操作系统结构风格研究*)

Study of Structural Styles of Operating Systems

耿 技 周明天 秦志光

(电子科技大学计算机学院 成都 610054)

Abstract Common structural styles of operating systems and their Base-Platform Subsystems are systematically discussed in this paper. As our opinion, it is helpful for designers of operating systems to systematically realize the structural knowledge of operating systems and their Base-Platform Subsystems.

Keywords Operating system, Architecutre, Structural style

一、引言

最近十年人们对软件体系结构及其设计方法和设计工具进行了系统研究并取得了一些重要研究成果。相关研究成果揭示[1]:在大型复杂软件系统的开发过程中强调软件体系结构设计有利于保证软件系统的质量、提高其开发效率、降低其开发成本,因此在此类软件系统的开发中软件体系结构设计至关重要。这一结论对各种大型复杂软件系统普遍适用。当然,大型复杂操作系统也不例外。事实上,经过多年的探索和实践,人们已经意识到操作系统体系结构设计在大型复杂操作系统开发过程中的重要性并对一些特定的操作系统结构设计技术给予了关注。近些年来人们对微核结构设计思想的重视表明了这一点。然而,迄今为止人们尚未对操作系统体系结构及其设计方法和设计工具进行系统研究。这使得操作系统体系结构设计仍然缺乏坚实的基础。不可否认,关于软件体系结构及其设计方法和设计工具的研究成果可以用来指导操作系统体系结构设计。但是,操作系统毕竟是一种特定的软件系统,其体系结构以及相关的设计方法和设计工具有自己的特殊之处。因此,对操作系统体系结构及其设计方法和设计工具进行系统研究仍然是必要的。

本文并不打算讨论与操作系统体系结构相关的所有问题。在我们看来,对一些具体的操作系统结构风格进行系统研究是人们对操作系统体系结构及其设计方法和设计工具进行系统研究的一个很好的起点。因此,本文将基于关于软件体系结构的基础研究框架[2]系统地讨论一些常见的操作系统结构风格。本文讨论的操作系统结构风格既包括操作系统总体结构风格,也包括操作系统基础平台子系统总体结构风格。为了方便讨论,本文把操作系统之外的其它软件系统统称为应用软件。

二、一种常见的操作系统总体结构风格

众所周知,通常情况下,操作系统的功能性需求可以分为两种类型[3,4]:一是,计算机用户需要的用户命令;二是,应用软件需要的系统服务。经常地,在操作系统体系结构设计阶段,人们根据操作系统功能性需求的这种分类结果对操作系统的内部语义逻辑以及行为逻辑进行分解并最终获得操作系统的某种总体结构。

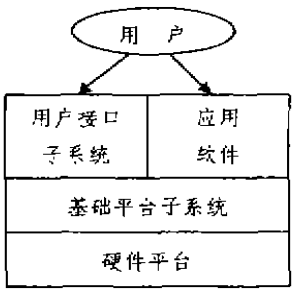

图 1 一种常见的操作系统总体结构风格示意图

十分明显,根据操作系统功能性需求的上述分类结果对操作系统的内部语义逻辑以及行为逻辑进行分解,人们最终获得的操作系统总体结构总是包含操作系统的两类子系统:一是,提供用户命令的子系统;二是,提供系统服务的子系统。我们把这两类子系统分别称为操作系统的用户接口子系统和基础平台子系统。一般说来,用户接口子系统与基础平台子系统之间的

*)本文为信息产业部/四川省经贸委专项课题"安全操作系统"系列研究报告之一。

相互关系具有单向性。具体地说，用户接口子系统在实现各种用户命令的语义逻辑和行为逻辑时能够引用基础平台子系统所提供的各种系统服务，但基础平台子系统在实现各种系统服务的语义逻辑和行为逻辑时不会引用用户接口子系统所提供的各种用户命令。显然，这里所给出的关于用户接口子系统与基础平台子系统其功能接口以及相互关系的特征描述实际上定义了一种操作系统总体结构风格。在操作系统工程实践中，这种总体结构风格为大多数操作系统所采用。图1示意性地描述了这种常见的操作系统总体结构风格。

必须指出，上述操作系统总体结构风格并不限制相关的操作系统总体结构包含多个用户接口子系统。事实上，一些已建成的采用上述总体结构风格的产品操作系统同时实现了多个用户接口子系统供计算机用户们选用。从基础平台子系统的角度看，用户接口子系统与应用软件没有什么不同：它们与基础平台子系统之间的相互关系是一致的。毫无疑问，计算机用户们可以象开发应用软件那样的基于基础平台子系统所提供的系统服务来开发自己的用户接口子系统。一般说来，基础平台子系统是大多数计算机系统中唯一与硬件平台直接交互的软件成份。因此，对于用户接口子系统与应用软件来说，基础平台子系统通常隐藏了硬件平台所提供的各种硬件设施。下一节我们侧重讨论基础平台子系统所采用的一些常见的总体结构风格。为了方便讨论，在下面的讨论中，我们把用户接口子系统也视作应用软件的一种。

三、常见的基础平台子系统总体结构风格

Druschel 等人主张[5]，在构造操作系统基础平台子系统总体结构时应当分离模块和保护。在我们看来，这种主张无异于提醒人们应当分别基于功能结构观点和并发结构观点构造操作系统基础平台子系统总体结构。的确，由于绝大多数操作系统其基础平台子系统最终以并发程序的形式加以实现，因此这些操作系统其合理的基础平台子系统总体结构应当分别基于功能结构观点和并发结构观点加以构造[5]。

毫无疑问，关于基础平台子系统总体结构风格的研究通常也应当基于功能结构观点和并发结构观点分别进行。在研究基础平台子系统总体结构风格时，功能结构观点和并发结构观点表述了人们应当关注的基础平台子系统的两组非功能性需求；这两组非功能性需求分别与基础平台子系统其内部语义逻辑以及内部行为逻辑的分解原则密切相关；它们是人们评价相关基础平台子系统总体结构风格的标准[5]。下面我们分别基于功能结构观点和并发结构观点讨论基础平台子系统采用的一些常见的总体结构风格。顺便指出，基础平台子系统在计算机系统中并非是孤立的；它与应用软件以及硬件平台存在着密切的联系。因此，在下面的讨论中，我们将把应用软件和硬件平台视作基础平台子系统总体结构所包含的两种基本成份。

3.1 功能结构观点下的基础平台子系统总体结构风格

习惯上，人们把基础平台子系统的功能性需求理解为为应用软件提供一组系统服务。事实上，基础平台子系统所提供的系统服务总是与一组基本概念相关。因此，我们可以说，基础平台子系统的功能性需求总是包含着一组基本概念以及与这组基本概念相关的一组系统服务。通常，这组基本概念以及与其相关的一组系统服务定义了多台或一台虚拟计算机[5]。其中，基本概念定义了虚拟计算机所提供的计算设施，系统服务则定义了施加于相关计算设施的基本操作。对于各种应用软件来说，基础平台子系统所实现的虚拟计算机为它们提供了实现平台与运行环境。

正是因为基础平台子系统的功能性需求总是包含着一组基本概念，所以基于功能结构观点构造基础平台子系统总体结构时人们总是可以针对这组基本概念对基础平台子系统的内部语义逻辑进行分解。我们把关于基础平台子系统内部语义逻辑的针对基本概念的分解原则称为概念分解原则。根据概念分解原则对基础平台子系统内部语义逻辑进行分解时，人们关注的是基础平台子系统所实现的基本概念的相对独立性和相对完整性；与基本概念相关的系统服务被视作依附于各个基本概念的操作属性。换句话说，在基本概念分解的基础平台子系统总体结构中，计算部件被视作一组基本概念以及依附于这组基本概念的包括操作属性在内的所有基本属性的实现体。很明显，概念分解原则与面向对象技术有着某种内在联系。因此，从原则上讲，人们可以用面向对象技术实现基于概念分解的基础平台子系统总体结构。

基于概念分解的基础平台子系统总体结构通常具有多种结构风格。如果只考虑基础平台子系统计算部件之间相互关系的结构特征，那么我们可以把基础平台子系统总体结构风格分为概念分层结构风格、概念分级结构风格、概念分块结构风格三种基本类型。下面我们分别介绍这三种基本结构风格并举例说明它们的应用。

·概念分层结构风格　使用概念分层结构风格的基础平台子系统总体结构包含若干叫做层的计算部件；其中，每一层实现了一组基本概念以及与其相关的所有基本属性。就层与层之间的相互关系而言，概念分层结构风格具有两个鲜明的结构特征：一是，所有各层其实现方案不会依赖其以上各层所提供的概念及其属

性而是只依赖其直接下层所提供的概念及其属性;二是,所有各层均对其以上各层隐藏了其以下各层的存在。

IBM 公司开发的 VM370 操作系统其功能结构观点下的基础平台子系统总体结构使用了概念分层结构风格[6]。从功能结构观点看,VM370 其基础平台子系统总体结构包含两种基础平台子系统计算部件。这两种基础平台子系统计算部件被 VM370 的开发者们分别命名为 CMS 和 CP。其中,CMS 用来提供交互类应用软件需要的运行环境,CP 则用来提供 CMS 需要的专用虚拟计算机。从原理上讲,VM370 其功能结构观点下的基础平台子系统总体结构包含多个层。其中,应用软件和硬件平台分别为顶层和底层;次顶层则由 CMS 组成;其余各层均由 CP 组成。VM370 其功能结构观点下的基础平台子系统总体结构之所以有这样一种组织形式是因为 CP 具有一个重要特征,即:CP 所提供的虚拟计算机精确地再现了 VM370 基础平台子系统所依赖的硬件平台。正是因为 CP 具有这样一个特征,所以任何一个原本在 VM370 基础平台子系统所依赖的硬件平台上运行的软件系统均可原封不动地在 CP 所提供的虚拟计算机上运行。由于 CP 原本在 VM370 基础平台子系统所依赖的硬件平台上运行,因此任何一个 CP 均可原封不动地在另一个 CP 上运行。必须指出,CP 对底层硬件平台的精确复制并非概念分层结构风格所固有的结构特征。

Microsoft 公司开发的 Windows NT 操作系统其功能结构观点下的基础平台子系统总体结构也使用了概念分层结构风格[7]。从功能结构观点看,Windows NT 基础平台子系统在其总体结构中被分为两层:一是,由各种环境子系统所组成的环境子系统层;二是,由各种执行部件所组成的执行部件层。Windows NT 其功能结构观点下的基础平台子系统总体结构由应用软件层、环境子系统层、执行部件层以及硬件平台层构成。这四个层之间的相互关系具有概念分层结构风格的结构特征。

·概念分级结构风格 使用概念分级结构风格的基础平台子系统总体结构包含若干叫做级的计算部件;其中,每一级实现了一组基本概念以及与其相关的所有基本属性。就级与级之间的相互关系而言,概念分级结构风格具有这样一个鲜明的结构特征:所有各级其实现方案不会依赖其以上各级所提供的概念及其属性而是只依赖其以下各级所提供的概念及其属性。表面上看,概念分级结构风格的这一结构特征与概念分层结构风格的第一个结构特征非常相似。实际上,它们之间存在着差别。正是因为存在着差别,所以概念分级结构风格不具有概念分层结构风格所具有的第二个结构特征。

从功能结构观点看,嵌套进程结构风格也是基础平台子系统总体结构可以使用的基于概念分解的一种结构风格[8,9]。该结构风格有两个重要的结构特征:一是,每个进程其状态均被封装在它的父进程数据区中;二是,父进程被允许控制其子进程的对外交往活动。Fluke[3]和 CAP[5]均采用了这种结构风格。不过,Fluke 的嵌套进程结构与 CAP 的嵌套进程结构并不相同。两者之间的差别主要在于它们对进程之间的相互关系做了不同限制:就父进程对其子进程对外交往活动的控制而言,在 Fluke 中是选择性的;在 CAP 中,是强制性的。换句话说,Fluke 中的每个进程均可能直接引用其父进程之外的任意祖先进程所提供的概念及其服务,CAP 中的每个进程只能引用其父进程所提供的概念及其服务。显然,在 Fluke 中,进程之间的相互关系仅仅符合概念分级结构风格的结构特征;在 CAP 中,进程之间的相互关系符合概念分层结构风格的结构特征。

元空间是一组元对象所组成的集合;它可以被视作一个对象所使用的专用的虚拟计算机或优化的运行环境[10]。在概念上,元对象与对象是相同的。因此,一个元对象有自己的元空间。从原理上讲,一个对象的元空间及它的所有元对象的元空间以及它的所有元元对象的元空间等形成了一种树型结构。这种树型结构具有两个突出特征:一是,它的每个结点都是对象的元空间;二是,它的拓扑结构是一棵树。显然,根据这两个结构特征,我们可以定义一种结构风格。我们把这种结构风格称为元空间树结构风格。这种结构风格也是基础平台子系统总体结构可以使用的基于概念分解的一种结构风格。Apertos[11]以及 MetaOS[12]均采用了这种结构风格,就元空间之间的相互关系而言,Apertos 的元空间树结构与 MetaOS 的元空间树结构并非是相同的。在 Apertos 中,一个对象的元空间与其元对象的元空间以及与其元元对象的元空间等可以共享同一对象,所以 Apertos 的元空间树结构仅仅具有概念分级结构风格的结构特征,与 Apertos 不同,MetaOS 不允许元空间之间共享同一对象,因而 MetaOS 的元空间树结构具有概念分层结构风格的结构特征。

·概念分块结构风格 使用概念分块结构风格的基础平台子系统总体结构包含若干叫做块的计算部件;其中,每一块实现了一组基本概念以及与其相关的所有基本属性。就块与块之间的相互关系而言,概念分块结构风格具有这样一个结构特征:所有各块其实现方案均可以任意引用其它各块所提供的概念及其属性。显然,依据该结构特征,我们很容易把概念分块结构风格与上述两种基础平台子系统总体结构风格区别

开来。

QNX[13,14]其功能结构观点下的基础平台子系统总体结构使用了概念分块结构风格。从功能结构观点看，QNX 基础平台子系统在其总体结构中被分解为若干相互协作的系统管理器，比如：进程管理器、文件管理器、设备管理器以及网络管理器等。Linux[15-16]其功能结构观点下的基础平台子系统总体结构也使用了概念分块结构风格。在其功能结构观点下的总体结构中，Linux 基础平台子系统被分解为进程调度、进程通信、存储管理、虚拟文件系统以及网络接口管理等相互依赖的系统模块。传统的 UNIX[17]其功能结构观点下的基础平台子系统总体结构同样使用了概念分块结构风格。从功能结构观点看，传统的 UNIX 其基础平台子系统总体结构包含文件子系统和进程子系统两个基本的基础平台子系统计算部件。这两个基础平台子系统计算部件彼此合作为 UNIX 应用软件提供了一个完整的运行环境。

总的说来，使用概念分层结构风格将使得基础平台子系统内部语义逻辑富有条理，不过，如果基础平台子系统内部语义逻辑十分复杂，那么为其构造一个纯粹的概念分层结构将非常困难。在这种情形下，使用概念分块结构风格可能更切合实际。比较而言，概念分层结构风格有利于实现基础平台子系统的可维护性、可扩展性、可移植性、部件可重用性等非功能性需求，但它不利于提高基础平台子系统的时间和空间效率。与之相反，概念分块结构风格有利于生成高效、紧凑的基础平台子系统可执行代码，但它不利于实现基础平台子系统的灵活性。相对于概念分层结构风格以及概念分块结构风格，概念分级结构风格的长处和不足介于两者之间。

不难看出，上述三种基本的结构风格存在着某种联系。具体地说，概念分层结构风格是一种特殊的概念分级结构风格，概念分级结构风格则是一种特殊的概念分块结构风格。必须指出，概念分解原则可以用来构造任意级别的基础平台子系统结构；因而，基础平台子系统的各种部件也可以采用上述三种基本的结构风格。顺便指出，基于概念分解的基础平台子系统结构风格并不只限于上述三种基本的结构风格；它还包括其它结构风格，比如嵌套进程结构风格、元空间树结构风格等。

3.2 并发结构观点下的基础平台子系统总体结构风格

大家知道，绝大多数操作系统是多任务操作系统。在安装多任务操作系统的计算机系统中常常有多个应用软件同时运行。这些应用软件或者由于相互合作或者因为共享资源而存在着种种制约关系。因此，它们中的任何一个产生病态行为均将对其它应用软件构成威胁。显然，如何防范这种威胁是多任务操作系统开发者们必须考虑的主要问题。在操作系统工程实践中，人们常常采取空间隔离、访问控制以及特权保护这三种基本的保护措施来防范这种威胁[4]。通常，这三种保护措施是在多任务操作系统基础平台子系统中实现的。通过实现这三种保护措施，多任务操作系统基础平台子系统可以为计算机系统同时运行多个应用软件提供保护框架。

事实上，在多任务操作系统中，应用软件的各种病态行为对基础平台子系统也构成了威胁。经常地，基础平台子系统本身也是利用上述三种基本的保护措施进行自我保护的；换句话说，基础平台子系统本身也是在它自己提供的保护框架中运行的。不难理解，上述三种保护措施可被视作关于基础平台子系统内部行为逻辑的三种分解原则；相应地，基础平台子系统所提供的保护框架可被视作并发结构观点下的基础平台子系统总体结构。在下面的讨论中，我们把空间隔离、访问控制以及特权保护这三种保护措施统称为关于基础平台子系统内部行为逻辑的保护分解原则。下面我们分别讨论基于这些保护分解原则的基础平台子系统总体结构风格。

顺便指出，从原理上讲，人们在构造并发结构观点下的基础平台子系统总体结构时并非总是需要同时使用上述三种保护分解原则。这意味着，人们在构造并发结构观点下的基础平台子系统总体结构时可以使用也可以不使用某种保护分解原则。严格说来，如果某种保护分解原则没有被使用，那么把相关的基础平台子系统总体结构称为基于该保护分解原则的基础平台子系统总体结构并不贴切。不过，为了使有关讨论具有系统性，我们将把没有蕴涵某种保护分解原则的所有基础平台子系统总体结构视作一类特殊的基于该保护分解原则的基础平台子系统总体结构。

· 基于空间隔离的基础平台子系统总体结构风格

这种总体结构包含若干称为地址空间的计算部件。所谓地址空间，简单地说，就是程序在运行过程中可以寻址的所有逻辑存储单元所组成的集合。根据基础平台子系统总体结构所包含的地址空间的数目，我们可以把并发结构观点下的基础平台子系统总体结构分为两种类别：一是，至少包含两个地址空间的基础平台子系统总体结构；二是，仅仅包含单个地址空间的基础平台子系统总体结构。大家知道，基础平台子系统总体结构类别实际上就是基础平台子系统总体结构风格[2]。在下面的讨论中，我们把并发结构观点下的基础平台子系统总体结构的这两种类别分别称为多地址空间结构风格和单地址空间结构风格。

显而易见，多地址空间结构风格蕴涵着空间隔离这一保护分解原则，而单地址空间结构风格则没有蕴涵。由于空间隔离是防范运行中的各个程序彼此干扰的一种基本的保护措施，因此多地址空间结构风格有利于实现基础平台子系统的可靠性、安全性等非功能性需求，然而，由于在不同地址空间中运行的各个程序相互引用时会引起地址空间切换，而地址空间切换会产生系统开销，因此多地址空间结构风格会降低基础平台子系统的性能。与多地址空间结构风格不同，单地址空间结构风格不会影响基础平台子系统的性能；不过，如果不考虑其它保护措施，单地址空间结构风格将不利于基础平台子系统的可靠性、安全性等非功能性需求的实现。从操作系统工程实践的角度看，如果硬件平台只支持基础平台子系统实现较小的地址空间，那么基础平台子系统总体结构使用多地址空间结构风格将是一种切合实际的选择；当然，如果硬件平台支持基础平台子系统实现很大的地址空间，那么基础平台子系统总体结构使用单地址空间结构风格也是可行的。在后一种情形下，选择哪一种结构风格主要取决于人们对基础平台子系统相关非功能性需求的折衷考虑。

从并发结构观点看，现有的大多数多任务操作系统其基础平台子系统总体结构使用了多地址空间结构风格。此类操作系统既包括产品操作系统（比如 Windows NT[7]、QNX[8,14]、Mach[18,19]、Chorus[20,21]等）也包括试验操作系统（比如 Apertos[11]、SPACE[12]、Kea[22]、Agies[24]等）。顺便指出，从形式上看 Linux 基础平台子系统其可执行代码位于同一地址空间中，但实质上 Linux 基础平台子系统所占据的地址空间只是被各个运行中的应用软件共享的一个地址空间区域[16]。换句话说，Linux 基础平台子系统其可执行代码实际上是在不同的地址空间中运行的。从这个角度看，并发结构观点下的 Linux 基础平台子系统总体结构使用了多地址空间结构风格。基于相同的理由，我们可以说，传统的 UNIX[1-7]以及 QS/2[25]等其并发结构观点下的基础平台子系统总体结构同样使用了多地址空间结构风格。

Psyche[26]、PANDA[27]、Angel[28]、Mungi[29]、Opal[30]、Nemesis[31]等试验多任务操作系统其并发结构观点下的基础平台子系统总体结构使用了单地址空间结构风格，这些操作系统有这样一个共同特点：它们所依赖的硬件平台均支持大地址空间（比如 64 位或 64 位以上的寻址空间）的实现。不难理解，把计算机系统中的所有软件成份装配在一个小的地址空间（比如 32 位或 32 位以下的寻址空间）中运行是不现实的。顺便指出，随着支持 64 位寻址空间的硬件平台在市场上流行，单地址空间结构风格势必因其在性能等方面的优势的成为主流操作系统其基础平台子系统所采用的并发结构观点下的主要总体结构风格。当然，使这一趋势成为现实尚需人们解决关于单地址空间结构风格的诸多问题，其中一个问题是，单地址空间结构风格在可靠性方面存在着缺陷，很明显，访问控制可以用来解决这一问题[25]。

• 基于访问控制的基础平台子系统总体结构风格

一般说来，访问控制这一基本的保护措施主要用于保护某个软件成份在运行过程中不受在同一地址空间中运行的其它软件成份的干扰。因此，访问控制既可以用于采用多地址空间结构风格的基础平台子系统，也可以用于采用单地址空间结构风格的基础平台子系统。从访问控制的角度看，并发结构观点下的基础平台子系统总体结构包含若干称为保护域的计算部件。所谓保护域，简单地说，就是具有明确存取权限的程序地址空间中的一个区域。保护域的存取权限或简单或复杂。在操作系统工程实践中，人们常常用能力或访问控制表的形式表达保护域的存取权限。

根据基础平台子系统总体结构所包含的保护域的数目，我们同样可以把并发结构观点下的基础平台子系统总体结构分为两种类别，一是，至少包含两个保护域的基础平台子系统总体结构；二是，仅仅包含单个保护域的基础平台子系统总体结构。在下面的讨论中，我们把并发结构观点下的基础平台子系统总体结构的这两种类别分别称为多保护域结构风格和单保护域结构风格。显然，多保护域结构风格蕴涵着访问控制这一保护分解原则，而单保护域结构风格则没有蕴涵。不难理解，一个采用单保护域结构风格的基础平台子系统一定同时采用单地址空间结构风格。

由于访问控制是防范运行中的各个软件成份彼此干扰的一种基本的保护措施，因此多保护域结构风格有利于实现基础平台子系统的可靠性、安全性等非功能性需求；然而，由于在不同保护域中运行的各个软件成份彼此引用时需要进行保护域存取权限的检查，而这样做会产生系统开销，因此多保护域结构风格会降低基础平台子系统的性能。与多保护域结构风格所具有的优缺点相反，由于在同一保护域中运行的各个软件成份彼此引用时无需进行保护域存取权限的检查，因此单保护域结构风格不会影响基础平台子系统的性能；不过，若不考虑其它保护措施，单保护域结构风格将不利于基础平台子系统的可靠性、安全性等非功能性需求的实现。

就我们所知，现有的多任务操作系统，不管是产品操作系统还是试验操作系统，其并发结构观点下的基础平台子系统总体结构均使用了多保护域结构风格。当然，某些多任务操作系统其并发结构观点下的基础

平台子系统总体结构所包含的保护域具有简单的存取权限。从并发结构观点看，众所周知的单任务操作系统DOS[3]其基础平台子系统总体结构使用了单保护域结构风格；当然，DOS的基础平台子系统总体结构同时使用了单地址空间结构风格；事实上，DOS的基础平台子系统总体结构还使用了下面将要介绍的单模式结构风格。DOS的并发结构观点下的基础平台子系统总体结构之所以有这样一种组织形式是因为：DOS是为个人计算平台开发的单任务操作系统；就这样一种操作系统而言，保护问题不是主要问题。

·基于特权保护的基础平台子系统总体结构风格

这种总体结构包含若干我们称之为模式模块的计算部件。其中，不同的模式模块将在不同的特权模式下运行。所谓特权模式，简单地说，就是程序在运行过程中使用的由硬件平台体系结构提供的程序执行模式。比如，用Intel80x86CPU构筑的硬件平台其体系结构为运行中的程序提供了四种特权模式[14]。通常，硬件平台体系结构为在不同特权模式下运行的程序彼此进行交互制定了相应的规则。无疑，基于特权保护的基础平台子系统总体结构所包含的模式模块相互引用时将遵循这些规则。根据基础平台子系统总体结构所包含的模式模块的数目，我们可以把并发结构观点下的基础平台子系统总体结构分为两种类别：一是，至少包含两个模式模块的基础平台子系统总体结构；二是，仅仅包含单个模式模块的基础平台子系统总体结构。在下面的讨论中，我们把并发结构观点下的基础平台子系统总体结构的这两种类别分别称为多模式结构风格和单模式结构风格。

顺便指出，使用多地址空间结构风格的基础平台子系统总体结构其包含的地址空间的数目在系统运行过程中是变化的。类似地，使用多保护域结构风格的基础平台子系统总体结构其包含的保护域的数目在系统运行过程中也是变化的。与此不同，使用多模式结构风格的基础平台子系统总体结构其包含的模式模块的数目在系统运行过程中是固定不变的。十分明显，只有多模式结构风格才蕴涵着特权保护这一保护分解原则。一般说来，只有实现特权保护这一基本的保护措施、空间隔离和访问控制这两种保护措施才能真正发挥效用[4]。由此可以推断，采用多地址空间结构风格或多保护域结构风格的基础平台子系统一定同时采用多模式结构风格。

由于特权保护能够防范运行中的各个程序彼此干扰，因此多模式结构风格有利于实现基础平台子系统的可靠性、安全性等非功能性需要。然而，由于模式模块之间的相互引用会导致特权模式之间的切换，而特权模式之间的切换会产生系统开销，因此多模式结构风格会降低基础平台子系统的性能。此外，多模式结构风格会给基础平台子系统的开发增加一定难度，因为在较高级别特权模式下调试程序是一件困难的事情。与多模式结构风格不同，单模式结构风格不会增加基础平台子系统的开发难度。同时，由于单模式结构风格不会导致特权模式切换的问题，因而单模式结构风格不会影响基础平台子系统的性能。不过，若不考虑其它保护措施，单模式结构风格将不利于基础平台子系统的可靠性、安全性等非功能性需求的实现。

从并发结构观点看，国产系统软件平台COSA其基础平台子系统COSIX的总体结构使用了三模式结构风格[15]。基于特权保护的COSIX总体结构所包含的模式模块被COSIX开发者们分别称为应用软件、服务器以及微内核。相应地，COSIX模式模块所使用的三种特权模式被COSIX开发者们分别称为用户态、系统态以及核心态。就我们所知，现有的多任务操作系统，不管是产品操作系统还是试验操作系统，其并发结构观点下的基础平台子系统总体结构均使用了多模式结构风格，尤其是双模式结构风格。因篇幅所限，我们将另行撰文深入讨论双模式结构风格。

参考文献

1 Shaw M, Garlan D. Software Architecture: Perspectives on an Emerging Discipline. Prentice Hall, Englewood Cliffs, NJ, 1996

2 耿技，周明天，秦志光. 解析嵌定软件体系结构研究与设计基础的一组基本概念. 计算机科学，2001，28(10)

3 Stallings W. Operating System Internals and Design Principles. Third Edition, Prentice Hall, Englewood Cliffs, NJ, 1998

4 耿技，周明天. 现代操作系统概念. 计算机科学，1999，26(3)

5 Druschel P, Peterson L L, Hutchinson N C. Beyond Micro-Kernel Design: Decoupling Modularity and Protection in Lipto. In: Proc. of 12th Intl. Conf. on Distributed System (ICDCS), 1992. 512～520

6 Creasy R J. The Origin of the VM/370 Time-Sharing System. IBM Journal of Research and Development 1981, 25(5): 483～490

7 Custer H. Inside Windows NT. Microsoft Press, Redmond, WA, 1993(见：程渝荣译. Windows NT技术内幕. 清华大学出版社，1993)

8 Ford B, et al. Microkernel Meet Recursive Virtual Machines. In: Proc. of the Second Symposium on Operating Systems Design and Implementation, 1996. 137～152

9 Wilkes M V, Needham R M. The Cambridge CAP Computer and Its Operating System. North Holland, NY, 1979

10 Yokote Y, et al. The Muse Object Architecture: A New Operating System Structuring Concept. ACM Operating System Review, 1991, 25(2): 22～46

11 Yokote Y. The Apertos Reflective Operating System: The Concept and Its Implementation. In: Proc. of Object-Ori-

ented Programmnng Systems, Languages and Applications in 1992, Oct. 1992
12 Horse M, et al. Designing Meta-Interfaces for Object-Oriented Operating Systems. In: Proc. of 1997 IEEE Pacific Rim Conf. on Communications, Computers, and Signal Processing, 1997 989～992
13 Hildebrand D. An Architecutral Overview of QNX. In, Proc. of the Usenix Workshop on Micro-kernels and Other Kernel Architectures, 1992. 113～126
14 侯业勤, 张菁编译. 分布式嵌入式实时操作系统 QNX. 宇航出版社, 1999
15 Bowman I. Conceptual Architecutre of the Linux Kernel. Jan. 1998 (Available from http: // plg. uwaterloo. ca/ ~itbowman/CS746G/al/)
16 陈莉君编著. Linux 操作系统内核分析. 人民邮电出版社, 2000
17 Bach M J. The Design of the UNIX Operating Systems. Prentice Hall, Englewood Cliffs, NJ, 1986 (见: 陈葆钰等译. UNIX 操作系统设计. 北京大学出版社, 1989)
18 Acetta M, et al. Mach: A New Kernel Foundation for UNIX Development In. Proc. of the Summer 1986 USENIX Conf., 1986. 93～112
19 Black D L, et al. Microkernel Operating System Architecture and Mach In: Proc. of the USENIX Workshop on Microkernels and Other Kernel Architectures. Seattle, Washington, Apr 1992 11～30
20 Guillemont M. The Chours Distributed Operating System: Design and Implementation. In, ACM International Sympsium on Local Computer Networks, Firenze, Apr. 1982. 207～223
21 Rozier M, et al. CHORUS Distributed Operating System. Computing Systems Journal, The USENIX Association, Dec. 1988, 1(4). 350～370
22 Probert D, Bruno J. Building Fundamentally Extensible Application-Specific Operating Systems in SPACE: [Technical Report TRCS95-06]. Computer Science Department, UC Santa Barbara, Mar. 1995
23 Veitch A C, Hutchinson N C. Kea-A Dynamically Extensible and Configurable Operating systems Kernel. In: Proc. of the Third Conf. on Configurable Distributed Systems, 1996
24 Engler D R The Design and Implementation of a Prototype Exokernel Operating System: [MA Thesis] MIT, Oct. 1998
25 Deitel H M, Kogan M S. The Design of OS/2. Addison Wesley, NY, 1992
26 Marsh B D. Muti-Model Parallel Programming: [PhD Thesis]. Department of Computer Science, University of Rochester, Jan. 1992
27 Assenmacher H, et al. The PANDA System Architecture--A Pico-Kernel Approach. In: Proc. of the Forth Workshop on Future Trends of Distributed Computing Systems, Sep. 1993. 470～476
28 Murray K, et al. Design and Implementation of an Object-orientated 64-bit Single Address Space Microkernel. In. Proc. of the 2nd USENIX Symposium on Microkernels and Other Kernel Architecutres, Sep. 1993. 31～43
29 Heiser G, et al. Mungi: A Distributed Single Address Space Operating System: [Technical Report UNSW-CSE-TR-9314]. School of Computer Science and Engineering, University of New South Wales, Sydney, Australia, Nov 1993
30 Chase J S, et al. Sharing and Protection in a Single Address Space Operating System. ACM Transaction on Computer Systems, 1994(12): 271～307
31 Roscoe T. The Structure of a Multi-Service Operating Systme: [PhD Thesis]. Queens' College, University of Cambridge, Apr. 1995
32 Wilkinson T, et al. Single Address Space Operating Systmes: [Technal Report UNSW-CSE-TR-9504]. School of Computer Science and Engineering, University of New South Wales, Sydney, Australia, Nov. 1995
33 张载鸿编. 局部网操作系统 DOS 高级技术分析. 国防工业出版社, 1988
34 INTEL 资料. Intel Architecutre Software Developer's Manual. 1997
35 孟庆余. 系统软件平台 COSA 的设计与实现. 电子学报. 1999, 27(2)

(上接第 5 页)

11 Dincer K, Fox G C. Design Issues in Building Web-Based Parallel Programming Environments. In: Proc. of the Sixth IEEE Intl. Symposium High Performance Distributed Computing. Portland State University, Portland, Oregon, Aug. 1997
12 Dincer K, Fox G C. Using Java and JavaScript in the Virtual Programming Laboratory: A Web-Based Parallel Programming Environment. Concurrency: Practice and Experience, June, 1997
13 Fox G C, Furmanski W. The Use of The National Information Infrastructure and High Performance Computers in Industry. [NPAC Technical Report SCCS-732]. July, 1995
14 Alexandrov A D, et al. Ufo: Personal Global File System Based on User-Level Extensions to the Operating System. In: Proc. of the 1997 USENIX Technical Conf. Anaheim, CA, Jan. 1997
15 Gadde S, Chase J, Rabinovich M. Web Caching and Content Distribution: A View From the Interior. In: Proc. of The 5th International Web Caching and Content Delivery Workshop, May 2000
16 Anderson E, Patterson D, Brewer E. The Magicrouter, an Application of Fast Packet Interposing. http.// HTTP.CS. Berkeley EDU/～eanders/magicrouter/
17 Tennenhouse D, Wetherall D. Towards an Active Network Architecture In ACM SIGCOMM Computer Communication Review, April, 1996. 5～18
18 Dongarra J, et al. High Performance Computing Today. To Appear in FOMMS 2000: Foundations of Molecular Modeling and Simulation Conference
19 Yu Haifeng, Vahdat A. Design and Evaluation of a Continuous Consistency Model for Replicated Services In Proc. of the Fourth Symposium on Operating Systems Design and Implementation (OSDI), Oct. 2000