[19] 中华人民共和国国家知识产权局

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

[12] 发明专利申请公开说明书

[21] 申请号 200510070030.0

[43] 公开日 2005 年 11 月 23 日

[11] 公开号 CN 1700171A

[22] 申请日 2005.4.29
[21] 申请号 200510070030.0
[30] 优先权
[32] 2004. 4.30 [33] US [31] 10/837,103
[71] 申请人 微软公司
地址 美国华盛顿州
[72] 发明人 E·P·沃伯 T·罗德
U·埃林松

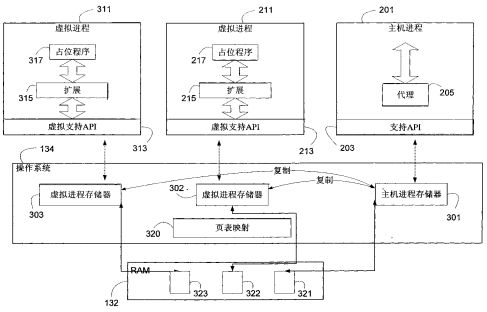
[74] 专利代理机构 上海专利商标事务所有限公司
代理人 张政权

权利要求书 6 页 说明书 35 页 附图 7 页

[54] 发明名称 提供从虚拟环境对硬件的直接访问

[57] 摘要

扩展或其它软件应用程序能够从虚拟机器环境内直接访问硬件。 硬件的物理地址可被映射到虚拟机器环境的进程空间。 类似地，可允许 I/O 端口通过到虚拟环境中。 虚拟机器可检测即将到来的直接存储器访问(DMA)，并可为 DMA 提供正确的地址，而同时可锁定必要的存储器。 如果当硬件中断到达时虚拟机器正在执行，则它可仿真其进程内的中断线。 因此，如果主机操作系统正在执行，则它可禁用中断并跟踪临时中断，随后将控制传递到虚拟机器进程，仿真临时中断并重新启用中断。 或者，主机操作系统可立即传递控制，或者它可执行其自己的进程内的中断服务例程。



ISSN1008-4274

1．一种具有计算机可执行指令的计算机可读介质，所述计算机可执行指令用于为虚拟环境中的扩展提供对硬件元件的直接访问，其中，所述虚拟环境使用一包括所述硬件元件的计算环境，所述计算机可读介质包括计算机可执行指令，用于：

检测与所述硬件元件相关联的存储器映射的输入/输出通信；

确定对应于所述存储器映射的输入/输出通信的所述硬件元件的一个或多个物理存储器地址；以及

修改一存储器映射，使得虚拟环境存储器的一个段被映射到所确定的一个或多个物理存储器地址。

2．如权利要求 1 所述的计算机可读介质，其特征在于，还包括计算机可执行指令，用于：在完成所述存储器映射的输入/输出通信之后，不修改所述存储器映射。

3．如权利要求 1 所述的计算机可读介质，其特征在于，还包括计算机可执行指令，用于：对所述存储器映射的输入/输出通信，确定是否应当向所述扩展授予对所述硬件元件的直接访问权限；其中，仅当应当向所述扩展授予对所述硬件元件的直接访问权限时，才执行所述存储器映射的修改。

4．如权利要求 1 所述的计算机可读介质，其特征在于，所述扩展是用于所述硬件元件的设备驱动程序。

5．如权利要求 1 所述的计算机可读介质，其特征在于，所述虚拟环境由在所述计算环境内执行的一虚拟机器应用程序创建。

6．一种具有计算机可执行指令的计算机可读介质，所述计算机可执行指令用于为一虚拟环境中的扩展提供对硬件元件的直接访问，其中，所述虚拟环境使用一包括所述硬件元件的计算环境，所述计算机可读介质包括计算机可执行指令，用于：

检测引用输入/输出端口的输入/输出通信；

修改所述虚拟环境和所述计算环境之间的接口，以使如果所述输入/输出端口与所述硬件元件相关联，能够从所述虚拟环境内访问所述输入/输出端口；以及

提供所述虚拟环境内一仿真的输入/输出端口，并且如果所述输入/输出端口不与所述硬件元件相关联，则通过所述接口访问所述输入/输出端口。

7．如权利要求 6 所述的计算机可读介质，其特征在于，用于修改所述接口的

计算机可执行指令包括计算机可执行指令，用于：修改一输入/输出保护位图，以避免由于与所述输入/输出端口相关联的指令而引起的俘获。

8．如权利要求 6 所述的计算机可读介质，其特征在于，所述输入/输出通信由所述虚拟环境中的扩展作出。

9．如权利要求 6 所述的计算机可读介质，其特征在于，所述输入/输出通信由所述接口代表所述扩展作出。

10．如权利要求 6 所述的计算机可读介质，其特征在于，所述扩展是用于所述硬件元件的设备驱动程序。

11．如权利要求 6 所述的计算机可读介质，其特征在于，所述虚拟环境由在所述计算环境中执行的一虚拟机器应用程序创建。

12．一种具有计算机可执行指令的计算机可读介质，所述计算机可执行指令用于为一虚拟环境中的扩展提供对硬件元件的直接访问，其中，所述虚拟环境使用一包括所述硬件元件的计算环境，所述计算机可读介质包括计算机可执行指令，用于：

检测所述虚拟环境中指示所述扩展的直接存储器访问的一个或多个事件，其中，所述直接存储器访问与所述硬件元件相关联；

确定与所述直接存储器访问相关联的适当的物理直接存储器访问地址；

在所述扩展的直接存储器访问之前，向所述虚拟环境中的所述扩展提供所述适当的物理直接存储器访问地址；

锁定所述计算环境中对应于所述适当物理直接存储器访问地址的物理存储器；以及

锁定所述虚拟环境中对应于所述适当物理直接存储器访问地址的虚拟存储器。

13．如权利要求 12 所述的计算机可读介质，其特征在于，用于检测所述一个或多个事件的计算机可执行指令包括计算机可执行指令，用于：

通过由所述虚拟环境展示的支持 API，标识提供给所述扩展的一个或多个功能，其中，所述一个或多个功能与所述扩展的直接存储器访问相关联；以及

检测所述扩展对所述一个或多个标识的功能的访问。

14．如权利要求 13 所述的计算机可读介质，其特征在于，用于检测所述扩展对所述一个或多个标识的功能的访问的计算机可执行指令包括计算机可执行指令，用于：修改所述一个或多个标识的功能以包括一个或多个非法指令，其中，当访问

所述一个或多个非法指令时生成俘获，所述俘获便于检测所述扩展对所述一个或多个标识的功能的访问。

15．如权利要求12所述的计算机可读介质，其特征在于，用于确定所述适当物理直接存储器访问地址的计算机可执行指令包括计算机可执行指令，用于：

确定与所述直接存储器访问相关联的物理直接存储器访问地址是否是受保护的地址；以及

确定作为与所述直接存储器访问相关联的物理直接存储器访问地址的不同地址的所述适当物理直接存储器访问地址为受保护的地址。

16．如权利要求12所述的计算机可读介质，其特征在于，还包括计算机可执行指令，用于：

在所述扩展的直接存储器访问完成之后，解锁所述计算环境中对应于所述适当物理直接存储器访问地址的物理存储器；以及

在所述扩展的直接存储器访问完成之后，解锁所述虚拟环境中对应于所述适当物理直接存储器访问地址的虚拟存储器。

17．如权利要求12所述的计算机可读介质，其特征在于，所述扩展是用于所述硬件元件的设备驱动程序。

18．如权利要求12所述的计算机可读介质，其特征在于，所述虚拟环境由在所述计算环境中执行的一虚拟机器应用程序创建。

19．一种具有计算机可执行指令的计算机可读介质，所述计算机可执行指令用于向一虚拟环境中的扩展提供对硬件元件的直接访问，其中，所述虚拟环境使用一包括所述硬件元件和处理器的计算环境，所述计算机可读介质包括计算机可执行指令，用于：

当所述虚拟环境在所述处理器上执行时，接收来自所述硬件元件的中断；

确定所述中断是否要由所述扩展处理；以及

如果所述中断要由所述扩展处理，则在所述虚拟环境中一仿真的物理中断线和所述计算环境中一物理中断线之间转换，所述物理中断线与所述中断相关联。

20．如权利要求19所述的计算机可读介质，其特征在于，用于确定的所述计算机可执行指令包括用于引用指示哪些中断可由所述扩展处理的中断表的计算机可执行指令。

21．如权利要求19所述的计算机可读介质，其特征在于，还包括计算机可执行指令，用于：在所述确定和转换之前，完成一指令仿真，其中，所述指令仿真在

接收所述中断之前启动。

22．如权利要求19所述的计算机可读介质，其特征在于，所述计算环境还包括至少一个额外处理器，所述计算机可读介质还包括计算机可执行指令，用于：如果所述中断要由所述扩展处理，且所述中断由所述至少一个额外处理器中的一个接收，则执行从所述至少额外处理器中的一个到所述处理器的处理器间中断。

23．如权利要求19所述的计算机可读介质，其特征在于，所述计算环境还包括至少一个额外处理器以及一中断控制电路，其中，所述中断控制电路被设计成中断到所述处理器而非所述至少一个额外处理器中的任一个。

24．如权利要求19所述的计算机可读介质，其特征在于，所述扩展是用于所述硬件元件的设备驱动程序。

25．如权利要求19所述的计算机可读介质，其特征在于，所述虚拟环境由在所述计算环境中执行的一虚拟机器应用程序创建。

26．一种具有计算机可执行指令的计算机可读介质，所述计算机可执行指令用于向一虚拟环境中的扩展提供对硬件元件的直接访问，其中，所述虚拟环境使用一包括所述硬件元件和处理器的计算环境，所述计算机可读介质包括计算机可执行指令，用于：

当所述虚拟环境不在所述处理器上执行时，接收来自所述硬件元件的中断；

确定所述中断是否要由所述扩展处理；以及

如果所述中断要由所述扩展处理，促使所述扩展的中断服务处理功能执行。

27．如权利要求26所述的计算机可读介质，其特征在于，用于促使所述扩展的中断服务处理功能执行的计算机可执行指令包括计算机可执行指令，用于：

禁用中断；

在传递控制之前完成一个或多个指令的执行；以及

传递控制使得所述虚拟环境在所述处理器上执行。

28．如权利要求27所述的计算机可读介质，其特征在于，用于促使所述扩展的中断服务处理功能执行的计算机可执行指令还包括计算机可执行指令，用于：

高速缓存要由所述扩展处理的瞬时中断，所述瞬时中断在所述禁用之后接收；

在传递控制之后仿真高速缓存的瞬时中断；以及

在所述仿真之后重新启用中断。

29．如权利要求26所述的计算机可读介质，其特征在于，用于促使所述扩展的中断服务处理功能执行的计算机可执行指令包括计算机可执行指令，用于：传递

控制，使得所述虚拟环境在所述处理器上执行，而不在接收所述中断时完成被执行的一个或多个指令的执行。

30．如权利要求26所述的计算机可读介质，其特征在于，用于促使所述扩展的中断服务处理功能执行的计算机可执行指令包括计算机可执行指令，用于：

传输控制，使得所述虚拟环境在所述处理器上执行；

仿真所述虚拟环境中的至少两个处理器，其中，所述至少两个仿真的处理器的至少一个被保留用于处理中断；以及

使用所述至少一个被保留的仿真的处理器来执行所述扩展的中断服务处理功能。

31．如权利要求26所述的计算机可读介质，其特征在于，用于促使所述扩展的中断服务处理功能执行的计算机可执行指令包括计算机可执行指令，用于：

将提供所述扩展的中断服务处理功能的计算机可执行指令复制到一当前执行的操作系统进程中，其中，当接收所述中断时，所述当前执行的操作系统进程在所述处理器上执行；

修改所复制的计算机可执行指令使用的一个或多个引用，以引用与所述虚拟环境相关联的存储器位置；

在所复制的计算机可执行指令中插入一个或多个计算机可执行指令，其中，所述一个或多个插入的计算机可执行指令监视或修改所复制的计算机可执行指令；以及

在所述当前执行的操作系统进程内，用所述一个或多个修改的存储器引用以及所述一个或多个插入的计算机可执行指令执行所复制的计算机可执行指令。

32．如权利要求31所述的计算机可读介质，其特征在于，所述修改一个或多个存储器引用和所述插入一个或多个计算机可执行指令提供了软件故障隔离。

33．如权利要求26所述的计算机可读介质，其特征在于，所述计算环境还包括至少一个额外处理器，所述计算机可读介质还包括计算机可执行指令，用于：如果所述中断要由所述扩展处理，且如果所述中断是由所述至少一个额外处理器中的一个接收的，则执行从所述至少一个额外处理器中的一个到所述处理器的处理器间中断。

34．如权利要求26所述的计算机可读介质，其特征在于，所述计算环境还包括至少一个额外处理器和一中断控制电路，其中，所述中断控制电路被设计成将所述中断发送到所述处理器而非所述至少一个额外处理器的任一个。

35．如权利要求26所述的计算机可读介质，其特征在于，所述扩展是用于所述硬件元件的设备驱动程序。

36．如权利要求26所述的计算机可读介质，其特征在于，所述虚拟环境由在所述计算环境中执行的一虚拟机器应用程序创建。

提供从虚拟环境对硬件的直接访问

相关申请

本申请涉及与本申请同一日提交的、代理案卷号为225654、名为“VEX－虚拟扩展框架（VEX-Vitual Extension Framework）”的共同提交的待决美国专利。

技术领域

本发明一般涉及虚拟机器，尤其涉及用于向虚拟机器环境内执行的扩展和其它软件应用程序提供对连接到底层主机计算装置的硬件设备的直接访问的系统和方法。

背景技术

随着计算硬件的性能的增加，虚拟机器技术成为对额外的硬件购买的一种可行且节省成本的替换方案。一般而言，虚拟机器可以是当在相同或不同硬件和软件上运行时试图仿真一种类型的硬件或软件环境的代码的集合。当计算机用户期望访问对其特定的硬件或软件配置可能不可用的软件或其它资源时，虚拟机器尤其有用。例如，在一种类型的计算硬件和操作系统上执行的虚拟机器可以仿真诸如在具有不同类型的硬件和操作系统的计算装置上找到的环境。因此，这类虚拟机器可允许第一种类型的硬件和操作系统的用户利用为第二种类型的硬件和操作系统创作的软件应用程序等，而不需要购买任何额外的硬件。

虚拟机器也可以对跨平台解决方案或向后兼容软件的开发是有用的。例如，使用最近的硬件和软件的软件开发者可通过仅执行虚拟机器并创建对应于现有硬件或软件的虚拟环境，在任何现有硬件和软件上测试其代码。类似地，诸如网站等需要跨平台兼容性的材料的开发者可通过执行虚拟机器并创建对应于为其设计浏览器的平台的虚拟环境，经由为各种平台所设计的web浏览器来测试网站。

一般而言，虚拟机器通过一代码集合执行硬件和软件抽象，该代码集合通常被成为“系统管理程序（hypervisor）”。系统管理程序可将来自虚拟机器环境的请求和执行命令转换成用于在其上执行虚拟机器应用程序的物理计算环境的适当

请求和命令。一般而言，这一转换可利用系统管理程序所执行的各种抽象。例如，系统管理程序可将许多不同的物理音频接口抽象成可向虚拟环境中的软件呈现的单个一般音频接口。虚拟环境中的软件然后可使用该一般音频接口，并且系统管理程序可在对一般音频接口的请求与硬件专用请求之间进行转换，硬件专用请求被发送到特定的底层物理音频接口，这些音频接口碰巧被连接到当前在其上执行虚拟机器的主机计算装置。

不幸的是，由于虚拟机器环境使用仿真和抽象的硬件，它可能无法主宿（host）与专有的、非寻常的或传统的硬件接口的扩展或软件。例如，现代的操作系统可能不再与用于传统设备的设备驱动程序兼容，这些传统设备诸如实验室设备、机器人接口以及可能不被经常更新的类似设备。在这一情况下，用户可能在虚拟机器环境中试图使用设备驱动程序用于这些传统设备。然而，由于虚拟环境依赖于所仿真的硬件，虚拟环境中的设备驱动程序可能无法与传统硬件正确地通信。类似地，非寻常硬件可能不能由系统管理程序正确地抽象，这仅仅是因为可能没有足够的需求来证明对这一抽象的尝试是正当的。因此，这类非寻常硬件的用户可能无法依赖于虚拟机器的便利性。

此外，由于系统管理程序仿真并抽象了硬件，因此对虚拟机器技术的作者和开发者有这样的负担，即他们要不断地仿真和抽象越来越多领域的硬件，以允许其虚拟机器尽可能地与现有的硬件兼容。这一负担经常分散了对诸如针对改进性能或减少编程错误等更重要虚拟机器技术的开发的注意力。因此，希望创建一种虚拟机器环境，它允许扩展或其它软件应用程序能够直接与在其上执行虚拟机器的底层硬件进行通信。

发明内容

本发明的实施例允许虚拟机器环境中的扩展和其它软件应用程序直接访问连接到主机计算装置的一个或多个硬件设备。

在一个实施例中，系统管理程序或底层硬件可将硬件设备的物理地址映射到虚拟机器进程，以使运行在虚拟机器进程中的扩展和其它软件应用程序能够直接访问硬件设备。

在另一实施例中，系统管理程序或底层硬件可修改诸如 I/O 保护位图等结构，以允许一个或多个 I/O 端口能够在虚拟环境中被正确地呈现，从而允许运行在虚拟机器进程中的扩展和其它软件应用程序能够向连接到硬件设备的物理 I/O 端口发

送 I/O 命令。

在又一实施例中，系统管理程序、虚拟操作系统或底层硬件可监视运行在虚拟机器进程中的扩展或其它软件作出的功能调用，以检测即将到来的直接存储器访问（DMA）。在检测到即将到来的 DMA 之后，系统管理程序或虚拟操作系统可以用即使从虚拟机器环境内也使用了正确的 DMA 地址的方式修改 DMA。要使用的物理存储器也可固定住（pin）以避免存储器冲突。

在再一实施例中，系统管理程序可通过在物理硬件中断线和虚拟机器环境中的硬件中断线之间进行转换，将硬件中断传递到虚拟机器环境中。如果当中断到达时主机操作系统进程正在执行，则它可禁用中断，并跟踪瞬时中断，以在将控制传递到虚拟机器进程之前完成一个或多个任务，在控制被传递到虚拟机器进程时，可仿真瞬时中断并可重新启用中断。或者，主机操作系统可立即将控制传递到虚拟机器进程，它可仿真一多 CPU 系统以具有可接收中断而没有延迟的至少一个 CPU。另一替换方案是主机操作系统将中断服务代码从虚拟机器进程中复制出来，并在主机操作系统进程上执行它，同时使用已知的软件故障隔离技术将存储器指针返回到虚拟机器进程。在具有多个物理 CPU 的计算系统中，中断可通过硬件被定向到在其上执行虚拟机器环境的物理 CPU。

当参考附图阅读以下说明性实施例的详细描述时，可以清楚本发明的其它特征和优点。

附图说明

尽管所附权利要求书以细节陈述了本发明的特征，然而，结合附图阅读以下详细描述，可以最好地理解本发明及其目的和优点，附图中：

图 1 是概括地示出了其中可实现本发明的各实施例的示例性设备体系结构的框图；

图 2 是概括地示出了依照本发明的各实施例用于隔离扩展的示例性环境的框图；

图 3 是概括地示出了依照本发明的一个实施例对用户模式环境的访问的框图；

图 4 是概括地示出了依照本发明的一个实施例对用户模式环境的另一访问的框图；

图 5 是概括地示出了依照本发明的一个实施例的相干状态的创建的流程图；

图 6 是概括地示出了依照本发明的一个实施例的相干状态的另一创建的流程

图；以及

图 7 是概括地示出了依照本发明的一个实施例用于向主宿在虚拟机器中的扩展提供对物理硬件的直接访问的示例性环境的框图。

具体实施方式

许多软件应用程序和操作系统依赖于扩展来向终端用户提供额外的功能、服务或能力。一种经常使用的扩展被称为设备驱动程序，它可提供主机软件应用程序（一般是操作系统）和硬件设备之间的接口。其它扩展包括用于 web 浏览器软件应用程序的小应用程序和插件、过滤器、用于图像编辑软件应用程序的效果和插件、以及用于音频/视频软件应用程序的编解码器。

以下描述的用于向扩展和其它软件应用程序提供从虚拟机器环境内部对硬件的直接访问的实施例可具有许多用途，包括简化虚拟机器设计以及使用户能够从虚拟机器环境内访问更大领域的硬件设备。提供从虚拟机器环境内对硬件的直接访问的另一好处是将包括操作系统设备驱动程序的一个或多个扩展从主机软件应用程序或操作系统中进行故障隔离的能力。在这一情况下，隔离的扩展可在虚拟机器环境内执行，虚拟机器环境可提供故障隔离，但是也可能需要维持对一个或多个硬件设备的直接访问以正确地操作。因此，详细描述以对可通过在一个或多个虚拟机器内执行将扩展从其主机进程中进行故障隔离的实施例的描述开始。随后，详细描述以对扩展或其它软件应用程序在虚拟机器环境内运行时可直接访问一个或多个硬件设备的实施例的描述而继续。

由于扩展与其主机软件应用程序紧密地协同操作，因此由扩展引入的不稳定性会使整个主机软件应用程序变得不可使用。一般而言，扩展通过可由主机软件应用程序使用的一个或多个应用程序接口（API）提供了对其能力的访问。扩展通过其来展示其功能的 API 一般被称为“服务 API”。如果扩展需要额外的信息、资源等，则扩展可通过一般被称为“支持 API”的一个或多个 API 向主机软件应用程序请求这些额外信息、资源等。万一扩展或主机软件应用程序不正确地使用了服务或支持 API，或者试图访问无事实证明的或不支持的 API，则任何导致的错误或无意识的人为因素将导致不稳定性。由于扩展一般在与其主机软件应用程序相同的进程内操作，因此当运行在该进程中的一个或多个扩展引入不稳定性时，主机软件应用程序很难继续正确运作。

如果扩展可以在单独的进程中执行，使得由该扩展引入的任何不稳定性可以

被隔离到与主机软件应用程序的进程无关的进程，则主机软件应用程序即使在面对不稳定的扩展时也能够继续正确运作。对于诸如操作系统等可主宿许多扩展的软件应用程序，隔离每一扩展可很大程度上提高操作系统的总体可靠性，因为故障的可能性与所使用的每一附加扩展成指数地增长。此外，隔离扩展允许应用程序作者全神贯注于识别和消除其自己的算法内的不稳定性来源。因此，本发明的实施例将扩展与其主机软件应用程序隔离，而继续对主机软件应用程序提供扩展的好处。

尽管并非所需，但本发明将在诸如由计算装置执行的程序模块等计算机可执行指令的一般上下文环境中描述。一般而言，程序模块包括例程、程序、对象、组件、数据结构等等，它们执行特定的任务或实现特定的抽象数据类型。在分布式计算环境中，任务可由通过通信网络连接的远程处理设备来执行。在分布式计算环境中，程序模块可以位于本地和远程计算机存储设备和/或介质中。本领域的技术人员可以理解，本发明可使用许多不同的计算装置单独地或作为分布式计算环境的一部分来实施，其中，这类设备可包括手持式设备、多处理器系统、基于微处理器或可编程消费者电子设备、网络 PC、小型机、大型机等等。

转向图 1，示出了可在其中实现本发明的示例性计算装置 100。计算装置 100 仅为合适的计算装置的一个示例，并非暗示对本发明的使用范围或功能的局限。此外，也不应将计算装置 100 解释为对图 1 所示的外围设备的任一个或其组合具有任何依赖或需求。

计算装置 100 的组件可包括但不限于，处理单元 120、系统存储器 130 以及将包括系统存储器的各类系统组件耦合至处理单元 120 的系统总线 121。系统总线 121 可以是若干种总线结构类型的任一种，包括存储器总线或存储器控制器、外围总线以及使用各类总线体系结构的局部总线。作为示例而非局限，这类体系结构包括工业标准体系结构（ISA）总线、微通道体系结构（MCA）总线、增强 ISA（EISA）总线、视频电子技术标准协会（VESA）局部总线以及外围部件互连（PCI）总线，也称为 Mezzanine 总线。此外，处理单元 120 可包含一个或多个物理处理器。

计算装置 100 通常包括各种计算机可读介质。计算机可读介质可以是可由计算装置 100 访问的任一可用介质，包括易失性和非易失性介质、可移动和不可移动介质。作为示例而非局限，计算机可读介质包括计算机存储介质和通信介质。计算机存储介质包括以用于储存诸如计算机可读指令、数据结构、程序模块或其它数据等信息的任一方法或技术实现的易失性和非易失性，可移动和不可移动介质。计算机存储介质包括但不限于，RAM、ROM、EEPROM、闪存或其它存储器技术、

CD-ROM、数字多功能盘（DVD）或其它光盘存储、磁盒、磁带、磁盘存储或其它磁存储设备、或可以用来储存所期望的信息并可由计算装置 100 访问的任一其它介质。通信介质通常表现为以诸如载波或其它传输机制等已调制数据信号形式的计算机可读指令、数据结构、程序模块或其它数据，并包括任一信息传送介质。术语“已调制数据信号”指以对信号中的信息进行编码的方式设置或改变其一个或多个特征的信号。作为示例而非局限，通信介质包括有线介质，如有线网络或直接连线连接，以及无线介质，如声学、RF、红外和其它无线介质。上述任一的组合也应当包括在计算机可读介质的范围之内。

系统存储器 130 包括易失性和/或非易失性存储器形式的计算机存储介质，如只读存储器（ROM）131 和随机存取存储器（RAM）132。基本输入/输出系统 133（BIOS）包括如在启动时帮助在计算机 110 内的元件之间传输信息的基本例程，通常储存在 ROM 131 中。RAM 132 通常包含处理单元 120 立即可访问或者当前正在操作的数据和/或程序模块。作为示例而非局限，图 1 示出了操作系统 134、应用程序 135、其它程序模块 136 和程序数据 137。

计算装置 100 也可包括其它可移动/不可移动、易失性/非易失性计算机存储介质。仅作示例，图 1 示出了对不可移动、非易失性磁介质进行读写的硬盘驱动器 141、对可移动、非易失性磁盘 152 进行读写的磁盘驱动器 151 以及对可移动、非易失性光盘 156，如 CD ROM 或其它光介质进行读写的光盘驱动器 155。可以在示例性操作环境中使用的其它可移动/不可移动、易失性/非易失性计算机存储介质包括但不限于，磁带盒、闪存卡、数字多功能盘、数字视频带、固态 RAM、固态 ROM 等等。硬盘驱动器 141 通常通过不可移动存储器接口，如接口 140 连接到系统总线 121，磁盘驱动器 151 和光盘驱动器 155 通常通过可移动存储器接口，如接口 150 连接到系统总线 121。

上文讨论并在图 1 示出的驱动器及其关联的计算机存储介质为计算装置 100 提供了计算机可读指令、数据结构、程序模块和其它数据的存储。例如，在图 1 中，示出硬盘驱动器 141 储存操作系统 144、应用程序 145、其它程序模块 146 和程序数据 147。注意，这些组件可以与操作系统 134、应用程序 135、其它程序模块 136 和程序数据 137 相同，也可以与它们不同。这里对操作系统 144、应用程序 145、其它程序模块 146 和程序数据 147 给予不同的标号来说明至少它们是不同的副本。

用户可以通过输入设备，如键盘 162 和定点设备 161（通常指鼠标、跟踪球或

触摸板）向计算装置 100 输入命令和信息。其它输入设备（未示出）可包括麦克风、操纵杆、游戏垫、圆盘式卫星天线、扫描仪等等。这些和其它输入设备通常通过耦合至系统总线的用户输入接口 160 连接至处理单元 120，或者可以通过其它接口和总线结构连接，如并行端口、游戏端口或通用串行总线（USB）。监视器 191 或其它类型的显示设备也通过接口，如视频接口 190 连接至系统总线 121。除监视器之外，计算机也可包括其它外围输出设备，如扬声器 197 和打印机 196，它们通过输出外围接口 195 连接。

由于接口技术可随着时间的推移而改进，某些计算装置可包含传统的接口以提供与传统设备的向后兼容性。图 1 的计算装置 100 被示出为具有传统接口 198，它可以是包括串行端口、并行端口、调制解调器端口等的多种接口中的任一种。传统接口 198 可使计算装置 100 能够与诸如传统设备 199 等传统设备进行通信，传统设备 199 可以是打印机、扫描仪、示波器、函数发生器或任何其它类型的输入或输出设备。如本领域的技术人员所已知的，大多数现代输入或输出设备通过诸如 USB 端口或 IEEE 1394 端口等依赖于新开发的标准的接口来连接。然而，传统设备可能没有这样的接口，因此必须依赖于传统接口以与计算装置 100 通信。

计算装置 100 可以在使用到一个或多个远程计算机的逻辑连接的网络化环境中操作。图 1 示出了到远程计算装置 180 的通用网络连接 171。通用网络连接 171 可以是各种不同类型的网络和网络连接的任一种，包括局域网（LAN）、广域网（WAN）无线网络、符合以太网协议、令牌环协议的网络、或包括因特网或万维网的其它逻辑、物理或无线网络。

当在网络环境中使用时，计算装置 100 通过网络接口或适配器 170 连接至通用网络连接 171，网络接口或适配器可以是有线或无线网络接口卡、调制解调器或类似的联网设备。在网络化环境中，相对于计算装置 420 所描述的程序模块或其部分可储存在远程存储器存储设备中。可以理解，示出的网络连接是示例性的，也可以使用在计算机之间建立通信链路的其它装置。

在以下描述中，将参考由一个或多个计算装置执行的动作和操作的符号表示来描述本发明，除非另外指明。由此，可以理解，这类动作和操作，有时称为计算机执行的，包括计算装置的处理单元对以结构化形式表示数据的电信号的操纵。这一操纵转换了数据或在计算装置的存储器系统中的位置上维护它，从而以本领域的技术人员都理解的方式重配置或改变了计算机的操作。维护数据的数据结构是存储器的物理位置，具有由数据的格式所定义的具体特性。然而，尽管在上述的上下文

环境中描述本发明，它并不意味着限制，如本领域的技术人员所理解的，后文所描述的动作和操作的各方面也可以用硬件实现。

转向图 2，示出了本发明的实施例所构想的用于将扩展与主机软件应用程序隔离的一种机制。如图 2 所示，主机进程 201 可调用代理 205 而非扩展 215 本身。扩展 215 可被主宿在不同于主机进程 201 的虚拟进程 211 中。虚拟进程 211 可试图至少在可提供类似于主机软件应用程序可提供的支持 API 203 的虚拟支持 API 213 的程度上，仿真主机进程 201。因此，运行在虚拟进程 211 中的扩展 215 可以用使用原始支持 API 203 的同一方式来使用虚拟支持 API 213。

本发明的一个实施例所构想的对代理 205 的一种设计可以是，至少在代理 205 可提供类似于扩展 215 所提供的服务 API 的服务 API 的程度上仿真扩展 215。主机进程 201 然后可使用由代理 205 提供的 API 来以它使用扩展 215 本身提供的服务 API 的同一方式访问扩展的功能。然而，如图 2 所示，当代理 205 使用这种服务 API 从主机进程 201 接收请求时，代理 205 可从主机收集相关信息，并将该信息转发到在虚拟进程 211 内执行的扩展 215。

本发明的一个实施例构想的对代理 205 的另一种设计可以是，与主机进程 201 接口，并转换或截取主机进程的某些功能，并使用扩展 215 来扩展主机进程 201 的功能。例如，扩展 215 可提供对诸如使用非寻常或传统文件系统格式的文件存储等特定类型的文件存储的访问。在这一情况下，代理 205 可被设计成检测主机进程 201 内的文件访问指令，并截取这些指令。代理 205 然后可将适当的信息转发到扩展 215，扩展 215 可访问使用传统文件系统格式的文件存储中的文件。信息然后可从扩展 215 返回到代理 205，并且代理 205 可向主机进程 201 呈现该信息。由此，代理 205 可扩展主机进程 201 的功能，诸如通过使主机进程 201 能够访问以传统文件系统格式保存的数据，即使主机进程未被设计成启用这一扩展的功能也能如此。由此，代理 205 不需要基于被设计成与主机进程 201 接口的预先存在的扩展，但是相反可被设计成担当主机进程和任一扩展之间的填补（shim）。

不论代理 205 是被设计成仿真预先存在的扩展，还是担当任一扩展的填补，代理 205 都可将适当的信息转发到扩展 215，以使扩展能够执行主机进程 201 的工作。本发明的一个实施例所构想的将信息从代理 205 转发到扩展 215 的一种方法要求代理 205 直接与扩展 215 通信。在这一情况下，代理 205 本身可调用扩展 215 的适当服务 API。本发明的一个实施例所构想的转发请求的一种替换方法要求代理 205 与在虚拟进程 211 内执行的占位程序（stub）217 通信。占位程序 217 然后可

调用扩展 215 的适当服务 API。如本领域的技术人员所已知的，某些扩展可能无法正确地处理通过进程间通信接收的请求。为避免这种困难，可使用虚拟进程 211 内诸如占位程序 217 等占位程序来提供一种机制，扩展 215 可使用该机制，通过其服务 API 经由进程内通信而非进程间通信来接收请求。

一旦扩展 215 从主机进程 201 接收了请求，它可继续响应该请求。取决于请求的特性，扩展 215 可访问一个或多个功能，这些功能通常由主机进程 201 通过支持 API 203 来提供，但是现在由虚拟进程 211 通过虚拟支持 API 213 来提供。如下文更详细地解释的，取决于主机请求的特性，扩展 215 可能需要直接访问计算系统 100 的资源，或者以直接的方式访问连接到计算系统的硬件设备。在这一情况下，可以作出规定（provision），以向扩展 215 授予对这类资源的访问权限，而同时将扩展 215 与主机进程 210 隔离。

为实现预期的隔离，仅具有两个单独的进程，如主机进程 201 和虚拟进程 211 可能是不够的。因此，本发明的实施例构想代理 205 可以用防止来自扩展 215 的不正确响应或部分扩展上的不正确行为影响主机进程 201 的方式来设计。例如，在本发明的一个实施例所构想的一种机制中，代理 205 可被设计成严格地遵守由扩展 215 呈现的服务 API。因此，如果扩展 215 试图向主机进程 201 返回不是主机所期望的形式或类型的数据，则代理 205 可识别潜在的问题，并且不将该数据传递到主机进程。

在本发明的一个实施例构想的另一种机制中，代理 205 可向要返回的数据应用进一步的智能来避免向主机进程 201 引入不稳定性。例如，如果扩展 215 遭受致命的错误并且发生故障，则代理 205 可维持一超时计数器或类似的机制，以检测扩展的故障，并可诸如通过提供错误响应或让主机进程温和地退化而不会例如丢失用户的工作产品，来向主机进程 201 通知该错误。代理 205 也可返回主机进程 201 给予扩展 215 的任一控制，以防止扩展的故障阻止主机进程的执行。例如，代理 205 可请求底层操作系统终止虚拟进程 211，并将控制返回给主机进程 201。或者，代理 205 可使用作为虚拟进程 211 的一部分的专用代码，来通知主机进程明显发生了故障，并请求虚拟进程终止并将控制返回给主机进程 201。

然而，如果扩展 215 正确地完成了向它请求的任何任务，则它可以服务 API 所指定的方式返回主机进程 201 预期的任何结果。由此，例如，如果结果是请求成功的指示，并且要在预定义的变量中返回给作出调用的程序，则扩展 215 可将该变量传递回占位程序 217，或直接传递给代理 205。从那里，该变量可由代理 205 返

回给最初作出调用的主机进程。以此方式，至少在涉及主机进程 201 前，代理 205 可能变得无法从扩展 215 中区分出来。当然，如本领域的技术人员所已知的，某些扩展可能不需要返回任何结果，在这一情况下，不需要实施用于接受返回值的任何规定。

如图 2 所示，扩展 215 在虚拟进程 211 中操作。因此，如果扩展 215 的行动导致不稳定性，则该不稳定性很可能包含在虚拟进程 211 内。在这一情况下，操作系统或某一其它代码，如代理 205，可检测到虚拟进程 211 中的错误，并可终止它，或试图重新启动它。在任一情况下，不稳定性不太可能影响主机进程 201，并且因此将不会导致对用户的有害故障。因此，上文描述的机制允许主机进程 201 即使在主机进程所使用的扩展 215 发生故障或变得不稳定时也能够正确地运作。

如上文详细描述的，代理 205 可以扩展 215 就如在主机进程中运行时一样的方式向主机进程呈现服务 API。在本发明的一个实施例构想的一种机制中，代理 205 可基于由扩展 215 实现的预定义服务 API 来创建。如本领域的技术人员已知的，扩展和主机软件应用程序能够通过其互操作的服务 API 一般在事先已知，因为软件应用程序的作者和扩展的作者通常是不同的实体。当安装扩展时，它可向主机软件应用程序或适当的信息存储，如注册数据库 221 注册其自身，并指示它支持哪些服务 API。使用这一信息，主机软件应用程序或底层操作系统能够在主机软件应用程序试图使用服务 API 之一时查找适当的扩展。这一信息也可用于创建代理 205，因为它指示了扩展 215 支持的服务 API 的完整组。代理 205 的创建也可以用后文更详细描述的方式改变例如注册数据库 221 中的条目。

本发明的一个实施例构想的另一种机制是基于预定义服务 API 的整个组创建可接受请求的的“超级代理”。这一超级代理然后可被调用，而不论主机应用程序试图使用哪一特定服务 API。在这一情况下，扩展 215 可在安装时执行的任何注册可包括向超级代理或底层支持体系结构的注册，使得超级代理可在主机软件应用程序使用特定的服务 API 时调用正确的扩展 215。

本发明的一个实施例构想的又一种机制是代理 205 可基于代理试图提供给主机进程 201 的扩展功能来创建。由此，代理 205 可被创建以检测、截取或连接主机进程 201 使用或主机进程 201 内的一个或多个功能，使得代理可向主机进程提供扩展 215 的功能的好处。使用上文描述的示例，如果代理 205 被设计成允许主机进程 201 通过扩展 215 访问传统文件系统，则代理可被设计成检测并截取主机进程使用的文件访问和类似的功能。代理 205 还可被设计成将相关信息从那些文件访问功能

转发到扩展 215，使得扩展可与传统文件系统接口。类似地，代理 205 可被设计成从扩展 215 接受响应，并将其转换成可由主机进程 201 识别为与主机进程的所截取的文件访问功能相关联的适当响应的格式。

在某些情况下，可能期望修改虚拟支持 API，以更准确地反映支持 API 203。例如，如果向虚拟支持 API 213 查询进程标识符，则它可返回虚拟进程 211 的标识符。然而，可能期望虚拟支持 API 213 返回主机进程 201 的标识符。在这一情况下，可使用“后向通道（back channel）”或“侧向通道（side channel）”通信来使虚拟支持 API 213 能够访问来自主机进程 201 的信息。

为确保对所请求的特定扩展调用了正确的代理，可使用注册数据库或类似的信息存储来将代理 205 链接到扩展 215。如上所述，注册数据库 221 或类似的信息存储可由主机进程 201 或操作系统来咨询，以确定用于调用扩展 215 的参数。然而，与标识扩展 215 本身相反，注册数据库 221 可指向代理 205。

一旦主机进程 201 调用了代理 205，代理 205 可继续调用或协调虚拟进程 211 内的扩展 215 的调用。如后文更详细地描述的，虚拟进程 211 可以已经是可操作的，或者可以处于各种就绪状态。如果虚拟进程 211 尚未可操作，则代理 205 可协调对虚拟进程 211 所必需的任何步骤的完成，以达到可操作状态。一旦虚拟进程 211 可操作，代理 205 可指令虚拟进程 211 调用扩展 215。例如，代理 205 可提供指向扩展 215 的位置的指针，并可传递该指针或主机进程 201 使用的类似参数。另外，如果确定扩展 215 使用了后向通道或侧向通道通信，则扩展所使用的任何附加资源也可在虚拟进程 211 内调用。

一旦虚拟进程 211 调用了扩展 215，和扩展 215 使用的任何其它代码，代理 205 就能够在必要时协调占位程序 217 的调用。或者，代理 205 可与扩展 215 直接建立通信链接。如果将使用占位程序 217，则代理 205 可向虚拟进程 211 提供占位程序 217 的位置以及调用占位程序时要使用的参数。一旦调用了占位程序 217，占位程序本身可与扩展 215 建立通信链接，并与代理 205 建立通信链接。代理 205 和占位程序 217 或扩展 215 之间的通信可使用任一类型的进程间或进程内通信协议，包括例如，已知的远程过程调用（RPC）机制。尽管可能事先决定所使用的通信协议，然而可实现同步交换（handshaking）过程，以确保代理 205 和占位程序 217 或扩展 215 能够适当地通信。

由于某些扩展可能依赖于用户模式环境来执行主机进程向它们请求的功能，因此可能必须提供虚拟环境中的扩展用于提供用户模式环境的机制。用户模式环境

一般指的是进程资源的总体状态，包括存储器、文件、注册表条目等等，使得给定用户模式环境中特定资源的引用是准确的，而当被传递到特定用户模式环境外时，这些相同引用可能涉及不正确的存储器位置，或者是不准确的。对于可接受或返回大量数据的扩展，发送和接收假定公用用户模式环境的存储器引用，通常比发送和接收数据本身更有效。因此，如果要正确运作使用这种数据传递方案的扩展，则在虚拟进程 211 和主机进程 201 之间维持公用用户模式环境是需要的。

转向图 3，示出主机进程 201 已经以上文详细描述的方式调用了在虚拟进程 211 和 311 内执行的两个扩展，即分别为扩展 215 和扩展 315。代理 205 可以是如上文详细描述的超级代理，并且可将来自主机进程 201 的请求定向到扩展 215 或扩展 315。或者，可使用第二代理（未在图 3 示出），使得扩展 215 和 315 的每一个可具有与主机进程 201 内的代理的一对一关系。

操作系统 134 也在图 3 中示出，它包括主机进程存储器 301 和虚拟进程存储器 302 和 303，它们分别对应于主机进程 201、虚拟进程 211 和虚拟进程 311。尽管图 3 和 4 示出的机制可依赖于主机进程 201 和虚拟进程 211 和 311 之下的公用操作系统，然而下文将更详细地描述的另外的机制也可提供主机进程和虚拟进程之间的公用用户模式，即使虚拟进程是与主机进程之下的操作系统 134 无关地执行的。当主机进程 201 和虚拟进程 211 及 311 的确共享公用操作系统 134 时，如图 3 所示的，操作系统也可包括页表映射 320 的集合，该页表映射将主机进程存储器 301 和虚拟进程存储器 302 和 303 映射到物理 RAM 13 的各段。尽管图 3 示出了段 321、322 和 323 分别对应于主机进程存储器 301 和虚拟进程存储器 302 和 303，本领域的技术人员可以理解，段 321、322 和 323 仅为说明性的，RAM 的物理段可能被分散，并且可能不是以所示的方式为连续的。

为在主机进程 201 和虚拟进程 211 和 311 之间维持公用用户模式环境，操作系统 134 或其它支持软件可向虚拟进程 211 和 311 提供对构成主机进程 201 的用户模式环境的某些或所有资源的访问。尽管以下描述集中在用于提供对用户模式环境的存储器资源方面的公用访问的机制上，然而本领域的技术人员将认识到，这些机制可适用于构成用户模式的其它资源，包括注册表资源、文件资源等等。

在本发明的一个实施例所构想的用于提供对用户模式环境的存储器资源方面的公用访问的一种机制中，操作系统 134 或类似的支持软件可将主机进程存储器 301 复制到虚拟进程存储器 302 和 303。如图 3 所示，将主机进程存储器 301 复制到虚拟进程存储器 302 和 404 可导致将 RAM 段 312 物理复制到新的 RAM 段 322

和 323。或者，I/O 管理器可将主机进程 301 复制到系统存储器的常驻非分页池（pool）中，并可向虚拟进程 211 或 311 提供对该非分页池的访问。

一旦扩展 215 或 315 完成了其任务，虚拟进程存储器 302 或 303 可向后与主机进程存储器 301 合并。例如，代理 205 可执行不同的功能，它可以是位置 322 和 323 中的虚拟进程存储器和位置 321 中的主机进程存储器之间的逐字节比较或更大级别的比较，以确定任何差异。那些差异可被验证为正确的，或者符合扩展 215 或 315 的预期行为，然后可被复制回主机进程存储器 301，或通过代理 205 变得对主机进程 201 可用。或者，如果 I/O 管理器仅将主机进程存储器 301 复制到系统存储器的常驻非分页池中，则 I/O 管理器可将非分页池复制回主机进程存储器。一般而言，这一复制可在每一请求的基础上完成。因此，与复制整个主机进程存储器 301 相反，本发明的一个实施例构想的一种更有效的机制要求操作系统 132 或其它支持软件仅复制主机进程存储器 301 中扩展 215 或 315 执行所请求的任务所需的那些缓冲区。当由操作系统 134 的 I/O 管理器执行时，这一到系统存储器的非分页池的缓冲区专用复制被称为“缓冲的 I/O（Buffered I/O）”或“缓冲的 I/O 方法（I/O Method Buffered）”。

转向图 4，示出了本发明的一个实施例构想的一种用于提供对用户模式环境的存储器资源方面的公用访问的替换机制。具体地说，如图 4 所示，与复制主机进程存储器 301 的某一些或全部相反，可修改由操作系统 134 维护的页表映射 320，以将虚拟进程存储器 302 和 303 定向到 RAM 132 中的物理位置 321，其中储存了代表主机进程存储器 301 的数据。由于消除了对复制数据的需要，图 4 所示的机制可以比图 3 所示的机制更有效。

然而，如果扩展 215 和 315 可影响构成主机进程存储器 301 的物理段 321，则扩展的一部分上的错误或不稳定性将导致主机进程 201 本身中的错误或不稳定性。因此，为将这一可能性最小化，可以“只读”方式修改页表映射，使得虚拟进程 211 和 311 可指向物理存储器 321，以读取它，但不允许修改它。因此，运行在虚拟进程 211 和 311 中的扩展的一部分上的任何错误或不稳定性不能将错误或不稳定性引入到主机进程 201 中，因为虚拟进程不允许修改主机进程存储器。

如上所述，由图 4 的机制所构想的对页表映射 320 的修改可在每一请求的基础上完成。然而，如果仅存在一个虚拟进程，则页表映射 320 可继续指向 RAM 132 的物理段 321，即使对不需要用户模式环境的请求也是如此。上述页表映射的修改一般被称为“即非缓冲也非直接 I/O（Neither Buffered Nor Direct I/O）”或“也不

是 I/O 方法（I/O Method Neither）”。

本发明的一个实施例构想的用于提供对用户模式环境的存储器方面的公用访问的另一替换机制可以是图 3 和图 4 中示出的替换方案的混合。具体地，如上所述，可向虚拟进程 211 和 311 提供对物理存储器 321 的只读访问。然而，如果扩展 215 或扩展 315 的任一个需要将数据写回存储器，则可执行“写复制（copy-on-write）”。如本领域的技术人员已知的，写复制可在向数据写入修改之前将正在修改的数据复制到新位置。由此，如果扩展 215 或扩展 315 需要将数据写回存储器 321，则存储器 321 的某一些或全部可被复制到一新位置，如图 3 所示的 322 或 323，并且扩展 215 或扩展 315 然后可修改存储器 322 或 323 中复制的数据。以此方式，运行在虚拟进程 211 和 311 中的扩展引入的错误或不稳定性将不会影响主机进程 201，因为虚拟进程将不被允许修改主机进程存储器。

代理 205 可跟踪可能由扩展 215 或扩展 315 使用上述写复制的机制来编辑的那些存储器段。当访问那些存储器段时，代理可适当地引用位置 322 或 323，而非位置 321。如果储存在位置 322 或 323 中的数据符合扩展 215 或 315 的预期行为，则代理 205 可诸如通过将数据复制到主机进程存储器 301 中，或通过向主机进程传递位置 322 或 323，来允许数据在主机进程 201 内使用。因此，可实现上述隔离，而同时允许代理 205 访问修改的数据。

如上所述，诸如图 2 的虚拟进程 211 等可主宿扩展的虚拟进程的初始化可在主机进程 201 调用了代理来替代扩展 215 之后由代理 205 协调。本发明的一个实施例构想的一种类型的虚拟进程是在与主机进程相同的操作系统 134 上执行的主机进程 201 的副本。这一虚拟进程可通过派生（fork）主机进程，然后使用克隆的进程作为虚拟进程来创建。或者，可指令操作系统再一次启动最初被调用来创建主机进程 201 的软件应用程序。由此，例如，如果主机进程 201 是 web 浏览器，则虚拟进程 211 可通过再一次启动 web 浏览器应用程序来创建单独的进程或通过派生当前运行的 web 浏览器进程来创建。

本发明的一个实施例构想的另一种类型的虚拟进程可以在虚拟机器环境的上下文中创建。如果扩展 215 是操作系统使用的设备驱动程序或其它扩展，则虚拟机器可提供最优解决方案。尽管诸如通过派生或重新执行使用操作系统来创建其本身的另一副本以担当虚拟进程是可能的，然而更高档的解决方案可以是启动虚拟机器，并引导虚拟机器环境中的操作系统，以担当用于主宿一个或多个扩展的虚拟进程。这一机制可能提供更好的隔离，并可允许一个操作系统使用为不同的操作系统

设计的扩展。例如，可能未对操作系统的较新版本更新的传统驱动程序可主宿于运行在虚拟机器环境内的操作系统的较旧版本内。以此方式，可使扩展的特征和能力仍对较新操作系统的用户可用，而同时对可能由传统扩展导致的不稳定性屏蔽了较新的操作系统。通过使用虚拟机器，或通过执行上述派生或重新执行，虚拟进程211可提供与主机进程201等效的支持API，而无需在个别的基础上考虑支持功能。

与接收来自底层操作系统134的支持的虚拟进程211和311不同，如本领域的技术人员所已知的，虚拟机器一般不以此方式利用操作系统。相反，为避免令每一虚拟机器指令通过完整的操作系统而导致的性能恶化，虚拟机器可仅依赖于系统管理程序，它可提供有限的操作系统功能，并可为运行在虚拟机器环境中的任何操作系统抽象计算装置的底层硬件。通过使用这一系统管理程序，虚拟机器可更有效地运作。然而，作为使用系统管理程序的结果，在虚拟机器进程能够在计算装置的处理器上执行之前，该计算装置的操作系统可被移除，并且该操作系统的基础可被储存。随后，当虚拟机器进程完成了任务时，它可从硬件中移除其基础，并且原始的操作系统可被再次储存。计算装置的操作系统和虚拟机器进程之间的这一硬件使用的交换可以每秒发生多次。由此，尽管用户可能察觉虚拟机器仅为使用操作系统的另一应用程序，然而虚拟机器进程一般仅与操作系统分时操作计算装置硬件。

为实现上述交换，虚拟机器可包括可由计算装置的操作系统调用的虚拟机器设备驱动程序或类似的扩展。虚拟机器设备驱动程序可提供必要的指令，用于从计算装置硬件中移除操作系统的基础，并高速缓存它们，直到允许操作系统恢复执行的时刻。另外，虚拟机器设备驱动程序可协调虚拟机器进程的调用。例如，当操作系统在执行时，它可接收令虚拟机器进程执行一项任务的用户命令。操作系统然后可向虚拟机器设备驱动程序发出一命令，以令虚拟机器进程执行所请求的任务，并将以有效的方式向操作系统返回控制。由此，操作系统可将向虚拟机器进程传递控制作为它将控制传递给当前由操作系统协调的任一其它线程来处理。虚拟机器设备驱动程序在接收这一命令后，可从计算装置的硬件中移除操作系统的基础、允许系统管理程序安装其基础、并将命令传递到虚拟机器进程。随后，当虚拟机器进程完成时，虚拟机器设备驱动程序可重新安装操作系统的基础，并允许它在计算装置硬件上恢复执行。

如上文详细描述的，代理205可检测虚拟进程211内的故障，并可试图防止该故障将不稳定性引入到主机进程201中。然而，如果虚拟进程211是运行在由虚拟机器创建的环境中的虚拟操作系统进程，则代理205可能很难检测或控制这种虚

拟操作系统进程，因为代理 205 所依赖的操作系统不在计算装置硬件上执行，但是相反被储存并等待虚拟机器完成其执行。因此，本发明的一个实施例构想的用于隔离错误的一种机制要求系统管理程序监视在由虚拟机器创建的环境中执行的软件，并检测该环境内的故障。如果检测到故障，则系统管理程序可停止执行、重新安装操作系统的基础、并允许它在计算装置硬件上恢复执行。系统管理程序也可提供适当的响应，以允许操作系统或依赖于虚拟环境中的扩展的其它软件温和地退化。

另外，由于操作系统在由系统管理程序允许它恢复执行之前一般无法恢复执行，因此系统管理程序也可维持一定时器或类似的机制，以确保虚拟机器环境中的故障不会阻止控制被返回到操作系统。尽管可以用上述方式使用定时器机制来检测故障，然而如果虚拟机器用于创建主宿一个或多个扩展的环境，则定时器机制可具有进一步的重要性，因为如果在虚拟机器环境中出现故障，可能没有用于将控制返回到操作系统的任何其它机制。

或者，与在系统管理程序中维护诸如定时器机制等用于检测故障的机制相反，这类机制可在计算装置 100 的硬件中维护，如果在由虚拟机器创建的环境中检测到故障，它可提示系统管理程序将控制返回给操作系统。例如，操作系统可在允许系统管理程序在硬件上执行之前，在硬件中设置一定时器。随后，如果在由虚拟机器创建的环境中发生了故障，则由硬件维护的定时器可超时，并提示系统管理程序将控制返回给操作系统。为将控制返回给操作系统，如果由硬件维护的定时器超时，可修改系统管理程序以中止任何执行，并将控制返回给操作系统。系统管理程序也可指示错误的存在，或可在控制以此方式返回时指示执行尚未完成。

如果虚拟进程 211 是在由虚拟机器创建的环境中运行的虚拟操作系统进程，则另一新增的问题是代理 205 和虚拟进程 211 或扩展 215 之间的通信可能无法依赖于进程间通信或 RPC 机制，如上所详细描述的。相反，代理 205 和虚拟操作系统进程 211 之间的通信可由系统管理程序或其它机制来协调，这些其它机制由虚拟机器设置用于与主机进程 201 之下的操作系统进程通信。这种机制可包括，例如，将消息储存在预定义的存储器位置中，以使当虚拟机器和操作系统的每一个在计算装置硬件上执行时，该消息对两者都是可访问的，或者，作为另一示例，当虚拟机器和操作系统都在计算装置硬件上执行时，提供保留在存储器内的线程。

另外，上文详细描述的可提供虚拟进程 211 或 311 和主机进程 201 之间的公用用户模式的机制也可能需要某些修改，以在虚拟进程 211 或 311 是运行在虚拟机器环境中的虚拟操作系统进程的环境中实现。例如，与依赖于公用操作系统 134

来执行对页表映射的修改相反，修改可在由虚拟机器的系统管理程序维护的页表映射中作出。由此，如果复制主机进程存储器 301 以创建虚拟进程存储器 302 和 303，这种复制可由系统管理程序而非图 3 所示的操作系统 134 来执行。更具体地，即使主机操作系统不再执行并且虚拟机器进程正在执行时，主机进程存储器 301 也可保留在物理存储器位置 321 中。系统管理程序可物理虚拟存储器位置 321，并可将该位置的内容复制到处于系统管理程序控制之下的物理存储器位置 322 或 323 中。

以类似的方式，如果主机进程 201 和虚拟进程 211 及 311 之间的公用用户模式是通过以上文参考图 4 详细描述的方式修改页表映射来实现的，则对页表映射的修改可由系统管理程序来执行。由此，即使主机操作系统当前不在执行，主机进程存储器 301 也可保留在物理存储器位置 321 中，并且系统管理程序可将虚拟进程存储器 302 和 303 映射到物理存储器位置 321。重要的是，需要被映射大物理位置 321 的虚拟进程存储器两者，如虚拟存储器 302 或 303，将处于系统管理程序的控制之下。因此，由于主机进程存储器 301 不需要任何修改，上述机制将不需要来自操作系统 134 的任何支持，因此该操作系统 134 可以是任一标准操作系统。

如果虚拟进程存储器被映射到由主机进程存储器使用的物理存储器位置，并且使用诸如上文详细描述的写复制方案，则系统管理程序也可执行必要的复制。例如，系统管理程序可留出储存作为写复制的一部分而写入的值的额外的物理存储器位置。此外，如上所述，可修改代理 205 以引用主机进程存储器 301 和用于写复制的额外位置两者。然而，由于由系统管理程序留出的额外存储器可能不是由代理 205 之下的操作系统使用的存储器，因此可修改代理，以使即使存储器位置不被底层操作系统正确地访问，也可特别地引用这些存储器位置。或者，由系统管理程序留出的存储器位置还可被进一步复制到代理 205 之下的操作系统可访问的存储器位置，作为虚拟机器停止在计算装置上执行并允许操作系统恢复执行的过程的一部分。

本发明的一个实施例构想的用于提供公用用户模式环境的另一替换机制要求代用主机进程在虚拟操作系统进程内运行。例如，类似于主机进程，代用主机进程可运行在虚拟机器环境内虚拟操作系统的顶层。代用主机进程的用户模式环境可以与处于虚拟机器环境外的主机进程的用户模式环境相同，由此自动提供了公用用户模式。该公用用户模式可由主机进程和代用主机进程之间的通信如使用上述技术来维护，而无需明确地访问或复制主机进程存储器 301。

本发明的一个实施例构想的用于创建虚拟操作系统进程的一种机制是调用主

机计算装置 100 上的虚拟机器软件应用程序，然后引导在执行虚拟机器软件应用程序时创建的环境的上下文中的适当操作系统。如本领域的技术人员所已知的，虚拟机器软件应用程序一般包括一操作系统扩展，它可用于从计算装置硬件移除操作系统 134 的基础，并将它们储存到临时存储中。虚拟机器软件应用程序也可包括一系统管理程序，在移除了操作系统 134 的基础之后，它可将其自己的基础安装在计算装置硬件上，并以适当的方式抽象该硬件以创建虚拟环境。虚拟操作系统可以与操作系统 134 相同或不同，它可在由系统管理程序提供的抽象硬件上引导。由此，系统管理程序可创建一虚拟机器环境，其中，虚拟操作系统进程可与操作系统 134 无关地执行。尽管这一虚拟操作系统进程可提供上文列举的好处，然而对虚拟机器软件应用程序的调用，包括所描述的对操作系统 134 的移除以及对虚拟机器环境内适当操作系统的引导，可以是慢得惊人的进程。

为避免由启动虚拟机器软件应用程序然后引导虚拟机器环境内的操作系统而引入的低效率，本发明的一个实施例构想的另一种机制要求在虚拟机器环境内初始化虚拟机器并引导操作系统，然后将所得的虚拟机器环境的最终状态保存并克隆，用于将来的使用。由此，例如，在计算装置 100 的初始设置过程中，在引导了操作系统 134 之后，可自动启动虚拟机器软件应用程序，并可在由虚拟机器创建的环境内引导虚拟操作系统。一旦这一虚拟操作系统被引导，可保存该虚拟机器环境的状态。如本领域的技术人员所已知的，这一状态可被容易地保存，因为虚拟机器软件应用程序可能在计算装置 100 的存储介质上仅创建包括虚拟机器环境的状态的少数文件。那些文件可被访问和复制，并且虚拟机器软件应用程序然后可保留在可操作状态，或者它可以预定的状态放置，例如睡眠模式，或者它甚至可被完全关闭。

随后当主机进程（可以是操作系统 134 或应用程序 145 的任一个）试图执行将导致使用扩展的操作时，这或者是被设计得如此，或者是因为代理作出这样的调停，则可复制保存的虚拟机器环境的状态，并且可以用有效的方式创建一新的虚拟机器环境。由于虚拟机器环境的状态已包括引导的虚拟操作系统，因此可容易地创建可主宿请求的扩展的虚拟进程。例如，如果所请求的扩展是操作系统扩展，则用于该扩展的虚拟进程已经以虚拟操作系统的形式存在。另一方面，如果请求的扩展是软件应用程序扩展，则可在虚拟操作系统上执行适当的软件应用程序，由此创建了一适当的虚拟进程。因此，通过在虚拟机器环境内引导了虚拟操作系统之后保存由虚拟机器软件应用程序创建的状态，然后在必要时克隆该保存的状态，可有效地创建用于主宿操作系统和软件应用程序扩展两者的虚拟进程。

为提供对虚拟进程创建的适当支持，虚拟机器软件应用程序可被设计成抽象硬件的超集（superset），它可以大于虚拟机器软件应用程序正常抽象的硬件集。类似地，在虚拟机器环境内引导的虚拟操作系统可实现一完整的操作系统 API 集。通过抽象这一硬件超集，并提供完整的操作系统 API 集，由虚拟机器创建的状态很可能可用于生成用于所请求的扩展的适当虚拟进程。因此，通过克隆保存的状态，可生成更多数量的有用虚拟进程，并且更少的虚拟进程将需要使用更大成本的机制来创建。

转向图 5，示出了本发明的一个实施例构想的用于创建虚拟操作系统进程的另一种机制。流程图 400 一般示出了诸如计算装置 100 等许多现代计算装置的启动过程。流程图 400 并非特定的计算装置或操作系统的启动过程的详细描述，而是相反，旨在提供通常在启动过程中找到的元素的一般说明，以更好地解释由本发明的一个实施例所构想的机制。

如可从图 5 中见到的，启动过程通过在步骤 405 向计算装置提供电源开始。在随后的步骤 410，中央处理单元（CPU）可开始执行只读存储器（ROM）基本输入/输出系统（BIOS）中找到的指令。ROM BIOS 可执行基本硬件测试，以确保计算设备的中央硬件元素正确地运作。在步骤 415，BIOS 可读取配置信息，它一般储存在互补金属氧化物半导体（COMS）存储器中。如本领域的技术人员所已知的，CMOS 存储器可以是一个小范围的存储器，当计算装置不在操作时，其内容由电池来维持。CMOS 存储器可标识可连接到计算装置的一个或多个计算机可读介质。如步骤 420 所指示的。BIOS 可检查各个计算机可读介质中的第一个扇区，以找出主引导记录（MBR）。

一般而言，MBR 包含某些或所有的分区加载器，它可以是用于查找引导记录并开始操作系统的引导的计算机可执行指令。由此，在步骤 425，在 MBR 处找到的分区加载器可取代 BIOS，并可检查计算机可读介质上的分区表或类似的记录，以确定要加载的适当操作系统。每一操作系统可具有与其相关联的引导记录，并且在步骤 430，如果引导记录没有任何问题，则分区加载器可启动操作系统的引导。

作为引导操作系统的一部分，分区加载器可调用硬件检测例程，该例程可开始执行硬件检测，如由步骤 435 所指示的。一般而言，在步骤 435 执行的硬件检测仅是初步的，并非必要地启用硬件，步骤 435 的硬件检测可以仅创建硬件设备的列表供以后使用。例如，这一列表可以储存在注册数据库或类似的信息存储中。在步骤 440，分区加载器可调用另一操作系统进程或子系统，以提供到计算装置的各个

硬件设备的通信和控制链接。有时候这一子系统被称为“硬件抽象层（HAL）”。另外，在步骤440，分区加载器也可加载操作系统的内核和注册表，或包含必要的硬件和软件信息的类似数据库。

由分区加载器在步骤 440 加载的注册表或类似的数据库也可包含操作系统内核访问诸如硬盘驱动器或存储器等需要的硬件所需要的设备驱动程序的列表。因此，在步骤445，分区加载器可加载这些设备驱动程序，以提供对操作系统内核的适当支持。一旦加载了设备驱动程序，分区加载器在步骤445也可将计算装置的控制传递到操作系统内核。

尽管流程图 400 的步骤 405 到 445 一般示出了大多数启动例程的元素，然而步骤 450 示出了本发明的一个实施例所构想的用于创建可主宿操作系统扩展或软件应用程序的虚拟操作系统进程的机制的第一部分。具体地，在步骤450，与引导记录相关联的HAL或信息可向操作系统内核指示在计算装置上存在比实际上物理存在更多的CPU。由此，例如，在仅具有单个CPU的计算装置中，操作系统内核可在步骤445接收在计算装置上存在两个或多个CPU的指示。类似地，对于已经具有两个 CPU 的计算装置，操作系统内核可接收在计算装置上存在三个或多个CPU的指示。如下文更详细地描述的，通过指示实际上不存在的CPU的存在，可更容易且有效地创建虚拟操作系统进程。

转向流程图400，在步骤455，操作系统内核可调用HAL来初始化操作系统内核相信在计算装置中存在的每一CPU。因此，初始化CPU的请求可包括实际上在计算装置中不存在的 CPU。一旦 HAL 完成了所有 CPU 的初始化，在步骤460可保存系统的状态，用于随后以下文详细描述的方式有效地创建虚拟操作系统进程。然后，操作系统的引导可以用标准启动操作来继续，包括例如，初始化操作系统的各个子系统、激活构成计算装置100的硬件设备、以及加载适当的设备驱动程序，如步骤465所指示的。尽管步骤465特别地列出了输入/输出（I/O）子系统的初始化，然而操作系统内核也可在步骤465初始化存储器管理器、进程管理器、对象管理器、操作系统的各个内核、以及类似的子系统。另外，操作系统内核可重新启用硬件中断，并可激活作为计算装置100的一部分被检测的各个硬件设备。如上所述，作为各个硬件设备的激活的一部分，操作系统内核也可加载用于那些设备的适当设备驱动程序。如本领域的技术人员所已知的，由于许多操作系统最初是为具有单个CPU的计算装置而设计的，因此这类操作系统一般仅用单个CPU执行图5所示的大部分步骤，并仅在几乎完成了所有的启动过程之后才激活任何额外的

CPU。因此，主 CPU 一般维护所有的硬件绑定，而其它 CPU 的任务可以被赋予将在计算装置上执行的各种进程的任务。

如上所述，在步骤 450，向操作系统内核通知额外的 CPU，即使 CPU 可能不在计算装置内物理地存在。由此，在步骤 470，可向操作系统内核通知，在步骤 450 所指示的，但不物理地存在的那些 CPU 发生故障。步骤 470 处这一发生故障的 CPU 的指示实际上撤消了步骤 450 处对额外 CPU 的指示，并允许操作系统内核使用与计算装置 100 上物理地存在的相同数目的 CPU 完成操作系统的引导过程。如上所述，由于各操作系统可在不同的时刻初始化额外的 CPU，因此步骤 470 并不意味着局限于在执行了步骤 465 所示的所有元素之后发生。相反，这意味着步骤 470 在额外的 CPU 被初始化，并且建立了适当的硬件绑定之后执行，只要它可以发生。继续流程图 400，在步骤 475，操作系统内核可启动适当的子系统来创建用户模式环境，并且在步骤 480，一旦创建了用户模式环境，操作系统可完成引导过程。

一旦在步骤 480 完成了引导过程，可引导虚拟环境，这可诸如通过经由其引导在步骤 480 完成的操作系统输入命令来执行虚拟机器而实现。为更有效地创建虚拟环境，可使用在步骤 460 引导操作系统期间保存的状态。由于保存的状态反映了在步骤 450 存在多个 CPU，并且不考虑步骤 470 处次级 CPU 的故障指示，因此虚拟环境可如同存在多个 CPU 那样被引导。因此，虚拟机器环境能够以下文示出的方式利用由主机操作系统建立的机制来更有效地启动。

如上所述，由于许多操作系统在引导过程几乎完成之前仅使用单个 CPU，因此该 CPU 一般被赋予处理大多数或所有系统设备的任务，包括处理诸如硬件中断等来自那些系统设备的任何通信。因此，具有多个物理 CPU 的计算装置上的操作系统一般提供引导过程期间未使用的 CPU 上执行的进程用于与引导过程期间使用的 CPU 进行通信的机制，以向那些进程提供与硬件通信的能力。图 5 示出了可充分利用这一能力以允许虚拟机器环境与底层硬件通信，而无需对硬件设备的任何运行时绑定的机制。具体地，当将保存的状态提供给虚拟环境时，可配置虚拟环境，使得在引导过程期间使用的CPU不被使用，或至少不被允许与输入/输出硬件通信。相反，虚拟环境可使用操作系统的机制以通过如同计算装置包括多个 CPU 那样运作来充分利用已对操作系统执行的硬件绑定。

作为一个示例，在仅具有单个 CPU 的计算装置中，虚拟操作系统进程将如同至少有第二 CPU 那样地运作，因为尽管操作系统在步骤 470 接收到了第二 CPU 发生故障的指示，然而虚拟环境未接收到任何这样的指示。由此，尽管计算装置中的

单个物理 CPU 仍执行所有的工作，虚拟机器的环境如同存在两个 CPU 系统那样运作，其中一个 CPU 具有对硬件设备的所有运行时绑定，而第二 CPU 主宿虚拟操作系统进程，由于第一 CPU 的存在，该虚拟操作系统进程不需要用对硬件的任何运行时绑定来初始化。结果，由于不需要初始化任何硬件，可有效地引导虚拟操作系统，并且由于不需要抽象任何硬件，虚拟机器本身可被十分有效地引导。如果主宿在虚拟操作系统进程中的扩展需要与硬件设备通信，则可使用上述为在多 CPU 系统中使用而建立的机制，从虚拟操作系统进程向主机操作系统作出请求。由此，扩展可以用标准方式来运作，并且可有效地创建虚拟环境。

然而，如本领域的技术人员已知的，对于某些扩展，例如操作系统设备驱动程序，上述机制可能无法提供令人满意的解决方案。特别地，如果主机操作系统遇到传统硬件，如传统设备 199，它可能无法找到适当的驱动程序，并且可能无法正确地识别硬件。由此，尽管适当的虚拟操作系统进程可主宿传统设备驱动程序，如传统接口 198，然而可能没有方法来与传统硬件通信，因为使用上述机制，操作系统将处理所有的硬件通信，并且操作系统将不能正确地连接到传统硬件。此外，即使底层操作系统的确正确地连接到计算装置的所有硬件，然而，即使使用上述机制将最小量的延迟引入到硬件通信中，某些扩展，如视频设备驱动程序也无法正确地运作。

因此，本发明的一个实施例构想的上述机制的一个变异要求其设备驱动程序将被主宿在虚拟操作系统进程中的硬件设备在底层操作系统的引导序列中标识，并且不绑定到底层操作系统，而是绑定到虚拟操作系统进程，从而向设备驱动程序提供了对该硬件设备的直接访问。更具体地，硬件设备的中断可被发送到被指示、但物理地不存在的次级 CPU。随后，当虚拟机器创建假定次级 CPU 的确存在的环境时，它能够初始化对硬件设备的运行时绑定，从而允许虚拟操作系统进程直接与硬件设备通信。由此，如图 5 所示，在步骤 499 完成虚拟环境的引导之前，可任选的步骤 495 可插入传统设备 199 的硬件配置，并可将正确的设备驱动程序，如传统接口 198 加载到虚拟环境中。

或者，虚拟机器可创建具有两个或多个虚拟 CPU 的环境，而不依赖于上述引导优化。不管用于创建多 CPU 虚拟环境的过程如何，其设备驱动程序由虚拟操作系统进程主宿的硬件设备可如同硬件设备向作为虚拟 CPU 的次级 CPU 发送中断那样被绑定。由此，在操作系统的初始引导期间，其驱动程序应当被主宿在虚拟环境中的硬件设备可被隐藏或延迟，如下文更详细地讨论的，使得硬件设备不被绑定到

加载操作系统的物理 CPU。然而，作为引导过程的一部分，虚拟环境可被绑定到硬件设备。如上所述，虚拟环境可如同至少一个第二 CPU 存在且虚拟环境正在使用它那样被创建。由此，对硬件设备的绑定将如同硬件设备向第二 CPU 发送中断那样被执行。由于仅存在单个物理 CPU，因此它可从硬件设备接收通信。然而，那些通信可被定向到虚拟环境而非主机操作系统，从而向虚拟环境提供了对硬件设备的直接访问。

本发明的实施例构想若干机制，其驱动程序应当主宿在虚拟操作系统进程中的硬件设备在流程图 400 的步骤 465 通过这些机制可被隐藏或延迟。本发明的一个实施例构想的一种机制要求捕捉可在步骤 465 被发送到应当主宿在虚拟操作系统进程中的设备驱动程序的任何控制信息。这一控制信息可被延迟，直到在步骤 490 建立了虚拟操作系统进程，然后被中继到设备驱动程序。本发明的一个实施例构想的另一种机制要求设备驱动程序的代理，它可由操作系统进程以上文参考主机进程 201 和代理 205 所描述的方式在步骤 465 调用，以返回“OK”指示，随后高速缓存发送到它的任何输入/输出请求分组（IRP），直到在步骤 490 建立虚拟操作系统进程。然后代理可将 IRP 转发到虚拟操作系统进程中的设备驱动程序。或者，代理可简单地延迟，直到建立了虚拟操作系统进程，然后可将任何 IRP 直接传递到设备驱动程序，而无需高速缓存。

本发明的一个实施例构想的又一种机制要求在步骤 465 硬件设备最初被绑定到操作系统，然后发送一“休眠”或类似的命令，它可干净地清洗队列中的任何 IRP，并保持硬件处于方便的状态。虚拟操作系统进程中的设备驱动程序然后可在步骤 495 试图与来自虚拟操作系统进程内的设备建立直接通信。本发明的一个实施例构想的这一机制的变异在步骤 465 要求从操作系统隐藏硬件设备，而非如上所述的绑定然后休眠。可通过向 HAL 或诸如即插即用管理器等各种其它子系统发送适当的命令来隐藏硬件设备。随后，在步骤 480 引导了操作系统并且建立的虚拟操作系统进程之后，可在步骤 495 激活硬件设备或令其可见，因此将其自身绑定到虚拟操作系统进程和其中主宿的设备驱动程序。

与以上文详细描述的方式试图仿真额外的 CPU 来充分利用多 CPU 操作系统的能力相反，本发明的一个实施例构想的用于有效地创建虚拟进程的一种替换机制在图 6 中概括地示出。图 6 所示的流程图 500 包含上文参考图 5 详细描述的许多相同的步骤。特别地，步骤 405 到 445 以及 465 到 475 概括地示出了上文详细描述的同样的基本启动过程。另外，尽管未在图 6 中具体示出，在步骤 445 和 465 之间，

操作系统内核可获知计算装置的 CPU，然后可调用 HAL 来初始化那些 CPU。然而，与图 5 所示的步骤 450 和 455 不同，上述步骤不导致向操作系统内核呈现比计算装置内实际存在的更多数量的 CPU。在步骤 475 之后，可执行新的步骤 505，由此可保存计算装置的状态。

在步骤 485 完成了操作系统引导之后，可启动虚拟机器，并且该虚拟机器可利用通过观察和记录代码收集的信息。由此，在步骤 485，虚拟机器可开始引导过程，并且在步骤 510，虚拟机器可使用在步骤 505 记录的状态来更有效地引导虚拟操作系统进程。更具体地，虚拟环境可仅使用它需要虚拟化的特定硬件设备的参数，从而允许它跳过其它硬件设备。此外，由于在操作系统引导期间，诸如在步骤 505，参数已被建立并记录，因此虚拟机器可更有效地虚拟化那些硬件设备。然而，如果诸如传统设备 199 等硬件设备未在步骤 465 被正确地初始化，则它可在可任选步骤 495 以上文详细描述的方式在虚拟环境中初始化。最终，由于虚拟机器可选择硬件设备的有限组来虚拟化，并可更有效地虚拟化它们，因此可更有效地创建虚拟环境。然而，本领域的技术人员可以认识到，如果引导的操作系统和虚拟操作系统是相同的，或者至少在其与硬件的接口方面是相似的，则上述优化将是最有效的。

在某些情况下，包括可由虚拟操作系统进程主宿的某些硬件设备驱动程序扩展，由虚拟操作系统进程提供的支持 API 的语义可能不是有用的。例如，某些硬件设备驱动程序可需要对物理硬件的访问，以正确地控制它。因此，在这些情况下，需要虚拟操作系统进程向主宿的设备驱动程序提供对物理硬件的访问。尽管上述某些机制可提供必要的直接访问，然而本发明的实施例构想了另外的机制，它可应用于任何虚拟进程，以允许主宿在该进程内的扩展能够直接访问硬件。

因此，可使用下文描述的机制，不仅提供扩展和主机进程之间的故障隔离，也可使虚拟机器在抽象硬件可能低效率或不可能的情况下能够提供对硬件的直接访问。例如，上述机制可允许虚拟机器主宿依赖于以下硬件的软件，即虚拟机器未被设计成抽象这些硬件。由此，上述机制向虚拟机器设计者和作者提供了缩小他们需要考虑的硬件的范围的能力，而同时向消费者提供了使用独特或传统硬件的能力。

转向图 7，示出了虚拟机器进程 617，它使用了系统管理程序 613 来与底层硬件 620 接口，并包括主宿扩展 615 的虚拟操作系统进程 611。如黑色箭头所指示的，本发明的实施例构想了一种虚拟机器环境，使得扩展 615 可直接从虚拟机器环境内访问硬件 620，从而绕过了由系统管理程序 613 执行的任何抽象。如上所述，系统

管理程序，如系统管理程序 613 可以是计算机可执行指令，它通过提供有限的操作系统功能，并通过提供对底层硬件，如硬件 620 的抽象访问，来管理虚拟机器环境。由此，系统管理程序 613 可用于将虚拟机器环境从底层硬件的细节屏蔽，从而允许虚拟机器软件应用为打算在其中执行的任何代码创建适当的虚拟机器环境。系统管理程序可在虚拟机器环境和底层硬件之间转换。

作为一个示例，虚拟机器环境可向虚拟操作系统进程 611 以及可在该进程中执行的任何程序呈现特定类型的 CPU，而同时，底层硬件 620 实际上包括一个完全不同类型的 CPU。系统管理程序 613 可被赋予将对虚拟机器环境内一种类型的 CPU 作出的请求转换成适当的请求，以与底层硬件 620 中存在的不同类型的 CPU 通信。然而，如上所述，由于某些操作系统扩展，如设备驱动程序，可能需要直接与底层硬件设备通信，因此由系统管理程序执行的抽象可阻止这类操作系统正确运作。因此，本发明的实施例构想了用于绕过系统管理程序并允许主宿在虚拟操作系统进程 611 中的扩展直接访问硬件的各种机制。

除虚拟机器进程 617 之外，图 7 也示出了主机操作系统进程 601，它也可使用硬件 620。硬件 620 被分割成两个块，以示出主机操作系统进程 610 和虚拟机器进程 617 之间的分时。由此，尽管虚拟机器进程 617 是通过系统管理程序 613 在硬件 620 上执行的，然而硬件 620 也不同时执行主机操作系统进程 601。相反，主机操作系统进程 601 的基础可被移除，并置于临时存储中。尽管未在图 7 中示出，这一基础可包括注册表条目、各种控制寄存器、中断分派例程、CPU 特权数据等等。一旦虚拟机器进程 617 结束了在硬件 620 上的执行，虚拟机器进程的基础可被移除，并置于临时存储中，并且主机操作系统进程 601 可被恢复，并允许在硬件上执行。

尽管图 7 的确示出了主机操作系统进程 601，然而采用代理 605，本发明的一个实施例构想的用于提供从虚拟环境内对硬件的直接访问的机制可在扩展故障隔离的环境外使用。具体地，上述机制可应用于一般的虚拟机器技术，从而允许虚拟机器主宿依赖于传统硬件设备、自定义硬件设备或非典型硬件设备的扩展和其它软件。通过消除了为这类设备设计抽象的需要，本发明的实施例提供了更简单的系统管理程序，以及更有效的虚拟机器设计。

本发明的一个实施例构想的用于提供从虚拟机器环境内对硬件的直接访问的一种机制要求系统管理程序修改页表映射，以允许访问对应于一个或多个硬件设备的物理存储器。如本领域的技术人员已知的，应用程序或扩展可通过访问适当的物理存储器来与硬件设备通信，这些适当的物理存储器通常可以是寄存器或位于硬件

设备本身或另一接口卡上的类似硬件。由此，例如，图 1 所示的说明性计算装置100通过向键盘设备驱动程序提供对用户输入接口60的物理存储器寄存器的访问，可允许键盘设备驱动程序与键盘162通信。或者，键盘设备驱动程序可访问RAM 132中的特定位置，并且可将额外的进程从键盘162传输到RAM中的该位置，以供设备驱动程序读取。

当虚拟机器环境中的代码，如虚拟机器进程617中的扩展615试图访问底层硬件时，系统管理程序613可执行对底层硬件适当的转换，并可访问物理寄存器本身，或将数据储存在虚拟机器进程存储器空间中，从该空间中，数据可由专用硬件或其类似物读取并复制到适当的物理寄存器中。为提供从虚拟机器环境内对底层硬件设备的直接访问，系统管理程序可避免执行任何转换，因为这类转换可能是不正确的，而是相反，系统管理程序可以用可将必要的物理存储器位置映射到诸如虚拟操作系统进程611所使用的存储器空间等适当的存储器空间中的方式，来修改页表映射。如上所述，页表映射确定了哪些物理存储器位置被分配给给定的进程。由此，通过修改页表映射以将对应于一个或多个设备的物理存储器位置放置到虚拟操作系统进程存储器空间中，系统管理程序使用虚拟操作系统可允许的扩展和应用程序直接访问硬件设备。

在一个示例中，扩展615可以是硬件设备驱动程序，并可由虚拟操作系统进程611主宿，它可使用已知的存储器读和写操作，可获得对作为硬件620的一部分的对应硬件设备的直接访问。提供硬件抽象的系统管理程序613可被设计成将来自扩展615的存储器读和写操作识别为应当被转换或被抽象的操作，并可允许他们通过以到达底层硬件。此外，由于系统管理程序613可在适当时修改页表映射，因此存储器读和写操作可在对应于扩展615试图控制的硬件设备的预期的寄存器或其它物理存储器位置上物理地执行。因此，扩展615具有对与硬件设备相对应的存储器寄存器或其它物理存储器位置的直接控制，并且由此，即使从虚拟机器环境内也可直接控制这些设备。

然而，通过改变页表映射，并允许扩展从虚拟机器环境内直接访问硬件，主机操作系统进程601可变得受由扩展引入的不稳定性的更多影响。例如，尽管虚拟机器进程617在硬件620上执行，然而扩展615可以用不正确的方式直接访问硬件620的某些组件，导致硬件组件不正确地运作，或甚至变得不可操作。随后，在主机操作系统进程601恢复在硬件620上执行之后，所访问的硬件组件可继续不正确地运作，并可能将不稳定性引入到主机操作系统进程中，或者它可保持不可操作，

并由此阻止主机操作系统进程执行所需要的任务。因此，本发明的一个实施例构想的一种机制提供了对上述页表映射修改的限制。例如，一种限制可以是仅在扩展所需的程度上修改页表映射。由此，如果扩展仅需要访问非常有限的地址范围，可能包括物理地位于硬件设备上或对设备的接口上的存储器寄存器的地址，则可仅在将有限的地址范围映射到虚拟机器进程存储器空间所需的程度上修改页表映射。另一中限制可以是临时限制，由此，可以仅在允许扩展完成其任务时修改页表映射。例如，当扩展 615 试图直接与硬件设备通信时，它可向系统管理程序 613 作出请求，指示它期望直接访问的时间的长度。这一请求可以直接作出，或通过主宿扩展 615 的虚拟操作系统进程 611 来作出。一旦系统管理程序 613 接收到该请求，它可为所请求的时间长度修改页表映射。

如本领域的技术人员所已知的，许多硬件设备通过接口硬件，如接口卡或其类似物连接到计算装置。通常，这类接口硬件被附加到已知的总线机制上，诸如上文描述的那些总线机制。总线地址可被映射到物理存储器，它可进一步由运行在计算装置上的软件来访问。因此，连接到总线的接口卡或其类似物的寄存器通常被称为“存储器映射的寄存器”，并可被映射到存储器的一个或多个物理页。然而，由于一组存储器映射的寄存器很少与另一组存储器映射的寄存器共享物理页，因此对页表映射的上述修改可在每一设备的基础上作出。

此外，本发明的一个实施例构想的一种机制要求使用虚拟地址转换，以允许某些存储器映射的寄存器仅对虚拟机器进程 617 可用。以此方式，主机操作系统进程 601 可避免处理对其它没有正确设备驱动程序的硬件，并且主宿在虚拟操作系统进程中的正确设备驱动程序可被授予对特定硬件设备的永久访问权限。

本发明的一个实施例构想的用于提供对硬件的虚拟机器直接访问的另一种机制允许从虚拟机器环境内访问输入/输出（I/O）端口，而不需要由系统管理程序 613 执行的仿真或其它修改。如本领域的技术人员所已知的，I/O 端口一般由地址或端口号来标识，并可通过已知的“IN”或“OUT”命令来访问。对于使用 I/O 端口来访问硬件设备的设备驱动程序或其它软件应用程序，IN 和 OUT 命令可以通过软件被转发到命令中指定的硬件设备上的物理端口或寄存器，或者它们可直接从发出命令的设备驱动程序或其它应用程序传递到所标识的端口或寄存器。某些类型的 CPU 允许通过使用任务段中的 I/O 位图来选择性地通过或直接访问，其中，I/O 位图指定了对其可通过软件来传递指令的地址，以及对其可直接将指令发送到物理端口或寄存器的地址。

在正常的操作中，虚拟机器的系统管理程序，如系统管理程序 613 将俘获 I/O 指令，或仿真 I/O 指令，以对虚拟机器环境内的软件正确地抽象底层硬件 620。如果系统管理程序 613 使用例如保护位图俘获了 I/O 指令，则本发明的一个实施例构想的一种机制要求修改保护位图，以提供“突破口（hole）”，或系统管理程序将不会俘获的 I/O 地址。由此，例如，如果扩展 615（可以是设备驱动程序）需要使用特定 I/O 地址直接访问硬件，则保护位图可从虚拟机器进程 617 内，如从扩展 615 内检测指定该 I/O 地址的 I/O 指令，并且保护位图可允许那些 I/O 指令通过系统管理程序而不被俘获。

然而，如果系统管理程序 613 仿真 I/O 指令，则本发明的一个实施例所考虑的机制要求修改系统管理程序，使得可在仿真前作出核查，并且对于指定特定地址的 I/O 指令不执行仿真。由此，例如，如果扩展 615 需要直接访问特定 I/O 地址处的硬件，系统管理程序 613 可核查接收的 I/O 指令中指定的 I/O 地址，并且如果接收到的 I/O 指令指定了扩展使用的特定地址，则系统管理程序可允许那些 I/O 指令通过而不被仿真。以此方式，即使从虚拟机器环境内，扩展也能直接访问硬件。

如可以见到的，上述机制可向扩展和其它软件应用程序提供甚至从虚拟机器内通过 I/O 端口对硬件的直接访问。然而，如果扩展或其它软件应用程序不被设计成通过 I/O 端口直接访问硬件，而是相反依赖于操作系统来执行这类硬件访问，则本发明的一个实施例构想的一种机制提供了系统管理程序 613 的修改，使得当虚拟操作系统进程 611 检测到来自扩展 615 或需要虚拟操作系统进程的其它软件应用程序对通过 I/O 端口直接访问硬件 620 的请求时，它可将请求传递到系统管理程序，后者然后可代表扩展或其它软件应用程序执行适当的 I/O 指令。或者，虚拟操作系统进程 611 可以自己执行 I/O 指令，并且系统管理程序 613 可诸如通过使用上文详细描述的机制让指令通过。

通常用于与硬件通信的另一种机制被称为直接存储器访问（DMA）。如本领域的技术人员已知的，DMA 可允许设备驱动程序或其它软件应用程序将数据传递到硬件设备或从硬件设备传递，而不对 CPU 形成负担。更具体地，DMA 提供了数据从一个或多个物理存储器段到硬件设备本身的物理寄存器或相似元件的传输。这一传输由计算装置上的电路协调，如专用 DMA 芯片，但是不需要 CPU 的协调。

一般而言，DMA 请求可以是由操作系统或软件应用程序提供给扩展的支持 API 的一部分。然而，由于上述虚拟支持 API 可由运行在虚拟机器环境内的虚拟操作系统进程提供，因此由虚拟机器环境内始发的 DMA 指定的存储器地址可能不是

应当向其定向硬件设备的正确物理地址。这可以由于众多因素，最值得注意的是，DMA 地址可能由系统管理程序作为系统管理程序执行的硬件抽象的一部分来修改。因此，对于要正确执行的 DMA，可在虚拟机器环境中使用正确的物理地址。

本发明的一个实施例构想的用于为 DMA 提供正确的物理地址的一种机制要求系统管理程序 613 或虚拟操作系统进程 611 向扩展 615 提供适合硬件的 DMA 访问的存储器区域。另外，为免于恶意或不正确的 DMA 请求，系统管理程序 613 也可阻挡或偏转到指向应当被保护的地址的任何 DMA 的正确地址。保护的地址可以例如在硬件 620 上首次执行系统管理程序 613 时事先确定。保护的地址也可仅仅是无法提供与其它硬件设备的 DMA 通信所必须的支持的那些存储器地址。作为又一替换方案，保护的地址可以是不参与当前 DMA 请求的地址的任一个或所有。通常，避免在 DMA 中使用受保护的地址可由计算装置 100 本身上的专用 DMA 芯片、存储器总线或类似的电路来实现。在这一情况下，系统管理程序 613 可获知这些块并使用它们，而非试图通过软件解决方案来阻挡或偏转 DMA。

为向扩展 615 提供适合 DMA 的存储器地址，本发明的一个实施例构想的一种机制要求系统管理程序 613 监视扩展 615 的操作，并检测即将到来的 DMA。或者，虚拟操作系统进程 611 可监视扩展的操作，并向系统管理程序 613 提供相关信息，或虚拟操作系统本身可检测即将到来的 DMA。如上所述，扩展一般使用支持 API 来获取对各种资源的访问。因此，即将到来的 DMA 可通过监控由扩展 615 通过虚拟操作系统进程提供的虚拟支持 API 调用的功能来检测。某些已知的功能一般用于设置 DMA，如，建立存储器块的请求，或对存储器的物理地址的请求。因此，向虚拟服务 API 请求这些功能的扩展可被确定为可能准备执行 DMA。

与连续地监视由扩展 615 作出的虚拟服务 API 功能调用相反，系统管理程序 613 或虚拟操作系统进程 611 可通过在调用一般用于设置 DMA 的已知功能时修改虚拟支持 API 以包括非法指令，来更有效地检测可能的 DMA。这一非法指令然后可生成俘获，并向系统管理程序或虚拟操作系统进程警告即将到来的 DMA。

一旦系统管理程序 613 或虚拟操作系统进程 611 诸如通过使用上述机制知道了即将到来的 DMA，则它可向扩展 615 提供适当的存储器地址范围，从而允许 DMA 正确地继续。在某些情况下，系统管理程序 613 可执行存储器交换或类似的存储器管理，以能够提供适当的存储器地址范围。或者，系统管理程序 613 可依赖于主机计算装置的已知分散/收集能力，以将要通过 DMA 发送到硬件设备或从硬件设备接收的信息放置到适当的存储器范围中。然而，因为扩展 615 由于一般由系

统管理程序 613 执行的转换而期望非寻常地址，上述的进一步图谋将不可能对扩展产生不利影响。

一旦向扩展 615 提供了存储器地址，可能必须防止额外的进程访问那些地址处的存储器，直到 DMA 完成。如本领域的技术人员已知的，在计算装置的正常操作期间，适合 DMA 的物理存储器一般不被映射出去。然而，虚拟机器环境内的存储器几乎总是通常由系统管理程序映射出去。因此，有必要以通常不需要对分配给虚拟机器环境中的其它进程的存储器完成的方式保护传递到扩展的存储器地址。这一保护可由系统管理程序来完成，它可使用一种通常被称为“锁定（pinning）”的机制来“锁定”指定的存储器地址，直到 DMA 完成。

当然，一旦 DMA 完成，系统管理程序可释放或“解锁（unpin）”指定的存储器位置。DMA 的完成可以用与上文详细描述的检测即将到来的 DMA 非常相同的方式来检测。例如，系统管理程序 613 或虚拟操作系统进程 611 可监视由扩展 615 调用的功能。诸如解除分配指定的存储器位置等功能可指示 DMA 已完成，并可用作对系统管理程序 613 能够解锁指定的存储器位置的指示。

本发明的实施例解决的与硬件的直接通信的另一方法涉及将硬件中断传送到在虚拟机器环境内执行的代码。如本领域的技术人员已知的，硬件中断可以是来自硬件设备、发送到适当的设备驱动程序或其它软件应用程序的信号，它一般需要某一类型的响应或确认。如上所述，由于主机操作系统可能无法支持特定硬件设备的正确的设备驱动程序或其它控制软件，因此需要将中断定向到在虚拟机器环境内执行的扩展。例如，图 1 的计算装置 100 被示出为连接到传统设备 199。如果操作系统 134 是现代操作系统，则它可能无法正确地支持传统设备 199 的设备驱动程序。因此，为使计算装置 100 的用户能够使用传统设备 199，可在虚拟环境内执行设备驱动程序或类似的控制软件。因此，从传统设备 199 接收到的任何中断可仅当它们被定向到虚拟机器进程时被正确地处理，并允许通过以到达设备驱动程序。

本发明的一个实施例构想的用于将中断定向到诸如扩展 615 等扩展的一种机制要求将接收到的中断与表或类似的结构进行比较，以确定虚拟机器进程 617 是应当处理该中断还是将其传递到主机操作系统进程 601。更具体地，在仅具有单个 CPU 的计算装置中，当虚拟机器进程 617 在 CPU 上执行时，或者当主机操作系统进程 601 在 CPU 上执行时，可接收中断。本发明的机制可应用于中断到达而虚拟机器进程 617 在 CPU 上执行的情况。在这一情况下，系统管理程序 613 可确定中断的原因或目标。系统管理程序 613 然后可确定中断是否由虚拟机器环境中的扩

展，如扩展 615，通过例如执行表查找来适当地处理。如果中断由扩展 615 适当地处理，则系统管理程序 613 可将中断传递到虚拟机器进程 617，并由此传递到扩展。如果中断由与主机操作系统进程 601 相关联的扩展或其它软件应用程序适当地处理，则系统管理程序 613 可完成硬件 20 上的虚拟机器进程 617 的执行，并允许主机操作系统进程恢复硬件上的执行，并以适当的方式中断。

如果系统管理程序 613 将中断传递到虚拟机器进程 617，它可修改中断所到达的中断线的数量，以维持与虚拟操作系统进程 611 的兼容性。由此，当启用中断线时，系统管理程序 613 可验证中断线信息对应于物理中断线。系统管理程序 613 然后可在物理中断线和仿真的中断线之间转换。

由于虚拟机器可仿真不同于其上执行虚拟机器进程 617 的硬件 620 的硬件，因此系统管理程序 613 可能需要将单个虚拟机器指令仿真为主机硬件上的多个指令。例如，如果虚拟机器正在仿真正在执行其上的物理 CPU 的不同类型的 CPU，则当由正在被仿真的 CPU 执行时可能仅需要单个 CPU 周期的指令在由物理 CPU 执行时可能需要多个 CPU 周期。在这一情况下，对系统管理程序 613 重要的是以单一的方式处理物理 CPU 的多个 CPU 周期，以维持与仿真的 CPU 的兼容性。由此，如果当系统管理程序 613 在执行物理 CPU 上与仿真的 CPU 的单个周期相关的一系列周期的中间时硬件中断到达，则系统管理程序可忽略、排队或延迟该中断，直到该 CPU 周期序列完成。

本发明的一个实施例构想的用于将中断定向到虚拟机器进程中的扩展的另外的机制要求主机操作系统进程在将控制传递到虚拟机器进程之前延迟中断、一旦接收中断就将控制传递到虚拟机器进程、或试图在主机进程内用指向虚拟机器进程的适当指针来执行扩展。如上所述，在仅具有单个 CPU 的计算装置内，当虚拟机器进程 617 在 CPU 上执行时，或者当主机操作系统进程 601 在 CPU 上执行时，可接收中断。本发明的机制可应用于当主机操作系统进程 601 在 CPU 上执行时中断到达的情况。最初，主机操作系统可能已经预定义了用于将中断定向到适当设备驱动程序的过程。例如，这类过程可在主机操作系统的引导过程期间建立，如当加载设备驱动程序时。因此，扩展 615 的调用可试图充分利用这些预定义的过程，并指示主机操作系统进程 601，从特定硬件设备接收到的中断应当被定向到虚拟机器进程 617。

因此，当主机操作系统进程 601 在 CPU 上执行时接收到应当被发送到扩展 615 的中断，主机操作系统进程可执行类似于当它接收任何其它中断时所执行的过程，

而它可确定处理该中断的适当软件正在虚拟机器进程 617 内执行的情况除外。主机操作系统进程 601 然后可试图例如通过禁用中断、完成一个或多个任务、将执行切换到虚拟机器进程 617 以及然后重新启用中断，将中断传递到扩展 615。由于当重新启用中断时虚拟机器进程 617 因此将会在 CPU 上执行，中断可由虚拟机器进程 617 接收，并由其以上文详细描述的方式来处理。

如本领域的技术人员所已知的，硬件设备一般可使用两种不同类型的中断：保持活动直到它被处理或响应的永久中断，以及可抛出闭锁然后结束的瞬时中断。使用上述机制，虚拟机器进程 617 可在一旦重新启用中断后就检测到永久中断，因为永久中断从未被去激活。由此，对于永久中断，虚拟机器进程 617 可使用上文详细描述的机制来以如同当虚拟机器进程在 CPU 上执行时它最初到达的相同方式来处理该中断。然而，对于瞬时中断，指示中断出现的闭锁会变得未完成。因此，除非出现另一中断以重新抛出闭锁，否则如果当主机操作系统进程 601 在 CPU 上执行时出现中断，虚拟机器进程 617 将从未获知该中断。由此，主机操作系统进程 601 可跟踪或储存在将执行传递到虚拟机器进程 617 之前出现的一个或多个瞬时中断。主机操作系统进程 601 可将信息传递到系统管理程序 613，以通知系统管理程序出现了瞬时中断，并可在适当时提供瞬时中断的数量。一旦虚拟机器进程 617 在 CPU 上执行，系统管理程序 613 然后可依次仿真瞬时中断，并允许扩展 615 以同样的方法响应它们。一旦系统管理程序 613 完成了对瞬时中断的仿真，它然后可重新启用中断。

在某些情况下，可能需要用高于上述过程可提供的速度来处理或响应硬件中断。在这一情况下，本发明的一个实施例构想的一种机制要求当检测到由诸如扩展 615 等运行在虚拟机器进程中的扩展正确处理的中断，主机操作系统进程 601 立即将执行传递到虚拟机器进程 617，而非禁用中断，并试图使用上述机制来完成一个或多个任务。然后，系统管理程序 613 可以是单线程的，它可延迟中断的检测，并且如果系统管理程序正在等待响应或某一其它信息，可因此延迟中断的服务。

为避免由于系统管理程序的单线程而引起的延迟，本发明的一个实施例构想的上述机制的变异要求系统管理程序 613 仿真多个 CPU 计算设备，并且虚拟操作系统进程 611 能够在多 CPU 环境中操作。另外，系统管理程序 613 可以至少一个被仿真的 CPU 保持在可接受中断的状态中的方式构造指令的执行。例如，如上所述，可通过将命令传递到虚拟机器进程，然后高速缓存主机操作系统进程的基础并在硬件 620 上执行虚拟机器进程，来从主机操作系统进程 601 调用虚拟机器进程

617。系统管理程序 613 可通过将从主机操作系统进程 601 接收到的命令传递到其它仿真的 CPU，将一个仿真的 CPU 保持在可接受中断的状态。因此，由于被保持的 CPU 不被允许来处理来自主机操作系统进程 601 的命令，它可维持它可立即处理接收到的中断的状态。

因此，如果中断要到达，而底层主机操作系统进程 601 正在硬件 620 上执行，并且中断需要低等待时间，则主机操作系统进程尽可能快地将控制传递到虚拟机器进程 617。一旦虚拟机器进程 617 开始在硬件 620 上执行，虚拟机器进程的至少一个仿真的 CPU 处于它可接受中断的状态。由此，即使其它仿真的 CPU 处于它们可执行功能或等待响应的状态，中断可以由被保留用于中断的至少一个仿真的 CPU 以有效的方式来处理。系统管理程序 613 和虚拟操作系统进程 611 然后可以用上文详细描述的方式执行必要的步骤以将中断传送到适当的软件，如扩展 615。此外，同样如上所述，由于系统管理程序 613 可能需要物理存储器被锁定，因此可允许接收中断的仿真的 CPU 在将控制返回到另一仿真的 CPU 或另一进程之前完成对中断的处理。以此方式，至少一个仿真的 CPU 可被保存用于及时处理中断。

本发明的一个实施例构想的提供硬件中断的低等待时间处理的另一种机制要求主机操作系统进程 601 从扩展 615 中取出用于中断服务例程的代码，并用指回虚拟机器进程 617 的适当数据指针自己执行该代码。例如，主机操作系统进程 601 可从虚拟机器进程 617 的存储器空间的起始跟踪适当的中断服务例程。一旦被定位，那些中断服务例程可被复制到主机操作系统进程 601 中，并在那里被执行以用非常低的等待时间来处理中断。

由于中断服务例程旨在虚拟机器进程 617 的处理空间内执行，因此主机操作系统进程 601 在它复制那些例程并执行它们时，可提供指回虚拟机器进程的数据指针，使得例程可正确地运作。例如，主机操作系统进程 601 可改变中断服务例程或页表映射的适当指令，以引用虚拟机器进程 617 内的存储器。可使用已知的软件故障隔离技术来修改适当的指令并提供故障隔离的措施。如本领域的技术人员所已知的，软件的执行可通过在被监视的软件的命令之间插入适当的命令来监视。为避免对重新编译被监视的软件的需求，插入的命令可以是可被插入到已编译代码的低级命令。例如，诸如通过将地址与已知的地址范围比较来核查要访问的存储器位置的地址的插入的指令可以优于通过将位置的内容复制到寄存器来访问特定存储器位置的低级指令。如果存储器位置是不正确的位置，例如，如果它位于适当的地址范围之外，则可作出修改以将适当的地址替代到访问请求中。以此方式，每一存储器

访问指令可被修改，以访问正确的存储器位置，尽管事实是中断处理例程可以在主机操作系统进程 601 而非虚拟机器进程 617 中执行。

如上所述，尽管中断处理例程直接在主机操作系统进程 601 中执行，软件故障隔离技术也可提供故障隔离的措施。例如，软件故障隔离的一方面通过在每一存储器写指令前插入低级指令以确保写执行所定向的位置是正确的位置来实现。如本领域的技术人员所已知的，由于故障缘由数据被写入到不正确存储器位置的数据中导致的故障，软件故障经常导致不稳定性。此外，这一不正确的写指令是很难检测到的，因为在完成直接前一指令之前，向其写入数据的地址可能无法被确定。通过紧靠任何存储器写之前插入上述指令，可例如通过将这些写指令所定向的存储器地址与已知的存储器地址范围进行比较，来核查这些存储器地址。因此，写被定向到已知范围之外的存储器位置的指示可表明写指令是不正确的，且可能导致不稳定性。因此，写指令可被修改或中止，并可实现故障隔离措施。也可使用故障隔离措施的其它方面，包括沙箱（sandbox）控制流、特权指令的使用等等。关于软件故障隔离的各方面的额外信息，包括上述信息，可在 Wahbe 等人的美国专利号 5,761,477 中找到，其内容通过整体引用结合于此，以进一步解释或描述包含在本发明中与其揭示相一致的任何教导或建议。

然而，某些计算装置可具有多个物理 CPU，在这一情况下，上述机制中的某一些可能不是必需的。例如，在具有多个物理 CPU 的计算装置中，单个 CPU 可能总是执行虚拟机器进程 617。在这一情况下，本发明的一个实施例所构想的一种机制要求硬件中断的控制机制（通常是作为计算装置本身的一部分的专用电路）将需要诸如扩展 615 等扩展的所有中断定向到虚拟机器进程 617 总是在其上运行的物理 CPU。即使虚拟机器进程 617 与其它进程共享物理 CPU，但是总是共享同一物理 CPU，将需要扩展 615 的所有中断定向到该物理 CPU 仍可在与上述用于将中断传递到适当的虚拟机器进程的机制相组合时提供最优的解决方案，即使它当前不在物理 CPU 上执行也是如此。

然而，如果虚拟机器进程 617 可以在多个物理 CPU 上的任何一个上执行，则可使用处理器间消息来允许任一处理器响应硬件中断。例如，如果虚拟机器进程 617 碰巧在第一物理 CPU 上执行，而可由扩展 615 处理的中断到达第二物理 CPU，则第二物理 CPU 可将相关信息传递到第一物理 CPU，以允许扩展处理硬件中断。如本领域的技术人员所已知的，很难物理地将硬件中断从一个物理 CPU 转发到另一个。因此，通过使用处理器间消息，中断可如同它到达正确的物理 CPU 那样被

处理。

鉴于可应用本发明的原理的许多可能的实施例，应当认识到，此处相对于附图所描述的实施例仅意味着说明性的，并不应当作为对本发明的范围的限制。例如，本领域的技术人员可以认识到，所示的实施例中以软件示出的某些元素可以用硬件来实现，反之亦然，或者所示的实施例可以在不脱离本发明的精神的情况下在排列和细节上作出修改。类似地，应当认识到，在虚拟机器环境中所描述的机制可以应用于在公用操作系统之上创建的虚拟环境，反之亦然。例如，上文结合虚拟机器环境描述的软件故障隔离技术可等效地应用于其中可能不期望过多环境变换的任何情况，包括即使两个进程共享一个公用的底层操作系统，也将扩展例程从虚拟进程复制到主机进程的情况。因此，此处所描述的本发明构想所有这样的实施例落入所附权利要求书及其等效技术方案的范围之内。

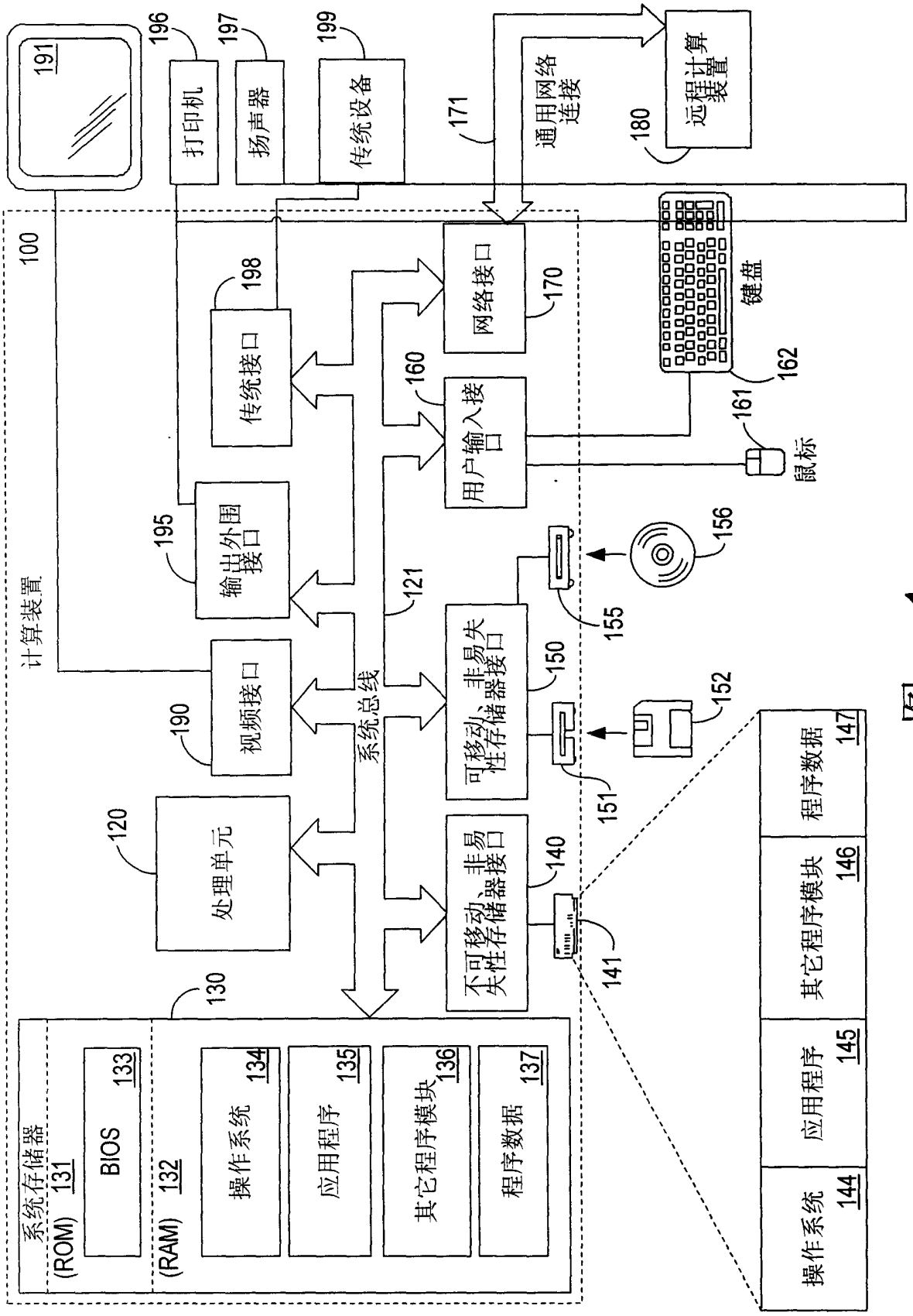

计算装置
100
191
196
打印机
197
扬声器
199
传统设备
171
通用网络连接
180
远程计算装置
198
传统接口
网络接口
170
160
用户输入接口
161
162
鼠标
键盘
195
输出外围接口
121
系统总线
190
视频接口
156
155
150
可移动、非易失性存储器接口
120
处理单元
152
151
140
不可移动、非易失性存储器接口
141
130
系统存储器
(ROM) 131
BIOS 133
(RAM) 132
操作系统 134
应用程序 135
其它程序模块 136
程序数据 137
操作系统 144
应用程序 145
其它程序模块 146
程序数据 147

图 1

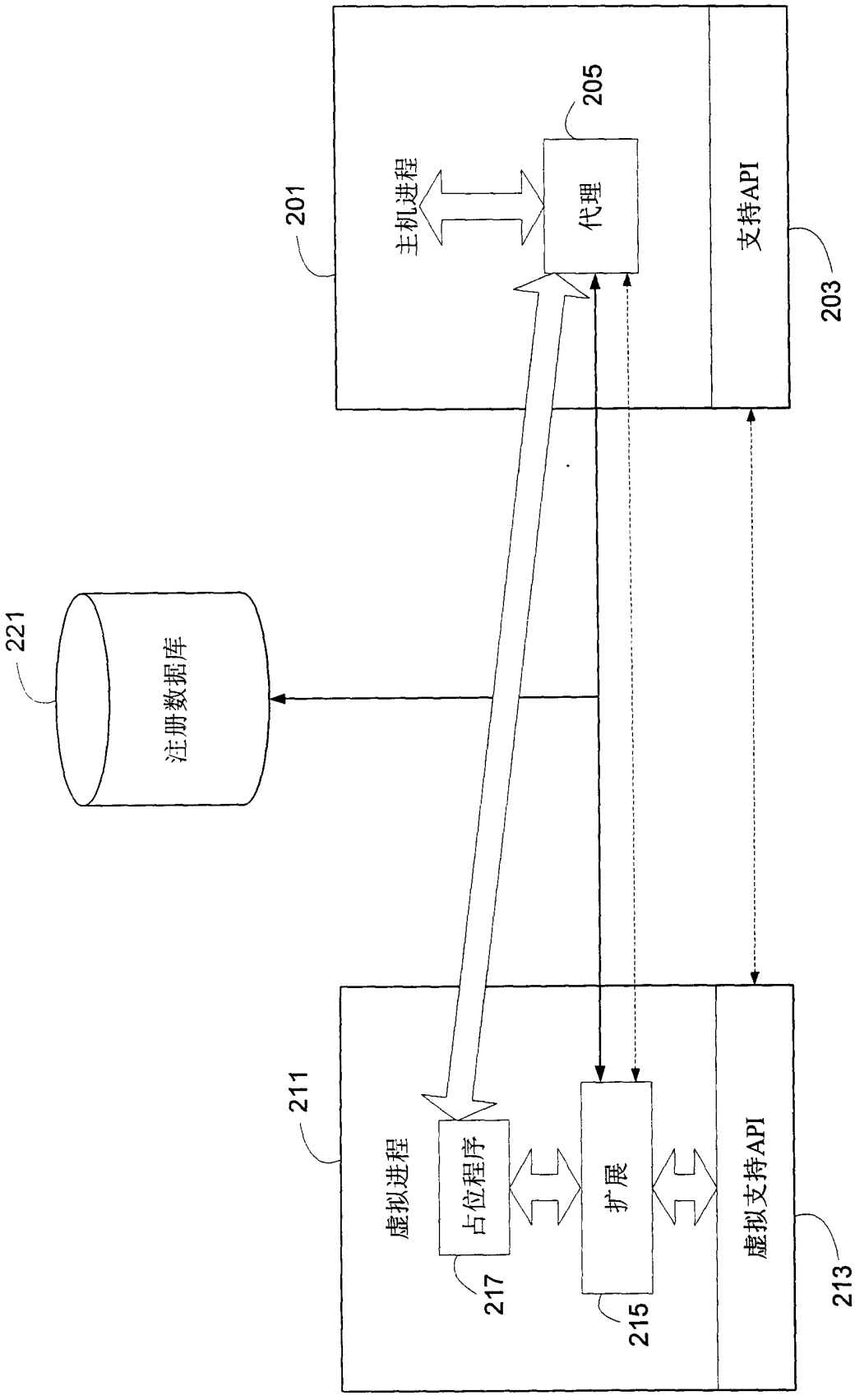

201
主机进程
205
代理
支持API
203
221
注册数据库
211
虚拟进程
占位程序
217
扩展
215
虚拟支持API
213

图 2

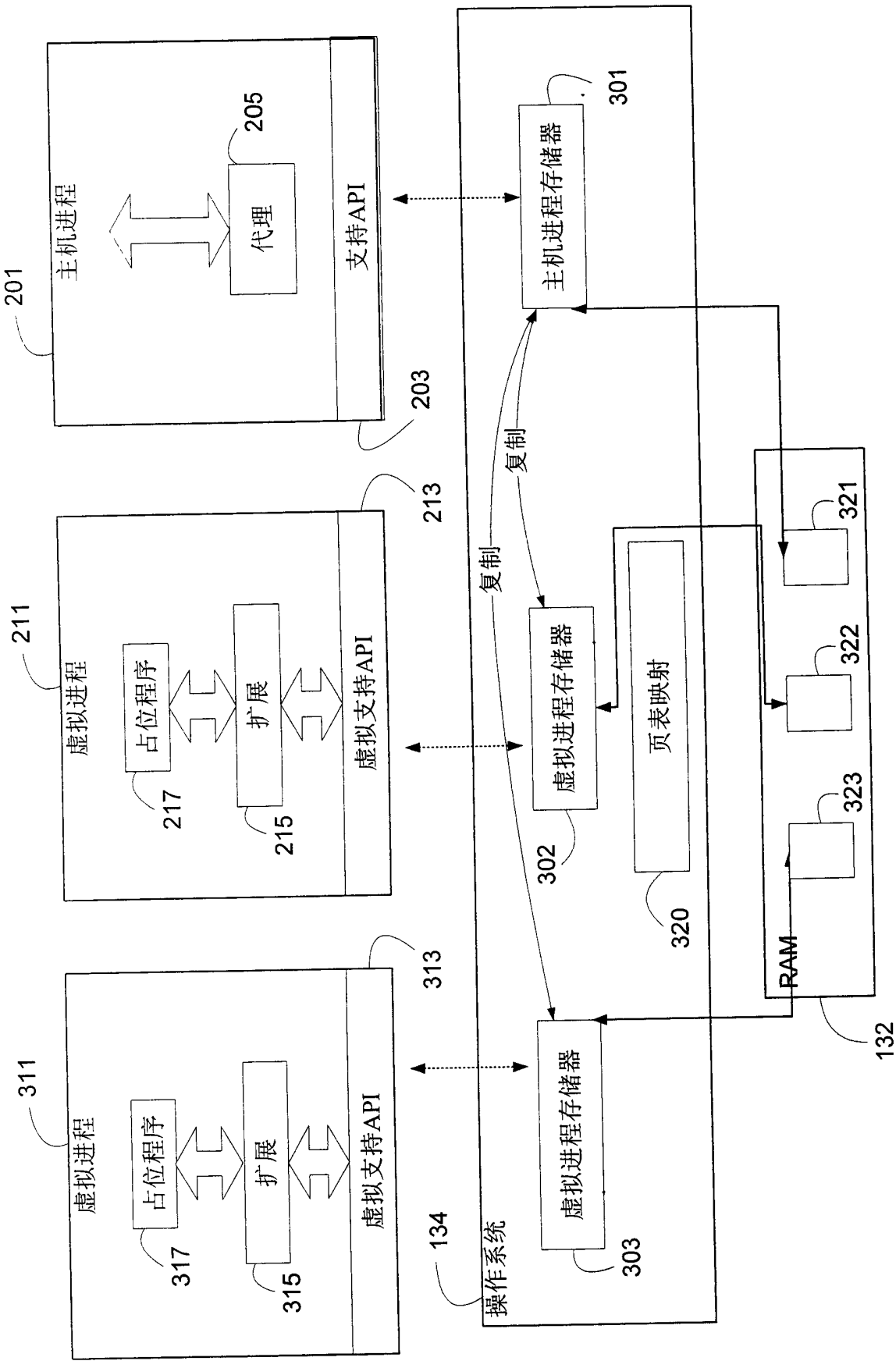

主机进程
201
代理
205
支持API
203
虚拟进程
211
占位程序
217
扩展
215
虚拟支持API
213
虚拟进程
311
占位程序
317
扩展
315
虚拟支持API
313
操作系统
134
主机进程存储器
301
复制
复制
虚拟进程存储器
302
页表映射
320
虚拟进程存储器
303
RAM
321
322
323
132

图 3

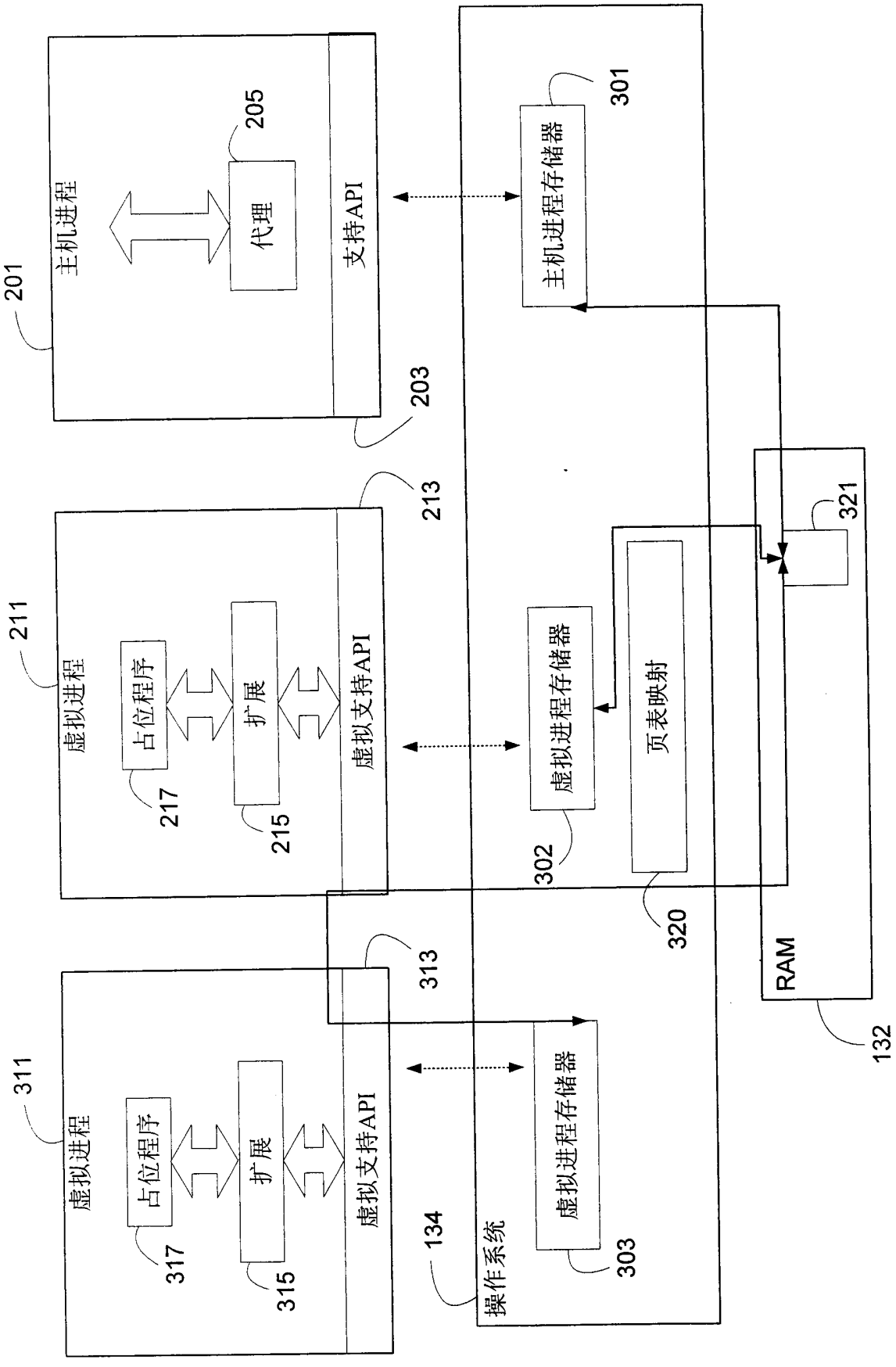

主机进程
201
代理
205
支持API
203
主机进程存储器
301
虚拟进程
211
占位程序
217
扩展
215
虚拟支持API
213
虚拟进程存储器
302
页表映射
320
321
RAM
132
虚拟进程
311
占位程序
317
扩展
315
虚拟支持API
313
操作系统
134
虚拟进程存储器
303

图 4

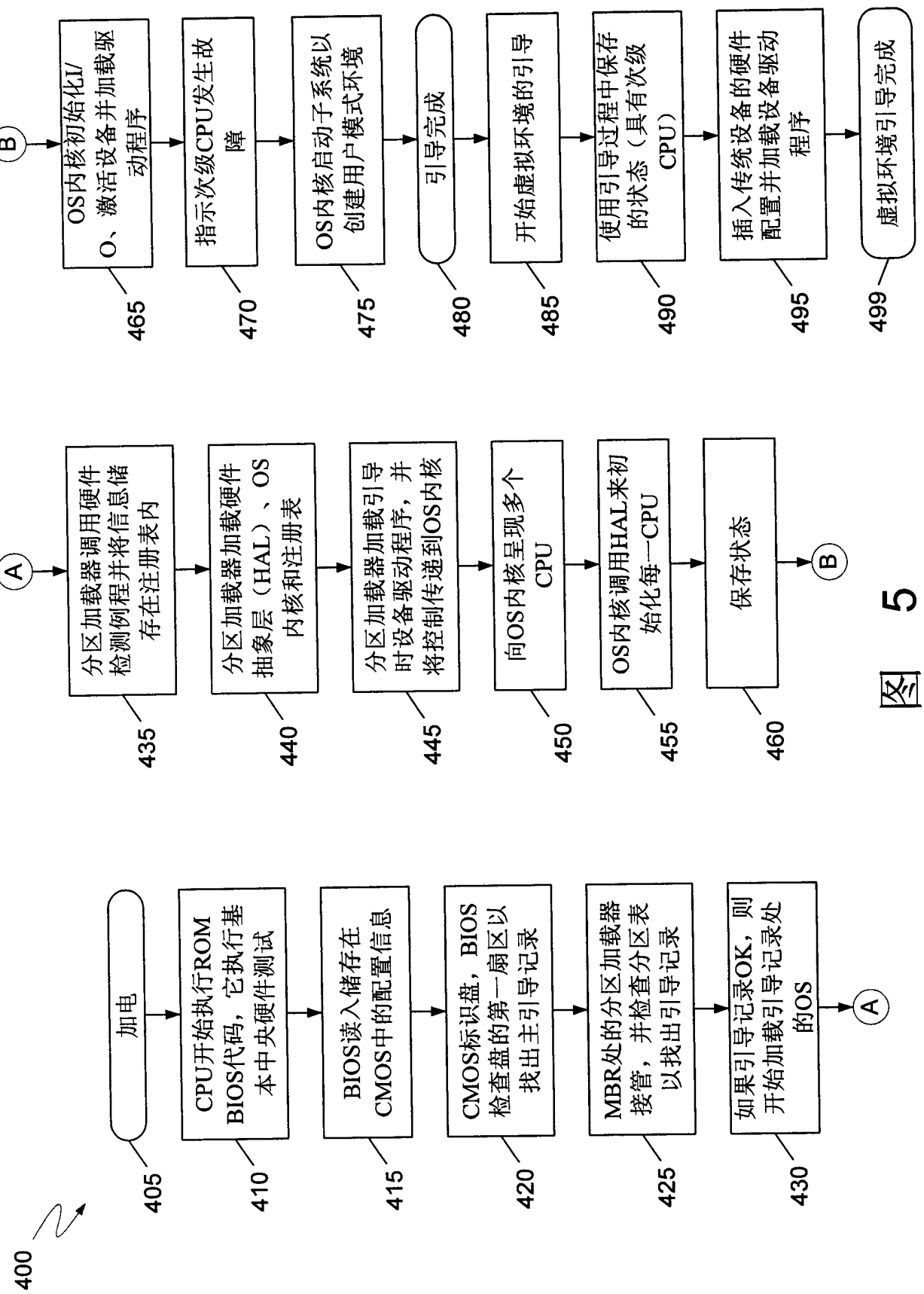

400
405
加电
410
CPU开始执行ROM BIOS代码，它执行基本中央硬件测试
415
BIOS读入储存在CMOS中的配置信息
420
CMOS标识盘，BIOS检查盘的第一扇区以找出主引导记录
425
MBR处的分区加载器接管，并检查分区表以找出引导记录
430
如果引导记录OK，则开始加载引导记录处的OS
A
435
分区加载器调用硬件检测例程并将信息储存在注册表内
440
分区加载器加载硬件抽象层（HAL）、OS内核和注册表
445
分区加载器加载引导时设备驱动程序，并将控制传递到OS内核
450
向OS内核呈现多个CPU
455
OS内核调用HAL来初始化每一CPU
460
保存状态
B
465
OS内核初始化I/O、激活设备并加载驱动程序
470
指示次级CPU发生故障
475
OS内核启动子系统以创建用户模式环境
480
引导完成
485
开始虚拟环境的引导
490
使用引导过程中保存的状态（具有次级CPU）
495
插入传统设备的硬件配置并加载设备驱动程序
499
虚拟环境引导完成

图 5

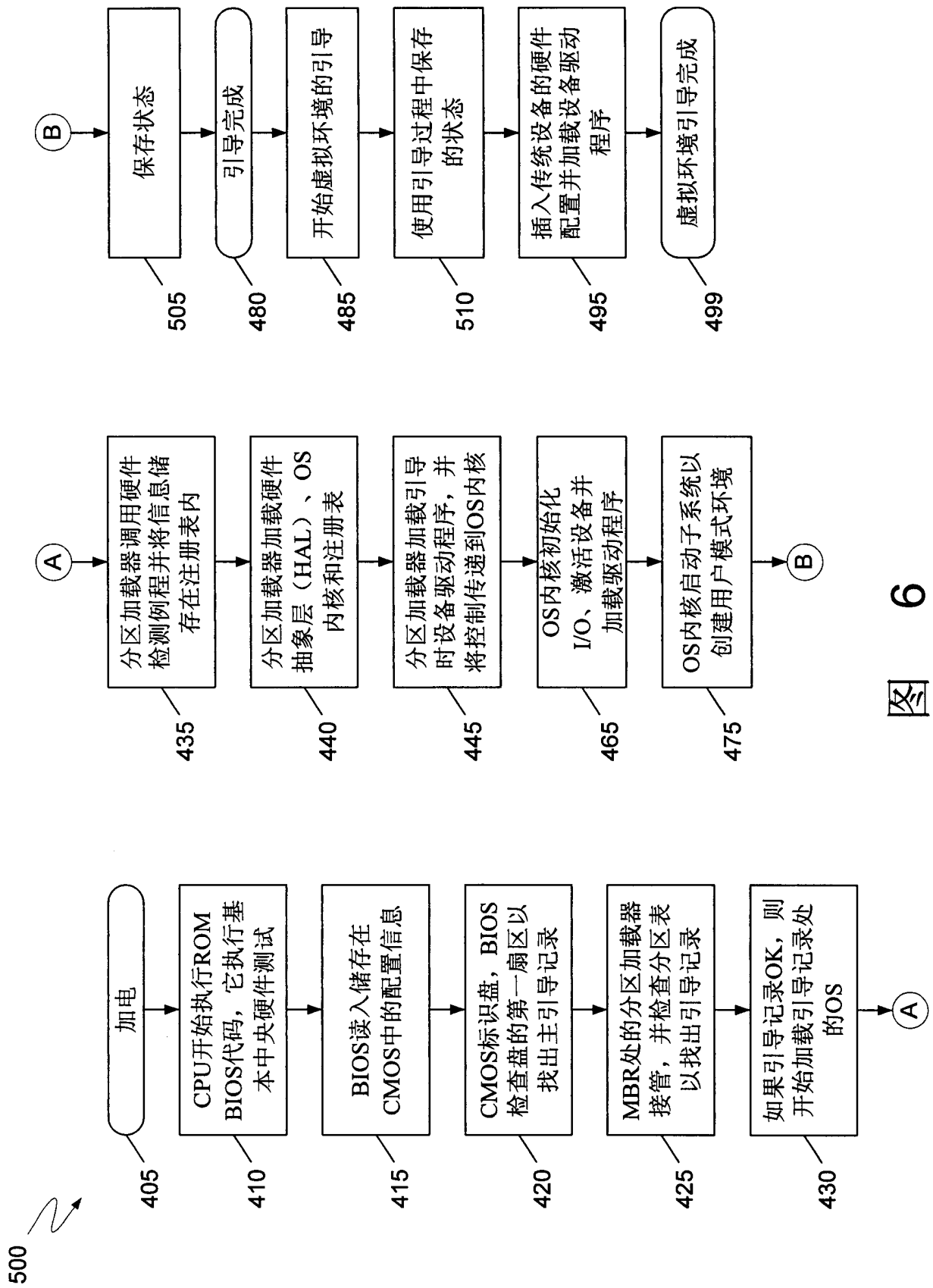

500
加电
405
CPU开始执行ROM BIOS代码，它执行基本中央硬件测试
410
BIOS读入储存在CMOS中的配置信息
415
CMOS标识盘，BIOS检查盘的第一扇区以找出主引导记录
420
MBR处的分区加载器接管，并检查分区表以找出引导记录
425
如果引导记录OK，则开始加载引导记录处的OS
430
A
A
分区加载器调用硬件检测例程并将信息储存在注册表内
435
分区加载器加载硬件抽象层（HAL）、OS内核和注册表
440
分区加载器加载引导时设备驱动程序，并将控制传递到OS内核
445
OS内核初始化I/O、激活设备并加载驱动程序
465
OS内核启动子系统以创建用户模式环境
475
B
B
保存状态
505
引导完成
480
开始虚拟环境的引导
485
使用引导过程中保存的状态
510
插入传统设备的硬件配置并加载设备驱动程序
495
虚拟环境引导完成
499

图 6

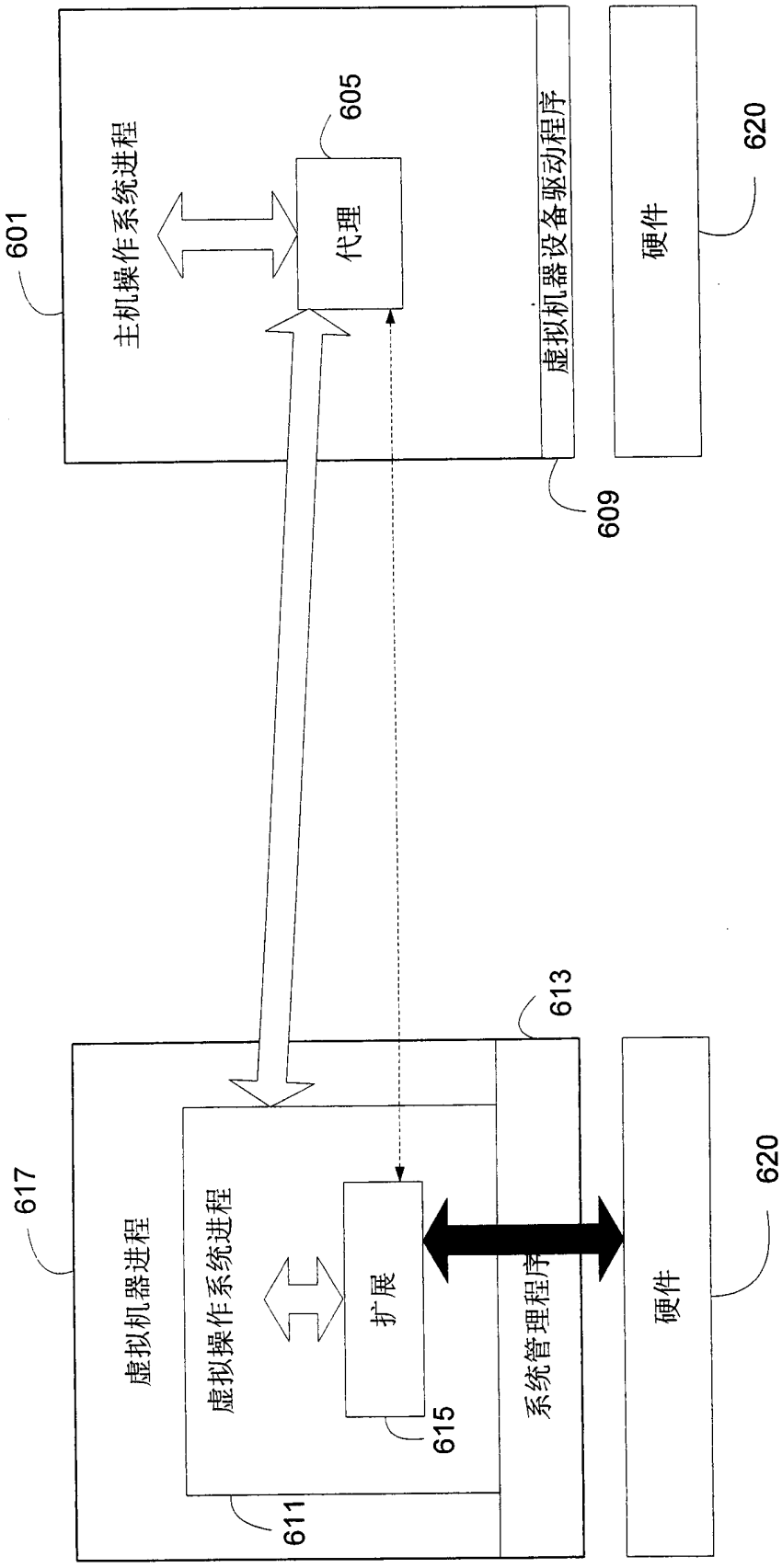

601
主机操作系统进程
605
代理
虚拟机器设备驱动程序
609
硬件
620
617
虚拟机器进程
虚拟操作系统进程
扩展
615
611
613
系统管理程序
硬件
620

图 7