[19] 中华人民共和国国家知识产权局

[51] Int. Cl[7]
G06F 9/44

[12] 发明专利说明书

[21] ZL 专利号 94190757.0

[43] 授权公告日 2003 年 6 月 18 日

[11] 授权公告号 CN 1111787C

[22] 申请日 1994.1.6 [21] 申请号 94190757.0
[30] 优先权
[32] 1993. 7.19 [33] US [31] 08/094,675
[86] 国际申请 PCT/US94/00197 1994.1.6
[87] 国际公布 WO95/03577 英 1995.2.2
[85] 进入国家阶段日期 1995.6.5
[71] 专利权人 塔里根特公司
地址 美国加里福尼亚州
[72] 发明人 德布拉·林·奥顿
尤金尼·李·博顿
丹尼尔·F·切尼科夫
戴维·布鲁克·戈德史密斯
克里斯托弗·P·蒙洛
审查员 杨 蕊

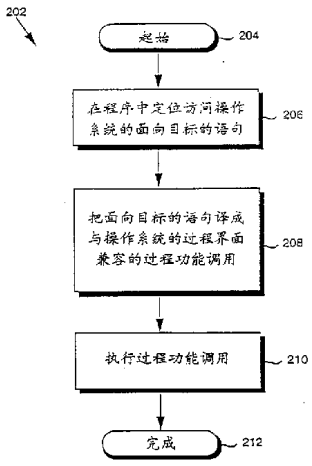
[74] 专利代理机构 北京康信知识产权代理有限责任公司
代理人 余 刚

权利要求书 5 页 说明书 70 页 附图 17 页

[54] 发明名称 使面向目标的应用程序与过程操作系统接口的方法与装置

[57] 摘要

一种使面向目标的应用程序以面向目标的方式访问具有本机过程界面的过程操作系统的装置，包括一个计算机和位于其中的存储器部件；一个编码库存于该存储部件之中。 该编码库包括实施一个面向目标类别库的计算机程序逻辑，该类别库包括使应用程序以面向目标的方式访问由操作系统提供的服务的面向目标的类别，该类别包括用与操作系统的本机过程界面兼容的过程功能调用访问操作系统服务的方法。 通过执行来自与面向目标的语句对应的类别库的方法，计算机对由类别库定义并包括在应用程序中的面向目标的语句进行处理。



ISSN1008-4274

1. 一种计算机实施方法，该方法使面向目标的应用程序(132)与包括过程功能(128)的过程操作系统接口，该过程功能被调用从而在具有存储部件(108)的计算机(102)执行面向目标的应用程序(132)的运行期间内访问由过程操作系统(114)提供的服务，该方法的特征在于包括以下步骤：

(a) 在面向目标的应用程序中定位一个面向目标的语句，该语句访问过程操作系统提供的服务；

(b) 将面向目标的语句译成与过程功能兼容并与该面向目标的语句相应的过程功能调用，所述过程功能被调用以访问过程操作系统提供的服务；以及

(c) 在计算机中执行与被调用以访问过程操作系统提供的服务的过程功能相兼容的过程功能调用，从而导致所述过程操作系统为面向目标的应用程序提供服务。

2. 根据权利要求1所述方法，还包括使面向目标的应用程序访问过程操作系统提供的服务以访问和操纵计算机中的虚拟存储器的步骤(408)。

3. 根据权利要求2所述方法,还包括使面向目标的应用程序访问一个虚拟存储范围以获取与该虚拟存储范围相关的存储器属性信息并将存储器目标映射到与一任务关联的地址空间的步骤(802)。

4. 根据权利要求1所述方法,包括使应用程序访问过程操作系统中的服务以便初始化,控制和刷新与任务关联的信息的步骤(404,406,1206,1320,1504,1604)。

5. 根据权利要求4所述方法,还包括示例一个第一面向目标的类别和第二面向目标的类别的步骤,该第一面向目标的类别用于封装需用来生成一个新任务的信息,该第一面向目标的类别是一抽象主类别,该第二面向目标的类别通过把第一类别的子类别送到第二类别的示例而使应用程序将一个新任务初始化,并使应用程终止、中止,恢复和调度一个现有任务,其中,第二类别的示例代表在计算机中的运行时间处理实体(404,406,1206,1320,1504,1604)。

6. 根据权利要求 1 所述方法,还包括以下步骤:

(a) 实现一个面向目标的类别库,该类别库包括面向目标的内部处理通信(IPC)类别(410);

(b) 通过内部处理通信类别与其他处理在计算机执行面向目标的主用程序的运行时间内进行通信, 该面向目标的类别包手利用过程功能调用访问过程操作系统的方式而对过程操作系统进行访问的方法(208,702),以及

(c) 把来自对应于面向目标的语句的面向目标的类别库的方法插入面向目标的应用程序以使面向目标的应用程序在计算机中执行面向目标的应用程序的运行时间中访问过程操作系统服务。

7. 用于使包括面向目标语句的面向目标的应用程序与包括过程的过程操作系统一起操作的装置,所述过程可以被调用以便在具有存储部件的计算机中执行面向目标的应用程序的运行时间内访问由过程操作系统提供的服务,该装置的特征在于包括:

(a) 位于所述存储部件中的类别库,所述类别库包括一目标类别,所述目标类别包括一个功能块,当被运行时,所述功能块执行一个过程功能调用,所述过程功能调用与所述操作系统过程兼容;和

(b) 一个第一环绕模块,其响应面向目标应用程序中的面向目标句,以使所述类别库中的目标类别定位,

所述目标类别响应面向目标语句；以及

(c) 一个第二环绕模块，其在存储部件中从所述目标类别建立一个目标，并运行该目标中的功能块，所述功能块反过来执行过程功能调用以便访问过程操作系统提供的服务。

8. 根据权利要求7所述装置，还包括使面向目标的应用程序访问面向目标的操作系统提供的服务以访问和操纵计算机的虚拟存储器的装置(408)。

9. 根据权利要求 8 所述装置，还包括使面向目标的应用程序访问虚拟存储器范围以获取与该范围相关的存储器属性信息并将存储器目标映入与一任务相关的地址空间的装置(802)。

10. 根据权利要求7所述装置，还包括使应用程序访问过程操作系统中的服务以初始化、控制、和刷新与任务关联的信息的装置(404,406,1206,1320,1504)。

11. 根据权利要求 10 所述装置,还包括用于示例一个第一面向目标的类别和一个第二面向目标的类别的装置,该第一类别封装用于生成新任务的信息,该第一类别是一抽象主类别,该第二类别用于通过把一个第一类别的子类别送到第二类别的示例而使应用程序初始化一个线索,并用于使应用程序终止、中止、恢复和调度一个当前任务,其中,第二类别的示例代表在计算机中的运行时间处理实体(404,406,1206,1320,1504,1604)。

12. 根据权利要求 7 所述装置,还包括:

(a) 用于实现面向目标的类别库的装置,该类别库包括面向目标的内部处理通信(IPC)类别(410);

(b) 用于在计算机执行应用程序的运行时间内通过该内部处理通信类别与其他处理通信的装置,所述面向目标的类别包括用过程功能调用访问过程操作系统的方式对过程操作系统服务进行访问的方法(208,702);以及

(c) 把来自对应于面向目标的语句的所述面向目标的类别库的方法插入面向目标的应用程序以使面向目标的应用程序在计算机执行该面向目标的应用程序的运行时间内访问过程操作系统服务的装置(302,1302)。

使面向目标的应用程序与过程操作系统接口的方法和装置

本发明涉及面向目标的计算机环境，尤其涉及为过程操作系统提供面向目标的界面的系统和方法。

本发明的部分内容受版权保护。版权拥有者允许该部分内容在专利局的文档案卷中出现但保留其他权利。

面向目标的技术（OOT）通常包括面向目标的分析（OOA），面向目标的设计（OOD），以及面向目标的编程（OOP），该技术在软件开发过程中作为最重要的新兴技术之一而赢得了一席之地。OOT 已经开始证明其有能力在编程及程序维护中导致巨大产出和增长，通过产生一个把数据以及操作在数据上的过程相结合而生成的目标环境，并采用命令目标与另一目标仅通过完善定义的信息通路进行通信的法则，面向目标的技术去除了大量复杂的传统的面向过程的编程。

以下的段落表达了 OOT 某些更重要方面的简述。有关 OOT 的详细叙述在许多公开发行的文章中均有记载，这些文章包括 Grady Booch 所著的“面向目标的应用设计（Object-Oriented Design with Applications）”，该书由本杰明/酷明出版公司（Benjamin/Cummings Publishing Company）在 1991 年出版，以及由 Donald G. Firesmith 所著的“面向目标的要求分

析以及逻辑设计(Object—Oriented Requirements Analysis and Logical Design)",该书由 John Wiley & Sons 公司在 1993 年出版。OOT 的基本部件为目标。一个目标包括(且其特征在于)一组数据(也称为属性)以及一组操作(亦称为方法),该组操作可在数据上进行操作。通常,一个目标的数据只可经由该目标的方法的操作而予改变。

通过把一信息传送给目标(该过程称为信息传送)的方式便可调用在一个目标中的方法。该信息指定一个方法名以及自变量表。当目标接收到该信息时,与所命名方法相关的码便和与自变量表中的对应值关联的方法的形式参数一起被执行。在 OOT 中的信息传递以及方法与面向过程的软件环境中的过程和过程调用类似。但是,过程操作是修改并返回传送的参数,方法操作则是修改相关目标的内部状态(通过修改包含于其中的数据)。在目标中的方法和数据的结合称之为封装。封装的最大好处之一在于任何目标的状态仅由已完善定义并与该目标相关的方法予以修改。当目标的行为限于这种完善定义区域及界面时,目标中的改变(即码修改)对系统中的其他目标和单元的影响是最小的。面向目标设计和编程良好封装的第二个好处则在于所得到的码较之常规技术生成的码更模块化并更易于维护。

对目标封装这一事实所产生的另一重要好处是数据抽象。抽象是一种处理。通过抽象,行为被结合而细节被去除,从而使复杂概念及结构易于理解。从软件观点来看,抽象在许多方面与硬编码相反,试想一个软件视窗:若一个图形用

户界面(GUI)程序的每个视窗的每个细节都出现在用户的屏幕上并且必须将其所有状态和行为硬编码成一程序，则该程序以及包含该程序的视窗会失去几乎所有的灵活性。通过把视窗的概念抽象成为一个视窗目标，面向目标的系统使得程序员只考虑生成一个唯一特定视窗的那些特定方面。由所有视窗共享的行为例如被拖曳和移动的能力，可由所有视窗目标所共享。

这导致了 OOT 的另一基本部件，即类别。一个类别包括一组数据属性及一组可在数据属性上进行的操作(即方法)。每个目标是某些类别的一个实例。作为封装和抽象的一个自然结果，OOT 支持继承。一个类别(称为子类别)可从另一类别(称为主类别，母类别等等)中衍生而来。其中，子类别继承了主类别的数据属性和方法。通过加入置换主类别的数据和/或方法或加入新数据属性和方法的编码的方式，该子类别可以限定该主类别，如此继承表达了一种操作，当子类别为更高级的限定而生成时，这种操作可使抽象更为具体。继承是对 OOP 提供的高效编程的主要贡献之一。继承使开发者得以将其用于生成应用程序的新编码数量减至最小。通过提供一个特定任务所需的大部分功能性，在继承层级中的类别为程序员的程序设计和生成提供了一个高起点。面向目标的环境的一个潜在缺陷是目标增生，其必须展示类似的行为以及哪个目标会用作一个单独信息名以说明。例如，考虑一个面向目标的图形环境：若一个 Draw 信息送到一个 Rectangle 目标，则该 Rectangle 目标通过绘制四边形而予以响应。另一

方面，一个 Triangle 目标的响应则是绘制一个三边形。理想地，送出 Draw 信息的目标应并不意识信息所寻址的目标类型，也不管接收信息的目标将如果响应以绘制出其本身。若可达到这一理想状况，则在稍后加入新的形状类型(如六边形)然后编码送出完全不变的 Draw 信息就相当简单的了。

通常，面向过程的语言会导致混乱。在 OOT 环境中，多形性(Polymorphism)，是指在面向目标的编程中，通过继承而关联的不同类别的目标唯一地响应相同成员功能调用的操作处理)的概念可使其完满解决而无损害。作为一个结果，方法可被编写只是概括地告知其他目标做某些事情而无需要求送出任何关于接收目标理解该信息的方式的知识。无论是面向目标，面向过程，及基于规则等的软件程序，它们几乎总是与操作系统交互以访问操作系统提供的服务。例如，一个软件程序可与操作系统交互以便对存储器中的数据进行存取，接收与处理器故障相关的信息，与其他处理通信，或调度处理的执行。

大部分常规操作系统是面向过程的且包括本机过程界面。结果，这些操作系统所提供的服务只可用它们各自的过程界面定义的过程所访问。若一个程序需要访问过程操作系统之一提供的一个服务，则程序必须包括一个语句以进行适当的操作系统过程调用。这与软件程序为面向目标，面向过程，或基于规则等无关。如此，常规操作系统提供了面向过程的环境，其中要进行软件开发和执行。某些 OOT 的优点在面向目标的程序在面向过程的环境中进行开发和执行时会失

掉。这是由于所有对过程操作系统的访问都必须用操作系统的本机过程界面所定义的过程调用来实施。结果，由于不能充分利用类别、目标、以及其他 OOT 特性，与面向目标的程序关联的某些模块特性、可维护特性以及自使用特性便会失去。

对该问题的一个解决方法是开发具有本机面向目标的界面的面向目标的操作系统。虽然这是一个最好的方案，但其现在这未能实用，因为需要修改所有主要过程操作的资源过于庞大，进而，对过程操作系统的修改会导致数以千计的面向过程的软件程序无用。因此，所需的是一种可使面向目标的应用程序以面向目标的方式与具有本机过程界面的过程操作系统进行交互的处理。

本发明是要提供一种使面向目标的应用程序以面向目标的方式对具有本机过程界面的过程操作系统进行访问的系统和方法。该系统包括一个计算机以及其中的存储部件。存储部件中存有一个编码库。该编码库包括实现一个面向目标的类别库的计算机程序逻辑。该面向目标的类别库包括相关的面向目标的类别，用于使应用程序以面向目标的方式访问由操作系统提供的服务。面向目标的类别包括利用与操作系统的本机过程界面兼容的过程功能调用，对操作系统进行访问的方法。系统还包括对面向目标的语句进行处理的装置，这些面向目标的语句包含在应用程序中，且由与其相应的类别库产生的执行方法所定义。

该类别库最好包括：

(1)线索类别，该类别用于使应用程序以面向目标的方式访问操作系统服务从而产生，控制和获取与线索有关的信息；

(2)任务类别，用于使应用程序以面向目标的方式访问操作系统服务以便引用和控制任务，其中每个任务都表示一个分别与任务相关的线索的执行环境；

(3)虚拟存储类别，用于使应用程序以面向目标的方式访问操作系统服务以访问和操纵计算机中的虚拟存储器；

(4)内部处理通信类别(IPC)，用于使应用程序以面向目标的方式访问操作系统服务从而在应用程序在计算机运行时间执行过程中与其他线索进行通信；

(5)同步类别，用于使应用程序以面向目标的方式访问操作系统服务以与线索的执行同步。

(6)调度类别，用于使应用程序以面向目标的方式访问操作系统服务从而调度线索的执行；

(7)故障类别，用于使应用程序以面向目标的方式访问操作系统服务以便处理系统和定义用户的处理器故障

(8)机器类别，用于使应用程序以面向目标的方式访问操作系统服务以定义和修改一个主机和处理器组。

本发明的特性和优点以后多种实施例的结构和操作将结合附图予以详述。在附图中，相同的数码指示相同或功能相似的部件。

图 1 示出了一个计算机结构的框图，本发明的环绕程序

(wrapper)操作于其中，

图 2 是展示本发明操作的高级流程图。

图 3 是展示本发明操作的更详细的流程图；

图 4 是包括本发明的面向目标类别库的编码库的框图；

图 5 是本发明的线索和任务类别的类别图；

图 6 是本发明的虚拟存储类别的类别图；

图 7～9 是本发明的内部处理通信类别的类别图；

图 10 是本发明的同步类别的类别图；

图 11 是本发明调度类别的类别图；

图 12～15 是本发明故障类别的烊别图；

图 16 是本发明的主机和处理器组类别的类别图；以及

图 17 示出了用于表示类别图中的类别关系以及重要性的已知图符。

计算环境

本发明提供了用于向具有本机过程界面的过程操作系统的提供面向目标的界面的系统和方法。本发明在一个具有过程操作系统的计算机结构上模仿面向目标的软件环境。尤其是，本发明提供了一种系统和方法，它们使面向目标的应用程序以面向目标的方式在应用程序在计算机中运行过程中对具有本机过程界面的过程操作系统进行访问。本发明最好是应用程序于其中执行的计算机的运行时间环境的一部分。在本专利申请中，本发明有时被称为面向目标的环绕程序，因为其操作环绕于具有面向目标软件层的过程操作系统，以致面向目标的应用程序可以面向目标的方式访问操作

系统。

图1示出了一个计算机结构102的框图，本发明的环绕程序128和129操作于其中。在此要提请注意的是本发明交替地把环绕程序128，129与计算机结构102相结合。计算机结构102包括硬件部件103，例如RAM108和CPU106。CPU106可以是一个单独的处理器，但最好是多个并行操作的处理器。计算机结构102还包括与硬件部件103相连的外设。这些外设包括输入设备(例如键盘，鼠标器，光笔等)数据存储设备120(例如硬盘或软盘)、显示124以及打印机126。数据存储设备120可与可擦除数据存储介质122(例如可擦除硬盘，盒式磁带或软盘)交互，其取决于所用数据存储设备的类型。计算机结构102还包括一个过程操作系统114，它有一未示出的本机过程界面。该过程界面包括被调用来访问操作系统102提供的服务的过程功能。

计算机结构102还包括设备驱动器116，并可包括微指令码(亦称为固件)。如图1所示，在执行需要的功能时，设备驱动器116可与操作系统114相交互。应用程序130，132，134(以下将一步叙述)最好通过操作系统114而与设备驱动器116相交互，但亦可交替地直接与设备驱动器116交互。应注意的是操作系统114可代表基本上全功能的操作系统，例如DOS和UNIX操作系统。但是，操作系统114可代表其他类型的操作系统。在本发明中，对操作系统114的唯一要求是具有本机过程界面的过程操作系统。操作系统114最好代表一个有限功能的过程操作系统，例如由CMU开发的

Mach 微核心程序(micro—kernel),它是本领域已知技术。为说明之目的,本发明仅引用 Mach micro—kernel,在本发明的最佳实施例中,计算机结构 102 是 IBM 或 IBM 兼容计算机。在本发明的替换实施例中,计算机结构 102 是一个苹果计算机。

环绕程序概述

应用程序 130,132,134 最好在计算机 102 上并行操作,而且它们最好用于不同操作环境下。例如,应用程序 130A 和 130B 可操作于面向目标的环境程序。应用程序 132 操作于微软视窗环境,IBM PS/2 环境,或 Unix 环境下。正如本技术领域的一般技术人员所理解的,程序 130A、130B 和 132 不能直接与操作系统 114 交互,除非操作系统 114 实施在 130A,130B 和 132 可以操作的环境。例如,若程序 132 操作于 IBM PS/2 环境下,则其不能与操作系统 114 交互,除非该操作系统 114 是 IBM PS/2 操作系统(或兼容系统)。若程序 130A 和 130B 操作于面向目标的环境,则其不能直接与操作系统 114 交互,因为操作系统有一过程界面。在图 1 所示实例中,程序 134 操作于操作系统 114 生成的计算环境中,因此其可与操作系统 114 直接交互。

环绕程序 128 是要为操作系统 114 提供具有面向目标的界面。环绕程序 128 使面向目标的程序 130A 和 130B 以面向目标的方式在计算机 102 执行程序 130A 和 130B 的运行时间期间直接访问过程操作系统 114。环绕程序 129 的概念与之相似,其为操作系统 114 提供 IBM PS/2 界面,从而使

程序 132 可以 PS/2 的方式直接访问过程操作系统 114(假设程序 132 操作于 IBM PS/2 环境下)。本发明的讨论将限于环绕程序 128,它为具有本机过程界面的过程操作系统提供了面向目标的界面。

环绕程序 128 最好象存在 RAM108 中的编码库 110 一样实施。编码库 110 也存在数据存储设备 120 和/或数据存储介质 122 中。该库 110 实现面向目标的类别库 402(见图 4)。根据本发明,面向目标的类别库 402 包括相关的面向目标类别,用于使一个面向目标的程序(例如 130A 和 130B)以面向目标的方式访问操作系统 114 提供的服务。面向目标的类别包括具有与操作系统 114 的本机过程界面兼容的过程功能调用。由面向目标的类别库 402 定义的面向目标的语句(例如调用类别库 402 的至少一个方法的面向目标的语句)被插入到程序 130 以使程序 30 可以面向目标的方式在计算机 102 执行程序的运行时间期间访问操作系统服务。面向目标的类别库 402 将在以下叙述。

编码库 110 最好包括编译过的可执行计算机程序逻辑,该逻辑实现面向目标的类别 402。编码库 110 的计算机程序逻辑不与应用程序链接。代之的是把编码库 110 的相关部分复制到运行时间期间的处理的可执行寻址空间。这在下面将作更详细解释。由于编码库 110 的计算机程序逻辑不与应用程序相连,所以可在任何时刻对其进行修改而不必修改、重新编译和/或重新连接应用程序(只要编码库 110 的界面不改变)。如上所述,本发明将参照 Mach micro—kernel 进行叙

述，虽然用本发明来环绕其他操作系统亦属于发明的范围。

Mach micro kernel 为用户提供了若干服务，它们按照下述范畴进行分组：线索，任务，虚拟存储器，内部处理通信(IPC)，调度，同步，故障处理，以及主机/处理器组处理。本发明的类别库 402 包括一组用于每个 Mach 服务范畴的相关类别。参见图 4，类别库 402 包括：

(1)线索类别(thread)，是指使一个应用程序以面向目标的方式访问操作系统的服务以产生、控制和获得与线索有关的信息；

(2)任务类别 406，用于使应用程序以面向目标的方式访问操作系统服务以引用和控制任务，其中的每个任务代表一个分别与任务关联的线索的执行环境；

(3)虚拟存储类别 408，用于使应用程序以面向目标的方式访问操作系统服务以访问和操纵计算机中的虚拟存储器。

(4)IPC 类别 410，用于使应用程序以面向目标的方式访问操作系统服务以便在程序在计算机中执行的运行时间期间与其他处理通信。

(5)同步类别 412，用于使应用程序以面向目标的方式访问操作系统服务以同步执行线索；

(6)调度类别 414，用于使应用程序以面向目标的方式访问操作系统服务以调度线索的执行；

(7)故障类别，用于使程序以面向目标的方式访问操作系统服务以处理系统和定义用户的处理故障；以及

(8)机器类别418,用于使程序以面向目标的方式访问操作系统服务以定义和修改一个主机和处理器组。

类别库402可包括用于Mach将来提供的其他服务范畴的其他类别。例如,正在为Mach开发保密服务。因此,类别库402也可包括保密类别420,用于使应用程序以面向目标的方式存取操作系统保密服务。如所知晓的,包括在类别库402中的类别的准确数量和类型取决于基本操作系统的实现。

最佳实施例的操作综述

本发明的操作将结合图2予以综述。图2示出了本发明高级操作流程图202。本发明是根据在计算机结构102上执行面向目标的应用程序130A的全过程而予叙述的。在步骤206中(它是流程图202的第一步骤),对操作系统114提供的服务进行访问的一条面向目标的语句在计算机102执行程序130A期间定位在程序130A中。该面向目标的语句由面向目标的类别库402定义。例如,面向目标语句可参照一个由类别库402的一个类别定义的方法。下述步骤叙述了计算机结构102执行该语句的方式。

在步骤208中,面向目标的语句被译成与操作系统114的本机过程界面兼容并与该面向目标的语句对应的一个过程功能调用。在步骤208处,该语句被译为实施该语句引用的方法并来自编码库110的计算机程序逻辑。如上所述,该方法包括至少一个与操作系统114的本机过程界面相兼容的过程功能调用。在步骤210中,来自步骤208的过程功能

调用在计算机结构102中执行，从而导致操作系统114提供为了应用程序130A的服务。步骤210是由执行步骤208中叙述的方法而实施的，从而导致调用过程功能调用。

一个最佳实施例的操作参照图3予以详述。图3示出了本发明的详细操作流程图302。进而，还根据在计算机结构102上执行面向目标的应用程序130A的全程来描述本发明。尤其是，本发明根据在计算机结构102上执行的面向目标应用程序130A的一条面向目标的语句的过程而予叙述。应用程序130A包括访问操作系统114提供的服务的语句，而且假设这些语句是由类别库402定义的(换言之，程序员参照类别库402来生成程序130A)。如以下将详述的，在Mach micro—kernel中可执行的实体称作为线索。在Mach micro—kernel处理组织实体称作为任务。一个任务包括至少一个线索(它可并行执行)以及一个代表任务的线索可被执行的虚拟存储器的块的地址空间。在任何时候都可有多个有效的任务在计算机结构102上。当在计算机结构102上执行应用程序130A时，该程序可代表一个完整任务(有至少一个线索)，或可代表作为一部分任务的几个线索(在这种情况下，任务有其它与程序130A的操作相关或无关的线索)。本发明范围包括当应用程序130A是一完整任务或只是一个任务的几个线索时的情况。

参见图3，步骤308确定从实施语句引用的方法的编码库110而来的计算机程序逻辑(亦称为计算机码)是否展示在与程序130A关联的任务地址空间中。若是，则步骤316

被处理(以下将叙述)。若否,则计算机程序逻辑在步骤310,312,和314中被送到任务地址空间。步骤310确定是否已知与编码库110关联的库服务器(未示出)。编码库110可代表多个与环绕程序128关联的编码库(未示出),其中,每个编码库包括用于类别库402的面向目标的类别之一的计算机程序逻辑。如本技术领域一般技术人员所知,亦可有其他与环绕程序128完全无关的编码库(未示出)。

库服务器与编码库相关,每个服务器管理指定的编码库的资源。希望对一个编码库的计算机程序逻辑进行访问的一个处理实体对编码库的库服务器发出一个请求。该请求可包括例如所希望的计算机程序逻辑的描述以及该计算机程序逻辑应被送至的目的地址。库服务器通过访问来自编码库的计算机程序逻辑并将其送至目的地址指定的存储器区域而处理该请求。库服务器的结构和操作是众所周知的。如此,步骤310确定与包含相关计算机逻辑的编码库110关联的库服务器是否是已知的。例如,通过引用标识已知的库服务器以及它们服务的编码库的库服务器表示实现步骤310。若库服务器已知,则进行步骤314的处理。否则去往步骤312。步骤312标识与编码库110有关的库服务器。库服务器的标识可以从正被处理的面向目标语句的内容中清楚得知。

在标识了与编码库110相关的库服务器之后或若库服务器已知,则进行步骤314的处理,其中,一个请求被送至库服务器以要求其把与语句引用的方法关联的计算机程序逻辑复制到任务地址空间。一旦完成步骤314,库服务器已把

所请求的计算机程序逻辑复制到了任务地址空间。编码库110最好是一个共同库。即编码库110可同时由多个线索访问。但是，编码库110的计算机程序逻辑最好物理地存在一个唯一的物理存储区中。库服务器虚拟地将编码库110而来的计算机程序逻辑复制到任务地址空间。也就是说，代之以物理把物理存储器的一部分中的计算机逻辑复制到另一个中，库服务器在任务地址空间中放置一个指向包含相关计算机程序逻辑的物理存储区的指针。在步骤316，在计算机结构102上执行与面向目标的语句关联的计算机程序逻辑。如上指出的，一旦面向目标的语句访问操作系统114，与方法关联的计算机程序逻辑包括至少一个与操作系统114的本机过程界面兼容的过程功能调用。如此这样执行方法的计算机程序逻辑来启动并执行过程功能调用，从而导致操作系统114为应用程序130A提供服务。

步骤306，308，310，312和314大部分实现于计算机结构102建立的运行时间环境。如所周知，计算机结构102的运行时间环境是由对程序1304进行编译的特定编译程序的运行时间规范所限定的。例如，运行时间规范可指示在遇到一个指令访问一个操作系统服务时，从编码库110而来的相应编码应被传送到任务地址空间(通过相关的库服务器)并被执行。编译程序运行时间规范是众所周知的。运行时间规范是针对所采用的特定编译程序的。在参照本发明的叙述，尤其是图3的流程图302之后，普通技术人员可以理解在本发明中和一特定编译程序中所用的运行时间规范。如上所

述，本发明的环绕程序 128 是作为编码库 110 实现的。编码库 110 包括实现面向目标的类别库 402 的计算机程序逻辑。替换之，环绕程序 128 可作为硬件装置实现。该硬件装置基本根据图 3 的流程图 302 把应用程序中的面向目标的语句（由类别库 402 定义）翻译成为与操作系统 114 的过程界面兼容的过程功能调用。或者，环绕程序 138 可作为操作在计算机结构 102 上的软件处理而予实现。该软件处理捕捉所有对操作系统 114 的访问（这由类别库 402 定义面向目标的语句完成）并将这些访问译为与操作系统 114 的过程界面兼容的过程功能调用。环绕程序 128 的其他实现可依据本发明阐述而予完成。

Mach 服务

本章节对 Mach micro—Kernel 提供的抽象和服务进行的综述。这些服务针对 Mach micro—kernel 的每个主要区域进行叙述。如上所述，这些主要区域包括线索、任务、虚拟存储器、IPC、调度、同步服务、硬件故障，以及主机/特权服务（也称为机器服务）。在许多公开发行的刊物和书籍中对 Mach micro—kernel 都有进一步的描述。它们包括 K. Loepere 在 Open Software Foundation and Carnegie Mellon University，Draft Industrial Specification 上发表的“Mach 3 Kernel 原理”，“Mach 3 Kernel 界面”，“Mach 3 服务器编写指南”，以及“Mach 服务器编写界面”（1992 年 9 月和 11 月）；A. Silberschatz，J. Peterson，P. Galvin 在 1992 年 7 月发表的“操作系统概念”（由 Addison—uesley 出版社出版），以及由 A. Tanen-

baum 在 1992 年发表的“现代操作系统”(由 Prentice Hall 出版)。

线索

在 Mach 中的可执行实体即为线索。线索有几个方面使其可在系统中执行。线索总是包含在任务中,它代表线索可利用的大部分主要资源(即地址空间)。一个线索有一个执行状态,它基本是机器寄存器组和组成其上下文其他数据。一个线索总是处于几个预定状态之一,即执行,准备执行,或某种原因之中上。线索应是轻加权(light—weight)执行实体。这是要促使程序员利用程序中的多个线索。因此把比传统操作系统更多的并发生性引入系统。虽然线索需要成本,但其实在太少而且在 Mach 环境中的典型应用程序或服务器可利用这一能力。

但是线索确实有某些与其相关的元素。包括的任务和地址空间以及执行状态已被讨论过。每个线索都有一调度策略。用于确定线索在何时或如何经常被给予到运行其上的处理器。调度服务将在其后予以评述。一个线索的调度策略与后选的处理器组指定密切相关,其可用于具有多处理器的系统中以便更严密地控制分配给处理器的线索以加强程序的功能。如前所述,一个地址空间(任务)可包括多个同时执行的线索或不包括线索。核心不对整个系统中或地址空间中的线索的关系进行推测。反之,它根据与之相关的调度参数以及系统中的可用处理器资源调度和执行线索。尤其是,在地址空间中没有线索配置(即层级)并没有关于它们之间如何

交互的假设。为了控制线索的执行以及其与某些有用终端的协调，Mach 提供了几种同步机制。最简单(最粗糙)的机制是线索级中止和返回操作。每个线索有一中止数，它由这些操作增量或减量。一个具有正值中止数的线索保持中断直至该计数为零。

可通过采用同步目标(信标，监视器和条件)的方式获得更为准确的同步。同步目标使多个不同的同步方式得以使用。线索亦可通过 IPC 进行交互。其后将对每一服务提供详细叙述。所存在的基本操作包括支持生成、终止以及获取和设置线索的属性。还有几个其他有关线索的控制操作它们可以由任何具有把权利送给属意的线索的控制端口的线索所执行。线索可以被明确地终止。它们亦可从多种可能的等待状态中被中断并在指示其处于中断的情况下被恢复执行。线索亦可被“连接”，其意为它们被标志为相对于核心资源的特权，即它们可以在可用存储器不足时用物理存储器。这用于预置页出道路的线索。最后，线索还有几个重要IPC 端口(更准确地说，是送出或从端口接收权利)，它们被用于某些功能。尤其是，每个线索都有一个线索自有端口，可用于在线索上自己执行某些功能。一个线索亦可有一组故障端口，用于当线索在执行期间碰到处理器故障之时。还有区别端口，用于收集线索的执行状态的取样以供其他线索(例如调试程序或程序简介表程序监测。

任务

用于管理 Mach 中的资源的基本组织实体叫做任务。任

务有许多与之相关的目标和属性。一个任务基本包括三件事:一个任务包括多个线索,它们在系统中是可执行实体;任务还包括一个地址空间,它代表线索在其中得以执行的虚拟存储器;任务还有一个端口名空间,用于代表线索在系统中与其他线索通信的有效IPC端口。在以下章节中将详细讨论在一个任务中的每个基本目标。要注意的是一个任务其本身不是可在Mach中执行的实体。但是,任务可包括线索,而其是可执行实体。一个任务除了以上所述的基本线索之外,还包括多个与之关联的其他实体。其中一些实体必须根据预定的决策完成核心需要以便生成任务包括的线索。调度参数,处理器组指定,以及主机信息都对任务的线索的调度有贡献。一个任务还有多个服务于某些预定功能的区别内处理通信端口。内部处理通信的特征及端口将在其后讨论。现在,应充分了解端口资源是在一个任务中按时间积累的。大部分资源是由程序员明确管理的。上面提到的区分端口必须在系统中建立与几个重要功能的连接。Mack提供给每个任务三个特定端口。第一个是任务自有端口,用于要求核心程序在任务上执行某些操作。第二个特定端口是引导程序端口,用于除了通常服务于对其他服务进行定位之外的所有事(它是OS特定环境的)。第三个特定端口为每个任务所有,它是主机名端口,用于使任务获取有关其运行的机器的信息。此外,Mach为每个任务提供几个"注册"的端口,以使任务中包括的线索与系统中的某些高级服务器(即网络名服务器,"服务"服务器,以及环境服务器)进行通信。

每个任务还有两个有用的端口组，以允许故障处理和程序状态取样得以完成。任务的故障端口为要处理的任务中的线索所碰到的处理器故障提供通用位置。其后将对故障处理进行详细叙述。PC 取样端口允许简介工具反复监视任务中的线索的执行状态。任务可能有许多操作。任务可被生成和结束。生成一个新任务涉及把某些现有任务指定为用于新任务地址空间的初始内容的模型机。亦可终止一个任务，它导致所有包括的线索被终止。在一个任务中包括的线索可被计算且有关线索的信息可被抽取。可以通过中止和恢复操作来控制一个任务(更准确讲是任务中的线索)的粗数式执行。每个任务都有一中止计数，它由中止和恢复操作增加和减少。任务中的线索可一直执行，直到所包括的中止计数成为零。当中止计数为正时，任务中的所有线索被中断，直到该任务最后恢复为止。最后，可按需要查询和设置与一任务(即预定优先级)关联的各种参数和属性。

虚拟存储器

Mach 支持虚拟存储器(VM)子系统中的一些特性。外部客户界面和内部实现都提供在许多其他操作系统中没有发现的特性。从广义而言，Mach 虚拟存储系统支持用于在系统中运行的每个任务的大的稀疏虚拟地址空间。为客户提供常规服务以管理地址空间的组成。VM 系统的某些方面实际是由在微核心程序之外的部件实现的，这使得在修改 不同系统环境的某些策略功能时具有很大灵活性。Mach VM 系统的内部结构已分为机器独立和机器相关模块以提供最大便

利。对新处理器/MMU 结构的端口通常在实现操纵基本硬件 MMU 结构的那些功能并非很重要。Mach 已经有端口到多个不同的处理器结构，表明整个核心程序和虚拟存储系统的灵活性。一个 Mach 任务的地址空间包括多个虚拟存储区，它们是以不同方式分配以供任务所用的若干地址空间。它们是存储器可以被合法访问的位置。所有对地址空间限定区域之外的地址引用都将导致存储器引用不当的故障。一个虚拟存储区有几个有用的特性。它包括一个页对齐起始地址和尺寸，它必须是多个系统页尺寸。在区域中的页面都有相同的访问保护，这些访问保护可以是只读的、读一写的或执行的。区域中的页面具有相同的继承特性，用于从当前任务中生成新任务之时。该页面继承特性可设置以指示一个新任务应继承该区域的读写复制，即其应该继承该区域的虚拟复制，或其不继承该区域的复制。在一个新地址空间中的一个区域的读写复制提供了任务之间的区域的完全共用映射，由虚拟复制提供了写复制映射，它基本上给出了每个任务的区域的自我复制但可有效地共享构成区域的页面的写复制。

每个虚拟存储区实际上是一个称为存储器目标的抽象实体的映射。存储器目标只是一个数据集合，它可以字节方式被访问并且核心程序不对其进行假定。最好想想可被直接存在某些地或可按需要以某些方式被生成的一些纯数据。许多事情都可作为存储器目标。其例包括文档，ROM 碰盘分区，或字形。存储器目标没有预定的操作或协议要遵从。包括在一个存储的目标中的数据只是在其通过映射已与 VM

区域相连时才被访问。在存储器目标已映射到一个区域后，可通过正常的存储器读写(装入和存储)操作访问数据。存储器目标通常由称为外部存储器管理程序或页面程序的一种特定的任务管理。页面程序是一个做微核心程序之外执行的任务，就象系统中的其他任务一样。它是一个用户方式实体，其工作是处理对于存储器目标支持的数据的请求。就象给定区域中客户任务引用的线索一样，核心程序逻辑地把来自相关存储器目标的相应章节地址中的数据填充到页面上，为完成这一任务，核心程序实际上将完善定义的(并且复杂的)协议与页面程序相关，这发生于只要当其需要获取页面故障数据或由于页面替代而需要页出数据之时。这一协议称为外部存储器管理界面(EMMI)，也用于管理由用户任务映射时的存储器目标起始顺序以及由用户任务重新分配相关的存储器区时的终止顺序。

根据不同用户任务所用的存储器目标而有各种在系统中运行的页面程序。页面程序通常与一给定时刻安装的各种文档系统关联。例如，页面程序亦可存在以支持某些数据库程序，这些程序可能需要某些超出文档系统支持的范围之外的操作。页面程序也可以用于某些希望以非标准方式(即生成计算数据而非从存储子系统中检索数据)为其客户提供数据的服务器。微核心程序总希望有在系统上运行系统的称为预置页面程序的不同页面程序。该预置程序负责管理与假虚拟存储器(例如堆栈、堆阵等)关联的存储器目标。这一存储器是暂时的且只用于运行客户任务的时候。如上所述，在

Mach VM 系统中的主要实体是区域，存储器目标，以及页面目标。但是，大部分客户将通过在存储器范围内的操作而与虚拟存储器关联。所谓范围可是区域的一部分或其它跨越地址空间中的多个连续区。由 Mach 提供的操作使用户得以在地址空间中分配新的虚拟存储器范围并按需要重新分配范围。另一重要操作使一存储器目标被映射成为上述的虚拟存储器的一个范围。进一步的操作是改变存储器范围上的保护，改变继承特性，以及连接(或锁定)一个范围的页面进入物理存储器。也可能从另一任务中读出存储器范围或将范围写入另一个任务，如果该任务的控制端口可用的话。其他服务还包括使用户指示所期望的存储器范围的引用方式。这可由核心程序所用以便对不同情况制定页面替代规定。另一服务涉及同步(或齐平)存储器范围的内容和支持它的存储器目标。最后的服务是获得有关区域的信息并计算根据其包括的区域所叙述的一个任务的地址空间的内容。

内部处理通信

Mach 有四个有关其内部处理通信的概念：端口，端口组，端口权，以及消息。其中之一的端口权(port rights)亦由 Mach 用作标识系统中的某些通用资源的装置(通用资源包括线索，任务，存储器目标等)。

端口

线索利用端口进行相互间的通信。端口基本上是在核心程序之内的消息队列，在适当许可下，线索可把消息加至其中或从其中去除消息。所谓“许可”即为端口权。其他与端口

相关的属性(除端口权外)包括可在端口上排队的消息数量的限制,可送至端口的消息的最大容量的限制,以及对端口的权利计数。端口只存在于核心程序中而且只可通过端口权操纵。

端口权

若端口有发送权时,线索可把一个消息加至该端口的消息队列。类似地,亦可在有接收权时由线索从该端口队列中去除消息。端口权被认为是任务的资源而非一个独立线索。对一个端口可以有多个发送权(由不同任务所有);但是一个端口只可能有一个接收权。事实上,端口是通过分配接收权生成而且只有在接收权被重新分配(在任务消亡时直接或间接)时才去除端口。此外,端口的属性由接收权操纵。多个线索(在相同或不同任务上)可在同时送至端口且多个线索(在同样任务)可同时从一个端口接收。端口权可视作为从端口接收或向端口发送消息的许可或能力,而且如此它们实现了系统保密的低级形式。端口的"拥有者"是持有接收权的任务。另一任务欲获取端口发送权的唯一方式是是否其直接从拥有者或任何具有该端口有效发送权的任务获得了该权利。这主要是由在一个消息中包括该权利并将该消息送至另一任务而实现的。把发送权赋予给一个任务就是给了它一个许可使其在需要时把许多消息送至端口。另一种端口权称为一次发送权,它只允许持有者给端口发送一个消息,然后其发送权失效并不可以再使用。注意,可以通过把一个消息中的端口的接收权送至另一任务的方法完成传送端口的拥有关

系。

任务通过在消息中生成或接收端口权以获取端口权。接收权只能直接生成(通过对上述的端口分配);而发送权可从现有的发送权或接收权中直接生成,即可从消息传送中间接生成,一次发送权只能从一接收权中直接或间接生成。当发送消息的权利时,发送端能指定其复制、移动,或由发送操作生成的新权利。(当然,接收权只可以被移动)。当移动一个权利时,发送程序就失去其权利而接收程序得到该权利。当复制时,发送程序保留其权利但生成复制的权利并将其赋予给接收程序。当生成时,发送程序提供一个接收权而且一个新的发送或一次发送权被生成并给予接收程序。当一个任务获取一个端口权时,Mach 赋予其一命名。注意端口本身并不命名,但其端口权要命名。(不管这一事实,Mach 的生成程序决定了要由"端口名"这一术语来称呼端口权的命名)。该命名是一个标量值(在 Intel 机器中是 32 位),该值在一个任务是确保是唯一的(其意味着在几个任务中,每个任务都具有同样数值的端口名但其代表了对于完全不同端口的端口权)而且是随机选择的。由一个任务持有的每个不同的权利不必有一个区别端口名。一次发送权对每一权利都有一个单独的名字。但是,对于同一端口的接收和发送权有相同的命名。

端口权有几个与之相关的属性:权利类型(发送,一次发送,接收,端口组,或无效名),以及对于每个上述权利类型的参考计数。当一个任务获取了已有发送或接收权的端口时,便对相关端口名的参考计数增值。当其相关端口被破坏时,

端口名成为无效名。即，表示其接收权被重新分配的端口的发送或一次发送权的所有端口名变成为无效名。任务可在其中一个权利变为无效时请求通知。核心程序保持每个端口发送或一次发送权数量的系统范围计数。任何持有接收权的任务(例如服务器)可在该数字为零时请求发送通知消息，以指示该端口没有更多的发送程序(客户)。这叫做“无发送程序”通知。该请求必须包括通知送经端口的发送权。

端口组

端口组提供从端口集同时接收的能力。即接收权可加到端口组以致当端口组的接收完成时，可从该组中的一个端口接收消息。其端口提供了消息的接收权的命名由接收操作报告。

消息

一条 Mach IPC 消息包括标题和直接数据部分，以及任选地某些间接存储器(out of line memory)区和端口权。若消息即不包括端口权又不包括间接存储器，则其为简单消息；否则即为复杂消息。简单消息包括消息标题以及其后的直接数据部分。消息标题包括一个目的地端口发送权，一个响应可送达的任选发送权(通常是一次发送权)，以及消息数据部分的长度。直接数据的长度可变(最大长度按端口不同而定)而且在复制时无需解释。一个复杂消息包括一个消息标题(其格式与简单消息的标题一样)，其后则为间接存储区和端口的计数，叙述这些区域和端口的核心程序处理的配置阵列，以及包括间接描述符和端口权的阵列。

端口权配置阵列包括对权利的处理，即其是否被复制，生成或移到目标任务。间接存储器配置阵列指示在消息排队时是否每个存储范围应被重新分配，而且是否存储器应被复制到接收任务的地址空间或通过一个虚拟存储器的梳利复制处理而将其映射成接收地址空间。间接描述符指示间接存储器区的容量，地址和对准。当一任务接收一条消息时，标题，直接数据和描述符阵列被复制到在参数中指定要接收调用的地址，若消息包括间接数据，则接收任务的地址空间中的虚拟存储器由核心程序自动分配以保持这些间接数据。接收任务有责任在接收数据时对存储区进行重新分配。

消息传送语义

Mach IPC 基本属于非同步性质。线索把消息送给端口，而且一旦消息在端口上排队时，发送线索继续进行。端口上的接收在没有端口消息排队时会被阻断。为有效起见，有一个组合的发送/接收调用，它可用来发送消息并在指定的响应端口上立即阻断消息等待(提供同步模式)。若在指定时段内消息不能送出(或若无消息可接收时)，则将所有将废止的操作的消息操作设定为超时。发送调用将在其使用与之相应的端口已达到消息的最大数目的发送权的情况下阻断。若发送采用一次发送权，则在端口已满的情况下将保证消息被排队，消息传送是可靠的，而且消息保证是以发送顺序被接收的。在 Mach 中的一个特殊情况是其在非同步模式上优化了同步模式；经由服务器在反复发送/接收循环之后进行接收并由客户在其他点进行相应的发送/接收循环，取得了最快

的 IPC 往返时间。

作为标识符的端口权

由于核心程序保证端口权不能被伪造且消息不能被误导或曲解，端口权提供了一个非常可靠保险的标识符。Mach 通过用端口权代表系统中的几乎一切(包括任务，线索，存储器目标，外存管理程序，进行系统优先操作的许可，处理器分配等等)的方式来利用这一点。此外，由于核心程序本身可发送和接收消息(它将其表示为“特殊”任务)，所以可通过 IPC 消息而非系统调用陷阱访问大部分核心程序服务。这使得服务在适当之处相当容易地迁移到核心程序之外。

同步

当前，Mach 对同步能力提供非直接支持。但是，常规操作系统提供例行同步服务。这类同步服务采用许多已知处理，例如信标和监视程序以及条件，它们叙述如下，信标(Semaphore)是允许对资源进行排比或共享访问的同步处理，它可被获取和释放(在上述任一种访问模式下)，而且它们可任意地指定获取操作上的超时时段。信标在保持有一信标的线索永久结束与该信标相关的计数内被调整以及等待线索被适当解阻断时可任意恢复。

监控程序和条件是实现比简单信标更严格(和安全)的同步方式的一种同步处理。一个监控程序锁定(也称为 mutex)实际上是一个二进制信标，它启动对某些数据的双向排他地访问。条件变量可用于等待和预示监控程序上下之中的某些程序员定义的布尔表达的真值。当一个具有监控程序锁

定的线索需要等待条件时，该监控程序锁定被释放且线索被阻断。稍后，当持有锁定的另一线索通知该条件为真时，等待的线索被释放然后在继续执行之前再次获取锁定。线索亦可在条件上执行广播操作，其释放所有等待该条件的线索。亦可在条件等待操作上设置任意的超时，以限制线索等待条件的时间。

调度

由于 Mach 具有多处理器能力，所以它在多处理器环境中用于线索调度。Mach 定义了处理器组从而将处理器分组，而且它定义了可与之关联的调度规范。Mach 提供了两种调度规范：时间共用和固定优先级。时间共用规范基于 CPU 线索使用的指数平均值。这一规范企图根据线索和处理器数量来优化时间总和。固定优先级规范不修改优先级，但在同一优先级的线索上进行循环调度。线索可利用其处理器组的预置调度规范或直接采用其处理器组能使用的调度规范之一。可为处理器组和线索设置最大优先级，在 Mach 中，优先级越低，其越紧急。

故障

Mach 故障处理服务是要为标准的和用户定义的处理器故障提供灵活的处理。线索，消息和端口的标准核心程序用于提供故障处理机能。（本文在 Mach 文件使用“异常”一词之处采用“故障”一词。这类术语在此被改变以区别 C++ 语言的异常机制和硬件故障）。线索和任务都有故障端口。它们在继承规则中不同并希望以稍微不同的方式被使用。最好以

每个线索为基础进行错误处理并以每个任务为基础处理调试。任务故障端口是由子任务从母任务继承而来，而线索故障端口不继承并且不预置给处理程序。线索故障处理程序优于任务故障处理程序。当一个线索导致一个故障时，核心程序阻断该线索并把一个故障消息通过故障端口送到线索的故障处理程序。故障处理程序是一个从故障端口接收消息的任务。该消息包括有关故障，线索，和导致故障的任务的信息。处理程序根据故障的类型实现其功能。若适当，则故障处理程序可以获得并修改导致该故障的线索的执行状态。可能的动作是要清除故障，终止线索，或将故障传送给任务级处理程序。由类型和数据来标识故障。Mach 定义支持所有 Mach 实现(即误访问，误指令，断点等)的某些机器独立的故障类型。其他故障类型可是实施相关的(即 f—行，协同处理器扰乱等)。

主机和处理器组

Mach 输出主机(host)的概念，它实际是在计算机上执行的一个抽象。根据一个任务对该主机所有的特定端口权而在主机上进行多种操作。可由持有对主机名端口发送权的任何任务获得不敏感的信息。这种信息例子包括为达到对系统时钟值的访问的核心程序或权利类型。几乎所有其他信息都认为是敏感的，而且要求较高级的的特权以获取或操纵该信息。当任务有对主机控制端口(也称为主机特权端口)的发送权时，该加入的特权级被蕴含其中。必须非常小心地并有选择选择将该权给予任务，因为具有此权可使一任务虚拟地为

核心程序做各种可能的事，从而通过 IPC 服务支持的系统的安全方面。这种加入的特权可完成各种操作，包括改变系统时钟设置，获得系统的整体性能和资源使用统计，并重新引导计算机。

Mach 要输出处理器和处理器组的概念。他们允许任务更仔细地指示何时和在什么处理器上执行线索。主机特权端口可计算和控制处理器和处理器组。一个处理器代表在系统中的一个特定处理器，而一个处理器组代表一个处理器集。服务存在以生成新处理器组并将处理器加至所希望的一组中或从该组中去除。还存在分配整个任务或特定线索给一个处理器组的服务。通过这些服务，程序员可控制（以粗粒式）构成应用程序的线索和任务实际要执行的时间。这使得程序员得到指定在一个处理器组中并行执行某些线索的时间。不直接使用这些能力的线索和任务的故障分配是对系统故障处理器组的，它通常包括不在其他组中使用的系统处理器。

保密

Mach 可包括除了上述之外的其他服务范畴。例如，Mach 可包括与保密相关的服务。根据 Mach 保密服务。每个任务携带一个保密标记，它是未由 Mach 翻译的标量值。有一个叫做主机保密端口的端口，它被给予引导任务并传送给可靠的保密服务器。任何具有对主机保密端口发送权的任务均可设置或改变一个任务的保密标记，而无需专门许可以确定任务保密标记（当然除了持有任务的控制端口之外）的值。在接收到一个 Mach IPC 消息之时，消息发送程序的保密标记作

为接收功能的输出参数之一而被返回。持有主机保密端口的任务可发送一消息并把一不同的保密标记分配给该消息，以致就象是来自于另一任务似的。这些服务可由系统的上部层级所用以实施不同的保密级。

环绕类别库

本节将对用于 Mach 做核心程序所提供的服务的面向目标接口进行逐区描述。到 Mach 服务的面向目标接口代表由编码库 110 实现的环绕类别库 402。库 402 包括线索类别 404，任务类别 406，虚拟存储器类别 408，IPC 类别 410，同步类别 412，调度类别 414，故障类别 416，以及机器类别 418。它还可包括其他类别，例如保密类别 420，其取决于操作系统 114 下所提供的服务。每一区由一说明每个类别功能和目的类别图的文本加以描述。选择的方法被表示和定义(适当时，还提供方法的参数表)。如此，这节对环绕类别库 402 提供了完整的操作定义和叙述。库 402 的实现方法在稍后章节中加以讨论。

展示类别图采用了表示类别关系和基本原理的已知 Booch 图符。这些图符示于图 17 中。Grady Bood 已在“面向目标的应用程序设计”一书中已对 Booch 图符进行了讨论。库 402 最好由 C++编程语言实现。但是，亦可用其他语言。最好把类别说明组成 SPI(系统编程界面)，API(应用程序界面)，内部(Internal)和“套索(Noose)”方法—由把有问题的编码归类的#ifndef 语句(或由用于套索方法的命名)指定。SPI 界面针对被使用的特定计算机结构。为说明之目的，参照根据

IBM Micro Kernel (基于 Mach 版本 3.0)或兼容的计算机结构操作来说明和表达环绕类别库 402。一般技术人员可根据本发明对 SPI 进行修改以适应其他计算机结构。

API 界面包括于库 402 之中,它与系统运行的结构无关。内部界面只由低级设备所用。套索方法主要用于使和环境程序 128 一起操作的程序 130 与直接运行在 Mach 114 上的编制的应用程序 134(或服务器)进行通讯。它们以位于环绕程序 128 所建立的面向目标编程模式之外的方式提供对原始 Mach 设备的访问。最不希望的是使用套索方法。SPI 和 API(也许内部)类别和方法已足以实现任何应用程序、部件或子系统。

线索类别

图 5 是线索类别 404 和任务类别 406 的类别图 501。线索类别把面向目标的界面提供给 Mach114 的任务和线索功能性。有些线索类别 404 是处理类别,其意为它们代表对相应核心程序实体的引用。处理类别上的空构成程序生成一个空处理目标。该空处理目标开始并不与核心程序实体对应—它必须由流动,分配,或复制操作初始化。在一个空处理上的调用方法将导致投掷一个异常。可以生成多个处理目标的复制,每一复制都指向同一核心程序实体。该处理目标是内部引用计数的,以致在最后一个代表它的目标被破坏之时可以去除核心程序实体。

TThreadHandle 是一个代表系统中的线索实体的具体类别。它为控制和确定有关线索的信息提供方法。它还为在系

统中生成新线索提供处理机制。控制操作包括取消，中断/恢复，以及进行停滞监视。构成 TThreadHandle 并在 TThreadProgram 目标中传送会导致在当前任务中构成一个新线索。在该新线索中运行的第一编码是 TThreadProgram 目标的 Prepare()和 Run()方法。破坏一个 TTreadHandle 并不破坏其代表的线索。可以在 TThreadHandle 目标上也有删除操作，注意每个 TThreadHandle 目标包括一个对于该线索的控制端口的发送权。该信息通常不是由界面出口，但由于它有一端口权，所以 TThreadProgram 可流入的唯一流入目标即为 TIPCMessageStream。试图流入其他 TStream 目标将导致扔掉一个异常。

TThreadHandle 为调试程序和运行时间环境提供了多种方法，并为支持由环绕程序 128 建立的环境外运行的 Mach 任务进行交互提供了各种方法。这些方法包括获取和设定线索的状态，在另一任务中生成一个“空”线索，获得线索的故障端口，将权利返回到线索的控制端口，并从一个线索控制端口发送权生成一个 TThreadHandle 处理。

如上所述，环绕程序 128 建立应用程序 130 操作的计算环境，为简洁起见，这样由环绕程序 128 建立的计算环境称为 CE。关于环绕程序 128，TThreadHandle 在当前任务中生成一个 CE 运行时间线索。一个线索可由另一任务生成而不只限于由当前任务，其过程是使用 TTaskHandle 类别和其子类别的 CreateTHread 方法。(但是，在另一任务上生成线索不是最好的常规编程模式)。为了在另一 CE 任务上生成 CE 线

索，把 TCETaskHandle：GreateThread 方法送至说明要运行的线索的 TThreadProgram。为生成非 CE 线索(即不在环绕程序 128 建立的计算环境中操作的线索)，CreateThread 方法被用于适当的 TTasKHandle 子类别(即已被生成以便和其他非 CE 计算环境操作的 TTaskHandle 子类别)。例如，为在 OS2 任务上生成一个 IBM OS2 线索，可以使用一个 TOS2TasKHandle：GreateThread 方法。不可能在非 CE 任务上运行 CE 线索，也不可能在 CE 任务上运行非 CE 线索。

TThreadHandle 包括下列方法：

TThreadHandle (const TTHreadProgram $ copy THread-Code)：生成调用任务中的新线索一进行 TThreadProgram 的内部复制，它在线索终止时被删除。

TThead Handle (TThreadProgram * adoptThreadCode)：在调用任务中生成一个新线索一ADOPT 采用随线索端止而被删除的 ThreadCode。由线索拥有的资源要被废弃。不复制 TThreadProgram。

TThreadHandle(EExecution yourself)生成用于调用线索的线索处理。

TSream 在 TThreadHandle 目标中流入到 TIPCMes-sageStream。

CopyThreadSchedule()将指针返回到调度目标的 Schedul-ing 目标(即 TServerSchedule，TUISchedule 等)。为必须由调用程序安排的 TThreadSchedule 目标分配存储器。

SetThreadSchedule(constTThreadSchedule & newSchedule)

把线索中的调度目标设置成 newSchedule 目标。这使得调度线索的方法得以受控。

GetScheduleState(TThreadHandle&theBlockedOnThread)允许在线索被阻塞时对线索(the BlockedOnThread)的当前状态进行查询。

CancelWaitAndPostException()const 致使一个阻塞核心程序调用被释放而且在该线索(*this)扔掉一个 TKernelException。

WaitForDeathOf()const 进行线索的停滞监视—阻塞调用线索直到该线索(*this)终止。CreateDeathInterest()生成有关线索(*this)停滞的通知。当线索终止时,指定的 TInterest 收到一个通知。

TThreadProgram 是一个把所有需要生成新线索的信息封装起来的抽象主类别。这些信息包括执行的代码,调度信息,以及线索的堆栈。为了使用,它必须进行子分类以及 Begin 和 Run 方法置换,然后将目标的示例传送到 TThreadHandle 的构成程序以生成线索。Begin 例行程序用于协助建立同步;Begin 在 TThreadHandle 构成程序完成之前在新线索中执行,而 Run 例行程序则在 TThreadHandle 构成程序之后执行。CopyThreadSchedule 和 GetStackSige 方法返回预置线索调度和堆栈容量。为提供与预置不同的值,这些方法应被置换以返回理想的线索调度和/或堆栈容量。TThreadProgram 包括以下方法:

TThreadProgram(const TText & task Description);TaskDe-

scription 提供可通过 TTaskHandle:GetTasKDescription 方法访问的任务的文本说明。这只有在目标传送给 TTaskHandle 构成程序时才有效。若代之以采用预置构成程序,则界面将合成一个唯一命名以供 TTasKHandle:GetTaskDescription 返回。

GetStackSize()返回将为该线索设置的堆栈的容量。若不想要"预置"的堆栈尺寸,可置换这一方法。

GetStack():用于建立线索的推栈。若希望提供自己的堆栈,则可置换这一方法。

Run()代表在线索上将运行的代码的入口指针。置换该方法以提供被执行代码的线索。

任务类别

参见图 5 关于任务类别 406 的类别图。

TTaskHandle 是一个封装一个基本 Mach 任务的所有操作和属性的主类别。它可用于引用和控制系统上的所有任务。但是,TTaskHandle 不能用于直接生成任务,因为其没有关于运行时间环境的知识。它通过保护的方法为具有被生成的特定运行时间知识的子类别提供足够的协约,这些子类别可生成任务(TCETaskHandle 即为这种类别的例子)。TTaskHandle 目标流入和流出 TIPCMessageStreams 并通过 IPC 送至其他任务,而且它们在与 TCETaskHandle 相关的集中被返回。与 TTaskHandle 关联的任务控制操作包括取消任务,中断和恢复任务,以及进行有关任务的停滞监视。信息方法包括获取其主机,获取和设定其登录端口,计算其端口或虚拟存储器区域获得其故障端口,计算其线索等等。

TTaskHandle 包括下述方法：

TTaskHandle(EEXecutionThread)生成指定线索的任务处理。

Suspend()中止任务(即所有任务包括的线索)Resume()恢复任务(即由该任务包括的所有线索)。

Kill()终止任务—所有由任务包括的线索都被终止。

WaitForDeathOf()进行任务的停滞监视—调用线索被阻断直到任务(*this)终止。CreateDeathInterest()生成用于任务停滞的通知关系(nohfication interest)。在 TInterest 目标中指定的线索当任务(*this)终止时获得一个通知。

AllocateMemory(size—t howManyBytes,TMemory—Surtogate & newRange)在任务地址空间中分配虚拟存储器的范围。字节的理想大小由 howManyBytes 指定。(在页对齐之后)最新指定的存储器的起始地址和实际容量返回到 newRange 之中。

AllocateReservedAddressMemory(const TMemorySurrogate & range, TMemorySurrogate &newRange)在任务的地址空间中以指定的保留地址确定虚拟存储器的范围。范围自变量指定请求的地址和容量。newRange 返回被分配的存储器的页对准地址和容量。

GetRemotePorts(TCollection＜TRemotePortRightHandle＞&thePortSet)获取任务(*this)上的端口表。调用程序负责在返回的 Collection 中重新分配存储器。

虚拟空白的 CreateFaultAssociation Collection(TCollection

＜FaultAssociation＞& where)返回这一任务登录的故障端口。

TCETaskHandle 是 TTaskHandle 的子类别，它代表由 CE 运行时间系统(CE 代表环境程序 128 建立的计算环境)执行的一个 Mach 任务，并体现需要建立 CE 目标环境的所有知识。它可通过把 TThreadProgram 送到其构成程序的方式生成一个新任务。该新任务由一个线索生成，而该线索又由送给 TCETaskHandle 构成程序的 TThreadProgram 目标来说明。也有一个允许从 TTaskHandle 中构成一个 TCETaskHandle 的构成程序。为保证非 CE 运行时间任务不被 TCETaskHandle 所环绕，构成程序查询操作在 CE 环境下的装载程序/库服务器以确保被环绕的任务已由其登录。这一行为自动完成(而无需用户干预)。TCETaskHandle 包括下述方法：

TCETaskHandle(const TThreadProgram& WhatToRun)生成新任务和线索以执行指定的代码。该新线索在'WhatToRun'中执行代码。

TCETaskHandle(EExecutionTask)环绕当前执行线索的任务。

TCETaskHandle(const TThread Program & whattoRun, const TOrderedCollection ＜ TLibrary Searcher ＞ &liberarySearchers)生成新任务和线索以执行具有指定库检索的特定码。LibrarySearchers 指定用于解决命名的库表。

TCETaskHandle(const TTask Handle & a Task)从一通用任务目标中生成一个 CE 任务目标。

AddLibrarySearcher (const TLibrary Search & newlib Searcher)加入一个用于任务的库检索程序—装入程序利用newlibrary Searcher 首先解决库引用,即把 newlibrarySearcher 放在用于解决引用的集的顶部。

GetTaskDescription(TText & description)const 返回任务的字符串说明—从根线索(送至构成程序)的相关 TThreadProgram 中获得字符串。该字符串保证是唯一的,而且若无说明被送到 TThreadProgram 构成程序,则由界面合成一个字符串。

NotifyUponGreation(TInterest * notifyMe)同步地通知调用程序有关系统中每个新任务的生成。没有(* this)任务目标被涉及。生成调用的任务是通知的接收方。

虚拟存储器类型

图 6 是虚拟存储器类别 408 的类别图。TTaskHandle 是代表一个任务上的类别,它已在任务类别 406 中讨论过了。对虚拟存储操作而言,TTasKHandle 型的目标用于指定操作发生的地址空间。大部分可在 Mach 中执行的虚拟存储器操作表示为 TTaskHandle 的方法。在虚拟存储器上操作的这些方法将 TMemory Surrogate 目标作为参数。进一步细节可参见 TTa kHandle 的多种方法说明。若干存储器类别具有复制构成程序和/或分配运算符。存储器类别包括对存储器而非实际存储器本身的引用。因此,当复制和流入存储器类别目标时,只有在其中而非实际存储器的引用被复制。TMemorySurrogate 类别包括复制引用的存储器的直接方法。

TMemorySurrogate 是一个代表虚地址空间中连续存储器范围的类别。它有一起始地址和容量(以字节计)。TMemorySurrogate 可用于指定在其上进行某些操作的存储器的范围。它们通常是作为自变量提供给在与任务相关的地址空间中操纵虚存的 TTaskHandle 的方法。这一类别用于指定/提供具有特定容量的存储区。该类别本身并不分配任何存储器。它只是封装现有存储器。调用程序的任务是提供该类别指定的实际存储器(自变量给构成程序)。该类别不可再分类。

TChunkyMemory 是以指定尺寸的块管理存储器的抽象主分类。存储器被(有指定块尺寸)分成块,便对用户而言存储器仍视为一系列字节。TChunkyMemory 包括下列方法。

locatechunk (size — t lwhere, TMemory Surrogate & the CoutainingRange)在块集中查找并在 theContainingRange 中返回存储器和块尺寸的地址。

CutBackTo(sige—t where)剪切回到包括"where"的块,即在表中将成为最后一块的偏移处的块。

AllocateMemoryChunky (TMemory Surrogate &theAllocateRange)由客户调用以按需要分配存储器的新块。返回分配的范围。

THeapChunkyMemory 是在堆阵上管理块存储器(chunky memory)的具体类别。

TVMChunkymemery 是用虚存管理块存储器的具体类别。

TMemoryRegionInfo 是在任务的地址空间中由虚拟存储器使用的类别。它提供了存储器属性信息(例如继承,保护等)。它还提供对与存储区相关的存储器目标以及对存储区中封装的实际存储范围进行的访问。在 TMemoryRegionInfo 中嵌套定义存储区的所有存储属性的类别 TMemoryAttributeBundle。这在想要获取/设置所有存储器属性时(或在最小改变时重新使用存储器属性时)很有用。TMemory Attribute Bundle 也用于类别 TTaskHandle 中以涉及把存储器目标映射到任务的地址空间。TMemoryRegionInfo 包括下述方法:

EMemoryProtection {kReadOnly, KReadWrite, KExecute} 指示了存储器的保护。

EMemoryInheritance {KDontInherit, kReadWriteInherit, kCopyInherit}指出存储器继属特性。

EMemoryAttribute {kCacheable,kMigrateable}指示如何管理存储器的机器指定特性。

EMemoryAdvice {KWillUse,KWontUse}指出如何使用存储器。

TMemory ObjectHandle 是代表 Mach 存储器类别的概念的主类别。它体现了可映射到虚拟存储器的数据。提供 TMemoryObjectHandles 给客户的系统服务器从 TMemoryObjectHandle 中分类以便定义文档、设备分区等的特定类型的存储器目标。对通用虚拟存储器服务的客户而言,EMemory-Behavior {kReferenceSequential, kReferenceReverseSequential,

kReferenceRandom}指示存储器如何引用。TMemoryObjectHandle 以及各种子类别的主要用处是提供可映射到任务地址空间的数据的通用类型和协约。

TChunkyStream 是体现由存储器块支持的随机存取流的具体类别(衍生于 TRandomAccessStream)。可指定块尺寸或使用的预置。可计算块。该类别在无需产生维护连续存储器的花费的情况下提供 TMemory 类别的通用功能。若需要保留 TMemory 的功能性,则可定义其他类别。

TContiguousMemoryStream 是采用连续存储器(由客户提供)的具体类别。由于其来自 TRndomAccessStream,所以全部随机存取操作(如 Seek)都可用于 TContiguousMemoryStream 目标。

内部处理通信(IPC)类别

IPC 类别 410 代表 Mach IPC 消息抽象。所有消息行为都在消息类别上;端口权类别基本上用于对消息寻址。常用模型最好如下:示例一个 TIPCMessageStream,将目标流入其中,并由代表目的地发送权(作为自变量传送)的目标调用 TIPCMessageStream::Send 方法为接收消息。为接收一个消息,示例 TIPCMessageStream 并调用其 Receive 方法,并在接收权目标中将其作为自变量传送。当 Receive 返回时,目标从 TIPCMessageStream 目标中流出。该 TIPCMessageStream 目标是可再使用的。参照图 7、图 8 和图 9 对 IPC 类别 410 进行更详细的说明,图 7 示出了 IPC 消息类别的类别图 702,图 8 示出了 IPC 间接存储器类别的类别图 802,而图 9 是 IPC 端口

权类别的类别图 902。

消息类别

MIPCMessage 是代表 Mach IPC 消息的抽象主类别。它为设置标题区，安排阵列，以及端口和间接存储器阵列提供所有方法。它还包括消息发送和接收的全部协约。它提供基本协约(作为保护界面出口)到建立直接消息数据的子类别。类别 TIPCMessageStream 和 TIPCPrimitiveMessage 从这一类别中衍生，并提供公共方法把数据加入消息中。MIPCMessage 包括下列方法：

GetReplyPort(TPortSendSideHandle&replyPort)只对接收侧有效。在其与消息一起发送时，返回一个响应端口目标。只有在接收消息后第一次调用时入返回。否则返回伪值。

TSecurityTokenGetSendersSecurityToken()只在接收侧有效。返回发送该消息的任务的保密标记。

SetSenders SecurityToken(const TSecurityToken & impostorSecurityToken, const TPortSendRight $ host SecurityPort)上在发送侧有效。下一次发达消息时，它将载带特定的保密标记而非实际进行发送的任务的标记。其中在下一发送中有效并转换回到实际发送程序的保密标记值。

发送/接收 IPC 消息的方法(注意所有这些方法具有任选的 TTime 超时值。若无需超时，则指定 KPositiveInfinity。所有这些方法取代位于 msg 标题中的现有响应端口值。对允许响应端口说明的方法而言，响应端口权以及端口权本身的安排通过目标 MIPCMessage:TReplyPortDisposition 传送。这是

设定响应端口的唯一方式，因为配置状态只在发送期间有效。其配置被移去的端口权的目标只是在发送发生时才无效）：

Send（const TPortSendSideHandle&destinatiomPort，const TTime&timeout =kPositiveInfinity）是单向非同步发送。

Send（const TPortSendSideHandle&destinationPort，const TReplyPortDisposition&replyPort，const TTime&timeout = kPositiveInfinity）是同步发送，具有指定的（一次）发送响应端口。

接收（const TPortReceiveSideHandle sourcePort，const TTime&timeout=kPositiveInfinity）是“阻断”接收。

SendAndReceive（const TPortSendSideHandle&sendPort，const TPortReceiveSideHandle&receivePort，const TTime&timeout=kPositiveInfinity）送出一条消息，阻塞和接收一个响应（响应编口是从 receivePort 构成的一次发送权）。

SendAndReceive（const TPortSendSideHandle&sendPort，const TPortReceiveSideHandlereceivePort，MIPCMessage&receiveMsg，const TTime&timeout = kPositiveInfinity）发送消息，阻断和接收响应（响应端口是从 receivePort 构成的一次发送权）。消息在一个新消息目标中接收以避免重写。

ReplyAndReceive（const TPortSendSideHandle&replyToPort，const TPortReceiveSideHandle & ReceivePort，const TTime &timeout=kPositiveInfinity）送回一个响应，阻断和接收一个新消息。

ReplayAndReceive(const TPortSendSideHandle & replyToPort, const TPortRcceiveSideHandle & receivePort, MIPCMessage & receciveMsg, const TTime & timeout=kPositiveInfinity)送回一个响应,阻塞和接收一个新消息。

用于在标题中获取/设置端口权区段的子类别的方法(Remote and Local Ports:在SEND侧,REMOTE PORT指定目的端口而LOCAL PORT指定响应端口。在RECEIVE侧,REMOTE PORT指定响应端口(在响应之前)而LOCAL PORT指定接收端口。传输(或要用来传输)端口的方法返回到Disposition。它可有以下值:MACH__MSG__TYPE__(MOVE__RECEIVE__MOVE__SEND, MOVE__SEND__ONCE, COPY__SEND, MAKE__SEND, MAKE__SEND__ONCE.)

GetRemotePort:在远地端口权中传送和指定配置。

PORT RIGHT方法:

MovePortRightDescriptor:发送方把端口权给予目的。在发送权,一次发送权和接收权上工作。

CopyPortSendRightDesCriptor:发送程序在目的地生成发送权的复制。

MakePortSendRightDescriptor:在目的地生成一个新发送权。

MakePortSendOnceRightDescriptor:将在目的地生成一个新的一次发送权。

TIPCMessageStream是提供基于流动的IPC消息抽象。这是用于IPC操作的建议类别。它衍生自MIPCMes-

sageDescriptor 和 TStream。为发送一个消息，TIPCMesageStream 在要发送的数据(包括端口权(TPort Right Handle 衍生结果)，间接存储区(TOutOfLineMemorySurrogate)，端口权阵列(TPortRightHandleArray)，包括任意或所有这些的目标，以及需要的任意其他目标或数据类型中流动。TIPCMessageStream 将自动地为端口权，端口权阵列，以及消息标题中的间接存储器建立适当的数据结构，并把一个位置保持程序放入该流动以便这些元素可在流动的适当位置处流出该消息。一旦数据已流入，就用 Send 方法发送消息，从而提供适当的目的端口权(TPortSenderHandle)和任意的响应端口。为接收消息可调用 Receive 方法，从而为接收端口提供接收权(TPortReceiverHandle)。刚接到的数据则自 TIPCMessageStream 中流出。

TIPCMessage 也提供两种方法进行组合的发送和接收操作，设计用于提供通用信息传送语义(并利用 Mach 微核心程序中的快速通路)。SendAndReceive 进行客户侧同步方式发送然后在接收中阻断以选取响应消息。ReplyAndReceive 进行服务器一侧的响应消息发送并立即阻断接收等待下一请求。两个调用都要求指定目的端口和接收端口。此外，SendAndReceive 方法自动地从提供的接收权中生成适当的一次发送权并将其作为响应端口一起发送。

TIPCPrimitiveMessage 是从 MIPCMessage 中衍生的具体类别并把一个更基本和低级的界面提供给 Mach 消息系统。通过获取和设定调用为消息标题和正文提供数据并从中得

到数据。它没有流动能力。这是一个代表 Mach IPC 消息的具体类别。通过在 TMemory Surrogate 中传送而将数据加到消息上。端口权，阵列，和 OOL 数据用适当方法被直接加入和抽取出来。

TOutOfLineMemorySurrogate 代表一个将包括在 IPC 消息中的间接存储器范围。它在实施过程中采用 TMemory Surrogate 并只把配置信息加到 Start Address 而且长度信息已包含在了 TMemorySurrogate 之中。这一类别与 TMemorySurrogate 一样，不同之处仅在于在发送消息时包括配置信息并可代表与范围相关的存储。该类别包括流动操作，设定/获取范围的方法以及设定/获取配置信息的方法。

端口权

以下类别代表 Mach 端口权的所有有效类型。这些类别都共享下列通用行为：通常，当端口权目标被示例时，它对该权的核心程序的引用计数增值，而当端口权被破坏时，将该引用计数减值。但是，端口权目标处理“真正”的核心程序端口权实体。它们可被复制，从而可以有两个引用同样核心程序端口权实体的目标。这些复制是内部引用计数的，从而当所有引用端口权的目标被删除时，该核心程序的端口权引用计数被减值。当端口权成为无效名(即其所属端口被破坏时)，采用代表目标的方法将扔掉一个异常(排除那些象设定引用计数之类的在无效名上有效的操作)。

TPortRightHandle 是一个代表端口权概念的抽象类别。它包括对每种端口权返回的协约，例如获取端口名，请求无

效名通知，测试检查是否端口权为无效名等。（端口名作为mach __port __name __t type 返回，并作为与 Mach 服务器交互的方式提供，该服务器不是由目标环绕程序编写的）。它还作为通用主类别以使代表所有类型端口的一个类型被多形地传送。TPortSenderHandle 和 TPortReceiverHandle 衍生于这些类别。该类别包括流动支持方法（这一类别以及任何包括它的类别只可在 TIPCMessageStream 类别中流入流出。想流入任何其他 TStream 的企图将在运行时间导致扔弃一个异常），获取程序/设置程序，以及请求通知的方法（必须提供通知要送达的一次发送权。通过传送（由引用）一个接收权而生成一个一次发送权。通过采用一个一次发送权而 MOVE 一个一次发送权）。

TPortSenderHandle 是代表 IPC 消息可送达的任意端口权的抽象类别，即这是 MIPCMessage:Send 作为目的和响应端口的类型。类别 TPortSendRightHandle 以及 TPortSendOnceRightHandle 这一类别而来。该类别包括流动支持的方法，以及获取程序和设置程序。

TPortSendRight Handle 代表端口发送权。它支持所有可在发送权上完成的典型操作。通过把一个有效的 TPortReceiveRightHandle 或 TPortSendRightHandle 传送到构成程序，或将其流出 TIPCMessageStream 而生成该类别。该类别还包括不影响核心程序的引用计数而生成空 TPortSendRightHandle 目标的方法，在当前任务中生成新的发送权的构成程序，流动支持方法，以及获取程序和设置程序。

TPortSendOnceRightHandle 代表端口一次发送权。它支持可在一次发送权上进行的所有典型操作。通过把一有效 TPortReceiveRightHandle 送给构成程序或将其流出 TIPCMessageStream 即可生成该类别。当把消息送到该类别的目标时，使一次发送权无效。所有要对该目标进行的发送都将导致丢弃异常。此外，目标被标记为无效且企图使用该目标时也将导致产生丢弃异常(除了将目标初始化的方法之外)。这一类别包括生成一个 TPortSendOnceRightHandle 目标的构成程序，生成当前任务上的新的一次发送权的构成程序，以及获取程序和设置程序。

TPortReceiveHandle 是代表 IPC 消息可从其接收的任意端口权的抽象类别。即，这是 MIPCMessge::Receive 用作端口进行接收的类型。类别 TPortRightReceiveHanlle 和 TPortSetHandle 自该类别衍生而来。该类别包括流动支持方法，以及获取程序和设置程序。

TPortReceiveRightHandle 代表端口接收权。它支持可在接收权上运行的所有典型操作，例如请求无发送程序通知，设置和获取端口的最大消息尺寸及队列长度，以及设置和获取其生成的发送计数等。若示例一个 TPortReceiveRightHandle(不是由空的或复制构成程序示例)，则其导致生成一个端口和接收权。该复制构成程序生成另一目标来引用同一接收权。这些目标被内部地引用计数，从而当引用一个特定接收权的最后目标被破坏时，也破坏了接收权和其代表的端口，进而导致所有对于该端口的权成为无效名。这

一类别是代表端口接收权的具体类别。由定义，在生成接收权时生成实际的核心程序端口实体，并在接收端口被破坏时破坏。由于该类别是一处理，所以接收权的生成和删除连系在一起。例如，预置构成程序(construtor 其意为 C++语言中，在类别的目标生成时自动调用的成员功能)实际上不生成新权利，而只生成一个空目标。该类别包括生成一个 TPortReceiveRightHandle 目标而不生成端口或影响核心程序引用计数的构成程序，生成新的接收权和端口的构成程序，使未初始化目标有效的方法，在处理中生成接收权(因此端口)，流动支持，接收权/端口操纵方法，获取程序和设置程序，以及请求通知的方法。

TPortSetHandle 代表一个端口集，它有对代表端口集中包括的接收权的 TPortReceiveRightHandle 目标进行加入，去除和计算的方法，获取和设置其进行发送计数的方法等等。若 TPortsetHandle 由预置构成程序示例，则将生成端口。若其由复制的构成程序示例，则将生成端口。若其由复制的构成程序示例，则生成同一端口组的别名。当表示一特定端口集的最后目标被删除时，会破坏该端口集。该类别不能流动。

TPortRightHandleArray 是代表可作为 IPC 消息中的间接描述符被发送的端口权阵列的具体类别。它可包括各种端口权，以及在阵列中的每个端口权的配置进行指定(即怎样将其传给目标任务)。这一类别实现了可在 IPC 消息中作为间接描述符(与端口权和间接存储)发送的端口权陈列。该类别包括流动支持的方法，将元素加入阵列(发送侧)的方法，

以及从阵列(接收侧)去除元素的方法。

TRemotoPortRightHandle 是用于引用另一任务中的端口权的具体类别。它不包括大部分有用的端口权方法,因为它不希望被用于执行那些功能而只想作为远地端口权的命名和处理。构成这一类别不生成端口权—它只代表已存在于另一任务中的端口权。

等待组

MWaitable 以及 TWaitGroup 是提供消息发送和同时等待多种消息源能力的类别。TWaitGroup 是提供能力以建立由 MWaitable 衍生的目标集以致线索可用该等待方法从任一 MWaitable 目标中接收消息。它还提供接收消息的自动发送。一个任务反复调用多次等待操作以接收信息,它们是多线索安全的,以致于可以有多于一个的服务于消息的线索。该类别包括操纵 TWaitGroup 成员的方法。例如,GetListOfWaitables 在该组中返回一个 MWaitables 表。MWaitable 是把端口和内部处理方法(HandleIPC Message)相关联的抽象主类别。它还提供公共主类别以通过 TWaitGroup 类别收集 Receive Right 以及其他基于 Receive Riglits 的类别。

TWaitablePortReceiveRightHandle 是从 TPortReceiveRightHandle 和 MWaitable 中衍生出来的方法类别。它是一个抽象主类别,其子类别可加至 TWaitGroup 以便为 Mach 消息 IPC 的多等待/发送提供其他 MWaitable 子类别。

同步类别

图 10 是同步类别 412 的类别图 1002,它用于调用 Mach

的同步服务。如上述，同步类别412采用信标，监控程序和条件。TSemaphore是一个提供计数信标的通用服务的类别。当获取一个信标(Semaphore，其意是指在多任务中，用于实施互异的锁定，从而在一个时刻只有一个线索可对共同数据进行访问，一个简单的信标无论是锁定的还是释放的，有时被称为mutex)时，若一些其他任务已获取了该信标，则阻断调用线索(无异常丢弃)。但若信标由于某种原因无效，则将丢弃一个异常。该类别包括下列方法：

Acquire：以排它模式获取信标。

Acquire(const TTime & maximumWait)：以互异模式获取有超时的信标。

AcquireShared()：以共享模式获取信标。

AcquireShared(const TTime& maximumWait)：以共享模式获取有超时的信标。

Release ()：释放以前获取的信标。

AnyThreadsWaiting()：若信标现有等待获取它的线索，则返回真值。

TLocalSemaphore是代表可以互异或共享模式获取的一个计数信标的类别。其主要操作是获取和释放。一个任选的超时值可在获取操作上指定以限制等待时间。该类别实现“本地”信标，其只可一个在任务中使用并且无恢复语义。

TRecoverable Semaphore Handle是代表其行为与TLocalSemaphore类似但其可以恢复的信标的类别。可恢复性是指当持有该信标的线索异常终止时，调整计数并适当地释放

任何等待的线索。在每个这种线索中提出了一个例外，它指示信标被恢复且有关的用户数据的整体性可能被怀疑。对于以共享方式获取了信标的线索的异常终止而言，无需在其它线索中产生例外，因为相关的数据只是以只读方式访问而且处于一致状态。这一类别包括下述方法：

GetCurrrentHolders：返回持有信标的当前线索集。

SetRecovered：设置信标状态为“恢复”，去除以前“损坏”的状态。

Destroy：从系统中去除可恢复的信标。

TMonitorEntry 是一个代表与一个监控程序(monitor，其意是指在多任务中如一个概念实体，它包括数据，操纵数据的功能，以及对数据访问进行控制的锁。监控程序亦可包括对数据访问提供更准确控制的条件)相关的锁(有时亦称为mutex)的类别。该类别的构成程序导致获取监控程序锁，且现存本地范围(它导致调用破坏程序)的动作导致放弃监控序锁。若在该监控程序中已有另一任务，则企图进入该监控程序的线索将在 TMonitorEntry 构成程序中被阻断，直到先前的线索离开该监控程序。这一类别包括运算符新和删除，它们是隐私的以致 TMonitorEntry 能在堆栈中确定，从而提供有范围入出口的自动进入的出口(而且相关的监控程序锁获取和释放)。

TMonitorCondition 是一个代表与某些监控程序相关的条件变量的类别。其主要操作是等待，通知和播放，等待操作导致当前的线索等待将被通知的条件，并在线索被阻断时放

弃监控程序锁定。由在监控程序中执行的线索调用通知和播放以指示在条件上等待的一个或全部线索在通知(或广播)线索退出监控程序时被释放。当释放一等待线索时,试图再次获取监控程序锁定(在广播时的一个线索),在该点恢复监控程序执行,一个任选超时值可被指定以限制等待条件所需的时间。除构成和破坏之外,所有 TMonitorCondition 方法必须只从监控程序中调用。

TMonitorLock 是表示在监控程序上的锁定的类别。它被传给 TMonitorEntry 和 TMonitorCondition 的构成程序以指定要获取的监控程序或与之相关的监控程序。

调度类别

图 11 是调度类别 414 的类别图,用于调用 Mach 的调度服务。

TThreadSchedule 是体现线索的调度行为的具体类别。它定义了线索的实际,预置和最优先权。优先权值越低,则其越紧急。每一处理器组都有一可被启动的 TThreadSchedule 集和一预置集。可将一个线索分配给任一 TThreadschedule,该 TThreadschedule 可由线索在其上运行的处理器组启动。优先级可设置成 TThreadSchedule 限定的最大值,但尽量不要采用这一特性。特定的调度类别(TIdleSchedule,TServerSchedule 等)可为使用。但是(由于在这一类别中没有纯虚拟功能),衍生的类别可在需要时自由生成该类别的目标(但其不可以被要求这样做)。TThreadSchedule 目标(采用多形性)用于指示线索的调度规范,以下展示的子类别应用于确定适当的优先

权和范围。

TIdleThreadSchedule 是在系统空闲时要运行的那些线索的 TThreadSchedule 的子类别。它们只在系统空闲时才运行。通常，这一范畴用于空闲时间，维护，或诊断线索。

TServerSchedule 是服务器线索的 TThreadSchedule 的具体子类别。服务器线索必须很负责。它们执行短时间后便阻断。对占用相当时间的服务而言，有不同 TThreadSchedule (TSupportSchedule)的应被使用。

TUserInterfaceSchedule 是用于响应和处理应用程序人类界面的那些应用程序任务的 TThreadschedule 的具体子类别。它们通常运行短时间并阻断直到下一交互。

TApplicationSchedule 是支持应用程序的较长运行部分的线索的使用的一个类别。这类线索运行相当长时间。当激励一个视窗或应用程序时，在相关任务中的线索变为更紧迫以致线索更具响应性。

TPseudoRealTimeThreadSchedule 是一个类别，它允许任务通过其范围内的级别设定指示它们在确定优先级中的相关紧急度。该任务调度出口可允许的级别数量以及预置的主级别。若要求一个级别后其值处于类别范围之外，则将丢弃一个异常。该类别包括下述方法：

SetLevel (PriorityLevelsthelevel)：设定任务级，数字越低越紧急。

ReturnNumberofLevels()：返回这一调度目标的紧急级的数量。

ReturnDefaultLevel()：返回调度目标的预置紧急级，它与调度类别的最高优先级相关。

故障类别

图12,13,14和15表示了故障类别416的类别图1202，1220,1302,1402,和1502,它们用于调用Mach的故障服务。对于代表故障消息的类别(如TIPCIdentityFaultMessage,TIP-CIdentityFaultMessage等)而言，必须决定每个消息类型的一个端口。即用户应确保只将在用于故障处理的任意给定端口接收一类消息。最好是故障类别416包括操作系统114运行于其上的每个处理器106的类别的特定处理器组。替换之，故障类别414通常可包括用于多处理器的类属类别。Motorola—68000类别展示于此以供说明而非限制。本领域技术人员可以发现对于基于本文的原理产生其它处理器类别是显而易见的。

TFault Type是代表一个故障的抽象主类别。它被再分类以提供处理器独有的故障值。它通过处理器和故障标识来辨识故障。以下是TFaultType的三个子类别：

TMC680X0FaultType代表在Motorola 68K处理器上的故障类型。它标识可能的68K类型值和CPU描述符。

TMC680X0BadAccessFautType代表在Motorola 68K处理器上的错误访问。

TMC680X0AddressFaultType代表在Motorola 68K处理器上错误地址类型。

TFaultDesignation是封装消息应被送往一个任务或线索

的目的，故障消息格式，以及故障类型的类别。该类别允许以任务或线索为基础指示用于特定故障类型的需要类型的故障消息应被送往发送权指定的端口。

TFaultTypeSet 封装一组故障类型。

TFaultDate 一类别，用于封装除处理器状态之外的核心程序提供的故障数据。并非所有故障都有故障数据。故障数据提供于故障消息中并来自线索状态。

TIPCFaultMessage 是一类别，用于封装核心程序发送的故障消息，它代表获得故障的线索。它用于接收和回答故障。对可以由故障消息发出三种可能的数据提供了三种子类别。该消息可包括标识故障任务和线索，或故障线索的状态，或两种信息。TIPCIdentityFaultMessage 封装包括获取故障的线索的标识的故障信息，它用来接收和应答该故障。TIPCStateFaultMessage 则封装包括获取故障的线索的线索状态的故障消息，它用于接收和应答该故障。TIPCStateAndIdentityMessage 对包括获得了故障的线索的线索状态和标识的故障消息进行封装，它用于接收和应答该故障。

TThreadState 是表示线索的 CPU 状态的抽象类别。子类别实际定义了处理器特定格式。在类别中没有信息。所有工作完成于衍生类别中。所有对 CPU 状态的询问返回一个 TMC680X0State 指针，该指针必须在运行时间形成正确的衍生类别目标。衍生子类别针对特定处理器，例如示于图 12、13、14 和 15 中的许多子类别，它们与 Motorola68xxx 处理器相关。这类子类别包括 TMC680X0CPUState，它是代表线索

的 680X0CPU 状态的具体类别。另一个例子包括 TMC680X0CPUState，它对可用于所有 68K 状态加以封装，以及 TMC680X0CPUFaultState，它封装用于所有 68K 状态的 68K 故障状态。

主机和处理器组类别

图 16 是用于机器类别 418 的类别图 1602。机器类别也称为主机和处理器组类别，它们用于调用与 Mach 机有关的服务以及多处理器支持。

TPrivilegedHostHandle 是体现对核心程序的主机目标的特权端口的具体类别。特权主机端口是 Mach 处理器管理的根。特权主机端口的持有者可以访问系统的任何端口。由心程序提供的基本特权处理是对任务持有所控制端口的特权操作的限制。因此，系统的整体性取决于这一特权主机端口的封闭保持。该类别的目标可以获取信息和主机统计，重新引导该系统，计算特权的处理器组，与非 CE 实体通讯，以及计算处理器。

THostHandle 是体现对于核心程序的主机目标的命名端口的非特权具体类别。该类别的目标可返回一些主机信息并返回预置的处理器组。这一类别的目标在从主机中获取信息(例如核心程序方案，最大 CPU 数量，存储器容量，CPU 类型等)时限有用，但不能导致对主机的任何损坏。用户应该被供给对该类别目标而非高等级特权的 TPrivilegedHostHandle 目标的访问。

TProcessorHandle 是代表一个处理器的具体类别。可将

一个处理器启动，激励，加到 TPrivilegedProcessorSetHandle 中，处理返回信息，并被送给与实施相关的控制。

TPrivilegedProcessorSetHandle 是为处理器端口提供协议的具体类别。这一类别的目标可以启动和禁止调度规范，设置处理器组的最高优先级，返回统计和信息，计算任务和线索，并将线索和任务分配给处理器组。客户对这一类别的目标的访问应被严格限制以保护单独的处理器和处理器组。

TProcessorSetHandle 是为处理器组命名端口提供协议的具体类别。这一类别的目标可返回有关信息组的基本信息（在处理器组中的处理器数量等），但它们不能导致对处理器组的任何损坏。

环绕程序方法的实现

如上所述，Mach 和 Mach 过程界面是众所周知的。环绕类别库 402 以及其操作方法已在上面定义和叙述过了。由库 402 定义的方法的实现通过考虑从库 402 中选择的方法而叙述如下。在相关领域的一般技术人员可根据 Mach 的已知说明，对库 402 的上述讨论，以及其后将叙述的环绕方法实现而清楚地理解实施库 402 的方法。下面的 Code Example2 示出了从线索类别 404 的 TThread Handle 类别实现的 Kill() 方法。Code Example 示出了一个称为“example1”的例行程序，它包括导致 Kill()方法被执行的分解语句。

```
© Copyright, Taligent Inc., 1993
void example1(TThreadHandle& aThread)
{
    TRY
    {
        aThread.Kill();        // terminates aThread immediatly
    }
    CATCH(TKernelException)
    {
        printf("Couldn't kill thread\n");  // error occured trying to kill
    }
    ENDTRY;
    //...
}
```

CODE EXAMPLE 1

```
void TThreadHandle::Kill()
{
    kern_return_t error;
    if((error = thread_terminate(fThreadControlPort)) != KERN_SUCCESS)
        THROW(TKernelException());     // Error indicator
}
```

CODE EXAMPLE 2

这里：fThreadControlPort 是包括类别代表的线索的 Mach 线索控制端口的 TThreadHandle 类别的示例变量(instance variable，是指在 C++中包括在目标中用于定义目标的属性的变量)。

TKernelException 是在核心程序获得错误时丢弃的 C++异常类别。

THROW，TRY，CATCH，以及 ENDTRY 是允许丢弃和捕获 C++异常的 C++语言的部分。下面的 Code Example 4 示出了从任务类别 406 的 TTaskHandle 类别中实现 suspend()方法。在 Code Example 3 中示出了一个将为“example2”的例行程序，该程序包括一条导致 suspend()方法被执行的分解语句。

```
void example2(TTaskHandle& aTask)
{
    TRY
    {
        aTask.Suspend();    // suspend all threads on task aTask
    }
    CATCH(TKernelException)
    (
        printf("Couldn't suspend threads\n");   // error occured
    }
    ENDTRY;
    //...
}
```

CODE EXAMPLE 3

```
void TTaskHandle::Suspend()
{
    kern_return_t error;
    if((error = task_suspend(fTaskControlPort)) != KERN_SUCCESS)
        THROW(TKernelException());    // Error indicator
}
```

CODE EXAMPLE 4

这里：

fTaskControlPort 是包括类别代表的任务的 Mach 线索控制端口的 TTaskHandle 类别的示例变量。

TKernelException 是在核心程序获取一个错误时丢弃的 C++异常类别。

THROW，TRY，CATCH 以及 ENDTRY 是丢弃和捕获 C++异常的 C++语言部分。以下的 CodeEXample 6 示出了从调度类别 414 的 TPseudoRealTime ThreadSchedule 类别中实现 GetLevel()方法。CodeExample 5 则示出了称为“example 3”的例行程序，该程序包括导致执行 GetLevel()方法的分解语句。

```
void example3(TPseudoRealTimeThreadSchedule& aSchedule)
{
    PriorityLevels curPriority;
    curPriority = aSchedule.GetLevel ();      // Get thread's current priority
    //...
}
```

CODE EXAMPLE 5

```
PriorityLevels TPseudoRealTimeThreadSchedule::GetLevel()
{
    struct task_thread_sched_info schedInfo;
    thread_sched_info schedInfoPtr = schedInfo;
    mach_msg_type_number_t returnedSize;
    returnedSize = sizeof (schedInfo);
    void thread_info (fThreadControlPort, THREAD_SCHED_INFO, schedInfoPtr,
                                    &returnedSize);
    return (schedInfo.cur_priority);
}
```

CODE EXAMPLE 6

这里：

fThreadControlPort 是 TPseudoRealTimeThread- Schedule 类别的示例变量。它包括类别是调度的线索的 Mach 线索控制端口。

CodeExample8 示出了从机器类别 418 的 THostHandle 类别中实现 GetKernelVersion()方法。在 Code Example7 中示出了叫做“example 4”的例行程序，它包括导致执行 GetKernelVersion()方法的分解语句。

```
void example4(THostHandle& aHost)
{
    kernel_version_t version;
    aHost.GetKernelVersion (&version);      // get version of kernel currently
running
    //...
}
```

CODE EXAMPLE 7

```
void THostHandle::GetKernelVersion (kernel_version_t& theVersion)
{
    void host_kernel_version(fHostPort, theVersion);
}
```

CODE EXAMPLE 8

这里：

fHostPort 是包括类别表示的主机的 Mach 主机控制端口的 THostHandle 类别的示例变量。

Code Example10 示出了从 IPC 类别 410 的 TPortReceiveRightHandle 类别实现 GetMakeSendCount()方法。Code Example 9 示出了称为“example 5”的例行程序，该程序包括导致执行 CetMakeSendCount()方法的分解语句。正如其名，GetMakeSendCount()方法访问 Mach 以检索与端口关联的发送计数。GetMakeSendCount()方法包括调用 mach__Port__get__attributes 的语句，它是 Mach 面向过程的系统调用，它返回有关端口的状态信息。在 GetMakeSendCount()中，fTheTask 是包括相关任务的任务控制端口的 TPortRightHandle 目标的示例变量，而 fThePortName 是包括由 TPortReceiveRightHandle 目标代表的端口的端口权名的 TPort ReceiveRightHandle 的示例变量。

```
void example5(TPortReceiveRightHandle& aReceiveRight)
{
    unsigned long count;
    count = aReceiveRight.GetMakeSendCount();
    //...
}
CODE EXAMPLE 9
unsigned long TPortReceiveRightHandle::GetMakeSendCount()
{
    mach_port_status_t theInfo;          // port status info returned by Mach
    mach_msg_type_number_t theSize;      // size of info returned by
    void mach_port_get_attributes(fTheTask, fThePortName,
                                  MACH_PORT_RECEIVE_STATUS,
&theInfo, &theSize);
    return(theInfo.mps_mscount);
}CODE EXAMPLE 10
```

本发明的改型与变化对本发明技术领域的一般技术人员而言是显示易见的。例如，本发明的范围包括使过程应用程序以过程方式在由计算机执行的程序的运行时间内访问有本机面向目标界面的面向目标的操作系统。本发明最好由定位应用程序中的过程语句以访问操作系统提供的服务，将过程语句译为与操作系统的本机面向目标界面兼容并对应于该过程语句的面向目标的功能调用。该功能调用在计算机中执行以使操作系统为应用程序提供服务。虽然这在本文中对本发明的实施例进行了阐述，但其只是为了说明本发明的原理而非限制。本发明的范围只由权利要求所限定。

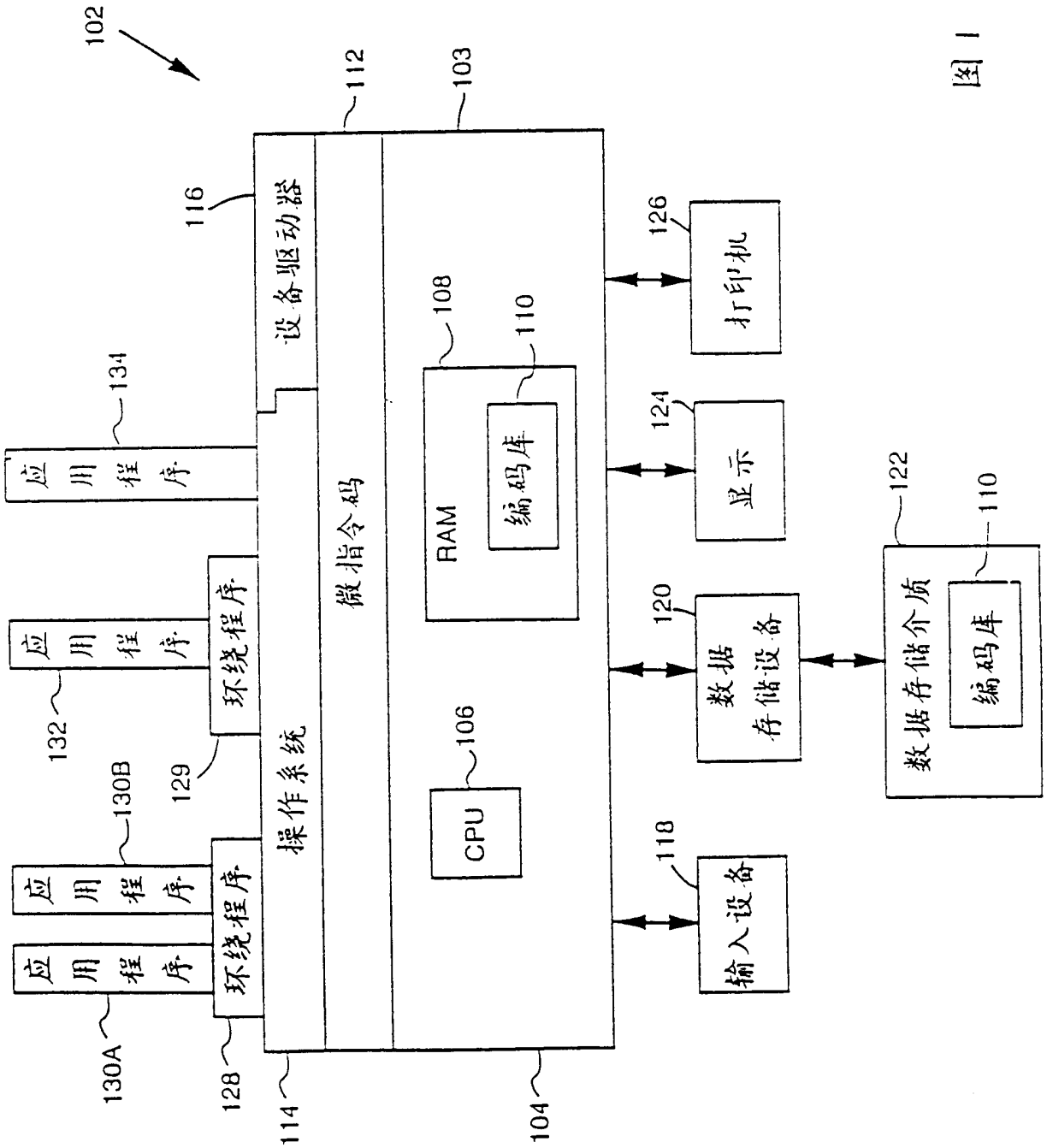

102
112
103
116
设备驱动器
微指令码
操作系统
RAM
编码库
CPU
108
110
106
134
应用程序
132
应用程序
129
环绕程序
130B
应用程序
130A
应用程序
128
环绕程序
114
104
126
打印机
124
显示
120
数据存储设备
122
数据存储介质
110
编码库
118
输入设备

图 1

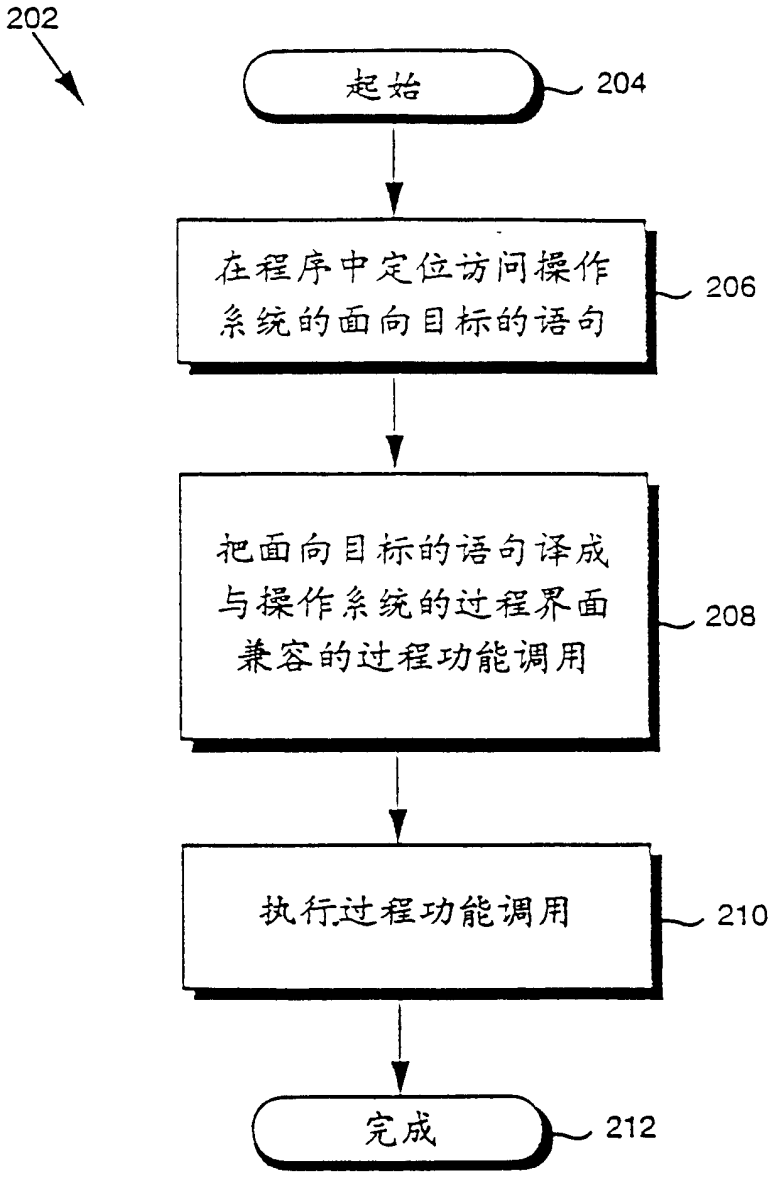

202
起始
204
在程序中定位访问操作系统的面向目标的语句
206
把面向目标的语句译成与操作系统的过程界面兼容的过程功能调用
208
执行过程功能调用
210
完成
212

图 2

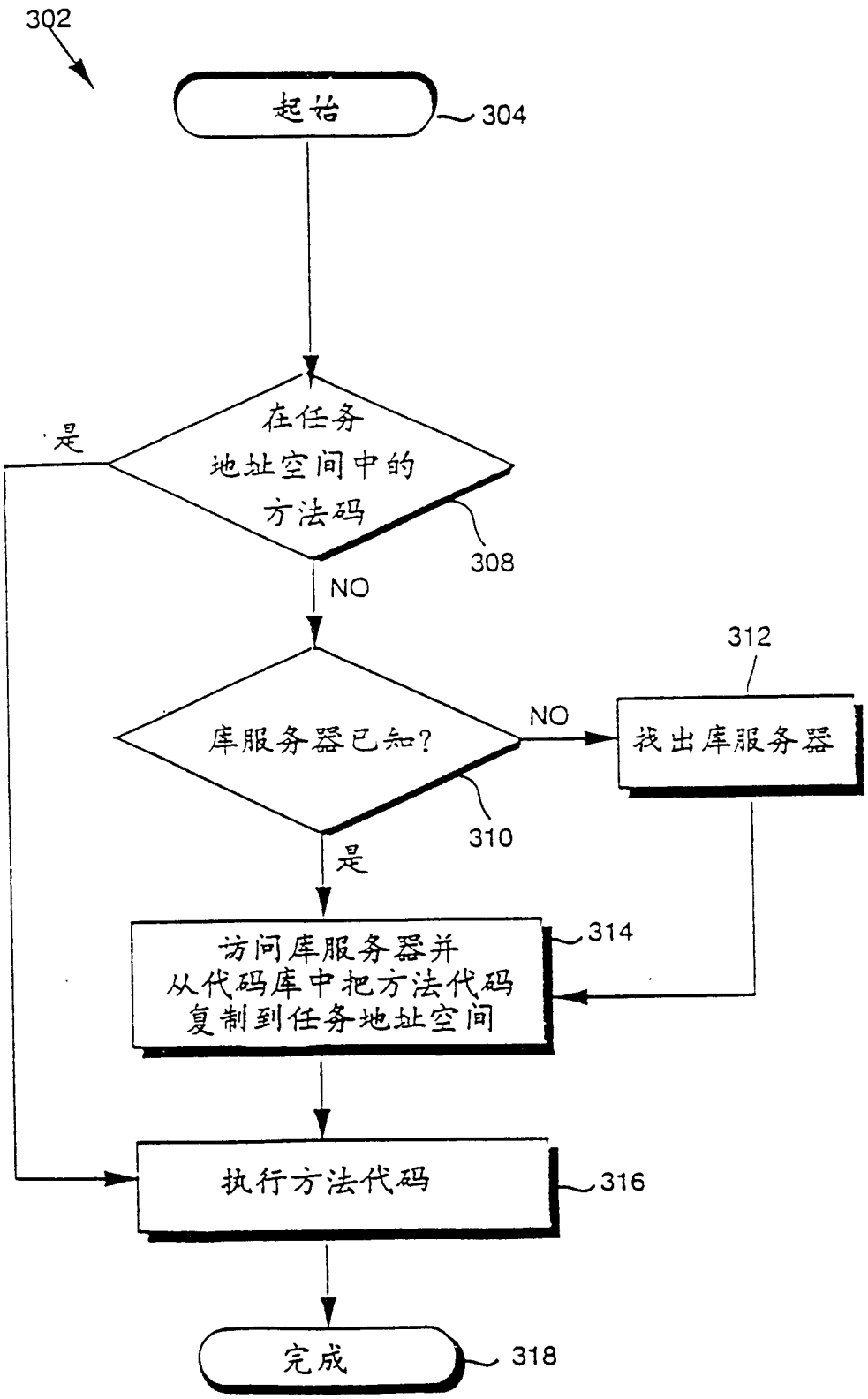

302
起始
304
在任务
地址空间中的
方法码
308
是
NO
库服务器已知?
310
NO
312
找出库服务器
是
访问库服务器并
从代码库中把方法代码
复制到任务地址空间
314
执行方法代码
316
完成
318

图 3

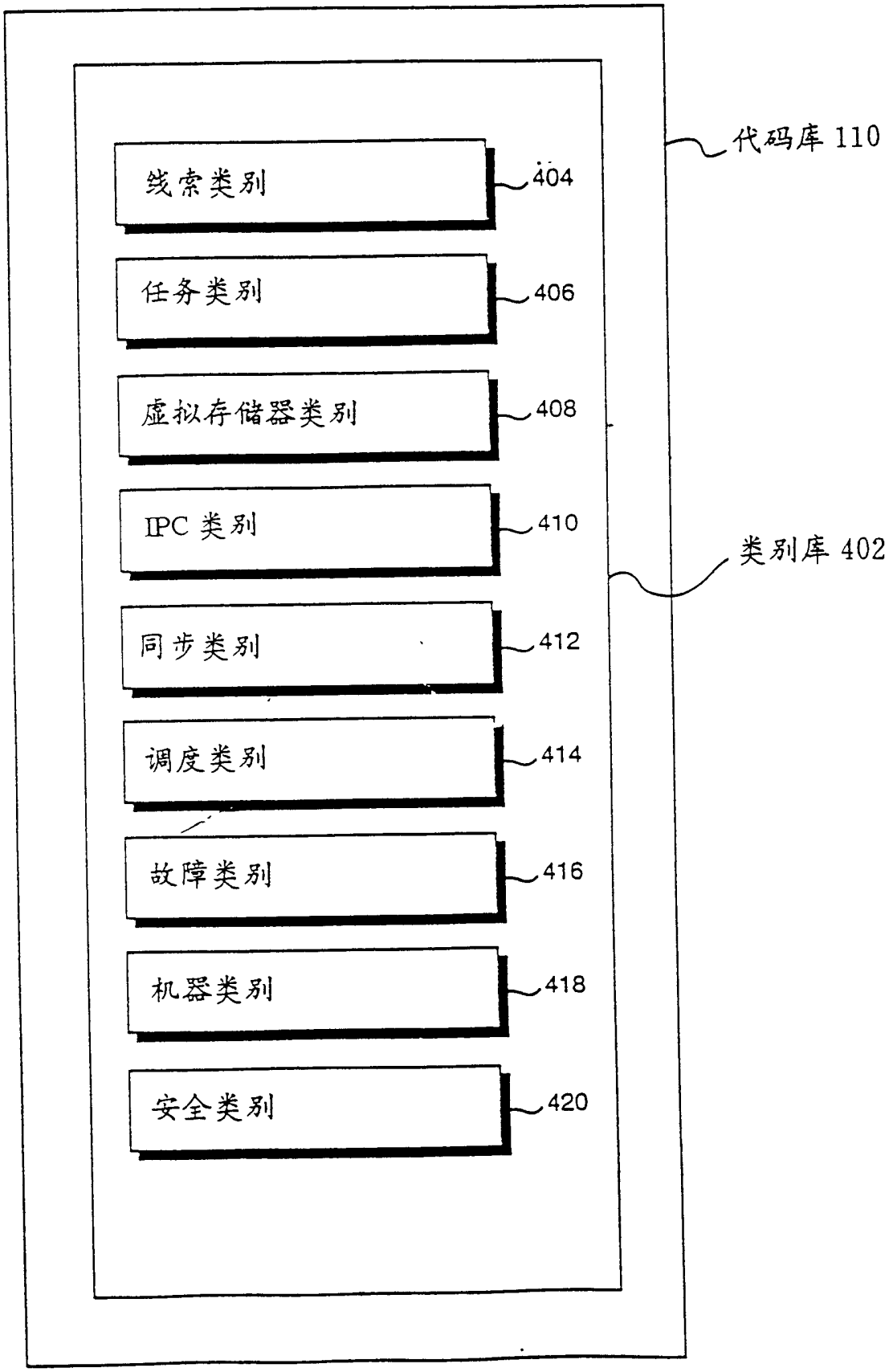

代码库 110
线索类别
404
任务类别
406
虚拟存储器类别
408
IPC 类别
410
类别库 402
同步类别
412
调度类别
414
故障类别
416
机器类别
418
安全类别
420

图 4

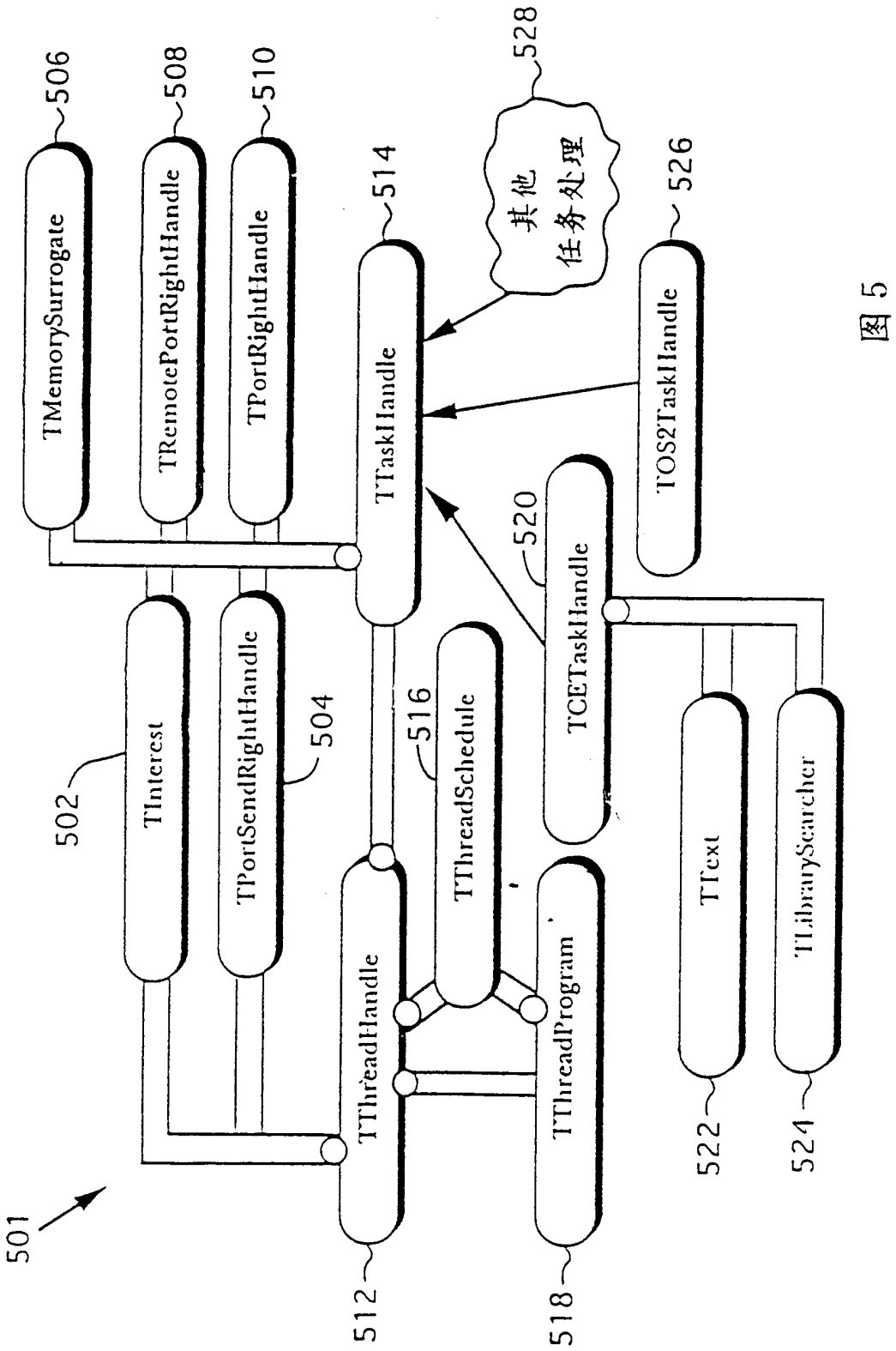

501
502
TInterest
504
TPortSendRightHandle
506
TMemorySurrogate
508
TRemotePortRightHandle
510
TPortRightHandle
512
TThreadHandle
514
TTaskHandle
516
TThreadSchedule
518
TThreadProgram
520
TCETaskHandle
522
TText
524
TLibrarySearcher
526
TOS2TaskHandle
528
其他
任务处理

图 5

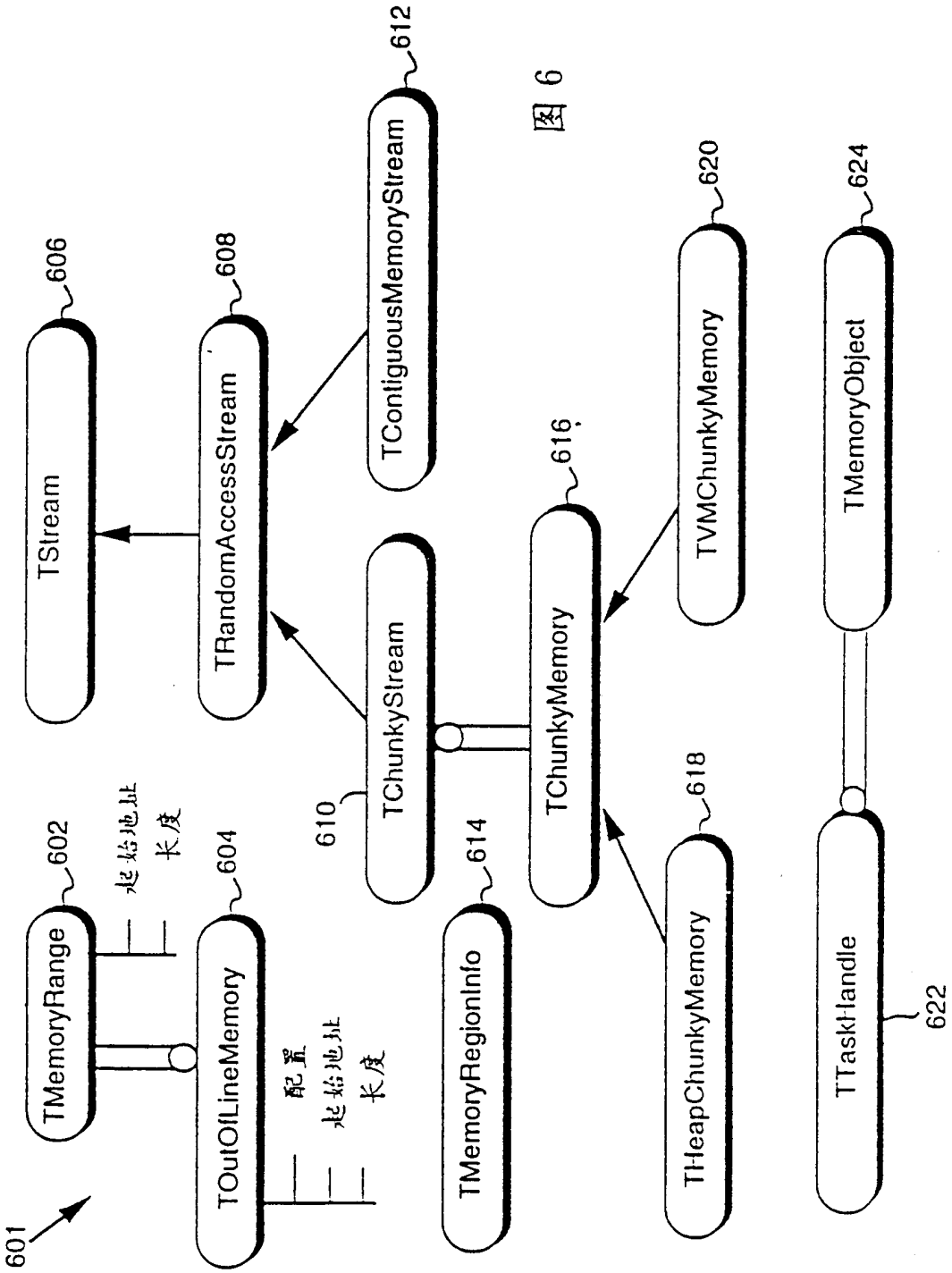

601
TMemoryRange
602
起始地址
长度
TOutOfLineMemory
604
配置
起始地址
长度
TStream
606
TRandomAccessStream
608
TContiguousMemoryStream
612
610
TChunkyStream
TMemoryRegionInfo
614
TChunkyMemory
616
THeapChunkyMemory
618
TVMChunkyMemory
620
TMemoryObject
624
TTaskHandle
622

图 6

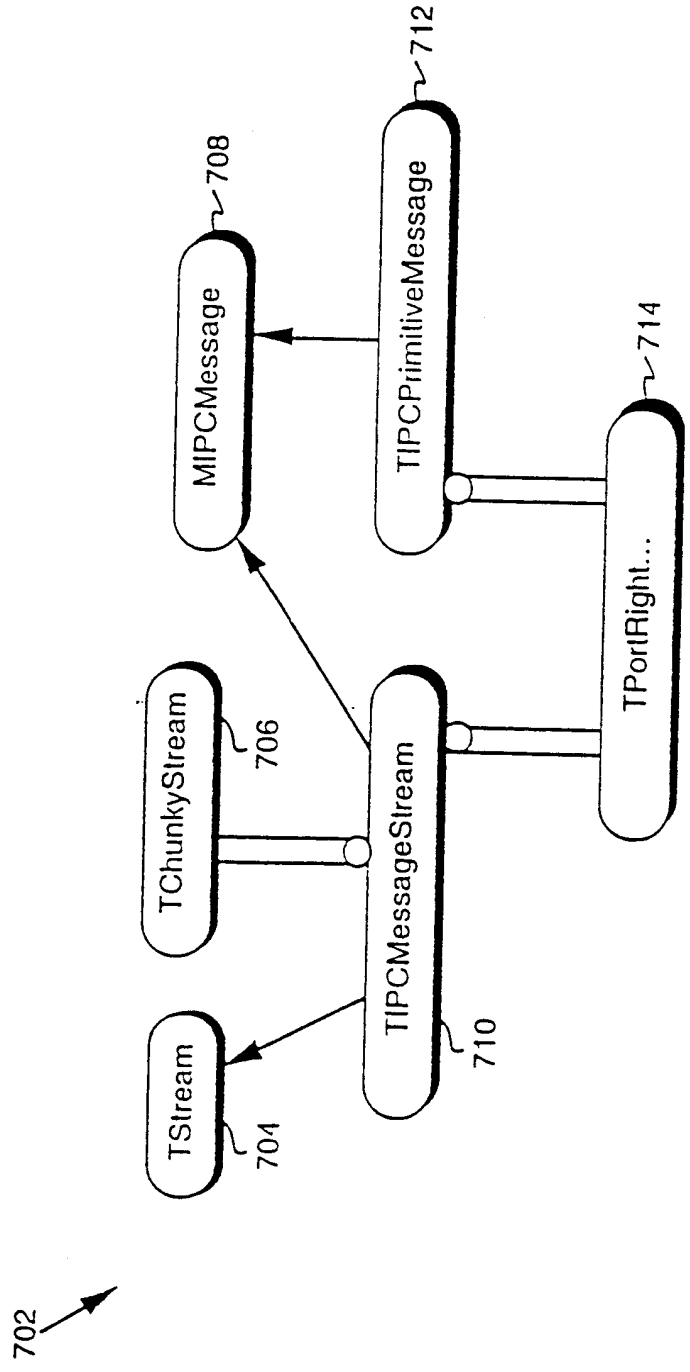

702
TStream
704
TChunkyStream
706
MIPCMessage
708
TIPCMessageStream
710
TIPCPrimitiveMessage
712
TPortRight...
714

图 7

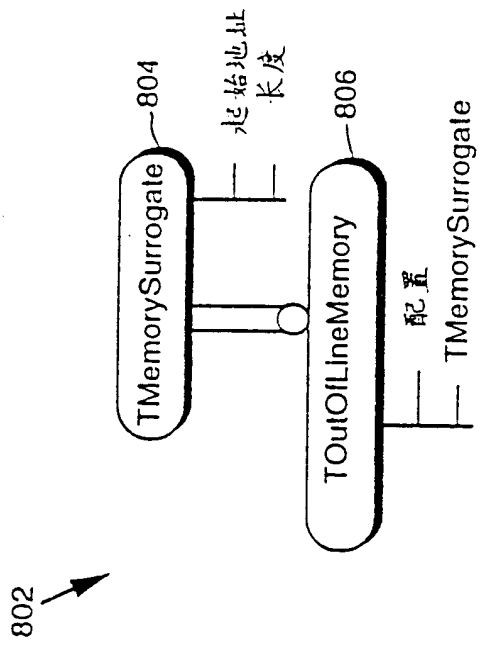

802
TMemorySurrogate
804
起始地址
长度
TOutOfLineMemory
806
配置
TMemorySurrogate

图 8

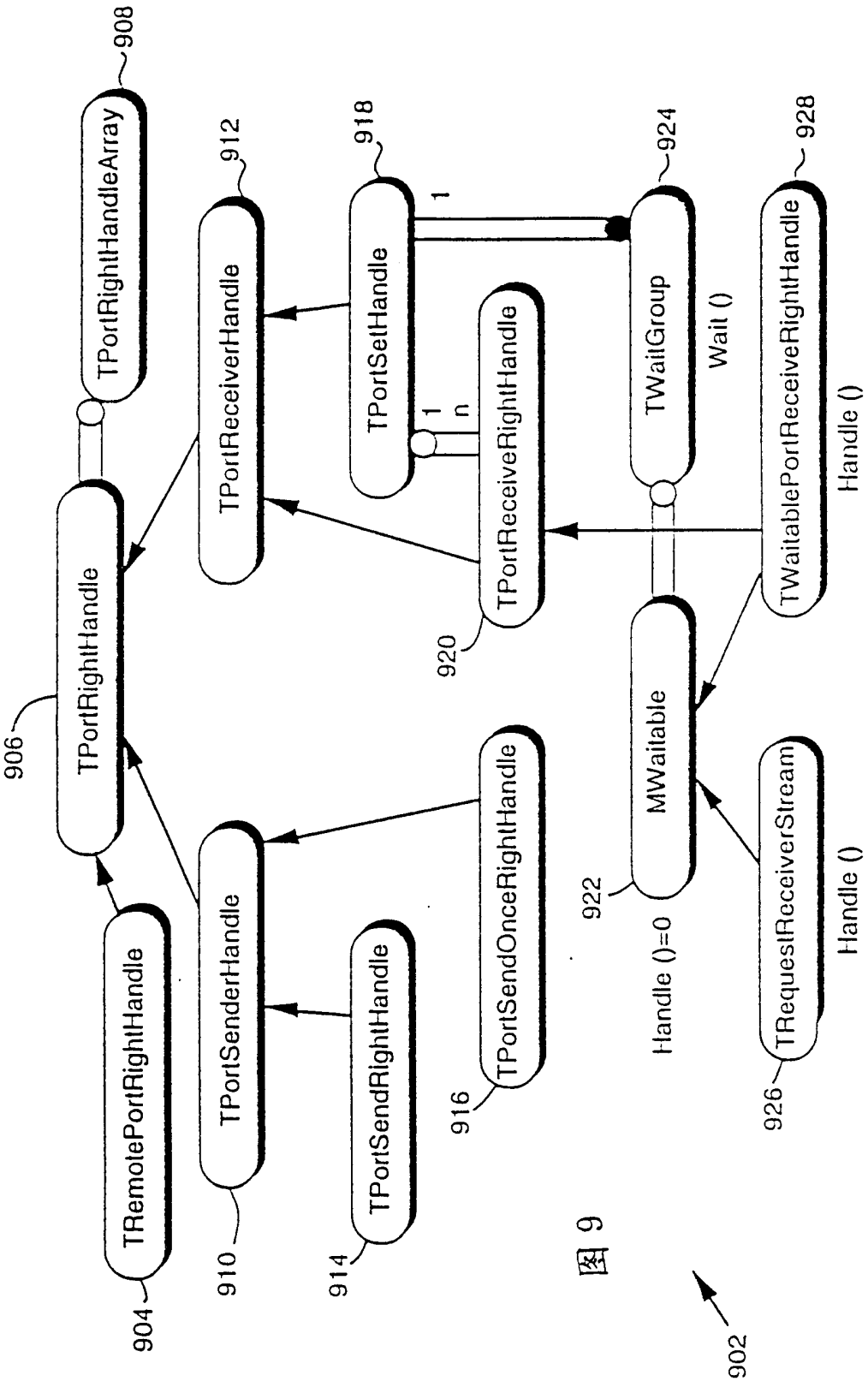

TPortRightHandleArray
908
TPortRightHandle
906
TRemotePortRightHandle
904
TPortReceiverHandle
912
TPortSenderHandle
910
TPortSendRightHandle
914
TPortSendOnceRightHandle
916
TPortSetHandle
918
1
1
n
TPortReceiveRightHandle
920
TWailGroup
924
Wail ()
MWailable
922
Handle ()=0
TWaitablePortReceiveRightHandle
928
Handle ()
TRequestReceiverStream
926
Handle ()
902

图 9

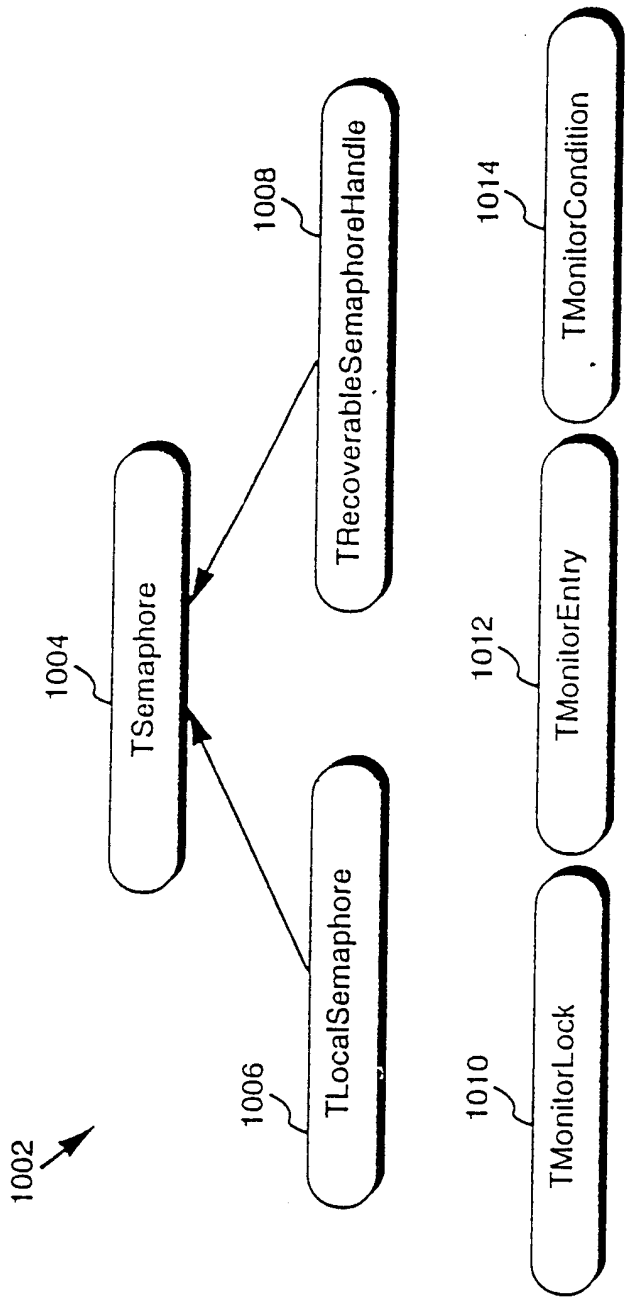

1002
1004
TSemaphore
1006
TLocalSemaphore
1008
TRecoverableSemaphoreHandle
1010
TMonitorLock
1012
TMonitorEntry
1014
TMonitorCondition

图 10

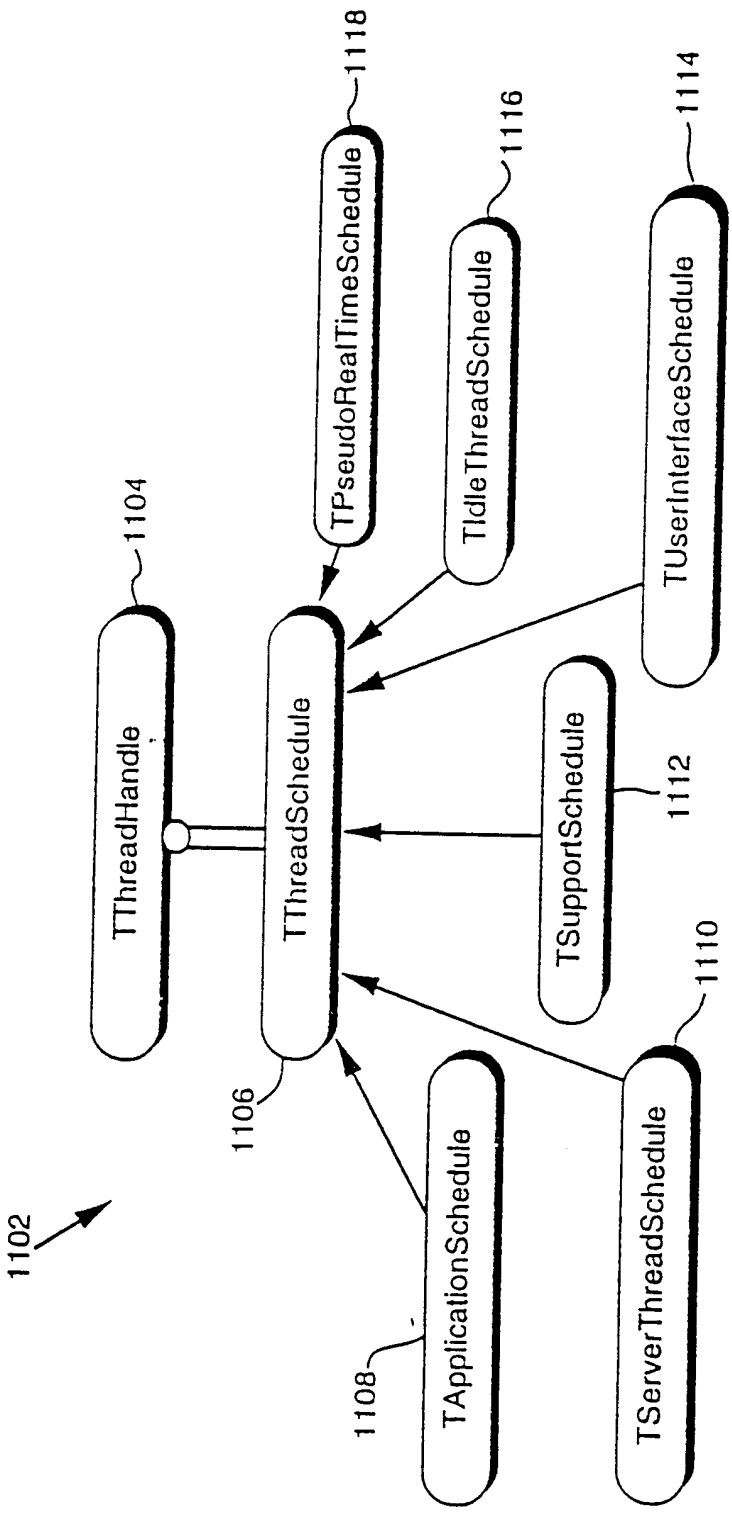

1102
1104
TThreadHandle
1106
TThreadSchedule
1108
TApplicationSchedule
1110
TServerThreadSchedule
1112
TSupportSchedule
1114
TUserInterfaceSchedule
1116
TIdleThreadSchedule
1118
TPseudoRealTimeSchedule

图 11

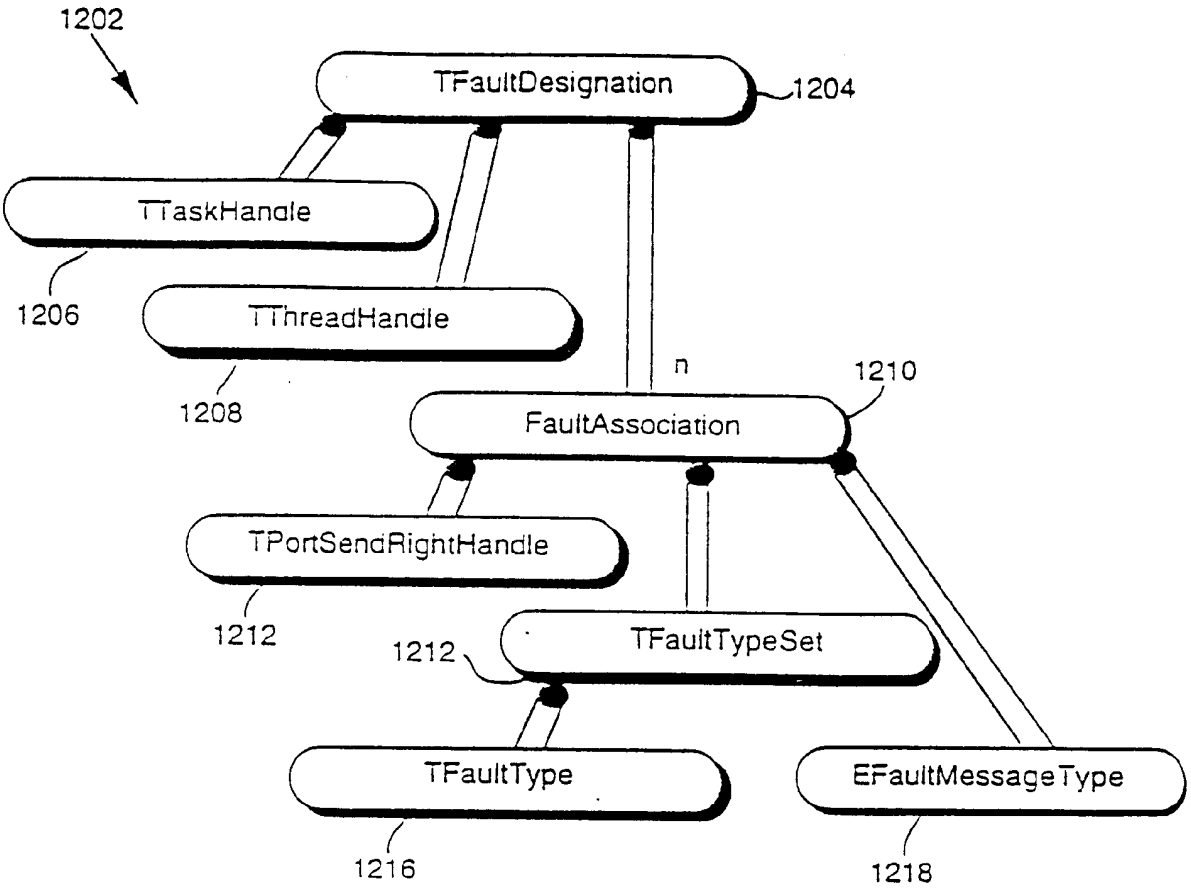
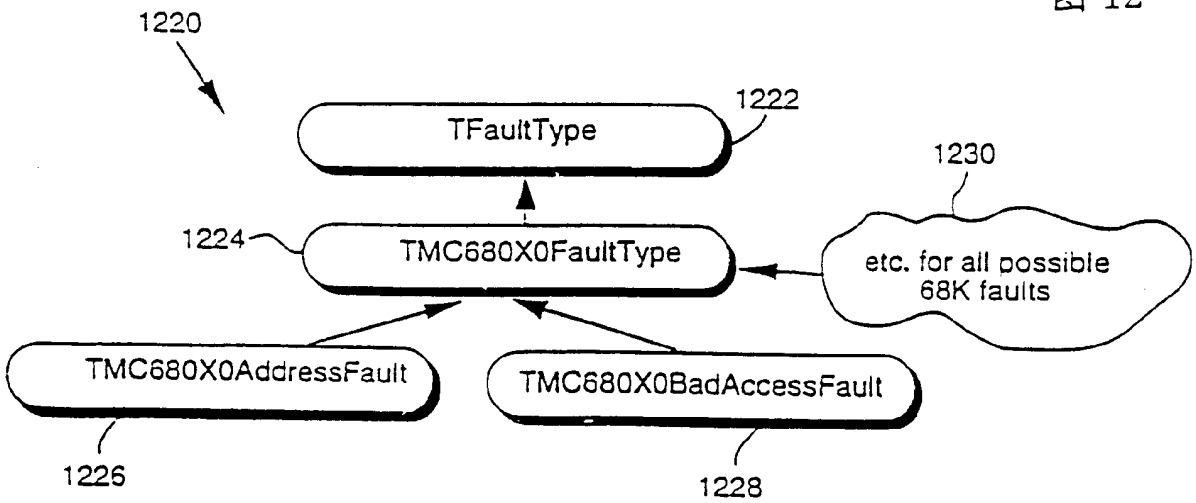

1202
TFaultDesignation
1204
TTaskHandle
1206
TThreadHandle
1208
n
1210
FaultAssociation
TPortSendRightHandle
1212
1212
TFaultTypeSet
TFaultType
1216
EFaultMessageType
1218

图 12


1220
TFaultType
1222
1230
1224
TMC680X0FaultType
etc. for all possible
68K faults
TMC680X0AddressFault
TMC680X0BadAccessFault
1226
1228

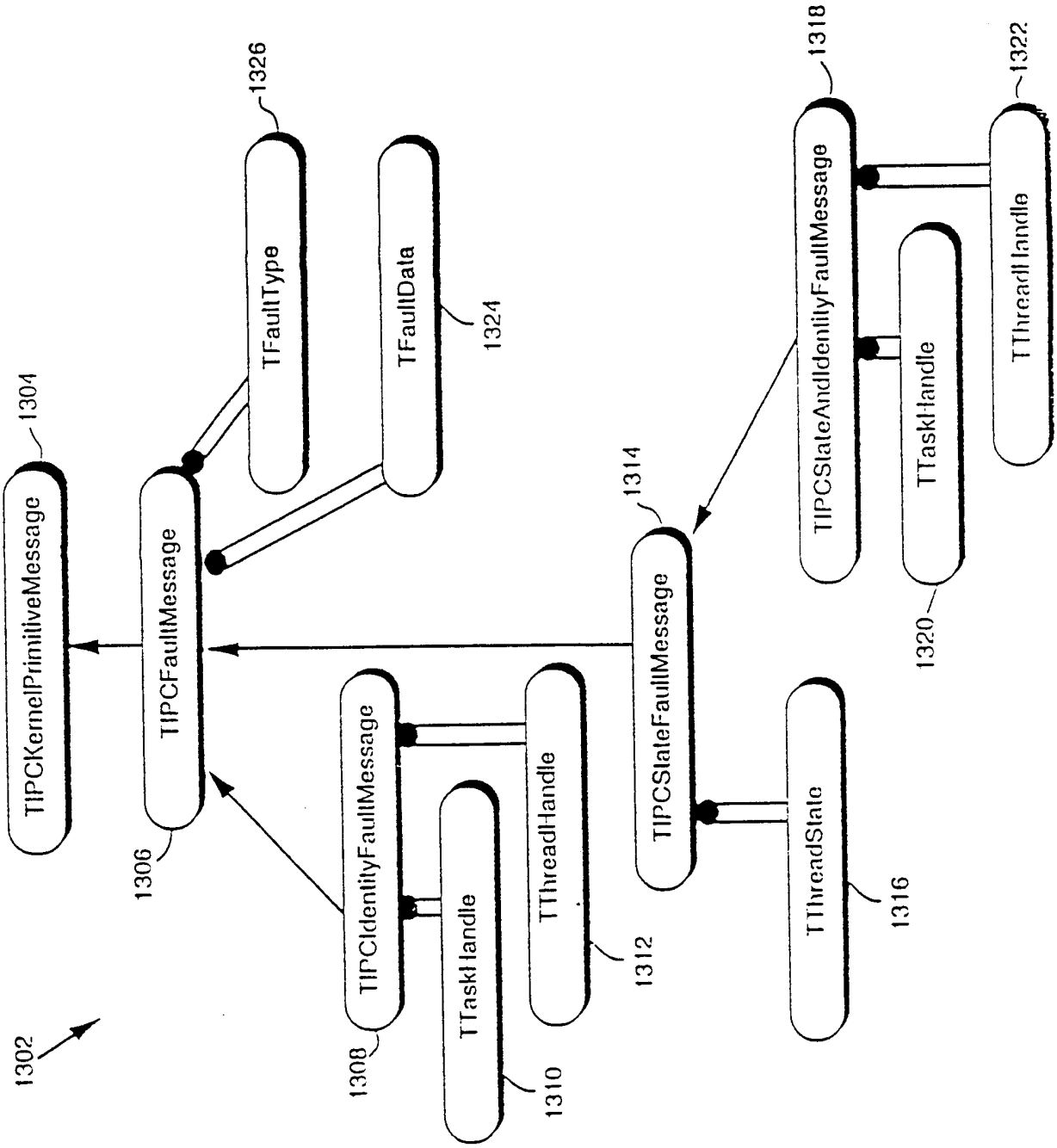

TIPCKernelPrimitiveMessage
1304
TIPCFaultMessage
1306
TFaultType
1326
TFaultData
1324
TIPCIdentityFaultMessage
1308
TTaskHandle
1310
TThreadHandle
1312
TIPCStateFaultMessage
1314
TThreadState
1316
TIPCStateAndIdentityFaultMessage
1318
TTaskHandle
1320
TThreadHandle
1322
1302

图 13

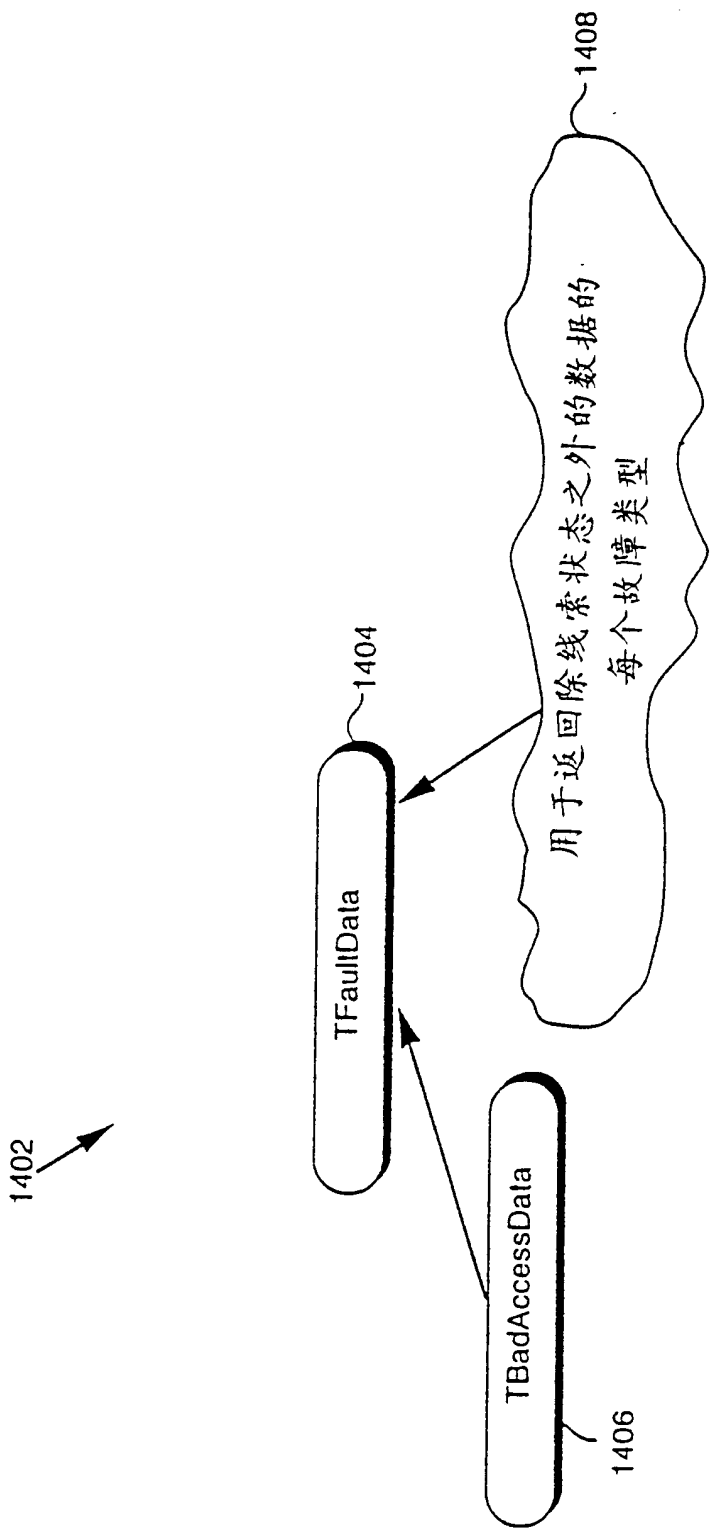

1402
1404
TFaultData
1406
TBadAccessData
1408
用于返回除线索状态之外的数据的
每个故障类型

图 14

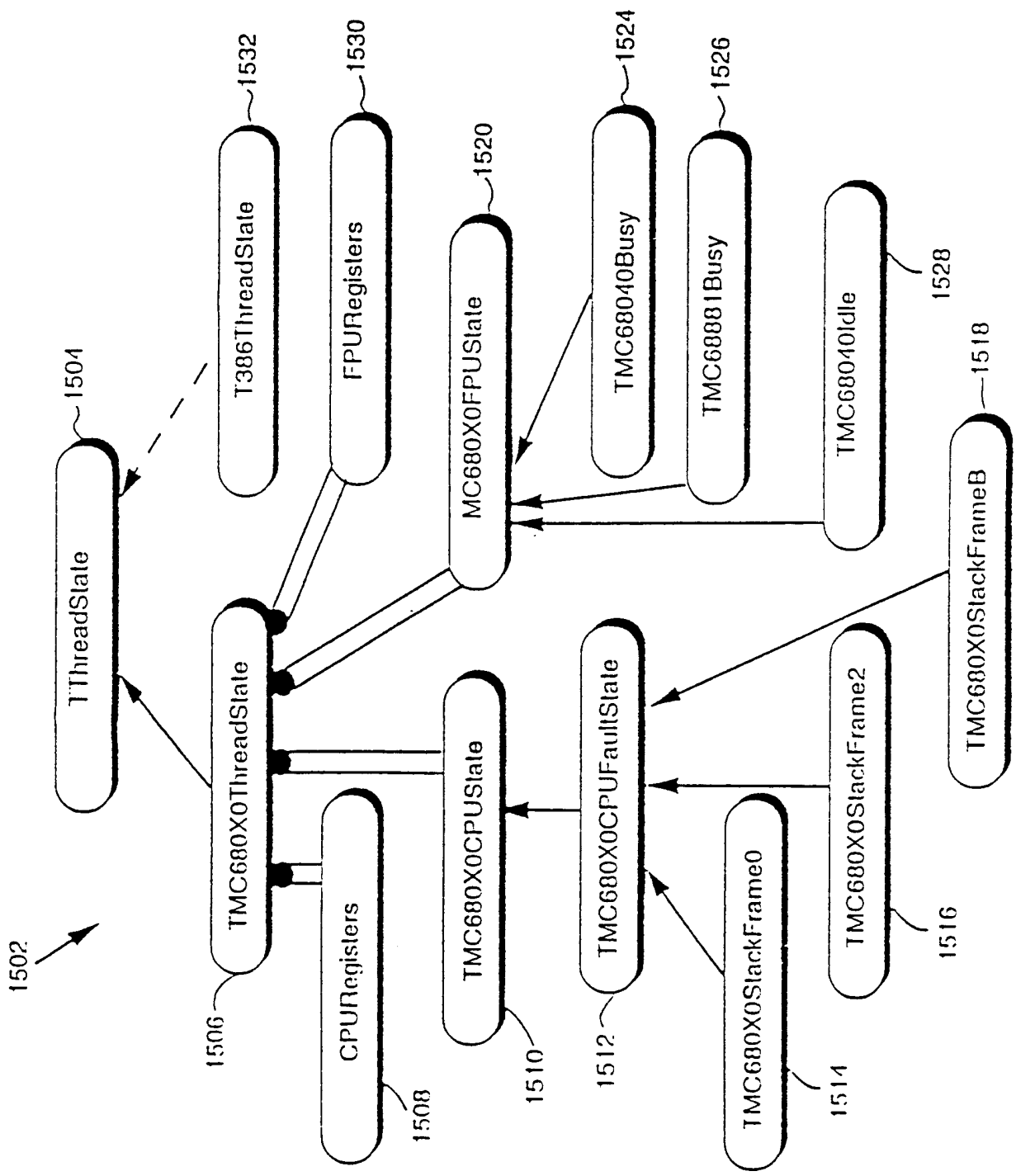

1502
TThreadState
1504
T386ThreadState
1532
FPURegisters
1530
TMC680X0ThreadState
1506
CPURegisters
1508
MC680X0FPUState
1520
TMC68040Busy
1524
TMC68881Busy
1526
TMC68040Idle
1528
TMC680X0CPUState
1510
TMC680X0CPUFaultState
1512
TMC680X0StackFrame0
1514
TMC680X0StackFrame2
1516
TMC680X0StackFrameB
1518

图 15

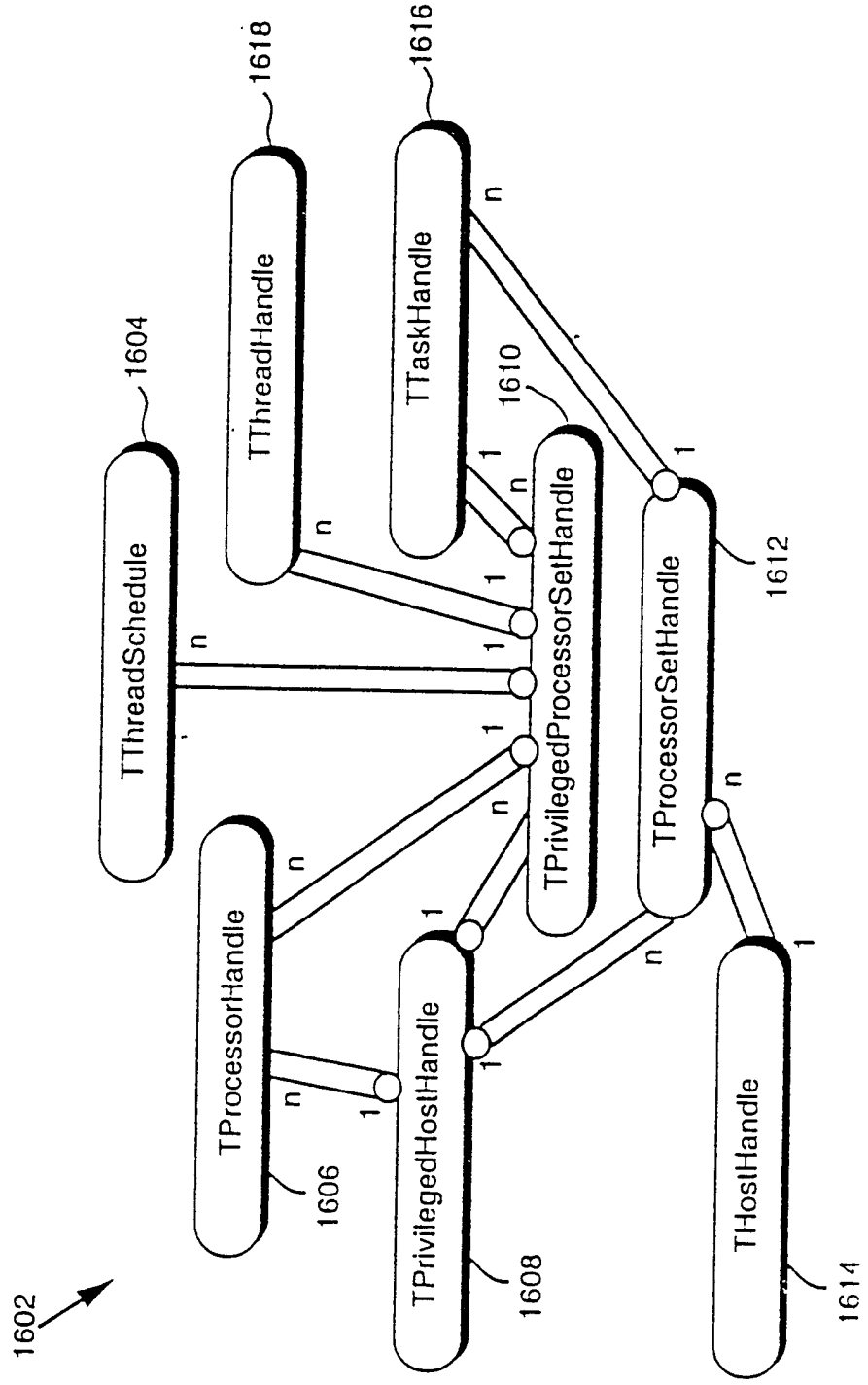

1602
TThreadSchedule
1604
TProcessorHandle
1606
TPrivilegedHostHandle
1608
TThreadHandle
1618
TTaskHandle
1616
TPrivilegedProcessorSetHandle
1610
TProcessorSetHandle
1612
THostHandle
1614
n
1

图 16

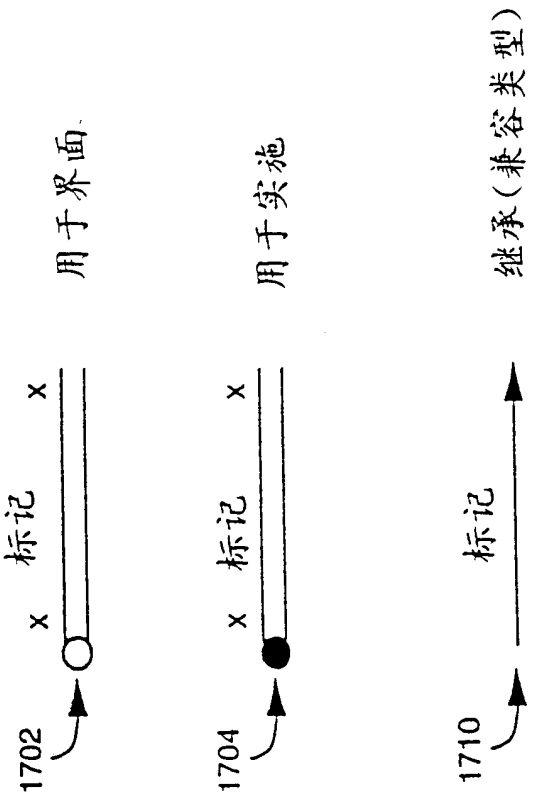


图 17