暴露命名：外展式核心应该输出资源的物理命名，这样，就无须进行虚到实的名字的转换。另外，外展式核心还应该输出一些事务处理的数据结构，象自由链表，cache 的 TLB 入口等。

·暴露回收：外展式核心让操作系统库知道资源的回收协议，这样一来操作系统库可以有效地进行资源管理。

4.2 策略

外展式核心将资源使用的决策权交给了操作系统库，应用可以根据其实际情况来决定采取何种策略去合理地使用资源。然而，当多个操作系统库之间发生竞争冲突时，得由外展式核心进行仲裁，外展式核心将决定哪个应用可以优先获得资源的使用权，资源应该如何共享。这种情况的处理不同于传统的操作系统内核，选择什么机制不是由操作系统体系结构决定的，而是由应用所处的环境决定的。比如，外展式核心将资源的管理交给应用，但却控制对资源的分配与回收。外展式核心利用对资源的控制能力，可以执行传统系统的一些策略，象限额策略或预留策略等。

4.3 技术特点

①安全捆绑。外展式核心的一个主要任务是安全地复合资源，对每个资源进行守护，为互斥的不可信赖的应用提供保护。为了完成这个任务，外展式核心使用安全捆绑技术，使得操作系统库与资源捆绑在一起。

安全捆绑实际上是一个资源使用的保护机制，从两个方面来完善系统的性能，第一，根据内核(硬件)能够迅速实现来表达包含在执行一个安全捆绑中的保护检测。第二，每个资源仅在被捆绑的时间里其安全检测才有效，这样使管理与保护分离开来。应用程序对各种资源都能响应，外展式核心只需对资源进行保护而不必知道该资源是干什么的。在操作上，外展式核心提供一系列原语支持安全捆绑技术，应用程序可以使用这些原语来表示保护检测。软硬件方法都可以实现这些原语。对 TLB 入口的使用就是一个最简单的用硬件的手段来安全捆绑资源的一次实际使用：当发生一次 TLB 故障时，通过操作系统库管理的页表来处理虚到实地址的映射，随后将此映射加载到内核，使它被捆绑上，以后可以对它进行多次访问。

现在，我们用三种技术来实现安全捆绑：硬件机制，软件缓冲，下载应用代码到内核里。

·硬件机制：安全捆绑被看作一个低级的保护操作，这样一来，最近使用的操作无需高级的授权信息就可以进行有效的检测。例如，一个文件服务器将数据缓存在贮存中，要访问这些主存页只需告诉外展式核心相关页的一些访问许可信息即可。

·软件 cache：外展式核心中也可以做安全捆绑，例如，一个外展式核心使用一个很大的软件 TLB 来缓存那些不适合用硬件 TLB 完成的地址转换。

·下载应用代码：一些在访问资源时必须频繁使用的应用代码可以直接下载到核心中去，这种方法可以避免应用代码在应用和核心之间来回进行切换而造成许多不必要的浪费。

下面我们来举个事例详细地说明安全捆绑的具体使用。

·复合物理存储：在一个实际的外展式核心中，采用自我鉴定访问权限和地址转换硬件相结合的方法实现安全捆绑。当一个操作系统库需要一个物理页时，外展式核心首先替这个物理页创建安全捆绑，也就是记录下这页的所有者和由应用给出的对这页的一些访问权限，该页的所有者可以对此页的访问权限进行修改，也可以要求将此页回收。应用每次访问该页，都必须向核心提供访问许可证，如果核心检测出所提供的访问许可不充分，此次访问被拒绝。

②显式的资源回收。与传统操作系统不同的一个地方是，外展式核心允许应用了解资源的回收情况。这样的好处在于应用可以及时地了解资源的使用情况，有利于对资源的管理，同时，对于特殊的应用，可以在资源回收前保护一些有用的资源信息。但是，这种方法要求外展式核心与应用之间发生会晤，增加了系统开销。因此，如果某个资源回收很频繁，就不要将此资源的回收情况通知给应用，否则会增加系统开销。例如 CPU 的回收就不应该采用显式的资源回收方法。

由于显式回收要在内核和操作系统库之间发生作用，操作系统库就应该组织一个资源表，以便资源能被快速回收。例如，操作系统库应该有一张它拥有的物理页的向量表，当内核表示有该库所拥有的一页必须被回收时，操作系统库就能通过这张向量表查找出相应的页，根据具体情况将此页从表中删除或先将此页写回磁盘，然后将此页从向量表中删除。

③流产协议。如果内核向操作系统库发送一个回收请求不被响应，则内核有权将该资源回收。外展式核心将回收协议定义成带时间限制的，如它将"请返回一个物理页"变成"请在50秒内返回一个物理页"。如果操作系统库没在规定的时间里返回指定的物理页，内核就强制性地将它与操作系统库之间相关的安全捆绑解除。

最简单的流产协议就是将相关的操作系统库和与之相关的应用程序杀死。我们觉得这种方式不好，因为每个应用的实时响应时间都是不同的，内核无法定义一个统一的响应时限。外展式核心现在只是解除与相关的操作系统库的所有安全捆绑，并将此信息通知给操作系统库。

在流产协议中，我们采用了一个重复使用向量表，当外展式核心向某个操作系统库强迫收回某个资源时，操作系统库就将在重复使用向量表中作个登记，并对使用该资源的映像作相应的修改，当这个资源被再次使用时，外展式核心就将先前记录的资源的状态写到某个存储器或磁盘中。在系统初起时，操作系统库可以预先准备一定程度的资源用于加载重复使用向量表。由于外展式核心无法裁决何种资源可以被重复使用，操作系统库必须使用一些物理存储器来存储一些重要的引导信息，如异常处理器，页表等，这些物理存储器不可以被重复使用。

五、实验情况

美国麻省理工大学计算机系的一个研制小组已经开发了一个原型系统，一个外展式核心 Aegis，一个库操作系统 Exos。另一个基于 SPARC 共享存储多处理机的原形系统 Glaze 和一个并行库操作系统 Phos 正在研制中。

Aegis 和 Exos 是在基于 MIPS 芯片的 DEC 工作站上实现的。Aegis 输出了处理器、物理存储器、TLB 表、异常和中断。除此之外，它还安全地输出网络界面。Exos 实现了进程、虚存、用户级异常、各种中断异常和几种网络协议。一个实现全局缓冲管理的文件系统正在实现中。

为了说明测试的普遍性，我们分别用了三种硬件平台，如表1所示。测试所用的时间均是墙上时间，所有的编译都是用的 gcc2.6.0，带编译开关-O2。这些测试都是在单机环境下，不考虑浮点运算，不考虑网络。每个操作都是通过大量的测试，取其平均值来获得。

通过一系列的测试，发现它与传统的操作系统 Ultrix 相比，性能在大多数方面均优于传统的，而 Ultrix 是当今一个很好的操作系统，它在一系列 I/O 的测试中，比 Mach3.0快二到三倍。而 Ultrix 的虚存速度是 Mach2.5的二倍，是 Mach3.0的三倍。从测试中可以看到，外展式核心可以很好地解决 Cache 冲突。

（下转第98页）