通用OS内核的构件化是否合适*)

李 航 郭敬林 刘西洋 陈 平
(西安电子科技大学软件工程研究所 西安710071)

摘 要 本文从OS研究的推动力——上层应用和底层硬件的角度,对过去OS的研究成果进行了详细的考察,并在该基础上得出结论:通用OS内核的结构不适合采用构件式。

关键词 OS,构件,构件式OS

Component Based General-Purpose OS Kernel, Good or Bad?

LI Hang GUO Jing-Ling LIU Xi-Yang CHENG Ping
(Institute of Software Engineering, Xi'an University of Electronic Science and Technology, Xi'an710071)

Abstract The paper surveys the past OS researches in the perspective of application and hardware and concludes that component based architecture is not suitable for general purpose OS.

Keywords OS, Component, Component-based OS

1 引言

近年来,OS研究领域中出现了一种新的OS结构,即基于构件的OS,这种技术主要是为了降低OS开发的复杂度,并灵活地适应上层千变万化的应用。但这种技术面临着一些诸如如何抽象合适的构件等难题,使人们对这种技术的前途产生了疑问,本文从OS研究的推动力——上层应用和底层硬件的角度,对过去的OS研究成果进行了分析,并在该基础上得出通用OS的内核结构不适合采用构件式。

论文的第2部分介绍了OS研究的推动力,第3部分对过去OS研究进行了总结,第4部分对第3部分的内容进行了分析,并得出最终的结论。

2 OS研究的推动力

传统的观点将OS看成是一个虚拟机或一个资源管理器,这两种观点容易使人对OS的研究方向和目的产生错觉——认为OS的研究所要解决的主要问题是:1. 如何对系统进行抽象。2. 如何优化OS的资源调度。事实上,这两点仅仅是OS研究的重要方面,OS研究真正需要解决的核心问题是各种应用中的问题(这里的应用包含OS自身,因为OS可以看成是直接依附于硬件上的应用,下面"应用"一词,除非特别声明,均指OS之上的应用,不包含OS自身),因为真正推动OS发展的动力是:1. OS的上层应用。2. OS的底层硬件。操作系统的研究从一开始到现在就一直是面向应用的,为什么?计算机一出现,用户就必须管理各种设备:键盘、鼠标、显示器,处理器……。用户有管理资源的这种"应用要求",因而才出现了操作系统。本文正是抱着这种观点,对OS研究的过去进行了详细的审视。

3 OS的分类

从OS内核结构的角度,可将过去的OS分为如下几大类(按出现时间的先后顺序排列):

3.1 巨内核结构

这种体系结构的最初目的是为了使上层应用开发者能够快速地开发出高性能的应用。它的特点在于:对底层硬件提供高层抽象,比如进程和文件系统。这种体系构架因为提供了定义良好的API接口(例如:Posix标准),所以它的通用性非常强。然而也正是这些定义良好的API接口,使得这种体系构架的灵活性降低,导致无法很好地匹配上层的某些应用。以数据库中的缓冲区管理为例:一般的操作系统本身有自己的一套缓冲区,不过缓冲区的大小和管理缓冲区的策略在编译内核时就已经确定了,而数据库上的各种应用却是千变万化的,所以数据库本身不得不维护自己的一套缓冲区,直接导致数据库设计的复杂和总体性能的下降。

另外,从内核开发者的角度而言,巨内核由于内部结构的无序性导致内部的交互很复杂,它们是难于维护的。

3.2 微内核结构

正是上述巨内核结构的那些弱点,促使人们提出微内核结构。它背后的理念在于将系统的设计目标和它的实现机制相分离,并将机制的实现交由用户处理,从而获得较好的可伸缩性。它的缺点在于[3]:

i. 大多数的微内核操作系统事实上不是"微"内核;因为它们提供一般性的系统服务,导致微内核的接口复杂,并且占用较大的内存。

ii. 微内核系统在性能上不足。由于许多功能是在内核以外实现的,那么,应用就会与内核频繁通信,从而加大了系统的整体开销。但也有不同的声音,有人认为这是因为IPC的低效实现造成的,L3,L4就是一个例子。

iii. 大部分微内核系统的自适应性不好。原因是微内核的系统接口在设计时,是用来提供通用的服务而非面向专门的应用领域,导致接口和实现机制的固定化。所以微内核系统并不能在不同的系统环境和应用下均表现出令人满意的适应

*)研究内容来源于"十五"军事预研项目(编号4133150501)及西安电子科技大学硕士生指导老师培养基金项目:"操作系统内数据管理算法的研究"并受上述两项项目资助。李 航 博士生,研究方向:嵌入式系统。郭敬林 博士生,研究方向:软件工程、数据库技术。刘西洋 副教授,研究方向:软件工程、分布式处理系统。陈 平 博士生导师,教授,研究方向:面向对象,软件工程。

性。

3.3 可扩展内核结构

正是因为微内核结构有这么多的问题，所以在微内核还未成熟之前，研究者就把重心移向了可扩展内核结构。可扩展内核结构按接口划分一共分为如下几种形式：

i. 硬件资源接口集

这种方法认为，不论向用户提供哪种接口，总存在某些应用，使得提供的接口不能很好地适应这些应用的要求，所以这种内核直接将底层的硬件资源提供给上层应用，这种内核唯一的职责是对底层硬件资源进行安全的复用。最典型的例子就是 exokernel 和 Ageis。

ii. 固定接口集

这种OS内核不像上一种OS内核那样，直接将底层的硬件资源暴露给上层，而是通过提供一个抽象接口集合，用户可以通过这些抽象接口集合实现特定应用的定制，最典型的是 spin 和 cache kernel。以 SPIN 为例，用户可以动态地将自己的代码通过这些接口动态地加入到内核中。

iii. 元接口集

这种OS内核提供一个元抽象接口层，用户可使用这些元抽象接口实现用户自己需要的接口。

iv. 构件接口集(即构件化OS内核)

这种OS内核将OS划分成各个构件，好处是：1. 上层的面向对象应用可以根据实际情况，通过诸如反射机制对下层OS作出调整，从而提高系统的自适应性。2. 节省OS的开发时间。

4 分析

虽然构件化OS的思想非常诱人，但它与其他的可扩展内核一样存在着如下缺陷。它们实际上对OS进行了3层抽象[9]：

1. 最高层抽象：应用程序界面抽象；
2. 最底层抽象：系统资源抽象；
3. 中间层抽象：位于上面两层之间的内容的抽象。

1、2两层的抽象的变动较小，而中间层抽象的变化最大。因为这一层位于高层 API 和底层最基本的硬件抽象之间，为了达到定制化的设计目标，就必须非常仔细地对这一层进行抽象(各种可扩展内核的区别之处也是在这里体现的)，从而得到可以针对具体应用进行裁剪的 building block。这虽然是一个非常诱人的思想，但在现实中难于实现。因为上层的应用总是千变万化的，不论你怎么对中间这一层抽象，总会存在一些这种抽象无法胜任的应用。更为致命的是，为了适应更多的应用，中间这一层抽象出来的 building block 就必须越细致，那么控制这些 building block 的组装机制也就越复杂，也就越容易出错。这与当前OS研究应用层面的驱动力——可信计算的理念是背道而驰的。

从硬件角度讲：最显著的证据是 code morphing，它是一种处理器使用的技术，其这种处理器的结构如图1所示。

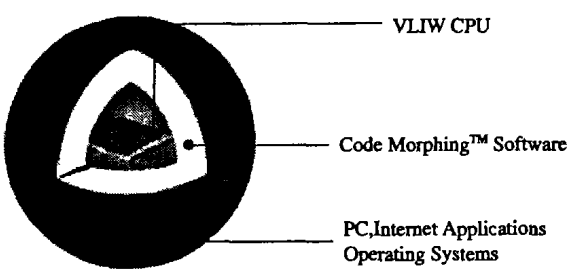

图1

通过 code morphing 层，这种处理器动态地将其它 CPU 的指令集，比如 x86指令集，翻译成自己的指令集，比如 VLIW，以提供良好的兼容性。这种 CPU 的出现，代表着一种新的观点：在设计一个系统时，可移植性的问题可以放在次要的位置上进行考虑。而构件化 OS 当初设计时的一个很重要的目标就是可移植性。

其次硬件价格的大幅下降，使得构件化 OS 当初设计的理念之一："用尽可能少的硬件，做尽可能多的事"发生了动摇。

基于以上的认识，本文认为：通用 OS 的内核结构仍然应该是基于模块式的，并且是开放的。这里的模块式内核不同于巨内核，巨内核内部的结构基本上是无序的，而且有很多冗余的部分，比如一些应用根本不使用的驱动程序也被放入内核中。而模块式内核，内部结构是模块化的，它可以动态地加载应用需要的模块，最典型的例子就是 Linux。有人批评，OS 内核不能做到完全的模块化，而且模块与模块之间的交互仍然很复杂，极易出错。我们赞同这样的观点，但我们认为这样的缺点可以通过开放的特性来弥补，早期的 Unix 是巨内核的，但最后却广泛流行，并且稳定性极佳就是最好的证明。不过，需要指出的是：严格地说，Linux 并不是一个真正意义上的开放系统，因为商业利益的关系，Linux 仅开放了源码，并未公开它的具体的设计构架和方案。而在当前可信计算的浪潮下，开放 OS 源码和其相关的设计方案具有特别重要的意义。

总结 本文通过分析指出通用 OS 的内核结构不适合采用构件式，应当采用模块式结构。但如何提高模块式结构的可重用性，依然是一个值得研究的问题。目前我们认为这个问题的解决方案是：在一个较为稳定的体系构架下，实现源代码一级的重用。另外，我们目前也认为嵌入式领域中的 OS 不应该构件化，重用的重点同样也应当放在体系构架一级和源代码一级上。我们下一步的工作是设计并实现一个 OS 内核以验证我们的想法。

参考文献

1 Milojicic D. Applied Operating Systems. IEEE Distributed Systems Online, 2000

2 Hulse D, Dearle A. Trends in Operating System Design: Towards a Customisable Persistent Micro-Kernel: [Technical Report Pastel RT1R4]. University of Stirling, 1998

3 Denys G. A Survey of Customizability in Operating Systems Research. CSUR, 2002

4 Steffens L, Van Loo S, Pérez C O. Trends in Operating Systems Resource Management for Future CE Systems. In: First Workshop on Embedded Systems for Real-Time Multimedia, 2003

5 Hume A. Operating System Reliability: Does Anyone Care Any More?. HotOS-IX, 2003

6 Gien M. Next Generation Operating Systems Architecture. LNCS 563. In: the Intl. Workshop on Operating Systems of the 90s and Beyond, 1991

7 Milojicic D. Operating systems-now and in the future. IEEE Concurrency, 1999

8 De Lara E, Wallach D S, Zwaenepoel W. Position Summary: Architectures For Adaptation Systems. HotOS-VIII, 2001

9 Cheung W H, Loong A H S. Exploring Issues of Operating Systems Structuring: from Microkernel to Extensible Systems. Operating Systems Review, 1995

10 Friedric L F, Stankovic J, Humphrey M. A Survey of Configurable, Component-Based Ooperating System for Embedded Applications. IEEE Micro, May 2001