



(12)发明专利

(10)授权公告号 CN 103443768 B

(45)授权公告日 2016.12.07

(21)申请号 200980135244.8

(22)申请日 2009.07.09

(65)同一申请的已公布的文献号  
申请公布号 CN 103443768 A

(43)申请公布日 2013.12.11

(30)优先权数据  
12/174,444 2008.07.16 US

(85)PCT国际申请进入国家阶段日  
2011.03.10

(86)PCT国际申请的申请数据  
PCT/US2009/050169 2009.07.09

(87)PCT国际申请的公布数据  
W02010/009003 EN 2010.01.21

(73)专利权人 苹果公司  
地址 美国加利福尼亚

(72)发明人 J·沙法尔 R·米斯拉

(74)专利代理机构 中国国际贸易促进委员会专利商标事务所 11038

代理人 邹姗姗

(51)Int.Cl.  
G06F 9/455(2006.01)  
G06F 9/44(2006.01)  
G06F 11/36(2006.01)

(56)对比文件  
US 6442752 B1,2002.08.27,  
US 6442752 B1,2002.08.27,  
US 6292843 B1,2001.09.18,  
CN 100388204 C,2008.05.14,  
Kuperman.“Generation of application level audit data via library interposition”.《CERIAS Tech Report》.1999,第2节第1段至第4节第2段.

审查员 汪见晗

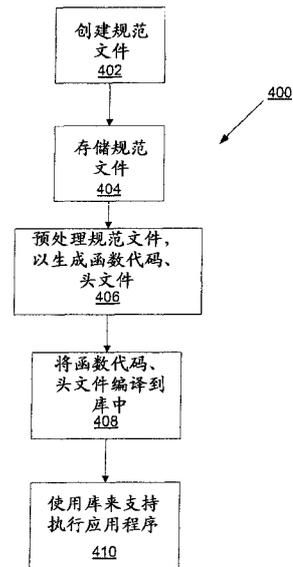
权利要求书2页 说明书18页 附图12页

(54)发明名称

用于调用转换和追踪的规范文件

(57)摘要

描述了用于存储函数规范文件的方法与装置。在示例性方法中,函数规范文件能够提供其它软件来在存在第一操作系统的情况下方便应用程序在第二操作系统中的执行,而且该应用程序是针对第一操作系统编译的。在另一示例性方法中,预处理器接收包括用于库函数的函数定义数据的函数规范文件。预处理器处理函数定义数据,以为该函数生成头信息和函数代码。在另一示例性方法中,预处理器基于函数定义数据为插入库生成自动记录框架。此外,插入库中的函数记录对相应库函数的调用。



1. 一种计算机化的方法,包括:

存储函数规范文件,其中该函数规范文件能够在利用软件进行处理时提供库中经编译的函数,以方便应用程序在第二操作系统中的执行,而不需要第一操作系统的存在,而且其中该应用程序是针对第一操作系统编译的,函数规范文件包括向软件指示将堆栈检查代码添加到所述经编译的函数的关键字,并且所述堆栈检查代码对于在所述经编译的函数被所述应用程序调用时所使用的调用堆栈操作实行公共动作。

2. 一种计算机化的设备,包括:

用于存储函数规范文件的装置,其中该函数规范文件能够在利用软件进行处理时提供库中经编译的函数,以方便应用程序在第二操作系统中的执行,而不需要第一操作系统的存在,而且其中该应用程序是针对第一操作系统编译的,函数规范文件包括向软件指示将堆栈检查代码添加到所述经编译的函数的关键字,并且所述堆栈检查代码对于在所述经编译的函数被所述应用程序调用时所使用的调用堆栈操作实行公共动作。

3. 如权利要求2所述的设备,其中第一操作系统是与第二操作系统不同类型的操作系统。

4. 如权利要求2所述的设备,其中第一操作系统和第二操作系统是相同操作系统的不同版本。

5. 如权利要求2所述的设备,其中第一操作系统和第二操作系统是相同的操作系统。

6. 如权利要求2所述的设备,其中所述库是插入库,该插入库包括调用运行时库、动态链接库和操作系统服务中的至少一个中的对应函数的函数。

7. 如权利要求2所述的设备,其中所述函数规范文件包括函数定义数据,该函数定义数据向处理软件指示生成如下函数,即该函数针对与该函数相对应的库函数执行函数调用转换。

8. 如权利要求7所述的设备,其中所述函数定义数据还包括函数模板调用定义,并进一步向处理软件指示根据函数模板调用定义执行多于一个函数生成。

9. 如权利要求2所述的设备,其中所述函数规范文件包括函数定义数据,该函数定义数据向处理软件指示生成在函数定义数据中定义的对象。

10. 一种计算机化的方法,包括:

根据函数规范文件生成包括多个函数的库,其中该库能够方便应用程序在第二操作系统中的执行,而不需要第一操作系统的存在,而且其中该应用程序是针对第一操作系统编译的,并且函数规范文件包括向软件指示将堆栈检查代码添加到所述多个函数中的一个的关键字,并且所述堆栈检查代码对于在所述多个函数中的所述一个被所述应用程序调用时所使用的调用堆栈操作实行公共动作。

11. 一种计算机化的设备,包括:

用于根据函数规范文件生成包括多个函数的库的装置,其中该库能够方便应用程序在第二操作系统中的执行,而不需要第一操作系统的存在,而且其中该应用程序是针对第一操作系统编译的,并且函数规范文件包括向软件指示将堆栈检查代码添加到所述多个函数中的一个的关键字,并且所述堆栈检查代码对于在所述多个函数中的所述一个被所述应用程序调用时所使用的调用堆栈操作实行公共动作。

12. 一种计算机化的设备,包括:

用于接收函数规范文件的装置,其中该函数规范文件包括用于第一库函数的函数定义数据,并且所述函数规范文件包括向软件指示将堆栈检查代码添加到库函数的关键字,并且所述堆栈检查代码对于在所述第一库函数被应用程序调用时所使用的调用堆栈操作实行公共动作;

用于处理所述函数定义数据,以生成用于所述第一库函数的头信息和函数代码的装置;及

用于基于所述函数定义数据,针对所述第一库函数生成自动记录框架的装置。

13.如权利要求12所述的设备,其中所述用于生成函数代码的装置还包括:用于生成执行对相应运行时库函数的函数调用转换的函数代码的装置。

14.如权利要求12所述的设备,其中所述用于生成函数代码的装置还包括:用于生成支持动态链接库的符号导出的代码的装置。

15.如权利要求12所述的设备,其中所述用于生成函数代码的装置还包括:用于生成支持将函数调用从第一操作系统库重定向到第二操作系统库的代码的装置。

16.如权利要求12所述的设备,其中所述用于生成函数代码的装置还包括:用于根据包括在函数定义数据中的函数模板调用定义生成多于一个的函数的装置。

17.如权利要求12所述的设备,还包括:

用于基于所述函数定义数据,针对所述库函数生成自动调试框架的装置。

18.如权利要求12所述的设备,其中第一操作系统是与第二操作系统不同类型的操作系统。

19.如权利要求12所述的设备,其中第一操作系统和第二操作系统是相同操作系统的不同版本。

20.如权利要求12所述的设备,其中第一操作系统和第二操作系统是相同的操作系统。

21.一种计算机化的系统,包括:

第二操作系统,在该第二操作系统中执行应用程序;以及

处理器,用于存储函数规范文件,其中该函数规范文件能够在利用软件进行处理时提供库中经编译的库,以方便应用程序在第二操作系统中的执行,而不需要第一操作系统的存在,而且其中该应用程序是针对第一操作系统编译的,并且函数规范文件包括向软件指示将堆栈检查代码添加到多个函数中的一个的关键字,并且所述堆栈检查代码对于在所述多个函数中的所述一个被所述应用程序调用时所使用的调用堆栈操作实行公共动作。

22.一种计算机化的系统,包括:

第二操作系统,在该第二操作系统中执行应用程序;以及

处理器,用于根据函数规范文件生成包括多个函数的库,其中该库能够方便应用程序在第二操作系统中的执行,而不需要第一操作系统的存在,而且其中该应用程序是针对第一操作系统编译的,并且函数规范文件包括向软件指示将堆栈检查代码添加到所述多个函数中的一个的关键字,并且所述堆栈检查代码对于在所述多个函数中的所述一个被所述应用程序调用时所使用的调用堆栈操作实行公共动作。

## 用于调用转换和追踪的规范文件

### 技术领域

[0001] 本发明总体上涉及应用程序执行,更特别地涉及函数调用转换和追踪,而且在有些实施例中涉及库的自动生成。

### 背景技术

[0002] 当操作系统执行应用程序时,该应用程序将对由函数库提供的函数进行函数调用。应用程序是在特定操作系统(OS)下被编译并链接以便执行的计算机程序。OS是启动计算机或者其它数据处理系统并且管理该计算机或者其它数据处理系统的函数和/或资源的软件。函数库是向应用程序提供服务的函数的集合。库可以是静态链接的,或者是在运行时动态链接的。静态链接的库是在编译时被链接并且是应用程序的一部分。动态链接的库是当应用程序执行时由该应用程序加载进来或者使用的。动态链接的库的例子是运行时库、动态链接库(Dynamic-Link Library, DLL)和操作系统(OS)服务。运行时库、DLL和OS服务在这里统称为系统库。在一个实施例中,运行时库是由编译器用来在应用程序执行过程中实现构建到编程语言中的函数的计算机程序库。DLL是在运行时加载到应用程序中的函数库。DLL函数可以是特定于操作系统的、特定于应用程序的,等等。在一个实施例中,OS服务是由应用程序在运行期间中使用并特定于特定OS的服务。OS服务可以用以管理诸如存储器的系统资源、文件系统资源、电力状态、图形用户接口、其它资源,执行应用程序间通信等的服务。

[0003] 图1(现有技术)是在操作系统环境中执行的应用程序102的框图。在图1中,应用程序102在利用运行时库104、OS服务106和/或DLL 108的OS环境100中执行。在图1中,应用程序102是专门为在OS环境100中执行而生成的。虽然如此,这个应用程序还可以在不同的OS环境中运行。图2(现有技术)是在一种操作系统环境210(OS2)中执行的应用程序202的框图,其中该应用程序202还在另一操作系统环境200(OS1)中执行。OS2可以是来自与OS1不同厂家的不同操作系统。可选地,OS2可以是OS1的不同版本。在图2中,基于OS2的应用程序202通过使用OS2服务环境210在OS1环境200中运行。在一个实施例中,OS2服务环境210是在OS1环境200中作为虚拟机执行的OS2的真实版本。本领域中已知的这种实施例的例子是VMWARE、PARALLELS和VIRTUALPC。在该实施例中,OS2应用程序202利用OS2服务、DLL和/或运行时库208在操作系统OS2中运行。

[0004] 可选地,OS2服务环境210提供了用于应用程序202的一组应用程序编程接口(API),而不需要在OS2服务环境中执行一种版本的OS2。在该实施例中,OS2服务环境210在OS1环境中利用OS2库206加载并执行基于OS2的应用程序。术语“基于OS2的应用程序”意味着,这个应用程序是针对OS2操作系统编译的,并打算在OS2操作系统下执行。此外,OS2服务环境210还可以使用OS2库208,例如OS2服务、DLL和/或运行时库。本领域中已知的该实施例的例子是WINE(例如,见<http://www.winehq.org>)。

### 发明内容

[0005] 描述了根据至少某些实施例的利用函数规范的方法与装置。在示例性方法中,函数规范文件能够提供其它软件,用以方便应用程序在第二操作系统中的执行,而不需要第一操作系统的存在;在这种方法中,应用程序是针对第一操作系统编译的。在另一示例性方法中,预处理器接收包括用于库函数的函数定义数据的函数规范文件。预处理器处理函数定义数据,以生成用于库函数的头信息和函数代码。在另一示例性方法中,预处理器基于函数定义数据生成用于插入库的自动记录框架。此外,插入库中的函数记录对相应库函数的调用。

### 附图说明

[0006] 通过示例例示了本发明,但本发明不限于附图,附图中类似的标号指示类似的元件。

[0007] 图1(现有技术)是在操作系统环境中执行的应用程序的框图。

[0008] 图2(现有技术)是在OS1环境中执行的应用程序的框图,该应用程序被设计成在OS2环境中执行。

[0009] 图3是在一个操作系统环境中执行的、针对另一个操作系统环境编译的应用程序的框图。

[0010] 图4是从函数规范文件生成用于执行应用程序的库的方法的一个实施例的流程图。

[0011] 图5是从函数规范文件生成函数代码和头信息的方法的一个实施例的流程图。

[0012] 图6是将来自应用程序的调用插入另一个库的插入库的框图。

[0013] 图7是一个操作系统的用于插入应用程序的库调用的插入库的框图,其中该应用程序是针对另一个操作系统编译的。

[0014] 图8是利用插入库将调用插入到库的方法的一个实施例的流程图。

[0015] 图9是为应用程序创建对象的一个操作系统的库的框图,其中该应用程序是针对另一个操作系统编译的。

[0016] 图10是记录对用于两个不同操作系统的操作系统库的调用的方法的一个实施例的流程图。

[0017] 图11是例示了开发者工具包的一个实施例的框图,其中所述开发者工具包生成用于针对相同或不同操作系统编译的应用程序的库。

[0018] 图12是例示了生成用于库的代码的函数规范文件预处理器的一个实施例的框图。

[0019] 图13是例示了记录对插入库的调用的插入库的一个实施例的框图。

[0020] 图14是适于实践本发明的运行环境的一个实施例的图。

[0021] 图15是适于在图14的运行环境中使用的数据处理系统(例如,通用计算机系统)的一个实施例的图。

### 具体实施方式

[0022] 在以下对本发明实施例的详细描述中,参考附图,附图中类似的标号指示类似的元件,而且其中通过例示的方式示出了本发明可以在其中实践的特定实施例。对这些实施例进行了足够详细的描述,以使本领域技术人员能够实践本发明,而且应当理解,其它实施

例也可以使用,而且,在不背离本发明范围的情况下,可以进行逻辑、机械、电子、功能以及其它变化。因此,以下详细描述不应当理解为限制,本发明的范围只由所附权利要求书来限定。

[0023] 在此描述从函数规范定义并生成库函数的方法与装置。图4例示了从函数规范文件生成库的方法的实施例。至少在某些实施例中,这些库可以称为插入库,它们用于支持针对第一OS设计的应用程序在无需第一OS执行副本的情况下在共同执行第二OS的系统上操作/执行。此外,该方法还从函数规范文件生成头信息和函数代码。图5进一步例示了生成头信息和函数代码的方法的一个实施例,其中该方法对函数规范文件进行预处理。图8例示了利用从函数规范文件生成的插入库记录对函数库的调用的方法的一个实施例。图10例示了生成和比较来自在两个不同操作系统上执行的应用程序的日志的方法的一个实施例。图11至13例示了实现图4、5、8和/或10的应用程序和执行系统的一个实施例。

#### [0024] 插入库概述

[0025] 图3是针对一个操作系统环境编译的、在另一个操作系统环境下执行的应用程序的框图。在图3中,基于OS2的应用程序316在不存在操作系统OS2的OS1环境300中执行。基于OS2的应用程序316在该环境下执行是因为OS1环境300使用了基于OS1的加载器(未示出),该加载器可以加载并执行应用程序,该应用程序是基于OS1或者OS2的应用程序。此外,如图1和2所述,基于OS2的应用程序316使用库310、312和/或314来执行通常由操作系统OS2(或者OS2环境)处理的、基于OS2的库函数调用。这些库插入在调用应用程序与这些库所插入的实际库之间,因此这些库310、312和314可以称为插入库。在一个实施例中,插入库310、312和314是将对在运行时库、DLL和/或OS2服务中的基于OS2的函数的调用转换成对在基于OS1的运行时库、DLL和/或基于OS1的服务中的函数的对应基于OS1的函数调用。例如并且作为例示,RTL插入库310将对运行时库中的基于OS2的函数的调用转换成对运行时库304中的函数的对应调用。RTL插入库310可以转换运行时库304中的一个、一些、许多或者全部函数调用。可选地,可以有多个RTL插入库310,其中每个库转换对一个或多个运行时库304的一个或多个函数调用。

[0026] 在一个实施例中,插入库312将对用于OS2服务的函数的调用转换成支持OS1服务库316中的那些OS2服务的对应函数。在一个实施例中,有一个插入库312,用以转换对OS1服务库306的函数调用。可选地,有若干个库312,其中每个库转换对一个或多个OS1服务库306的一个或多个函数调用。

[0027] 在一个实施例中,DLL插入库314将对DLL中所包含的函数的调用转换成支持用于应用程序316的DLL 308中对应函数的对应函数。DLL可以是针对OS1、OS2或者某个其它操作系统编译的DLL。在一个实施例中,DLL插入库314导出DLL 308中函数的符号。在一个实施例中,有一个DLL插入库314,用以转换对DLL库308中对应函数的函数调用。可选地,有多个DLL插入库314,其中每个库转换对一个或多个DLL库308的一个或多个函数调用。

#### [0028] 插入库的生成

[0029] 图4是从函数规范文件生成用于执行应用程序的插入或其它库的方法400的一个实施例的流程图。在一个实施例中,OS1环境300执行生成这些库的方法400,其中这些库包括但不限于运行时插入库310、OS1服务插入库312和/或DLL插入库314。在一个实施例中,软件开发者使用开发者工具包来开发应用程序、操作系统部件、库,等等。开发者工具包是开

发者将用以开发软件的工具集合。工具包可以包括编辑器、编译器、链接器、文件预处理器，等等。这里在图11中描述结合了本发明实施例的开发者工具包的例子。

[0030] 在图4中，在块402，方法400创建函数规范文件。在一个实施例中，函数规范文件包括函数定义数据，该函数定义数据向方法400指示生成软件以执行用于与操作系统OS2关联的库函数的函数调用转换。在一个实施例中，函数定义数据包括用于每个函数的、定义要转换的函数调用的数据。对于每个函数，函数定义数据可以包括函数样板(prototype)、关键字、库名称、可选参数和用于支持插入库生成的其它信息中的一个或多个。在一个实施例中，函数样板是定义函数名称、函数参数列表和返回值类型的本领域中已知的编程语言样板。所支持的编程语言样板的例子是C、C++、JAVA、PASCAL，等等。关键字是函数定义数据中由方法400用于修改由方法400生成的函数头和代码的词或者短语。在一个实施例中，关键字可以用于向方法400指示添加堆栈检查代码、符号导出代码，等等。方法400使用堆栈检查代码来加强由OS1或OS2所使用的特定堆栈变量管理策略。这种代码对于为基于一个操作系统或者为一个操作系统所写的应用程序生成在另一个操作系统中使用的库是有用的。对于何时将函数参数放到调用堆栈上和何时从堆栈拿掉这些参数，不同的操作系统可以具有不同的约定。使用堆栈检查指定了对所有堆栈管理要采取的公共动作。方法400使用符合导出关键字来生成支持函数名称和其它函数符号的导出的函数代码，以便使调用应用程序可以获得这些符号。下面将参照图5中的块508、510、512、514和532进一步描述函数定义。

[0031] 此外，函数定义数据可以包括可选参数。在一个实施例中，这些参数可以包括修改函数模板生成的参数。方法400可以使用这些参数中的一些来帮助从一个函数定义生成多个函数。例如，方法400可以从处理不同类型字符串参数与返回值(例如，ASCII、Unicode，等等)的单个函数定义生成多个字符串函数。下面将参照图5中的块512进一步描述函数调用模板生成。

[0032] 在块404，方法400存储函数规范文件。方法400可以将这个文件存储在与执行方法400的处理器本地或者远程的计算机、其它存储设备或者其它数据处理系统中的硬盘、存储器、闪存存储器等中。

[0033] 在块406，方法400预处理函数规范文件，以基于该函数规范文件中的函数定义生成函数代码和头信息。预处理是处理输入数据以便产生由另一个程序使用的输出的动作。在一个实施例中，方法400利用函数定义数据生成头信息。如本领域中已知的，头信息是函数、变量和/或其它标识符的前向声明(declaration)。这种信息可以存储在文件、存储器等中。编译器使用头信息来编译函数代码，以便产生库、可执行文件、应用程序，等等。

[0034] 此外，方法400利用函数定义数据生成函数代码。在一个实施例中，函数代码是该函数的完整实现，这包括函数调用定义、参数定义、记录框架、调试框架及允许开发者无需进一步编辑就能编译和使用该函数代码的其它函数软件。在一个实施例中，这种生成的函数代码支持调用转换。函数调用转换是将从一个应用程序对函数的调用转换成对作为执行该应用程序的操作系统的库中函数的调用。例如，在该实施例中，可以在系统库中生成与函数定义数据中具有相同名字和参数列表的函数。可选地，方法400可以生成作为函数部分实现的函数代码。在该实施例中，开发者编辑所生成的函数代码，来完全地实现该函数。在另一实施例中，方法400生成在运行时实例化函数定义中所述对象的代码。

[0035] 在块408，方法400将函数代码和头信息编译到库中。例如，方法400可以生成插入

运行时库310、OS1服务312和/或DLL 314,如参考图3所描述的。如本领域中已知的,方法400使用编译器来生成库。在块410,方法400使用这些插入库来执行应用程序。在一个实施例中,方法400利用用于编译应用程序的相同操作系统中的插入库来执行该应用程序。在该实施例中,插入库可以用于生成由执行应用程序所使用的系统库的具体追踪。可选地,方法400利用与用于编译应用程序的操作系统不同的操作系统中的插入库来执行应用程序。该不同的操作系统可以与用于生成插入库的操作系统是相同的类型,但是不同的版本。例如,执行应用程序的操作系统可以是由APPLE公司提供的MAC OS X 10.5.1操作系统,而该应用程序是针对MAC OS X 10.5.0编译的。可选地,该不同的操作系统可以是来自相同或者不同厂家的不同类型的操作系统。例如,执行应用程序的操作系统可以是MACINTOSH OS 10.5.1,而应用程序是针对由微软公司提供的WINDOWS XP操作系统编译的。不同OS的例子是但不限于MAC OS 9、MAC OS X 10.x、WINDOWS 95、WINDOWS 98、WINDOWS XP、WINDOWS VISTA、本领域中已知的许多不同类型的LINUX和UNIX,等等。

[0036] 在图4中,方法400使用函数规范文件来生成可以由一个或多个应用程序(例如,图3中的应用程序302)使用的插入库,以进行并记录对OS服务、运行时和/或DLL库中对应函数的调用。在这个图中,在块406,方法400预处理函数规范,来生成头信息和函数代码,以便支持插入库的生成。这种预处理功能一般是一组开发者工具中的一个。图5是从函数规范文件生成函数代码和头信息的方法500的一个实施例的流程图。方法500可以用于生成头信息和函数代码,来支持针对与方法500在其下运行的操作系统相同或不同的操作系统编译的应用程序的执行。在图5及这里其它图的讨论中,参考“C”编程语言给出了例子。本发明不限于这种编程语言,因为本领域技术人员将认识到,本发明可用于本领域中已知的其它编程语言使用,例如C++、JAVA、PASCAL,等等。

[0037] 在图5中,在块502,方法500接收函数规范文件定义。在一个实施例中,通过从如图4的块404中所描述的存储在盘、存储器等中的函数规范文件检索定义,方法500接收函数规范文件定义。方法500可以接收一个或多个函数定义。

[0038] 在块504至534,方法500执行处理循环,来为接收到的每个函数定义生成函数代码和头信息。在块506,方法500利用本领域中已知的一种解析策略来解析函数定义。通过解析,方法500确定与每个函数定义关联的函数样板、关键字、库名称、可选参数和/或其它信息。

[0039] 在块508,方法500确定该函数定义是否是运行时库函数调用定义。在一个实施例中,方法500确定该函数定义的函数样板是否与所指定库中的函数是相同的样板。如果是,则在块516,方法500生成用于运行时定义的函数代码和头信息。例如,方法500接收函数定义:

[0040] `stdcall void function2(void)libc;`

[0041] 在这个定义中,stdcall是关键字,void是从对function2的调用返回的参数类型,function2(void)是用于function2的函数样板,而libc是库的名称。在这个例子中,方法500检查function2是否在运行时库libc中定义为具有void的返回参数类型和自变量void。如果存在另一个具有相同样板的函数,则方法500为function2生成头信息和函数代码。

[0042] 如上所述,在块516,方法500为在块508确定的运行时库函数调用定义生成头信息和函数代码。在一个实施例中并且利用以上的示例函数定义,方法500将针对function2的

头信息生成为：

[0043] void STDCALL pe\_function2(void)\_force\_align\_arg\_pointer;

[0044] 这个头信息包括返回类型(“void”)、函数名称(“pe\_function2”)和函数参数列表(“void”)。此外,方法500还向头信息添加了STDCALL和“\_force\_align\_arg\_pointer”参数。STDCALL参数向编译器指示使用stdcall调用约定。在这种调用约定中,被调用者负责在返回到调用者之前清空堆栈。“\_force\_align\_arg\_pointer”参数用于堆栈检查。在一个实施例中,编译器使用“\_force\_align\_arg\_pointer”参数在函数的开始处生成对准堆栈的代码。在该实施例中,方法500将函数从“function2”重命名为“pe\_function2”。这样做是为了避免命名空间冲突,因为libc已经有一个名称为“function2”且被函数pe\_function2调用的函数。在另一实施例中,方法500不重命名function2。可选地,方法500生成适于开发者所使用的编程语言的头信息。

[0045] 此外,方法500为在函数定义中定义的插入运行时库函数生成函数信息。在一个实施例中,并且使用以上的示例函数定义,方法500生成具有包括以下内容的结构的函数代码:

[0046] 函数序言

**void STDCALL pe\_function2(void){**

*定义变量*

*初始记录代码*

*初始调试代码*

[0047] *RTL 函数代码*

*附加调试代码*

*附加记录代码*

*函数返回*

**}**

[0048] 在该实施例中,如本领域中已知的,函数序言是一组用于定义头文件导入、全局变量、DLL定义等的代码。定义变量是一组定义用于支持RTL函数代码调用、记录代码、调试代码的变量及pe\_function2中使用的其它变量的代码。初始记录代码是一组用于定义初始记录的代码,其中初始记录用于记录对运行时库函数进行调用之前的信息。在一个实施例中,初始记录代码记录函数名称、参数名称和所传递的值、函数被调用的时间,等等。初始调试代码是一组用于输出在对运行时库函数进行调用之前的调试信息的代码。RTL函数代码是用于调用运行时库函数的代码。附加调试代码是一组用于输出在调用运行时库函数之后的调试信息的代码。附加记录代码是一组用于定义附加记录的代码,其中附加记录用于记录在调用运行时库函数之后的信息。函数返回是一组设置返回参数的值并将该参数返回到调用应用程序的代码。尽管该实施例例示了利用以上所列结构的全部来生成函数代码,但是在可选实施例中,方法500可以利用所列结构中的一个或者一些来生成函数代码。执行前进到下面的块524。

[0049] 如果函数定义不是运行时库函数调用定义,则在块510,方法500确定该函数定义是否是DLL函数调用定义。在一个实施例中,方法500通过关键字“reexport”的存在来确定该函数定义是否是对DLL函数的函数调用。这个关键字指示将DLL中所使用的函数调用的符号导出到调用应用程序。如果函数定义是这种函数调用,则方法500在块518生成函数代码和头信息。例如,方法500接收函数定义:

[0050] reexport function1 libraryb;

[0051] 在这个定义中,reexport是指示DLL函数调用定义的关键字,function1是名称,而libraryb是包括function2的DLL。响应于reexport关键字,方法500在块518为function2生成头信息和函数代码。

[0052] 如上所述,在块518,方法500为在块510确定的DLL函数调用定义生成头信息和函数代码。在一个实施例中并且使用以上的DLL函数定义,方法500生成头信息:

[0053] extern“C”void pe\_function1();

[0054] 这个头信息包括extern关键字、void返回类型和函数名称(“pe\_function1”)。尽管在这个例子中,pe\_function1没有参数,但是在可选实施例中,方法500可以根据需要在头信息中生成参数列表。此外,方法将function1重命名为“pe\_function1”,以避免命名空间冲突,如以上参考块516所描述的。

[0055] 此外,方法500为DLL函数调用定义生成函数代码。在一个实施例中,并且使用以上的示例函数定义,方法500生成具有包括以下内容的结构的函数代码:

[0056] 函数序言

**static void reexport\_init(){**

[0057]

*定义变量*

*初始记录代码*

*初始调试代码*

*DLL 加载代码*

*DLL 函数调用代码*

[0058]

*附加调试代码*

*附加记录代码*

*函数返回*

**}**

[0059] 由方法500为DLL函数调用生成的函数代码类似于在块516中描述的运行时库函数。如本领域中已知的,函数序言定义了头文件导入、全局变量、DLL定义等。定义变量定义了支持DLL函数代码调用、记录代码、调试代码的变量和/或reexport\_init中使用的其它变量。初始记录代码定义了用于记录DLL函数调用之前的信息的初始记录代码。初始调试代码输出调用DLL函数之前的调试信息。附加调试代码输出在调用DLL函数之后的调试信息。附加记录代码定义用于记录在DLL函数调用之后的信息的附加记录代码。函数返回设置返回参数的值,并将该参数返回到调用应用程序。

[0060] 此外,方法500还生成建立对DLL函数的调用和调用该DLL函数的DLL加载代码和DLL函数调用代码。DLL加载代码是一组用以加载适当DLL的代码。DLL可以是由方法500所使用的操作系统的DLL,或者也可以是针对另一操作系统(不同版本、类型,等等)编译的DLL。DLL函数调用代码是一组用以调用所加载的DLL中的函数的代码。

[0061] 如果函数定义不是DLL函数调用定义,则方法500在块512确定该函数定义是否是函数调用模板定义。函数定义通过某种类型的参数和/或函数名称来指示函数调用模板定义。在一个实施例中,函数调用模板定义是通过函数定义参数列表、函数名称和/或其组合中字符串变量的存在来指示的。例如,以下函数定义可以是函数调用模板定义:

[0062]

```
stdcall void function7(LPTSTR string) genaw;
stdcall void function8(LPTSTR string) nocf A=0x0003
W=0x0004;
stdcall void function9(LPTSTR string) callthrough;
stdcall void function7(SAMPLE_STRUCT* structure);
```

[0063] function7函数调用模板定义包括stdcall关键字、返回类型(“void”)、函数样板(“function7(LPTSTR string)”)和可选参数(“genaw”)。function8、function9和function10具有类似的函数定义。在这个例子中,用于function7至function9的函数定义通过参数列表中参数类型LPTSTR的使用来指示函数调用模板定义。而用于function10的函数定义通过经参数列表传递的特定SAMPLE\_STRUCT来指示函数调用模板定义。在该实施例中,genaw、nocf和callthrough是可选参数,用以指示当方法500生成用于这些函数的头信息和函数代码时要使用哪个模板。

[0064] 在块520,方法500为函数调用模板中定义的函数调用生成头信息和函数代码信息。在一个实施例中并使用以上的function7函数调用模板定义,方法500生成以下的头信息:

[0065]

```
void STDCALL pe_function7CF(CFMutableStringRef string)
_force_align_arg_pointer;
void STDCALL pe_function7A(LPSTR string)
_force_align_arg_pointer;
void STDCALL pe_function7W(LPWSTR string)
_force_align_arg_pointer;
```

[0066] 在这个例子中,方法500生成三个头,来处理可用于function7的三个不同变体的三个不同的字符串自变量。在这里,方法500生成函数,来处理类型为CFMutableStringRef(pe\_function7CF)、LPSTR(pe\_function7A)和LPWSTR(pe\_function7W)的字符串。如上,每个函数都以预定的“pe\_”重命名,以避免命名空间冲突。此外,每个函数都具有使用STDCALL参数来指示标准调用约定和“\_force\_align\_arg\_pointer”可选参数来加强堆栈检查的头信息。在一个实施例中,这种函数调用模板定义用作转换对具有类似命名约定但处理不同

类型相关参数的函数库的调用的快捷方式(shortcut)。

[0067] 在一个实施例中,方法500使用可选参数“genaw”、“nocf”和“callthrough”来控制生成哪些函数。在该实施例中,“genaw”指示生成名称为impl\_functionnameCF的单个函数,并使用类型为CFMutableStringRef的字符串。相反,“nocf”指示生成分别具有LPSTR和LPWSTR类型字符串的、名称为“impl\_functionnameA”和“impl\_functionnameW”的两个函数。“callthrough”指示生成接受额外参数的单个函数,其中额外参数指示实际调用哪个输入点(A、W或者CF)。

[0068] 此外,方法500为函数调用模板定义中定义的每个函数生成函数信息。在一个实施例中,并且使用从函数调用模板定义生成的示例函数中的一个,方法500生成具有包括以下内容的结构的函数代码:

[0069] 函数序言

```
void STDCALL pe_function7A(LPSTR string){
```

*定义变量*

*初始记录代码*

*初始调试代码*

[0070] *函数调用代码*

*附加调试代码*

*附加记录代码*

*函数返回*

```
}
```

[0071] 由方法500为这个函数调用生成的函数代码类似于在块516中描述的运行时库函数代码和在块518中的DLL函数代码。如本领域中已知的,函数序言定义了头文件导入、全局变量、DLL定义等。定义变量定义了支持函数代码调用、记录代码、调试代码的变量和/或pe\_function7各个版本中所使用的其它变量。初始记录代码定义了用于记录在OS服务函数调用之前的信息的初始记录代码。初始调试代码输出在调用OS服务函数之前的调试信息。附加调试代码输出在调用OS服务函数之后的调试信息。附加记录代码定义用于记录在调用OS服务函数之后的信息的附加记录代码。函数返回设置返回参数的值,并将该参数返回到调用应用程序。此外,方法500还生成对相应函数进行调用的函数调用代码。执行前进到下面的块524。

[0072] 如果函数定义不是对OS服务的函数调用模板定义,则方法500在块514确定该函数定义是否是对OS服务的基本函数调用定义。在一个实施例中,对OS服务的基本函数定义是用于对该OS服务的调用转换的函数定义。OS服务可以用于与方法500在其下执行的相同或者不同OS。在一个实施例中,基本函数调用定义利用函数名称和样板及可选参数和内联代码一起来指示函数调用转换。在可选实施例中,其它函数定义类型可以使用内联代码。例如,以下函数定义可以是基本函数定义:

```

cdecl void function3(void) ordinal=0x0001;
stdcall void function4(void) ordinal=0x0002;
[0073] stdcall void function5(int* number) body={*number=1};
stdcall int function6(void) result=1;

```

[0074] function3函数定义包括cdecl关键字、void的返回类型、函数样板(“function3(void)”)和可选参数(“ordinal=0x0001”)。在一个实施例中,ordinal指示对相同函数的交替访问。例如,function3可以通过名称(pe\_function3)或者通过值(1)来查找。function4至6具有类似的函数定义。此外,函数定义可以定义内联代码和返回值,如分别在function5和function6函数定义中所使用的。在一个实施例中,方法500生成用于每个基本函数定义的一组头信息和函数代码。

[0075] 在块522,方法500为在基本函数定义中定义的函数调用生成头信息和函数代码信息。在一个实施例中并且利用以上的function3函数定义,方法500生成以下头信息:

```
[0076] void pe_function3(void)_force_align_arg_pointer;
```

[0077] 该头信息包括void返回类型、函数名称(pe\_function3)。尽管在这个例子中,pe\_function3没有参数,但是在可选实施例中,方法500可以根据需要在头信息中生成参数列表。此外,方法500将function3重命名为“pe\_function3”,以避免命名空间冲突,如以上参考块516所描述的。

[0078] 此外,方法500为基本函数调用定义生成函数信息。在一个实施例中,并且使用以上的示例函数定义,方法500生成具有包括以下内容的结构的函数代码:

[0079] 函数序言

```
void pe_function3(void){
```

```
    定义变量
```

```
    初始记录代码
```

```
    初始调试代码
```

```
[0080]    函数调用代码
```

```
    附加调试代码
```

```
    附加记录代码
```

```
    函数返回
```

```
}
```

[0081] 由方法500为基本函数调用生成的函数代码类似于在块516中描述的运行时库函数。如本领域中已知的,函数序言定义了头文件导入、全局变量、DLL定义等。定义变量定义了支持基本函数代码调用、记录代码、调试代码的变量和pe\_function3中所使用的其它变量。初始记录代码定义了用于记录在OS服务调用之前的信息的初始记录代码。初始调试代码输出在OS服务调用之前的调试信息。附加调试代码输出在OS服务调用之后的调试信息。附加记录代码定义用于记录在OS服务调用之后的信息的附加记录代码。函数返回设置返回

参数的值,并将该参数返回到调用应用程序。此外,方法500还生成建立基本函数调用的函数调用代码。执行前进到下面的块524。

[0082] 如果函数定义不是对OS服务的基本调用,则方法500确定该函数定义是否是对象定义。在一个实施例中,对象定义定义由与执行方法500的操作系统相同或不同的操作系统所使用的对象。例如并且作为例示,对象可以是如本领域中已知的基于微软WINDOWS的组件对象模型(COM)对象。例如,以下对象定义是用于新对象类ExampleObject1的COM声明:

[0083]

*//在以下例子中,我们声明一个新的对象 ExampleObject1  
//spec 发生器将创建在运行时实例化一个对象所需的全部代码  
//并且将创建用于 IExampleClass 和 IUnknown 接口中所有方法的缺省实现*

*//除 SampleMethod1 之外,它在下面标记为“imp”*

*//以便指示我们已经提供了实现*

```
class ExampleObject1 : public unimp IExampleClass {
    imp SampleMethod1;
    void AdditionalSampleMethod(void);
private:
    int fPrivateData;
};
```

[0084] 在这个例子中,类ExampleObject1实现了IExampleClass接口。ExampleObject1类有两个定义的函数,即,未实现的SampleMethod1和AdditionalSampleMethod。一方面,用于SampleMethod的“imp”关键字指示开发者将提供SampleMethod的实现。另一方面,方法500生成AdditionalSampleMethod的缺省实现。此外,ExampleObject1定义了一个私用的int,即fPrivateData。

[0085] 在块532,方法500为对象定义中定义的类生成头信息和对象函数代码。在一个实施例中并且使用以上的ExampleObject1定义,方法500为类ExampleObject1和该类的任何继承接口与类定义生成头信息。例如,方法500生成以下头信息:

**class ExampleObject1:**

*初始声明*

[0086] *继承的类/接口定义*

*ExampleObject1 定义*

**};**

[0087] 在该实施例中,初始声明是由类ExampleObject1使用的函数、变量等的初始声明。继承的类/接口定义是用于ExampleObject1继承的类和/或接口的函数、变量等的定义。

ExampleObject1定义是由类ExampleObject1使用、未在继承的类/接口定义中定义的附加函数、变量等的定义。在另一个实施例中,由ExampleObject1继承的类和/或接口还可以利用函数规范文件中的函数定义数据来定义。在该实施例中,这种附加的数据是以类似于其它对象定义的方式处理的。

[0088] 在一个实施例中,并且利用以上的示例对象定义,方法500为类ExampleObject1生成具有包括如下内容的结构的函数代码:

[0089] 接口定义函数

[0090] 继承类定义函数

[0091] 特定于类的函数

[0092] 在该实施例中,接口定义函数是为作为ExampleObject1实现的任何接口的一部分定义的ExampleObject1函数生成的一组代码。继承类定义函数是为作为ExampleObject1继承的任何类的一部分定义的ExampleObject1函数生成的一组代码。特定于类的函数是为专门针对ExampleObject1定义的函数生成的一组代码。这些生成的函数可以包括为其它类型函数定义(运行时库函数调用定义、DLL调用定义、函数模板调用定义、基本函数调用定义,等等)中的一个定义的任何或者全部代码结构。例如,以这种方式定义的类函数可以具有函数序言、变量定义、记录代码、调试代码、函数调用代码和函数返回代码中的一个或者多个,如针对这些其它类型函数定义所生成的。此外,为类ExampleObject1生成的函数还可以是完全实现的函数或者可以是部分实现。执行前进到下面的块524。

[0093] 如果函数定义不是对象定义,则方法500在块528发信号通知不支持的定义错误。在一个实施例中,方法500通过发信号通知错误、打印错误、停止执行等来发信号通知不支持的定义。

[0094] 在块524,方法500可选地接收对所生成函数代码的手动改变。在一个实施例中,方法500接收对函数调用代码、记录代码、调试代码等的改变。此外,方法500还将该变化集成到所生成的函数代码中。在块526,连同可选的函数代码变化一起,方法500存储所生成的头信息与函数代码。处理循环在块530结束。

[0095] 插入库应用程序

[0096] 具有在图5中生成的记录框架的自动生成的插入库可以在许多情况下使用,例如但不限于:记录对OS服务、DLL和/或运行时库的调用;帮助使得可以在不同的操作系统下执行应用程序;及比较在许多不同操作系统上执行的基于单个OS的应用程序的日志。

[0097] 在一个实施例中,自动生成的插入库支持OS服务、DLL和/或运行时库的自动记录。这种记录可以用于为与控制应用程序执行的操作系统相同或不同的操作系统编译的应用程序。此外,这种记录还可以用于记录由应用程序对OS服务、DLL和/或运行时库的调用,其中应用程序带或者不带调试信息地进行了编译。图6是在OS1环境600中插入从应用程序602对系统库604的调用的插入库606的框图。在图6中,应用程序602是针对OS1环境600编译的,而且应用程序602对基于OS1的服务、DLL和/或运行时库进行调用。系统库604可以是一个或多个如参考图1及库102、104和106所述的OS1服务、DLL和/或运行时库。系统库604还包括函数614A。在一个实施例中,函数614A可以实现OS1服务或者在DLL和/或运行时库中找到的函数。插入库606可以是用于如参考图3及库310、312和314所述的OS1服务、DLL和/或运行时库的一个或多个插入库。插入库606还包括函数614B。函数614B是对应于函数614A的插入函

数。在一个实施例中，函数614B是基于如图4和5中所述的函数规范文件生成的。

[0098] 在图6中，应用程序602对函数614A进行调用612A。在缺少插入库606的情况下，函数614A返回到应用程序602，包括任何经612B返回的参数。利用为系统库604插入的插入库606，OS1环境600将应用程序602对函数614A的调用经608A重定向到插入库606中的对应函数614B。函数614B经608B将这种调用记录到记录仓库616中。在一个实施例中，函数614B记录进行调用的时间、库函数名称、和/或传递到库函数614B的参数的名称与值。记录仓库616可以是本领域中用于存储函数调用记录的文件、存储器、数据库，等等。函数614B经608C调用函数614A。

[0099] 函数614A经610A返回到插入库606中的函数614B。在一个实施例中，函数614A将参数返回到函数614B。函数614B将任何附加的记录，例如函数614A返回值、时间戳、返回的参数等，经610B记录到记录仓库616。

[0100] 在图6中，应用程序602是针对执行这个应用程序的操作系统相同的操作系统编译的。此外，在另一实施例中，该应用程序可以利用插入库来执行，其中应用程序是针对与执行它的操作系统不同的操作系统编译的。图7是一个操作系统的用于插入应用程序的库调用的插入库的框图，其中该应用程序是针对另一操作系统编译的。在图7中，应用程序702是针对OS2编译的，但是在OS1环境700中加载并执行。在一个实施例中，应用程序702是应用程序602。此外，应用程序702使用系统库704来实现可以别的方式由OS2提供的库函数。系统库704可以是如参考图1及库102、104和106所述的一个或多个OS1服务、DLL和/或运行时库。系统库704还包括函数714A。函数714A可以实现OS1服务或者在DLL和/或运行时库中找到的函数。插入库706可以是如参考图3及库310、312和314所述的用于OS1服务、DLL和/或运行时库的一个或多个插入库。在一个实施例中，插入库706是利用与图6所述的插入库606相同的函数定义数据生成的。在该实施例中，插入库706是针对OS1环境700生成的，而插入库606是针对不同的OS环境生成的。插入库706还包括函数714B。函数714B是对应于函数714A的插入函数。在一个实施例中，函数714B是基于如图4和5中所述的函数规范文件生成的。

[0101] 在图7中，应用程序702对函数714A进行调用712A。在缺少插入库706的情况下，函数714A返回到应用程序702，包括任何经712B返回的参数。利用为系统库704插入的插入库706，OS1环境700将应用程序702对函数714A的调用经708A重定向到插入库706中的对应函数714B。函数714B将这种调用经708B记录到记录仓库716。在一个实施例中，函数714B记录进行调用的时间、库函数名称和/或传递到库函数714B的参数的名称和值。记录仓库716可以是本领域中用于存储函数调用记录的文件、存储器、数据库，等等。函数714B经708C调用函数714A。

[0102] 函数714A经710A返回到插入库706中的函数714B。在一个实施例中，函数714A将参数返回到函数714B。函数714B将任何附加的记录，例如函数714A的返回值、时间戳、返回参数等，经710B记录到记录仓库716。

[0103] 图6至7例示了插入对库的调用的插入库的两个不同实施例。图8是利用插入库插入从非调试应用程序对库的调用的方法800的一个实施例的流程图。在一个实施例中，非调试应用程序针对其编译的操作系统可以与执行该非调试应用程序的操作系统相同或者不同。此外，非调试应用程序和库不需要为了让插入库记录库函数调用而进行任何重新编译。因此，插入库可以在非调试应用程序和/或库的开发之前、期间或者之后开发。在图8中，在

块802,方法800利用插入库截取来自非调试应用程序的库函数调用。在一个实施例中,方法800利用插入库606(或者706)截取库函数调用,如参考图6(7)所描述的。

[0104] 在块802,方法800记录对库函数的调用。在一个实施例中,方法800利用插入库606(706)的函数614B(714B)将调用记录到记录仓库616(716),如参考图6(7)所描述的。在另一个实施例中,方法800记录库函数调用中的一个、一些或者全部。对哪些库函数调用要记录的控制是在运行时控制的,因此不需要对插入库或者非调试应用程序的重新编译。此外,对记录函数调用的控制可以是单个函数级别、API级别和/或库级别的。在块806,方法800利用插入库对库函数进行调用。在一个实施例中,方法800利用插入库函数614B(714B)调用库函数614A(714A),如参考图6(7)所描述的。在块808,方法800从被调用的库函数接收返回值。此外,方法800接收传递到被调用库函数的任何修改过的参数。

[0105] 在块810,方法800记录返回值以及插入库函数的退出。在一个实施例中,方法800记录所传递的参数、插入库退出的时间戳和关于插入库函数调用的其它信息,如参考图6(7)所描述的。在块812,方法800将返回值和任何修改过的参数传递回非调试应用程序。

[0106] 在图7中,基于OS2的应用程序702当在OS1环境700中执行时使用库704和706。在这个图中,应用程序702将参数传递到这些库,并接收回返回值和任何修改过的参数。一个可能的实施例是系统库704创建对象并将这些对象通过插入库706传递回应用程序702。在应用程序使用基于OS2的服务来创建当在OS1环境700中执行时提供给应用程序702的OS2对象的情况下,这种实施例会是有用的。例如,非微软WINDOWS的操作系统可以利用系统库704创建基于微软WINDOWS的COM对象。

[0107] 图9是为应用程序902创建对象的操作系统OS1的系统库904的框图,其中应用程序902是针对另一操作系统编译的。此外,应用程序902是针对OS2编译的,但是在OS1环境900中执行。在可选实施例中,应用程序902是基于OS1的应用程序。在这个图中,基于OS2的应用程序902是由OS1环境900加载并执行的。此外,应用程序902使用系统库904来实现将以别的方式由OS2提供的库函数。系统库904可以是如参考图1及库102、104和106所述的一个或多个OS1服务、DLL和/或运行时库。系统库904还包括函数914A。函数914A可以实现OS1服务或者在DLL和/或运行时库中找到的函数。插入库906可以是如参考图3及库310、312和314所述的用于OS1服务、DLL和/或运行时库的一个或多个插入库。插入库906还包括函数914B。函数914B是对应于函数914A的插入函数。在一个实施例中,函数914B是基于如图4和5中所述的函数规范文件生成的。

[0108] 在图9中,应用程序902对系统库904进行对象请求908A。对象请求可以针对本领域中已知的任何类型对象:COM、公用对象请求代理程序体系结构(CORBA)、JAVA BEANS,等等。函数914B接收对象请求908A,并将对象请求908B记录到记录仓库908。在一个实施例中,函数914B利用函数914B被调用的时间、函数914A的名称和/或传递到函数914A的参数的名称和值来记录对象请求。函数914B将对象请求908C传递到系统库904中的函数914A。响应于对象请求908C,函数914A创建所请求的对象。在一个实施例中,函数914A创建一般在除执行应用程序902的操作系统之外的另一操作系统中创建并使用的对象。例如并且作为例示,函数914A创建在非微软WINDOWS环境中执行的COM对象。函数914A经910A将所创建的对象返回到调用函数914B。函数914B经910B将所创建的对象和其它信息记录到记录仓库908。在一个实施例中,函数914B将所创建对象的字符串或者其它合适的表示记录到记录仓库908。在另一

个实施例中,函数914B将返回值、时间戳、修改后的参数等记录到记录仓库908。函数914B经910C将所创建的对象返回到应用程序902。

[0109] 图6和7例示了利用插入库来插入用于针对一个或另一个操作系统编译的应用程序的系统库函数调用的两个不同操作系统。在这些图中,插入库记录执行应用程序的库函数调用。在一个实施例中,不同的操作系统执行相同的应用程序。通过在不同的操作系统上执行相同的应用程序并且记录这些调用,开发者可以使用这个来调试、评估、开发等要在与用于编译该应用程序的操作系统不同的操作系统上执行该应用程序的一组库。在一个实施例中,调试、评估、开发等是通过比较图6和7中生成的日志来进行的。

[0110] 图10是方法1000的一个实施例的流程图,其中方法1000记录并比较用于在两个不同操作系统中执行的相同应用程序对操作系统库的库函数调用。在图10中,在块1002,方法1000在OS2环境中利用插入库执行基于OS2的应用程序。在一个实施例中,方法1000在OS2环境中利用插入库606执行基于OS2的应用程序602,如参考图6所描述的。在一个实施例中,方法1000在一系列可以重复的步骤中执行基于OS2的应用程序,而且,这一系列步骤也可以在块1006使用。在块1004,方法1000利用插入库记录对OS2服务、DLL和/或运行时库的函数调用。在一个实施例中,方法1000利用插入库606将这些函数调用记录到记录仓库616,如参考图6所描述的。

[0111] 在块1006,方法1000利用插入库在OS1环境中执行基于OS2的应用程序。在一个实施例中,方法1000在OS1环境中利用插入库706执行基于OS2的应用程序702,如参考图7所描述的。在一个实施例中,方法1000在与块1002相同的一系列步骤中执行基于OS2的应用程序,以便潜在地生成与块1004中类似的日志集合。在块1008,方法1000记录利用这些插入库对OS2服务、DLL和/或运行时库的函数调用。在一个实施例中,方法1000将利用插入库706的这些函数调用记录到记录仓库716,如参考图7所描述的。

[0112] 在块1010,方法1000比较在块1004和1008由插入库生成的基于OS1和OS2的日志。在一个实施例中,方法1000比较这两个日志,以确定日志是否有和哪里有分歧。这可以是一种调试工具,用以可视化应用程序跨两个操作系统所采取的路径,并通过观察两个日志的分歧来确定是否出现了错误。在一个实施例中,比较过程是利用日志分析器自动实现的。

[0113] 图11是例示在计算机1100上所使用的开发者工具包1104的一个实施例的框图,其中该工具包生成用于针对相同或不同操作系统编译的应用程序的插入库。在图11中,计算机1100包括OS1环境1102。OS1环境1102包括开发者工具包1104。开发者工具包1104包括规范文件创建模块1106、规范文件存储装置1108、规范文件预处理器1110、编译器/链接器1112和应用程序执行模块1114。规范文件创建模块1106创建函数规范文件,如参考图4的块402所描述的。在一个实施例中,规范文件创建模块1106是编辑器。规范文件存储模块1108存储函数规范文件,如参考图4的块404所描述的。规范文件预处理器1110预处理函数规范文件,以便产生用于插入库的头信息和函数代码,如在图4的块406和图5中所描述的。编译器/链接器1112编译插入库,如参考图4的块408所描述的。应用程序执行模块1114利用插入库执行应用程序,如参考图4的块410所描述的。

[0114] 图12是例示规范文件预处理器1110的一个实施例的框图,其中规范文件预处理器1110生成用于插入库的函数代码和头信息。在图12中,规范文件预处理器1110包括规范文件解析器1202、函数定义类型模块1204、函数运行时代码发生器1206、函数DLL代码发生器

1208、函数调用模板代码发生器1210、函数OS服务代码发生器1212和函数接口发生器1214。规范文件解析器1202解析函数规范文件定义,如参考图5的块506所描述的。函数定义类型模块1204确定函数定义的类型,如参考图5的块508、510、512、514和532所描述的。函数运行时代码发生器1206生成运行时头信息和函数代码,如参考图5的块520所描述的。函数DLL代码发生器1208生成DLL头信息和函数代码,如参考图5的块522所描述的。函数调用模板发生器1210利用函数调用模板定义生成函数头信息和函数代码,如参考图5的块516所描述的。函数OS服务代码发生器1212生成OS服务头信息和函数代码,如参考图5的块520所描述的。函数接口发生器1214生成对象与接口的头信息和函数代码,如参考图5的块534所描述的。

[0115] 图13是例示包括函数1308的插入库1300的一个实施例的框图。在一个实施例中,插入库1300可以是操作服务、DLL和/或运行时库插入的一个或多个库。每个函数1308都包括记录模块1302、调试模块1304和调用模块1306。在一个实施例中,记录模块1302利用关于时间戳、所传递的参数、返回值和/或修改过的参数的信息记录对特定函数1308的调用,如参考图6至9中一个或多个所描述的。调试模块1304输出调试信息,如参考图6至9中一个或多个所描述的。调用模块1306调用对应于操作系统服务、DLL和/或运行时库中函数1308的函数。

[0116] 在实践当中,在此所述的方法可以构成由机器可执行指令组成的一个或多个程序。参考图4、5、8和10中的流程图描述方法使得本领域技术人员能够开发这种程序,包括执行在适当配置的机器上由逻辑块所表示的操作(动作)的这种指令(机器的处理器执行来自机器可读介质的指令,可读介质例如RAM(例如,DRAM)、ROM、非易失性存储介质(例如,硬盘驱动器或者CD-ROM),等等)。机器可执行的指令可以用机器编程语言来写或者体现在固件逻辑或者硬件电路中。如果在编程语言中的编写遵循共识的标准,则这种指令可以在多种硬件平台上执行并用于接口到多种操作系统。此外,本发明不是参考任何特定的编程语言描述的。应当认识到,多种编程语言都可以用于实现在此所述的本发明的教义。此外,当采取动作或者造成结果时,在本领域中以一种或另一种形式来说软件是很普通的(例如,程序、过程、处理、应用程序、模块、逻辑,...)。这种表达仅仅是说由机器对软件的执行造成机器的处理器执行动作或者产生结果的简略方式。还应当认识到,在不背离本发明范围的情况下,更多或更少的处理可以结合到在流程图中所例示的方法中,而且在此所示和描述的块的布置不暗示任何特定次序。

[0117] 图14示出了通过网络1402(例如,互联网)耦接到一起的若干个计算机系统1400。如在此所使用的,术语“互联网”指使用某些协议的网络的联网,其中的协议例如TCP/IP协议及有可能其它协议,例如用于构成万维网(网络)的超文本标记语言(HTML)文档的超文本传输协议(HTTP)。互联网的物理连接及互联网的协议和通信过程是本领域技术人员众所周知的。对互联网1402的访问一般是由互联网服务提供商(ISP)提供的,例如ISP 1404和1406。例如客户端计算机系统1412、1416、1424和14212的客户端系统上的用户通过例如ISP 1404和1406的互联网服务提供商获得对互联网的访问。对互联网的访问允许客户端计算机系统的用户交换信息、接收和发送电子邮件并观看文档,例如以HTML格式准备的文档。这些文档常常是由网络服务器提供的,例如被认为是“在互联网上的”网络服务器1408。这些网络服务器常常是由诸如ISP 1404的ISP提供的,但是如本领域中众所周知的,计算机系统无需也是ISP就可以建立并连接到互联网。

[0118] 网络服务器1408一般是至少一个计算机系统,该计算机系统作为服务器计算机系统运行并配置成利用万维网的协议运行并耦接到互联网。可选地,网络服务器1408可以是为客户提供系统提供对互联网的访问的ISP的一部分。网络服务器1408示出为耦接到服务器计算机系统1410,其中服务器计算机系统1410本身又耦接到网络内容1412,其中网络内容1412被认为是一种形式的媒体数据库。应当认识到,尽管在图14中示出了两个计算机系统1408和1410,但网络服务器系统1408和服务器计算机系统1410可以是具有不同软件部件的一个计算机系统,其中不同的软件部件提供网络服务器功能性和由服务器计算机系统1410提供的服务器功能性,如以下将进一步描述的。

[0119] 利用合适的网络浏览软件,客户端计算机系统1412、1416、1424和14212每个都可以观看由网络服务器1408提供的HTML页面。通过可以看作客户端计算机系统1412一部分的调制解调器接口1414,ISP 1404提供到客户端计算机系统1412的互联网连接。客户端计算机系统可以是个人计算机系统、网络计算机、网络电视系统、手持式设备或者其它这种计算机系统。类似地,ISP 1406为客户端系统1416、1424和1426提供互联网连接,但是如在图12中示出的,连接对于这三个计算机系统是不同的。客户端计算机系统1416是通过调制解调器接口1418耦接的,而客户端计算机系统1424和1426是LAN的一部分。尽管图12将接口1414和1418总体上示为“调制解调器”,但是应当理解,这些接口中的每一个都可以是模拟调制解调器、ISDN调制解调器、线缆调制解调器、卫星发射接口或者用于将一个计算机系统耦接到其它计算机系统的其它接口。客户端计算机系统1424和1416通过网络接口1230和1232耦接到LAN 1422,其中网络接口1230和1232可以是以以太网或者其它网络接口。LAN 1422还可以耦接到网关计算机系统1420,其中网关计算机系统1420可以为局域网提供防火墙和其它互联网相关的服务。这种网关计算机系统1420耦接到ISP14012,以便提供到客户端计算机系统1424和14212的互联网连接。网关计算机系统1420可以是传统的服务器计算机系统。而且,网络服务器系统1408也可以是传统的服务器计算机系统。

[0120] 可选地,如众所周知的,服务器计算机系统1428可以通过网络接口1434直接耦接到LAN 1422,以便向客户端1424、1426提供文件12312和其它服务,而不需要通过网关系统1420连接到互联网。此外,客户端系统1412、1416、1424和1426的任意组合都可以利用LAN1422、互联网1402或者其组合作为通信介质在对等网络中连接到一起。总的来说,对等网络跨多台用于存储和检索的机器的网络分布数据,而不使用一个或多个中央服务器。因此,每个对等网络节点都可以结合上述客户端与服务器两者的功能。

[0121] 以下对图15的描述是要提供适于执行上述本发明的方法的计算机硬件和其它运行部件的概述,但不是要限制可以应用的环境。本领域技术人员将马上认识到,本发明的实施例可以利用其它计算机系统配置实践,包括机顶盒、手持式设备、消费者电子设备、多处理器系统、基于微处理器或者可编程消费者电子设备、网络PC、微型计算机、大型计算机,等等。本发明的实施例还可以在分布式计算环境中实践,其中任务是由通过诸如对等网络基础结构的通信网络链接的远程处理设备执行的。

[0122] 图15示出了可以在本发明的一个或多个方面中使用的传统计算机系统的一个例子。计算机系统1500通过调制解调器或者网络接口1502接口到外部系统。应当认识到,调制解调器或者网络接口1502可以被认为是计算机系统1500的一部分。这种接口1502可以是模拟调制解调器、ISDN调制解调器、线缆调制解调器、令牌环接口、卫星发射接口或者用于将

一个计算机系统耦接到其它计算机系统的其它接口。计算机系统1502包括处理单元1504，这可以是传统的微处理器，例如Intel的Pentium微处理器或者Motorola的Power PC微处理器。存储器1508通过总线1506耦接到处理器1504。存储器1508可以是动态随机存取存储器(DRAM)，而且还可以包括静态RAM(SRAM)。总线1506将处理器1504耦接到存储器1508，还耦接到非易失性存储装置1514、显示控制器1510和输入/输出(I/O)控制器1516。显示控制器1510以传统方式控制显示设备1512上的显示，其中显示设备可以是阴极射线管(CRT)或者液晶显示器(LCD)。输入/输出设备1518可以包括键盘、盘驱动器、打印机、扫描仪及其它输入和输出设备，包括鼠标或其它指向设备。显示控制器1510和I/O控制器1516可以利用传统的众所周知的技术来实现。数字图像输入设备1520可以是数码照相机，为了允许将数码照相机的图像输入到计算机系统1500，该数码照相机耦接到I/O控制器1516。非易失性存储装置1514常常是磁性硬盘、光盘或者用于大量数据的其它形式存储装置。这种数据中的一些常常是在计算机系统1500中软件的执行期间通过直接存储器存取处理写到存储器1508中的。本领域技术人员将马上认识到，术语“计算机可读介质”和“机器可读介质”包括可以被处理器1504或者被诸如蜂窝电话或个人数字助理或MP3播放器等的其它数据处理系统访问的任何类型存储设备。

[0123] 网络计算机是可以用于本发明实施例的另一类型的计算机系统。网络计算机通常不包括硬盘或者其它大容量存储装置，而且可执行程序是从网络连接加载到存储器1508中的，以便由处理器1504执行。根据本发明的实施例，本领域中已知的网络电视系统也被认为是一种计算机系统，但它可能缺少图15中所示的一些特征，例如某种输入或者输出设备。典型的计算机系统将通常至少包括处理器、存储器和将存储器耦接到处理器的总线。

[0124] 应当认识到，计算机系统1500是具有不同体系结构的许多可能计算机系统的一个例子。例如，基于Intel微处理器的个人计算机常常具有多条总线，其中一条可以是用于外围设备的输入/输出(I/O)总线，一条直接连接处理器1504和存储器1508(常常被称为存储器总线)。总线通过执行由于不同总线协议而导致的任何必要转换的桥部件连接到一起。

[0125] 还应当认识到，计算机系统1500是由操作系统软件控制的，其中操作系统软件包括作为操作系统软件一部分的文件管理系统，例如盘操作系统。具有其相关文件管理系统软件的操作系统的例子是称为来自位于加州Cupertino的Apple公司的MAC OS X的操作系统系列及其关联的文件管理系统。文件管理系统一般存储在非易失性存储装置1514中并且使处理器1504执行操作系统所需的各种动作，来输入和输出数据并在存储器中存储数据，包括在非易失性存储装置1514上存储文件。

[0126] 应当认识到，计算机系统1500可以是照相机、摄像机、扫描仪或者任何其它类型的图像获取系统。在一个实施例中，图像获取系统包括透镜、图像传感器或者一般与照相机、摄像机或其它类型图像获取系统关联的其它硬件。

[0127] 在以上说明中，本发明是参考其特定的示例实施例描述的。很显然，在不背离如以下权利要求中所述的本发明更广泛主旨与范围的情况下，可以对其进行各种修改。因此，应当从例示的角度而不是约束的角度来看待说明书与附图。

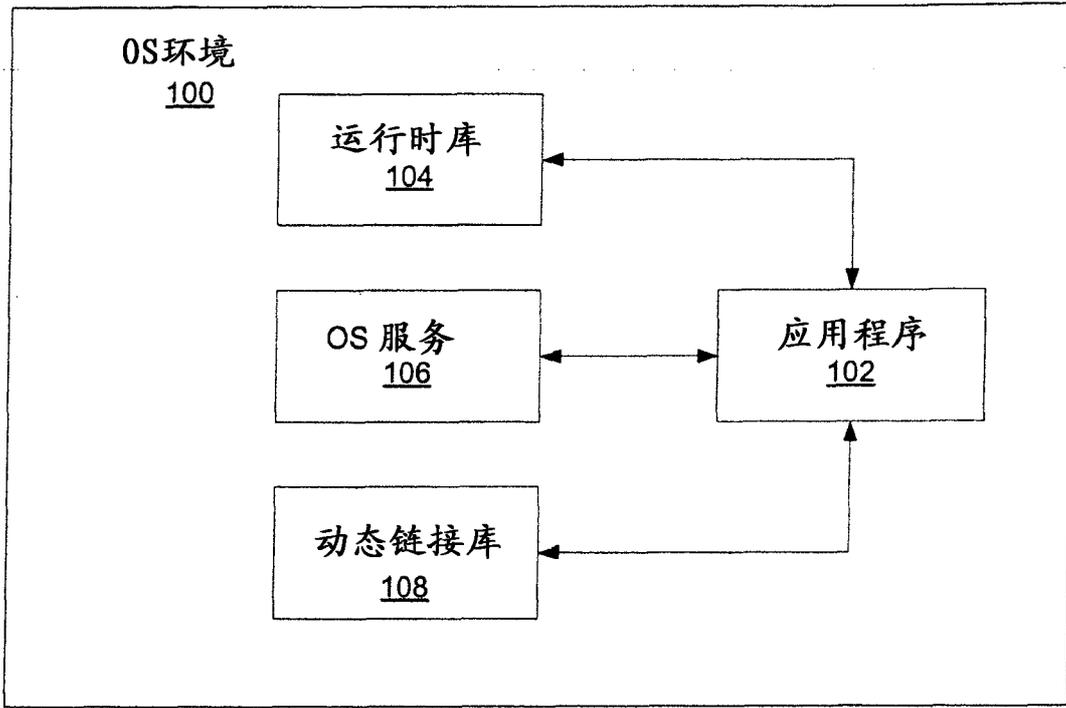


图1(现有技术)

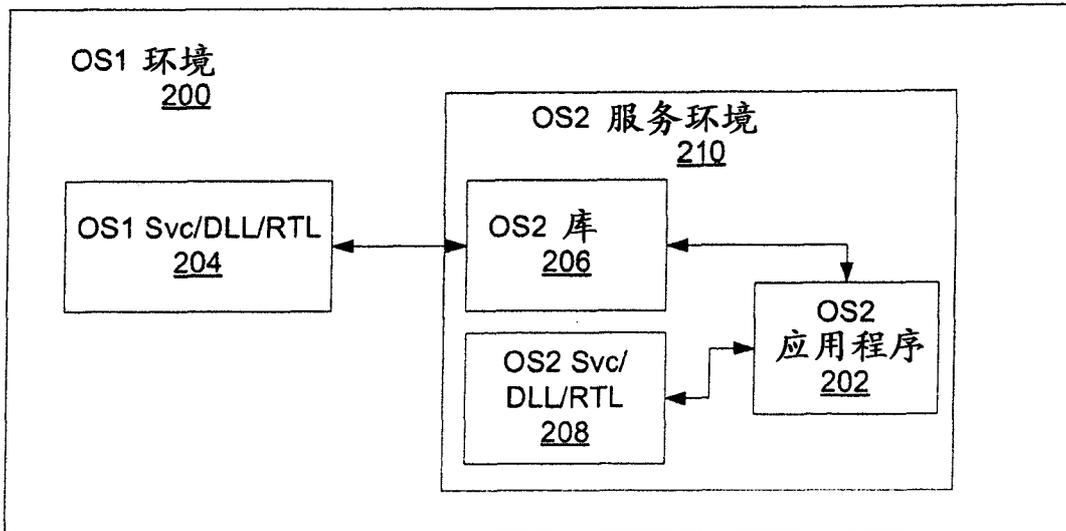


图2(现有技术)

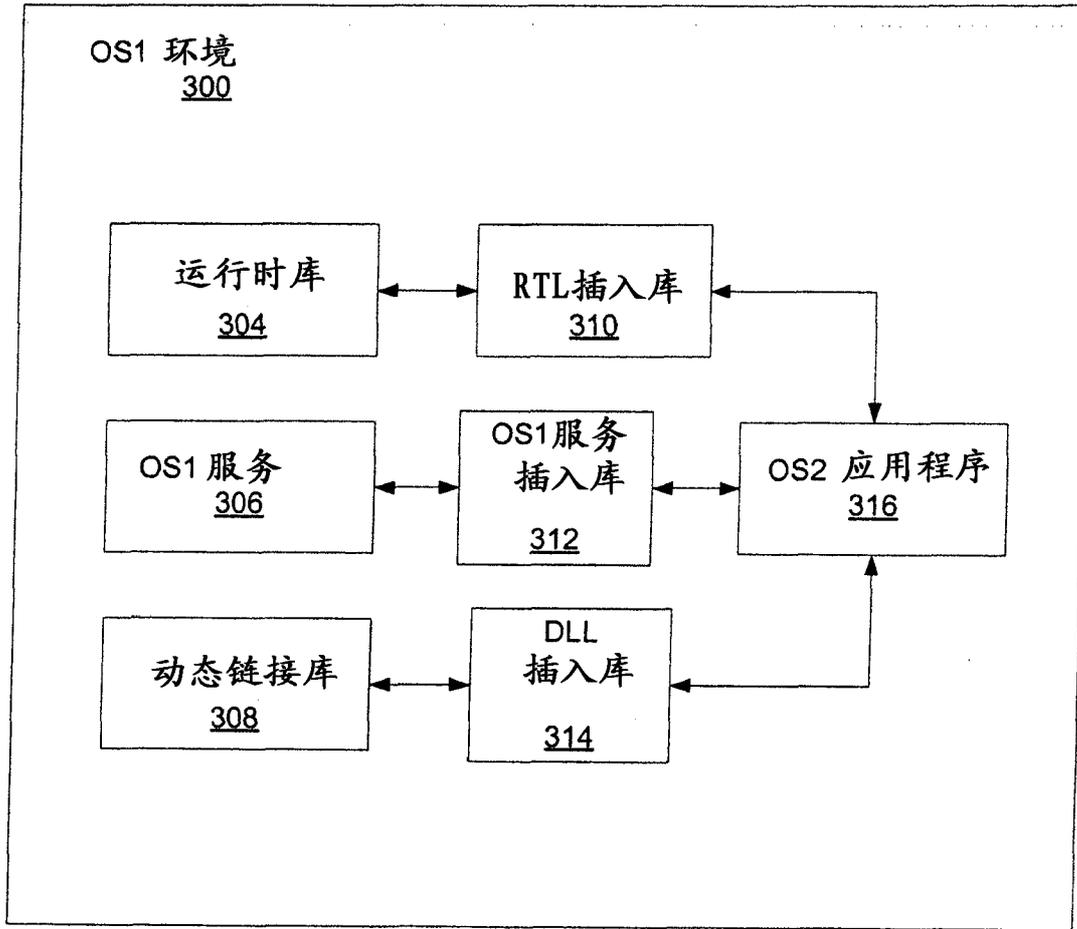


图3

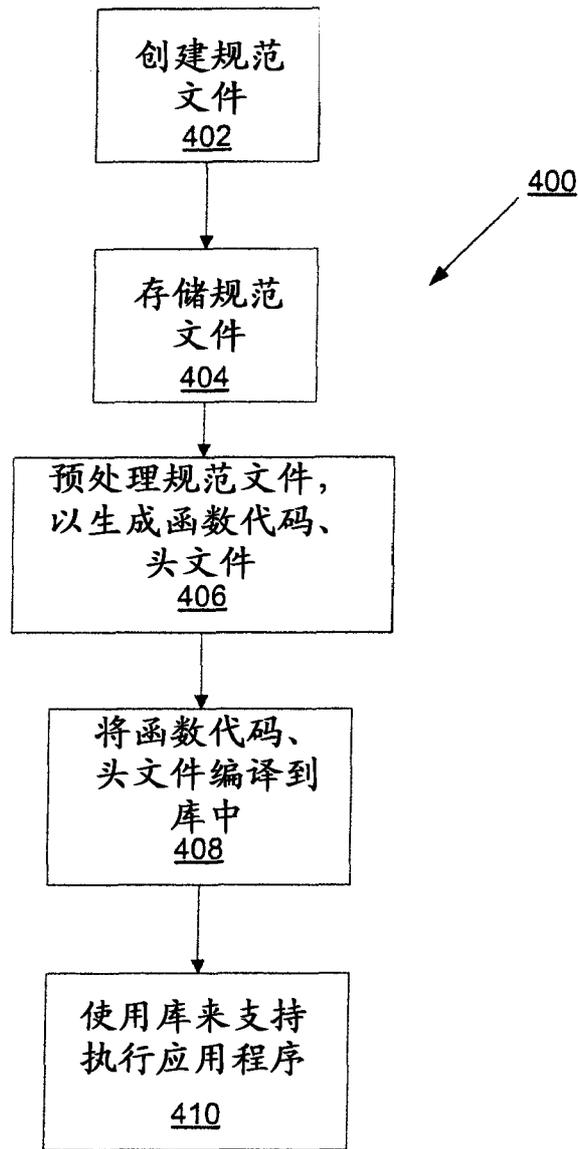


图4

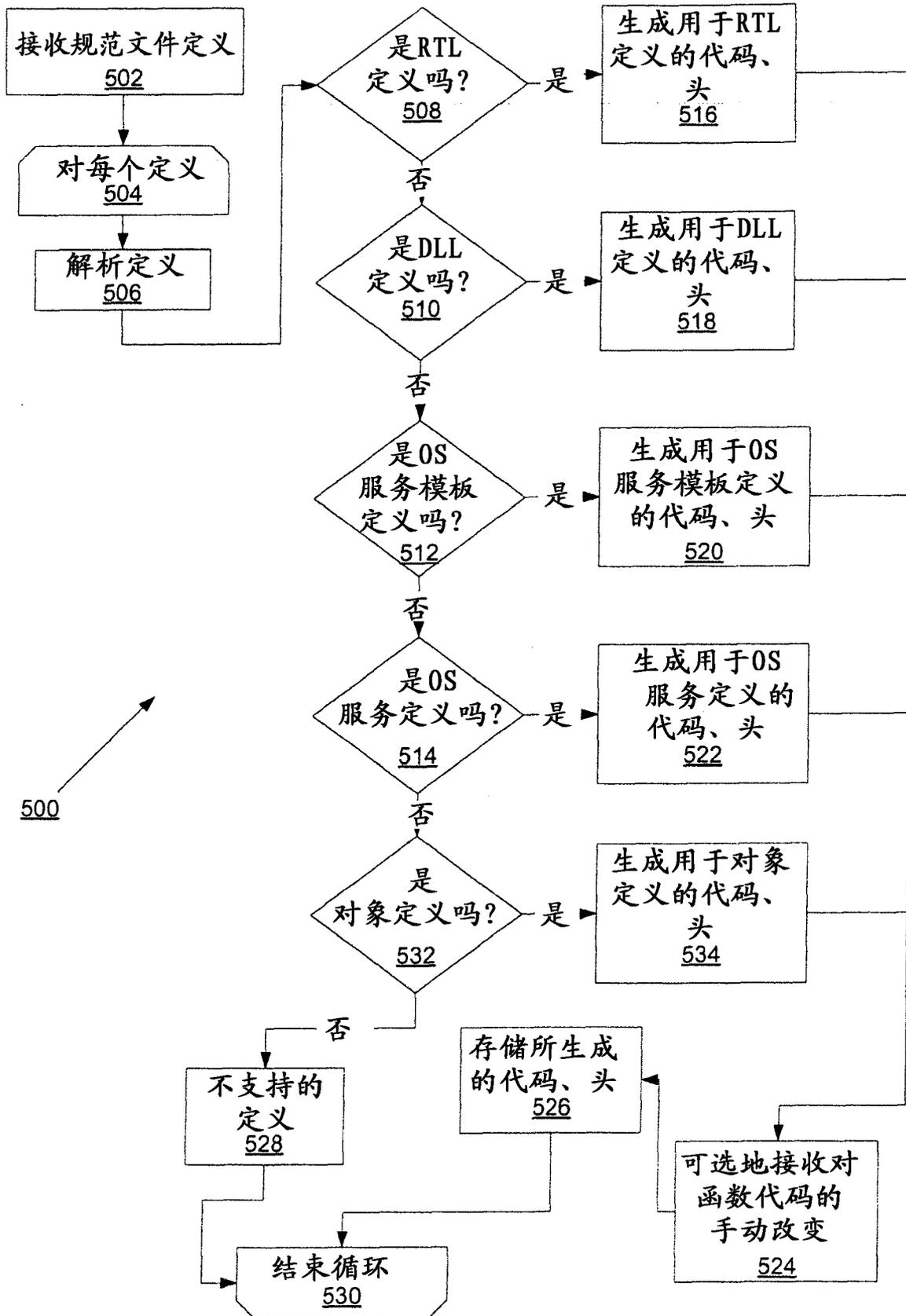


图5

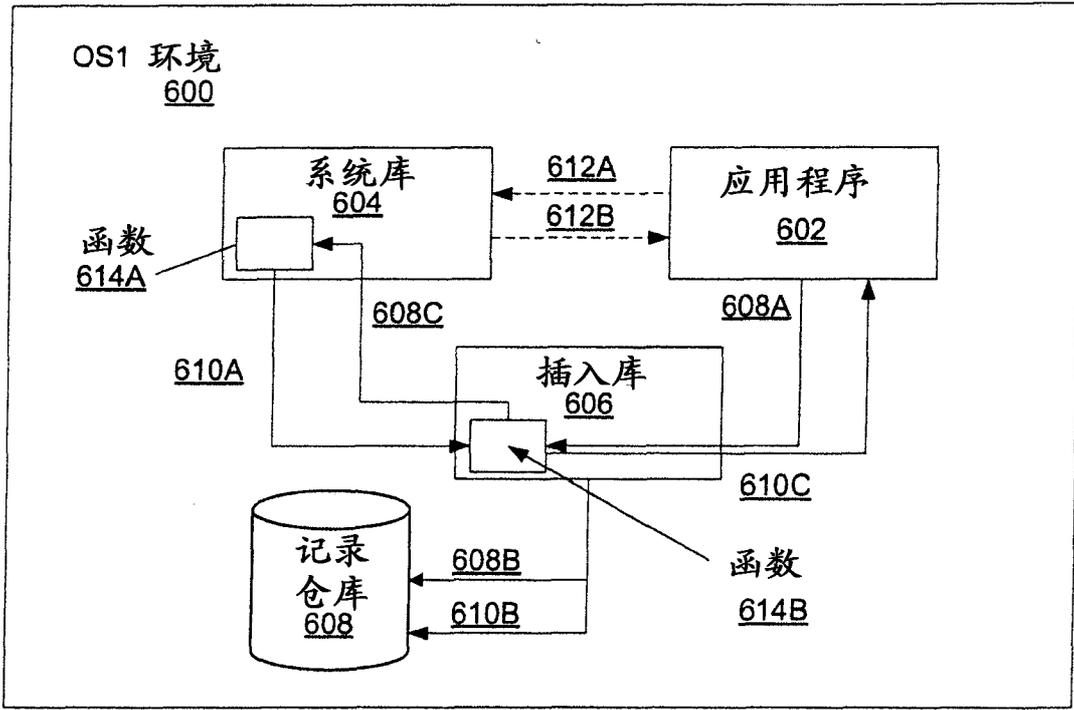


图6

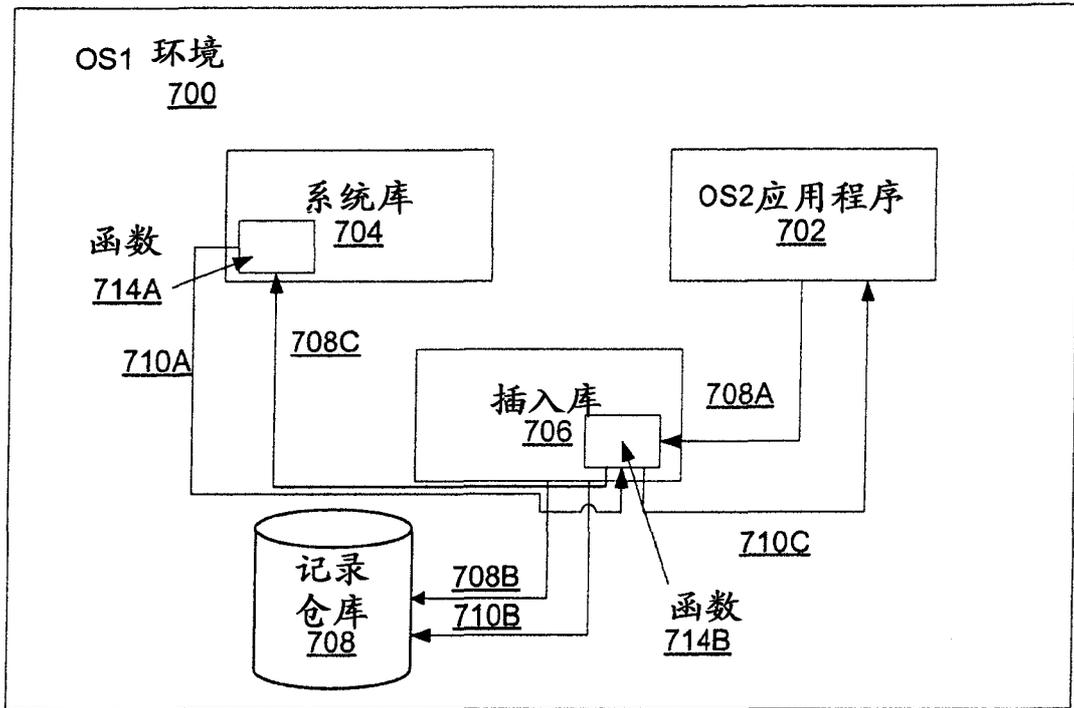


图7

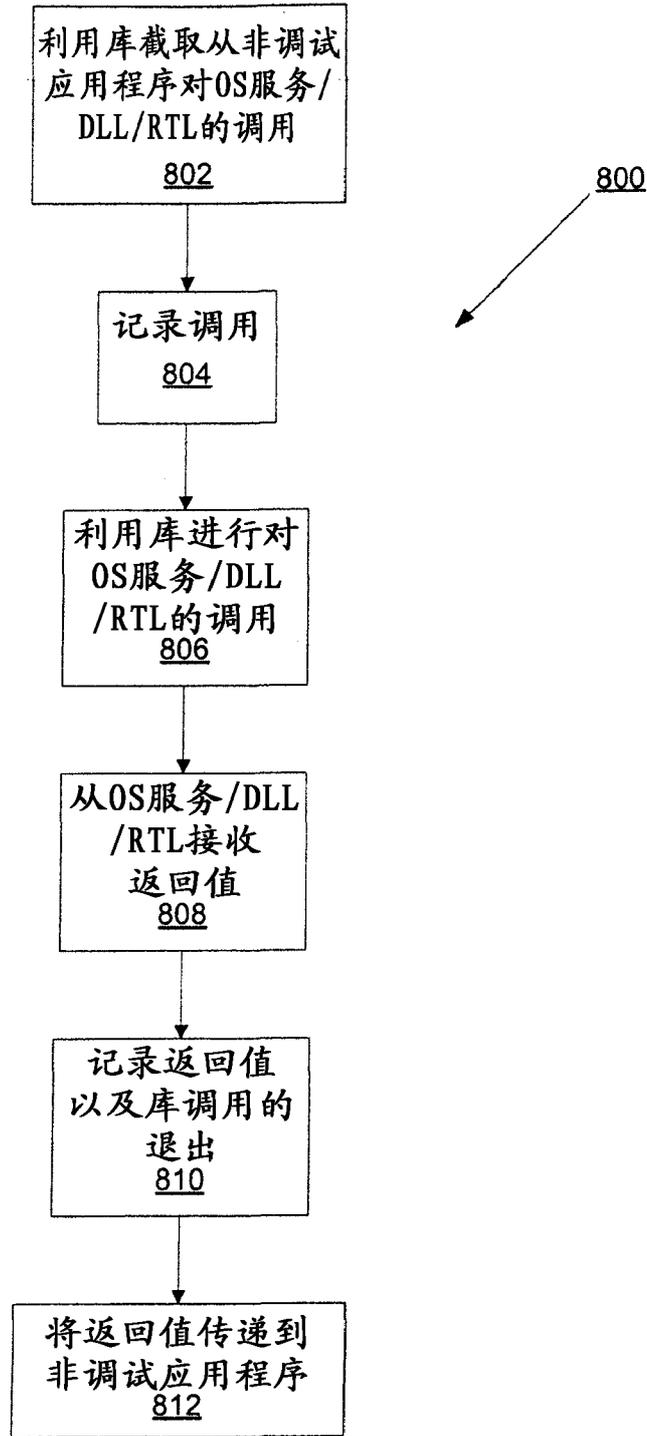


图8

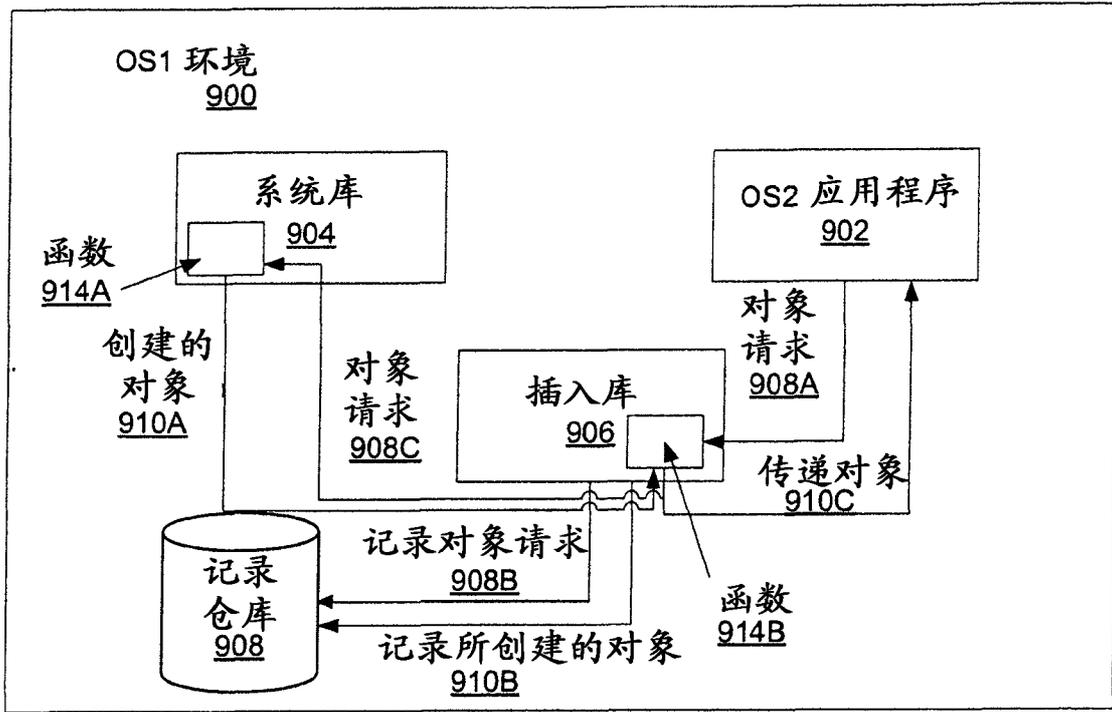


图9

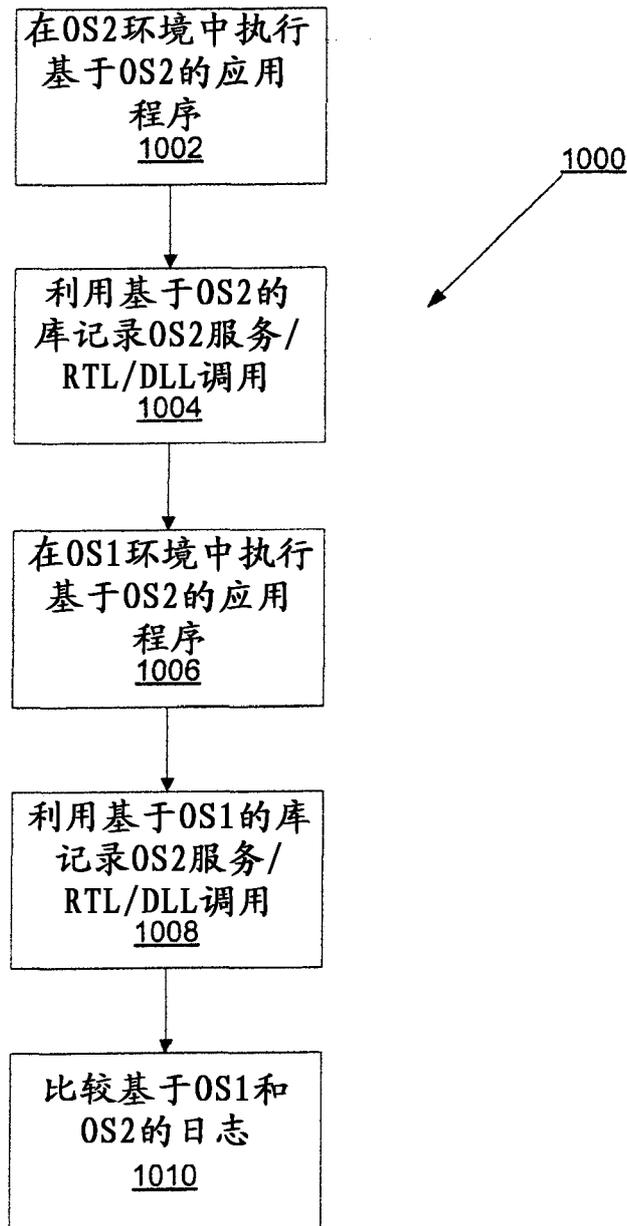


图10

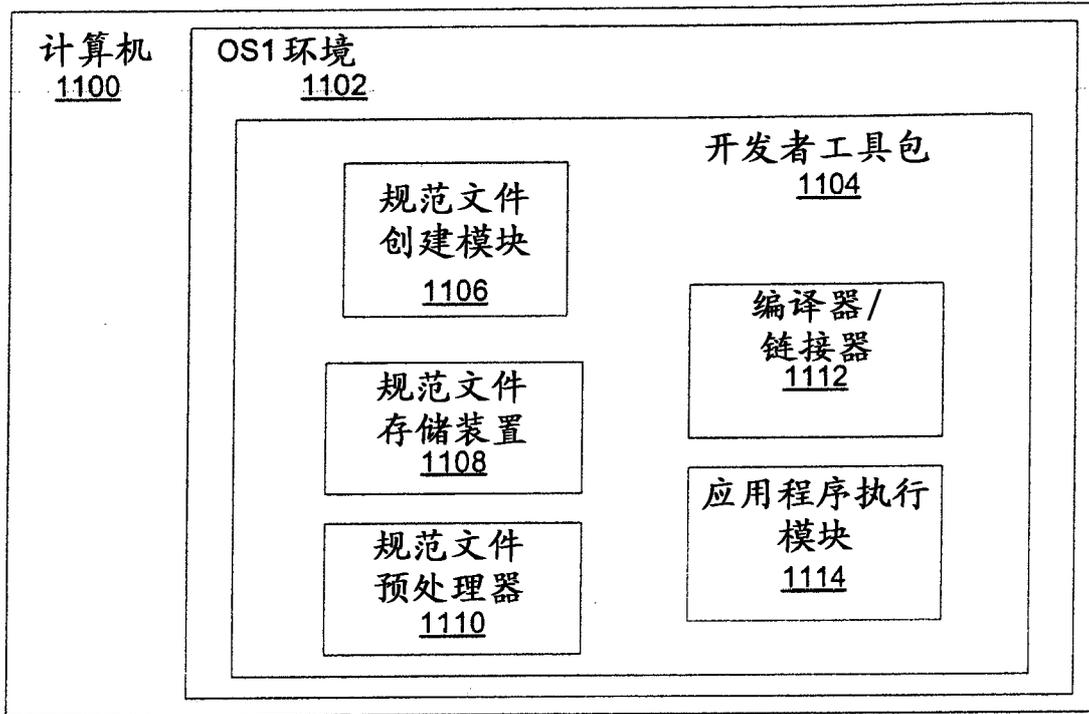


图11

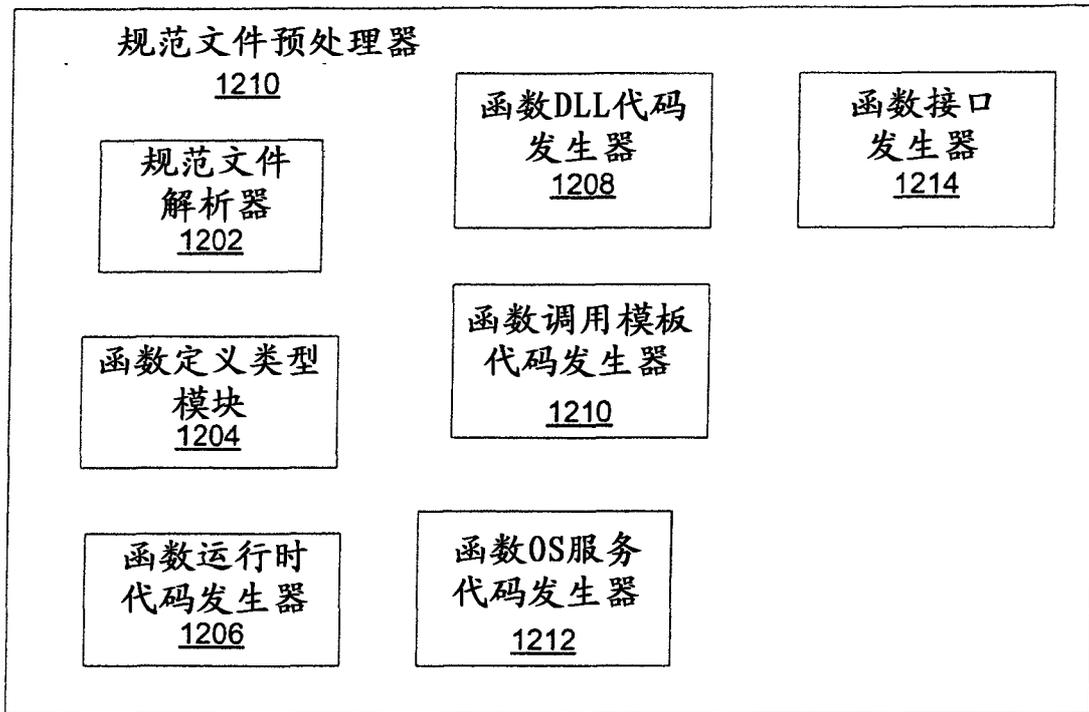


图12

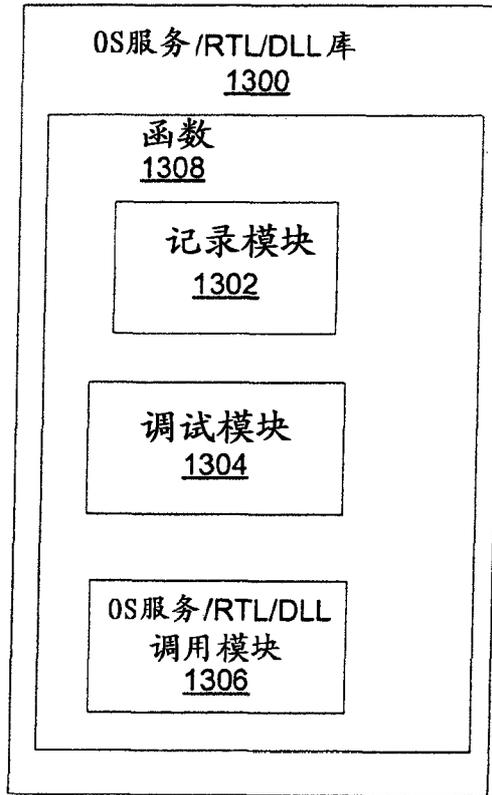


图13

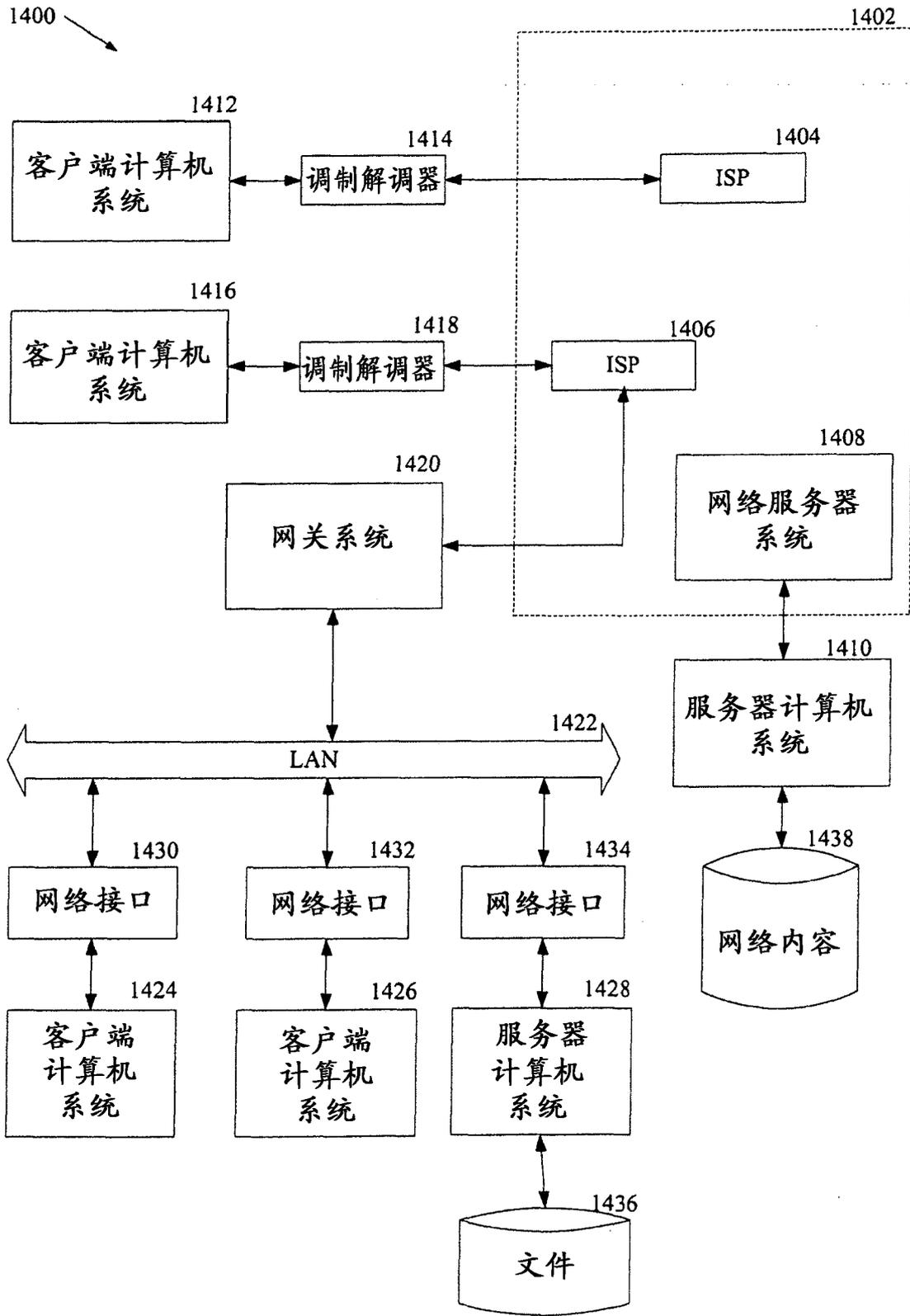


图14

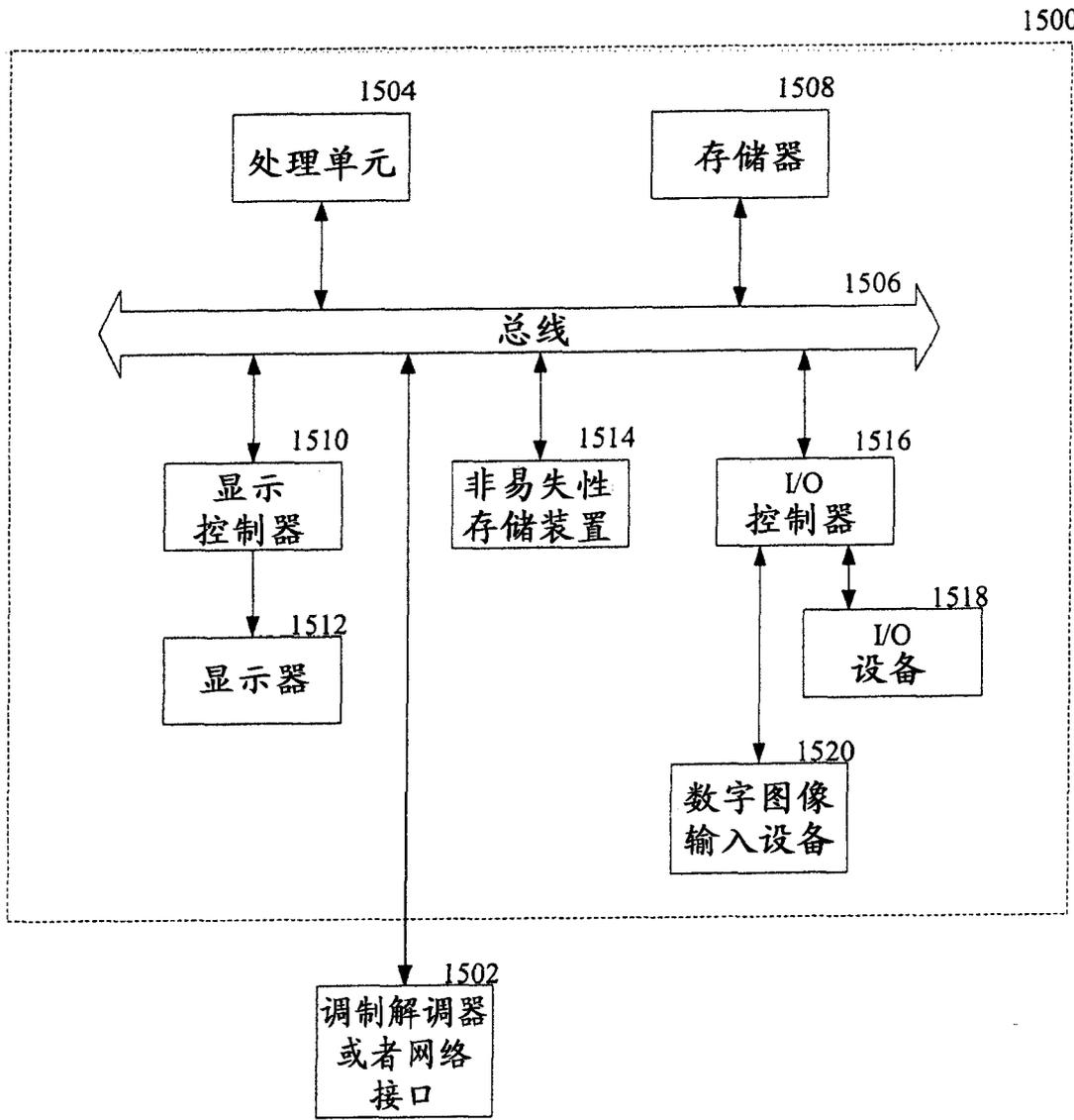


图15