

(12) 发明专利

(10) 授权公告号 CN 101826030 B

(45) 授权公告日 2012. 12. 05

(21) 申请号 201010149539. 5

(22) 申请日 2001. 12. 20

(30) 优先权数据

09/752587 2000. 12. 27 US

(62) 分案原申请数据

01822837. 2 2001. 12. 20

(73) 专利权人 英特尔公司

地址 美国加利福尼亚州

(72) 发明人 S·仇 G·奈格尔

E·科塔-罗布勒斯 S·耶亚辛

R·乌利 A·卡吉 S·舍恩博格

M·科祖彻

(74) 专利代理机构 中国专利代理(香港)有限公司

司 72001

代理人 刘春元 徐予红

(51) Int. Cl.

G06F 9/455(2006. 01)

(56) 对比文件

GB 2256513 A, 1992. 12. 09, 全文.

US 5504922 A, 1996. 04. 02, 全文.

lawton et al. 《le operating systems concurrently on an IA32 PC using virtualization techniques》. 《http://www.plex86.org/research/paper.txt》. 1999, 全文.

审查员 何明伦

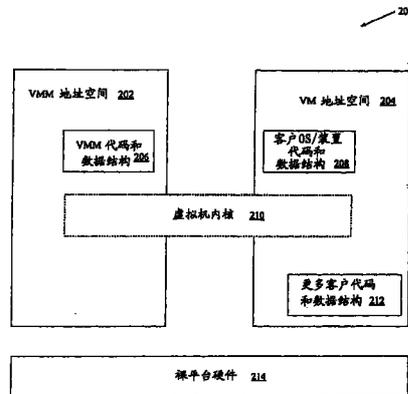
权利要求书 3 页 说明书 7 页 附图 7 页

(54) 发明名称

解决虚拟机监控器和客户操作系统间地址空间冲突的方法

(57) 摘要

在一个实施例中,一种用于解决地址空间冲突的方法包含:检测一个客户操作系统试图访问由一个虚拟机监控器的一个第一部分占据的一个区域,以及在该第一地址空间中重新定位该虚拟机监控器的第一部分以允许客户操作系统访问先前由该虚拟机监控器的第一部分占据的区域。



1. 一种用于解决虚拟机监控器 VMM 和客户操作系统 OS 间地址空间冲突的方法,包含:
 - 为客户操作系统分配第一地址空间;
 - 为虚拟机监控器 VMM 分配第二地址空间;
 - 将 VMM 的第一部分映射到第一地址空间和第二地址空间;
 - 在第二地址空间中定位 VMM 的第二部分;
 - 检测客户操作系统试图访问在第一地址空间内由 VMM 的第一部分占据的区域;以及
 - 在第一地址空间内重定位 VMM 的第一部分以允许客户操作系统访问先前由 VMM 的第一部分占据的区域。
2. 如权利要求 1 所述的方法,其中 VMM 的第一部分包含一组要求驻留在第一地址空间中的 VMM 代码和数据结构。
3. 如权利要求 1 所述的方法,其中 VMM 的第一部分包含一组陷阱处理程序和一种中断描述符表 (IDT)。
4. 如权利要求 1 所述的方法,进一步包含将 VMM 分成第一部分和第二部分。
5. 如权利要求 1 所述的方法,进一步包含:
 - 当由客户操作系统发起的一个事件能够引起在客户操作系统和 VMM 之间的一个地址空间冲突时,接收对该事件的控制。
6. 如权利要求 5 所述的方法,其中接收控制进一步包含:
 - 把由 VMM 的第一部分占据的部分的访问权限设置为一个比与客户操作系统有关的特权级别更高的特权级别;以及
 - 接收由客户操作系统试图访问具有比与该客户操作系统有关的特权级别更高的特权级别的硬件资源所引起的陷阱。
7. 如权利要求 6 所述的方法,进一步包含:
 - 确定该陷阱能够由 VMM 的第一部分处理;
 - 执行与该陷阱有关的代码;以及
 - 把对该事件的控制返回到客户操作系统。
8. 如权利要求 6 所述的方法,进一步包含:
 - 确定该陷阱应当由 VMM 的第二部分处理;
 - 把该陷阱传送到 VMM 的第二部分;
 - 在与该陷阱有关的代码由 VMM 的第二部分执行了之后把对该事件的控制传递到客户操作系统。
9. 如权利要求 1 所述的方法,其中重定位 VMM 的第一部分进一步包含:
 - 在第一地址空间内查找未使用的区域;以及
 - 把该 VMM 的第一部分重新映射到该未使用的区域中。
10. 如权利要求 1 所述的方法,其中重定位 VMM 的第一部分进一步包含:
 - 确定在该第一地址空间中没有未使用的区域存在;
 - 在第一地址空间内选择随机区域;
 - 复制位于该随机区域处的存储器内容到第二地址空间;以及
 - 把该 VMM 的第一部分重新映射到该随机区域中。
11. 如权利要求 10 所述的方法,进一步包含:

接收对起源于客户操作系统的事件的控制,该事件对应于客户操作系统试图访问先前位于该随机区域处的存储器内容;以及

访问在第二地址空间中的、复制的存储器内容。

12. 如权利要求 11 所述的方法,进一步包含周期性地把该 VMM 的第一部分重定位到在第一地址空间内的随机区域,直到找到一个很少被访问的区域为止。

13. 一种用于解决虚拟机监控器 VMM 和客户操作系统 OS 间地址空间冲突的装置,包含:

用于为客户操作系统分配第一地址空间的部件;

用于为虚拟机监控器 VMM 分配第二地址空间的部件;

用于将 VMM 的第一部分映射到第一地址空间和第二地址空间的部件;

用于在第二地址空间中定位 VMM 的第二部分的部件;

用于检测客户操作系统试图访问在第一地址空间内由 VMM 的第一部分占据的区域的部件;以及

用于在第一地址空间内重定位 VMM 的第一部分以允许客户操作系统访问先前由 VMM 的第一部分占据的区域的部件。

14. 如权利要求 13 所述的装置,其中 VMM 的第一部分包含一组要求驻留在第一地址空间中的 VMM 代码和数据结构。

15. 如权利要求 13 所述的装置,其中 VMM 的第一部分包含一组陷阱处理程序和一种中断描述符表 (IDT)。

16. 如权利要求 13 所述的装置,还包括用于把 VMM 划分成为第一部分和第二部分的部件。

17. 如权利要求 13 所述的装置,还包括用于当起源于客户操作系统的事件能够引起在客户操作系统和 VMM 之间的地址空间冲突时将接收对该事件的控制的部件。

18. 如权利要求 13 所述的装置,还包括接收部件,所述接收部件用于通过把由 VMM 的第一部分占据的部分的访问权限设置为一个比与该客户操作系统有关的特权级别更高的特权级别,以及通过接收由客户操作系统试图访问具有比与该客户操作系统有关的特权级别更高的特权级别的硬件资源所引起的陷阱,来接收控制。

19. 如权利要求 18 所述的装置,还包括用于决定该陷阱能够由 VMM 的第一部分处理,执行与该陷阱有关的代码,以及把对该事件的控制返回到客户操作系统的部件。

20. 如权利要求 18 所述的装置,还包括用于确定该陷阱应当由 VMM 的第二部分处理、把该陷阱传递到 VMM 的第二部分、以及在与该陷阱有关的代码由 VMM 的第二部分执行了之后把对该事件的控制传递到该客户操作系统的部件。

21. 如权利要求 13 所述的装置,还包括用于通过在第一地址空间内查找未使用的区域以及把该 VMM 的第一部分重新映射到未使用的区域中来重定位 VMM 的第一部分的部件。

22. 如权利要求 13 所述的装置,还包括用于通过确定在该第一地址空间中没有未使用的区域存在、在该第一地址空间内选择随机区域、复制位于该随机区域处的存储器内容到第二地址空间中、以及把该 VMM 的第一部分重新映射到该随机区域中来重定位 VMM 的第一部分的部件。

23. 如权利要求 22 所述的装置,还包括用于接收对起源于客户操作系统的事件的控制

以及访问在第二地址空间中的、复制的存储器内容的部件,其中该事件对应于客户操作系统试图访问先前位于该随机区域处的存储器内容。

24. 如权利要求 13 所述的装置,还包括用于周期性地把该 VMM 的第一部分重定位到在第一地址空间内的随机区域直到找到一个很少被访问的区域为止的部件。

解决虚拟机监控器和客户操作系统间地址空间冲突的方法

[0001] 本申请是申请日为 2001 年 12 月 20 日、申请号为 01822837.2、发明名称为“用于解决在一个虚拟机监控器和一个客户操作系统之间的地址空间冲突的方法”的发明专利申请的分案申请。

技术领域

[0002] 本发明通常涉及虚拟机,更具体地说涉及解决在一个虚拟机监控器和一个客户操作系统之间的地址空间冲突。

背景技术

[0003] 传统的虚拟机监控器 (VMM) 通常运行在一台计算机上并且向其它软件给出一个或多个虚拟机抽象。每个虚拟机可以起一个自给平台的作用,其运行它自己的“客户操作系统”(即,一个由 VMM 作主的操作系统)。客户操作系统期望就好像它正运行在一台专用计算机而不是一个虚拟机上那样进行操作。即,客户操作系统期望控制各种计算机操作并且在这些操作期间对计算机物理存储器和存储器映射的 I/O 设备具有不受限制的访问。然而,在一个虚拟机环境中,VMM 应当能够具有对计算机资源的最终控制以提供对虚拟机以及在虚拟机之间的保护。为了实现这个,VMM 一般截取和判优所有由客户操作系统进行的、对计算机资源的访问。

[0004] 利用现有的处理器(例如,IA-32 微处理器),VMM 不能截取客户操作系统对硬件资源的访问,除非该 VMM 代码和/或数据结构的一部分位于和客户操作系统相同的虚拟地址空间中。然而,客户操作系统不期望 VMM 代码和/或数据结构驻留在客户操作系统的地址空间中并且能够试图访问由该 VMM 在这个地址空间中占有的一个区域,这会导致在该客户操作系统和 VMM 之间的地址空间冲突。这个冲突可能导致由 VMM 或者客户操作系统执行的操作的异常终结。

[0005] 因此,需要一种将检测和解决在一个 VMM 和一个客户操作系统之间的地址空间冲突的机制。

附图说明

[0006] 本发明在附图的图表中通过示例进行说明但不局限于此,其中类似的参考数字涉及类似的单元,附图包含:

[0007] 图 1 说明了一个虚拟机环境的一个实施例;

[0008] 图 2 是依据本发明的一个实施例、一个用于解决在一个虚拟机监控器和一个客户操作系统之间地址空间冲突的系统的一个框图;

[0009] 图 3 是依据本发明的一个实施例、一种用于解决在一个虚拟机监控器和一个客户操作方法之间的地址空间冲突的方法的一个流程图;

[0010] 图 4 是依据本发明的一个实施例、一种用于重定位在一个虚拟机地址空间内的虚拟机内核的方法的一个流程图;

[0011] 图 5 说明了依据本发明的一个实施例、一个支持客户去特权的虚拟机内核的操作；

[0012] 图 6 是依据本发明的一个实施例、一种用于处理由一个客户操作系统产生的虚拟化陷阱的方法的一个流程图；以及

[0013] 图 7 是一个处理系统实施例的一个框图。

[0014] 实施例描述

[0015] 描述了一种用于分解地址空间冲突的方法和装置。在下面的描述中，阐述了许多细节，诸如在组件之间距离、制模类型、等等。然而，对于在本领域的技术人员来说，显然可以实践本发明而不用这些具体的细节。在其它实例中，以框图形式而不是详细地显示了众所周知的结构和设备，以避免弄模糊本发明。

[0016] 在下面描述中，为了说明起见，阐述了许多具体的细节以便提供对本发明的一个彻底了解。然而，对于在本领域的技术人员来说，显然能够实践本发明而不用这些具体的细节。

[0017] 随后的某些部分详细说明依据在一个计算机存储器中的数据位上的操作的算法和符号表示给出。这些算法描述和表示是由在数据处理领域的哪些技术人员使用来最有效地把他们的工作实质传达给在本领域其它人员的装置。在这儿的的一个算法，通常被认为是一个导致一个期望结果的、有条理的步骤序列 (self-consistent sequence of steps)。这些步骤是那些需要物理量的物理操作的步骤。通常，虽然不是必要的，这些量采取电的或者磁的、能够被存储、传送、组合、比较、以及否则被操作的信号的形式。有时已经便利地证明，主要是由于公共用途的原因，把这些信号称为位、值、元素、符号、字符、项、数字、等等。

[0018] 然而应当意识到：所有这些和类似的项与适当的物理量有关而且仅仅是应用于这些量的适当标记。除非另有具体地如从下面讨论中那样明显的说明要理解在整个本发明中，使用诸如“处理”或者“计算”或者“计算”或者“确定”或者“显示”等等的术语的讨论可能涉及一个计算机系统、或者类似电子计算设备的动作和处理过程，这些计算机系统或者计算设备把被表示为在该计算机系统的寄存器和存储器中的物理（电子）量的数据操作和转换为被类似地表示为在计算机系统存储器或者寄存器或者其他这样的信息存储器、传输或者显示设备中的物理量的其它数据。

[0019] 本发明还涉及用于执行在此的操作的装置。这个装置可以为所要求的目的特别构造，或者它可以包含一台有选择地由保存在计算机中的一个计算机程序激活或者重新配置的通用计算机。这样的计算机程序可能保存在一个计算机可读存储介质中、这些存储介质诸如，但不局限于，包含软盘、光盘、CD-ROM、和磁性光盘的任何类型磁盘、只读存储器 (ROM)、随机访问存储器 (RAM)、EPROM、EEPROM、磁或者光卡、或者任何类型适合于存储电子指令的介质，这些中的每一个都和一条计算机系统总线耦合。指令可使用一个或多个处理设备（例如，处理器、中央处理单元、等等）执行。

[0020] 在此给出的算法和显示不是内在地与任何特定的计算机或者其它装置相关。可以和依据在此的示教的程序一起使用各种通用的机器，或者它可以证明构造一个更专用的装置以执行所要求的方法步骤是方便的。将从在下面的描述中发现用于各种这些机器的所需要的结构。此外，本发明不参考任何特定编程语言加以描述。将要理解：各种编程语言可以用来实现在此描述的本发明示教。

[0021] 在下面实施例的详细说明中参考附图,附图通过例子说明了可以实施本发明的具体实施例。在附图中,遍及几个视图,类似的数字实质上描述了类似的组件。足够详细地描述了这些实施例,以便本领域的那些技术人员能实施本发明。可以使用其它的实施例以及可以进行结构、逻辑、和电的变化而没有背离本发明的范围。此外,将要理解:本发明的各种实施例,虽然是不同的,但不是必须是互斥的。例如,在一个实施例中描述的一个特定属性、结构、或者特征可以包含在其它实施例中。以下详细说明,因此,不以一个有限的观念被采用,而且本发明的范围仅仅由附加权利要求以及这样的权利要求的等同物的全部范围所定义。

[0022] 本发明中的方法和装置提供了一种用于解决在一个客户操作系统和一个虚拟机监控器 (VMM) 之间的地址空间冲突的机制。图 1 说明了一个其中本发明可以进行操作的虚拟机环境 100 的一个实施例。在这个实施例中,裸平台硬件 116 包含一个计算平台,其可以能够,例如,执行一个标准操作系统 (OS) 或者一个诸如 VMM 112 的虚拟机监控器 (VMM)。VMM,虽然一般地以软件形式实现,可以向一个高层软件导出一个诸如一个模拟的裸机接口。这样的高层软件可能包含一个标准或者实时 OS,虽然本发明在这方面不局限于这个范围,而且,例如做为选择,一个 VMM 可以在另一个 VMM 中,或者在另一个 VMM 之上运行。VMM 和它们的典型特征和功能对于在本领域的那样技术人员是众所周知的,而且可以例如以软件、固件或者各种技术的一个组合的形式实现。

[0023] 如上所述, VMM 向其它软件 (即,“客户”软件) 给出一个或多个虚拟机 (VM) 的抽象。图 1 中显示了两个 VM, 102 和 114 每个 VM 包含一个诸如客户 OS 104 或者 106 的客户 OS 以及各种客户软件应用 108-110。每一个客户 OS 104 和 106 都期望控制对在客户 OS 104 或者 106 在其上运行的硬件平台中的物理资源 (例如,存储器和存储器映射的 I/O 设备) 的访问,以及执行其它的功能。然而,在一个虚拟机环境中, VMM 112 应当能够具有对物理资源的最终控制以提供对 VM 102 和 114 以及在 VM 102 和 114 之间的保护。VMM 112 通过截取客户 OS 104 和 106 对计算机物理资源的所有访问来实现这个目的。例如,一个客户去优先级技术可以用来允许 VMM 112 截取上述的访问。客户去优先级迫使所有的客户软件在一个不允许软件访问某些硬件资源的硬件特权级别处运行。结果,每当客户 OS 104 或者 106 试图访问任何硬件资源时,它被“陷入”到 VMM 112 中,即,如果一个由客户操作系统发起的操作涉及访问这样的硬件资源的话,则 VMM 112 接收对这个操作的控制。应当注意到:任何在本技术领域已知的其它技术可以用来把对一个类似操作的控制从该客户 OS 104 或者 106 转移到 VMM 112。

[0024] 当使用允许 VMM 112 截取客户 OS 104 和 106 对计算机物理资源的访问的客户去特权 (deprivileging) 或者其它技术时,一部分 VMM 代码和 / 或数据结构可以在结构上被要求驻留在和每一个客户 OS 104 和 106 相同的虚拟地址空间中。然而,因为客户 OS 104 和 106 不知道 VMM 的存在,它们可以试图访问在与该客户 OS 104 或者 106 有关的虚拟地址空间中、由 VMM 代码和 / 或数据结构占据的一个区域。这样一个尝试可能导致在客户 OS 中的代码和数据结构和在该虚拟地址空间中的 VMM 代码和数据结构之间的冲突,这导致由客户 OS 104 或者 106、或者 VMM 112 执行的一个操作的异常终结。本发明提供了一个用于解决这样的地址空间冲突的机制。

[0025] 图 2 是依据本发明的一个实施例、一个用于解决在一个 VMM 和一个客户 OS 之间地

址空间冲突的系统 200 的一个框图。系统 200 包含裸平台硬件 214,其包含一个能够执行一个客户 OS(例如,客户 OS 104 或者 106)、一个 VMM(例如,VMM 112)、等等的计算平台。为客户软件和 VMM 分配两个单独的地址空间 204 和 202。即,VM 地址空间 204 被分配来保持客户 OS 及其它客户软件的代码和数据结构,以及 VMM 地址空间 202 被分配用于 VMM 代码和数据结构。

[0026] 如上所述,VMM code 和 / 或数据结构中的某些组件可以在结构上被要求驻留在和客户 OS 相同的地址空间中,以允许 VMM 截取客户 OS 对硬件资源的访问。例如,对于 IA-32 指令系统结构 (ISA),当客户去特权被用来确保 VMM 对客户 OS 访问硬件资源的控制时,一个包含指向陷阱处理例程的指针的中断描述符表 (IDT) 在结构上被要驻留在和客户 OS 相同的地址空间中。将在下面结合图 5 和 6 更详细地描述支持客户去特权的本发明的一个实施例。对于其它 ISA,VMM 代码和 / 或数据结构的各种其它部分可以在结构上被要求驻留在和客户 OS 相同的空间地址中,以允许 VMM 控制客户 OS 对硬件资源进行的访问。

[0027] 在一个实施例中,VMM 代码和结构被分成两个部分。VMM 的第一部分包含一组被要求驻留在客户 OS 的地址空间,即 VM 地址空间 204 中的代码和 / 或数据结构。VMM 的第二部分包含其余的 VMM 代码和数据结构。在一个实施例中,一个软件程序(被称为虚拟机内核 210) 收集被要求位于在和客户 OS 相同的地址空间中的 VMM 代码和 / 或数据结构的一个最小集合。其余的 VMM 代码和数据结构被编译为一个单独的程序并且位于 VMM 地址空间 202 中。虚拟机内核 (VMK) 210 然后把它自己映射成为 VM 地址空间 204 和 VMM 地址空间 202。

[0028] 随后,当客户 OS 试图访问在 VM 地址空间 204 中、由 VMM 代码和 / 或数据结构占据的一个区域时,VMK 210 检测到客户 OS 的这个尝试。在一个实施例中,如果一个由客户 OS 发起的事件可能导致在客户 OS 和 VMM 之间的一个地址空间冲突的话,VMK 210 接收对这个事件的控制。客户去特权或者在本技术领域已知的任何其它硬件或者软件机制可以用来把对这样一个事件的控制从客户 OS 转移到驻留在 VM 地址空间 204 中的 VMM 代码和 / 或数据结构。

[0029] VMK 210 然后计算这个事件以确定它的原因。当检测到该事件是由客户 OS 试图访问由 VMM 代码和 / 或数据结构占据的区域所引起的时,该 VMK 210 把它自己重新映射到在 VM 地址空间 204 中的一个不同区域以允许客户 OS 访问先前由 VMK 210 使用的区域。一种用于在 VM 地址空间 204 内重定位 VMK 210 的方法的一个实施例在下面结合图 4 进行了更详细的描述。

[0030] 图 3 是依据本发明的一个实施例、一个用于解决在一个 VMM 和一个客户 OS 之间地址空间冲突的方法 300 的一个实施例的流程图。方法 300 以把 VMM 划分成为第一部分和第二部分开始(处理块 304)。如上所述,第一部分包含一组在结构上被要求驻留在和客户 OS 相同的地址空间中的 VMM 代码和 / 或数据结构。VMM 的第二部分包含其余的 VMM 代码和数据结构。在一个实施例(在下面更详细描述)中,VMM 的第一部分包含一组陷阱处理程序和一个中断描述符表 (IDT)。在替换的实施例中,第一部分包含必须驻留在和客户 OS 相同的地址空间中的、VMM 的各种其它数据结构和代码。

[0031] 接下来,创建一个第一地址空间(即,VM 地址空间 204)来保持客户 OS 及其它客户软件的代码和数据结构(处理块 306),以及创建用于 VMM 代码和数据结构的一个第二地址空间(即,VMM 地址空间 202)(处理块 308)。在一个实施例中,在引导处理过程期间创建

这些地址空间。

[0032] 此外, VMM 的第一部分被映射到 VM 地址空间和 VMM 地址空间中(处理块 310), 以及 VMM 的第二部分被载入到 VMM 地址空间中(处理块 312)。

[0033] 在处理块 314 处, 检测到客户 OS 试图访问由 VMM 的第一部分占据的一个区域。在一个实施例中, 如果由客户 OS 发起的一个事件可能导致在客户操作系统和 VMM 之间的一个地址空间冲突的话, 通过把对这个事件的控制转移到 VMM 的第一部分来检测这样一个试图。在下面结合图 5 和 6 更详细地描述检测一个可能的地址空间冲突的一个实施例。

[0034] 此后, 在处理块 316 处, VMM 的第一部分被重定位到在该 VM 地址空间中的另一个区域, 以允许客户 OS 对先前由 VMM 第一部分占据的区域的访问。任何后续的试图对由 VMM 第一部分占据的新区域的访问将再次导致它在 VM 地址空间内的重新定位。在图 4 中显示了一种用于重定位一个包含 VMM 第一部分的 VMK 的方法的一个实施例。

[0035] 参见图 4, 在检测到在客户 OS 和 VMM 之间的一个地址空间冲突之后(处理块 404), 为一个未使用的区域搜索该 VM 地址空间(处理块 406)。在判定框 408 处, 进行有关一个未使用的区域是否存在于 VM 地址空间中的确定。如果该确定是肯定的, 则包含 VMM 代码和数据结构第一部分的 VMK 被重新映射到这个未使用的区域中, 而且控制被转移回到客户 OS, 其现在可以访问先前由 VMK 使用的区域。

[0036] 做为选择, 如果在 VM 地址空间中不存在未使用的区域, 即客户 OS 已经使用了整个 VM 地址空间的话, 则在该 VM 地址空间内选择一个随机区域(处理块 412), 位于被选定区域处的存储器内容被复制到在 VMM 地址空间中的一个缓存器(处理块 414), 而且 VMK 被重新映射到在 VM 地址空间中的选定区域中(处理块 416)。由在包含新 VMK 区域的原有内容的 VMM 地址空间中的缓存器通过模拟的存储器访问来服务对这个选定区域(即, 新的 VMK 区域)的后续存储器访问。在一个实施例中, 通过定期地把 VMK 重定位到在 VM 地址空间内的随机区域直到发现一个很少被使用的区域为止, 来减少这样的模拟存储器引用的频率。

[0037] 图 5 说明了依据本发明的一个实施例、一个支持客户去特权的 VMK 的操作。如上所述, 客户去特权导致客户 OS 在一个较低特权级别处运行, 以便每当客户 OS 试图发布有关该处理器系统状态的操作的特许指令时, 它被“陷入”到 VMM 中。在一个实施例中, 支持客户去特权的 VMM 在中断描述符表(IDT) 514 中安置指向陷阱处理例程(即, 陷阱处理程序 552)的指针。某些 ISA(例如, IA-32 ISA) 要求 IDT 514 驻留在当前活动的虚拟地址空间(即, VM 地址空间 504) 中。在一个实施例中, 在 IDT 514 中的入口(entry) 是任务门, 其提供了一个地址空间切换。即, 当产生一个陷阱时, 为一个指向陷阱处理例程的指针搜索 IDT 514。如果这个指针是一个任务门, 它将允许一个到 VMM 地址空间的直接切换, 该 VMM 地址空间包含用于产生陷阱的一个陷阱处理例程。因此, 虽然任务门它自己必须驻留在该 VM 地址空间中, 一个对应于任务门的陷阱处理程序不必要驻留在 VM 地址空间中。在另一个实施例中, 在 IDT 514 中的入口是陷阱门或者中断门, 其不提供地址空间切换。因此, 与这样的 IDT 入口有关的陷阱处理程序必须驻留在 VM 地址空间中。此外, VMM 可以在 VM 地址空间中放置其它数据结构(例如, 总体描述符表)的阴影版本。

[0038] 在一个实施例中, VMK 510 收集必须位于 VM 地址空间中的陷阱处理程序和 / 或数据结构(例如, IDT 514) 的一个最小集合在一起, 把它们映射成为 VM 地址空间 504 和 VMM 地址空间 502, 并且把保持 VMK 510 的页的访问权限设置为最特权的级别(例如, 对于 IA-32

微处理器,具有环=0的“超级用户”特权级别)。如上所述,客户 OS 运行在去特权模式中(例如,对于 IA-32 微处理器,具有环=3的“用户”模式)。结果,在一个实施例中,每当客户 OS 试图访问有特权的机器资源时,客户 OS 产生虚拟化陷阱,有特权的机器资源包括保持用最高特权访问权限保护的 VMK 510 的页。

[0039] 在一个实施例中,当产生一个虚拟化陷阱时,为一个指向陷阱处理程序的相应指针搜索 IDT 514。在一个实施例中,陷阱可以需要由 VMM 驻留陷阱处理程序处理。在这个实施例中,VMK 执行两个地址空间切换——一个把该陷阱传送到在 VMM 地址空间 502 中的陷阱处理程序的切换,以及一个在该陷阱已经由 VMM 驻留的陷阱处理程序服务之后转换回到 VM 地址空间 504 的第二切换。

[0040] 做为选择,能够在 VMK 驻留处理程序中处理一个陷阱。例如,一个陷阱可以由客户 OS 中的一条指令引起以复位在该处理器的寄存器中的一个标记。这样的陷阱能够完全在陷阱处理程序 552 中处理,而不用把控制转移到在 VMM 地址空间 502 中的 VMM,而且这样一个实现将导致更好的性能。

[0041] 一种类型的虚拟化陷阱是当客户 OS 试图访问当前由 VMK 510 使用的 VM 地址空间 504 中的一个区域时产生的一个冲突错误。如上面结合图 4 更详细地描述的那样,VMK 510 通过把它自己重新映射到在该 VM 地址空间 504 中的一个新区域中来处理这些冲突错误。

[0042] 图 6 是依据本发明的一个实施例、一种用于处理由一个客户 OS 产生的虚拟化陷阱的方法 600 的一个流程图。方法 600 以把由 VMK 占据的区域的访问权限设置为一个比和该客户 OS 有关的特权级别更高的特权级别开始(处理块 604)。例如,所有 VMK 页可以用仅仅是超级用户的特权(环=0)进行映射,而客户 OS 可以被设置为运行在一个去特权的用户状态(环=3)中。

[0043] 在处理块 606 处,由客户 OS 产生的一个陷阱被接收了。该陷阱由客户 OS 试图访问特权的硬件资源引起。在判定框 608 处,进行有关该陷阱是否能够由 VMK 内部处理(例如,在一个 VMK 驻留的陷阱处理程序中)的确定。如果该陷阱太复杂以致不能由 VMK 处理,它被转交到 VMM 地址空间(例如,被转交给 VMM 驻留的陷阱处理程序)(处理块 610),然后在陷阱已经由 VMM 服务之后继续回到 VM 地址空间(处理块 612)。此后,对引起该陷阱的事件的控制返回到客户 OS(处理块 620)。

[0044] 做为选择,如果陷阱由 VMK 内部处理,则做出有关该陷阱是否由在 VMK 代码和数据结构和客户 OS 的代码和数据结构之间的一个地址空间冲突引起的一个确定(判定框 614)。如果该陷阱实际上是由一个地址空间冲突所引起,则 VMK 代码和数据结构被重定位到在该 VM 地址空间中的一个新区域(处理块 618)。做为选择,在一个相应陷阱处理程序中处理该陷阱(处理块 616)。此后,对引起该陷阱的事件的控制返回到客户 OS(处理块 620)。

[0045] 图 7 是一个处理系统实施例的一个框图。处理系统 700 包含处理器 720 和存储器 730。处理器 720 能够是能够执行软件的任何类型处理器,诸如微处理器、数字信号处理器、微控制器、等等。处理系统 700 能够是一台个人计算机(PC)、大型机、手持设备、便携式计算机、机顶盒、或者任何其它包含软件的系统。

[0046] 存储器 730 能够是一个硬盘、软盘、随机存取存储器(RAM)、只读存储器(ROM)、闪存存储器、或者任何其它类别可由处理器 720 读取的机器介质。存储器 730 能够存储用于执行本发明各种方法实施例实现的指令,诸如方法 300、400 和 600(图 3、4 和 6)。

[0047] 将要理解：以上描述是用于说明性的，而不是限制。依据以上描述的读取和理解，许多其它的实施例对于本领域那些技术人员是显而易见的。本发明的范围，因此应当参考附加的权利要求，以及这样的权利要求的等同物的全部范围，来进行确定。

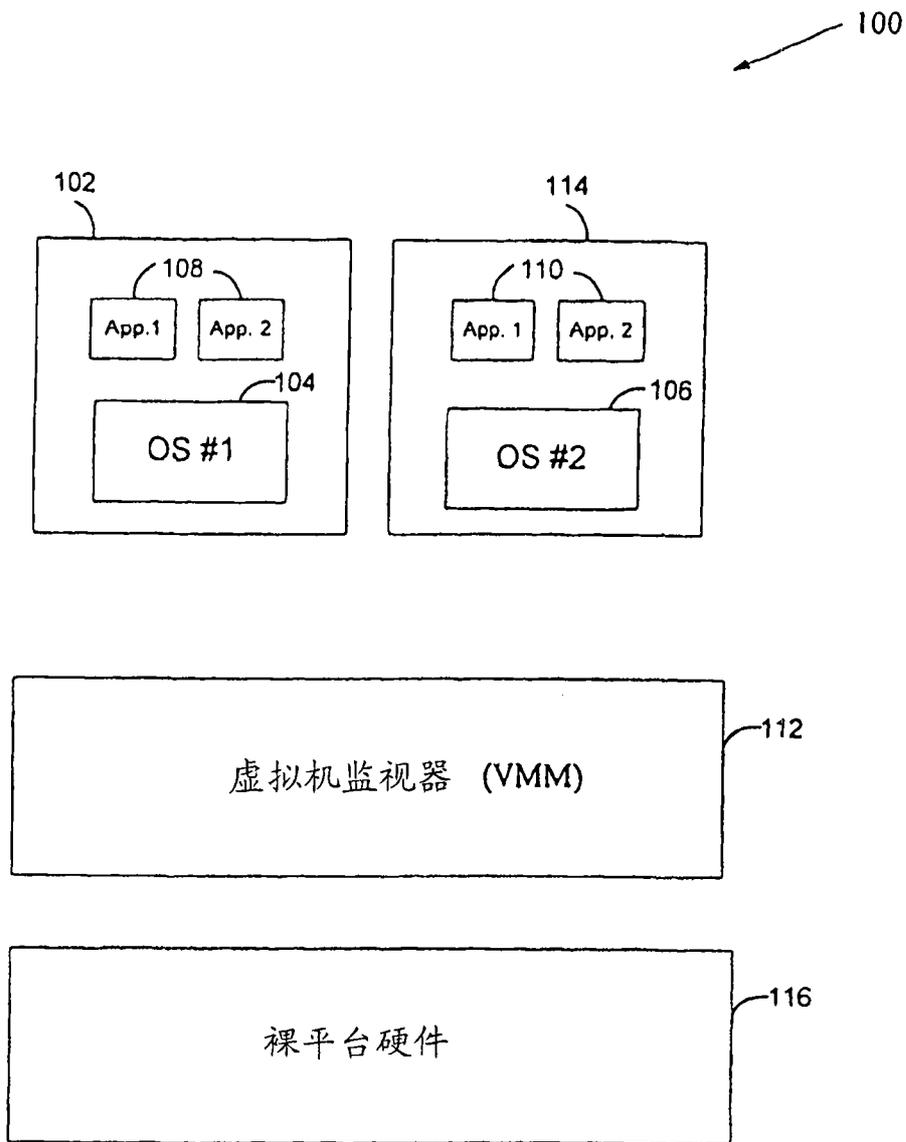


图 1

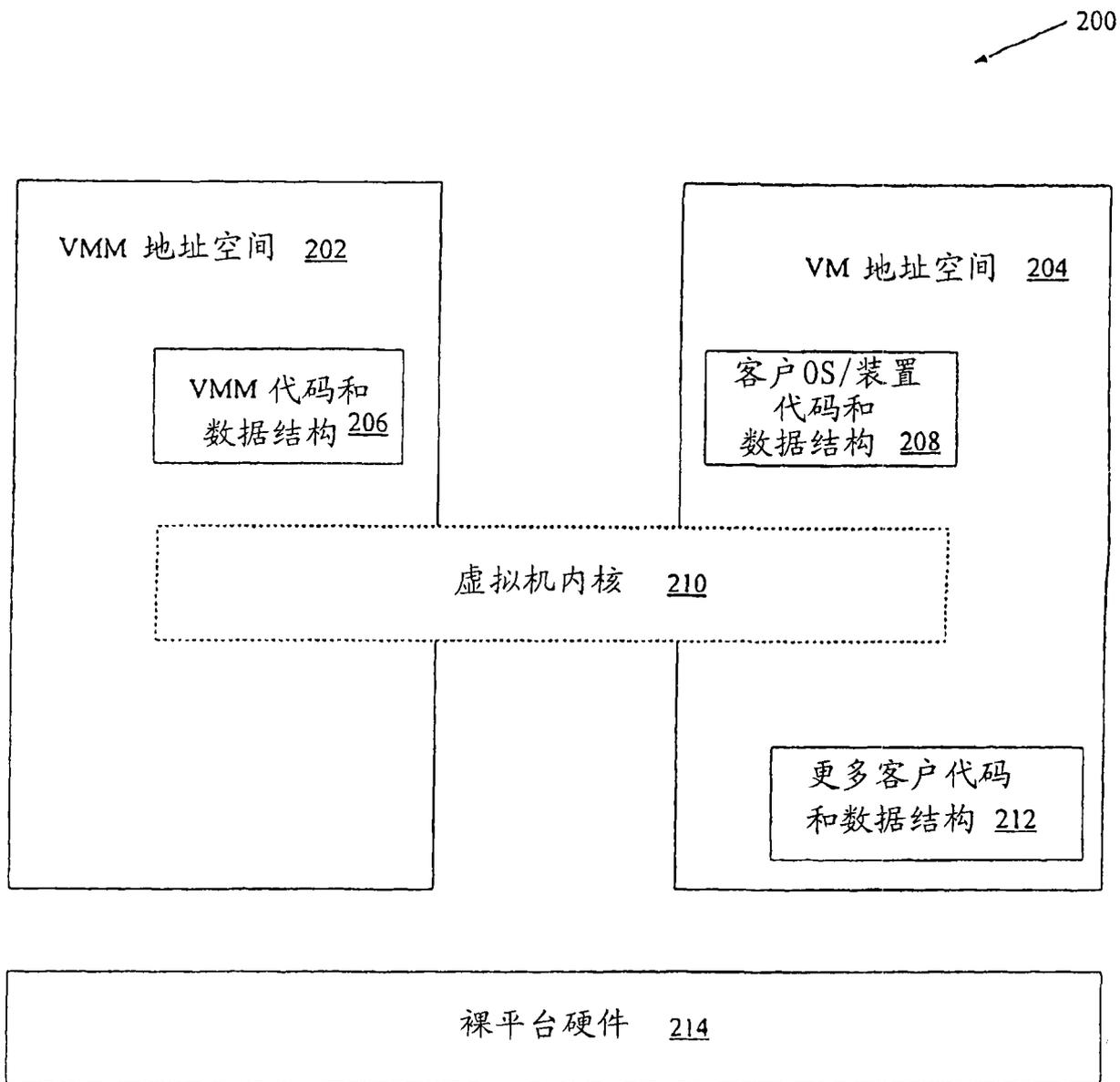


图 2

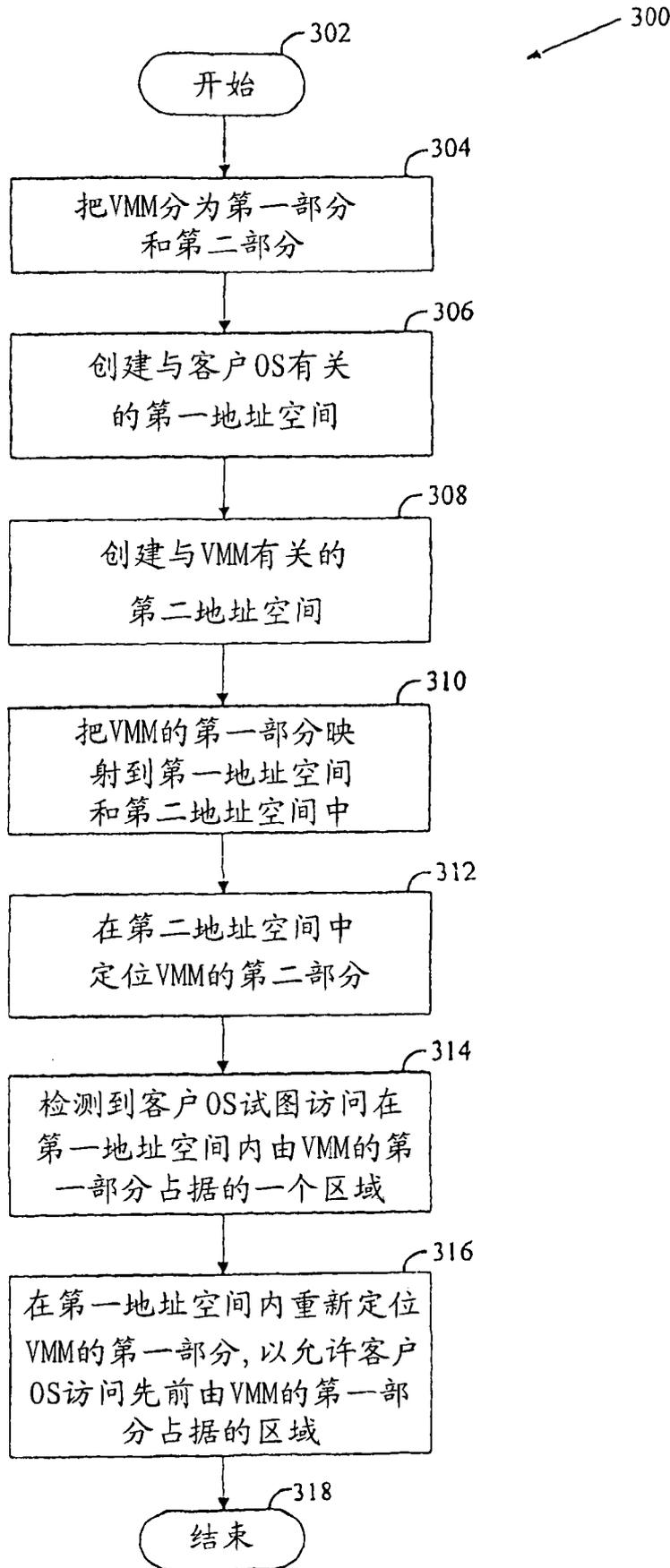


图 3

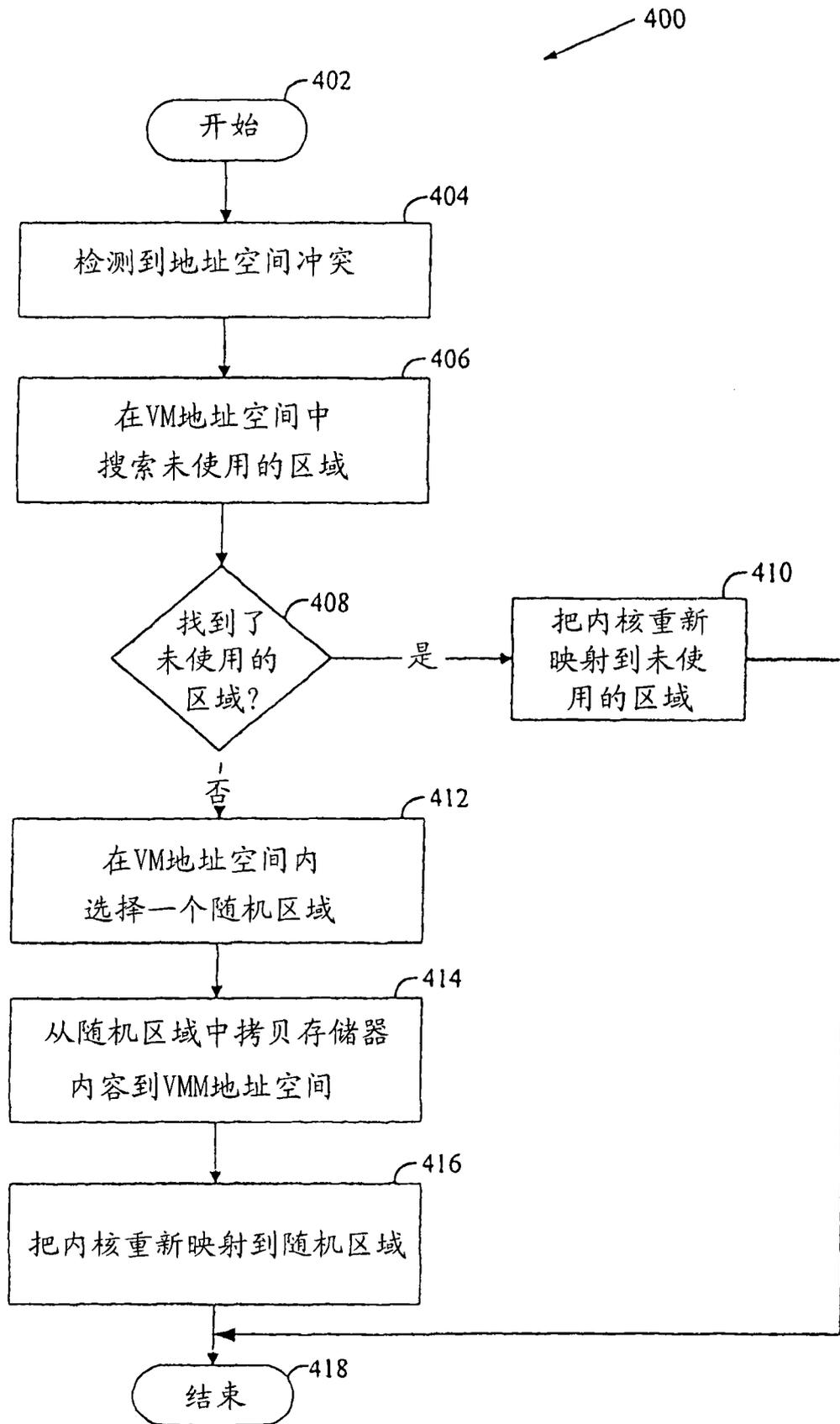


图 4

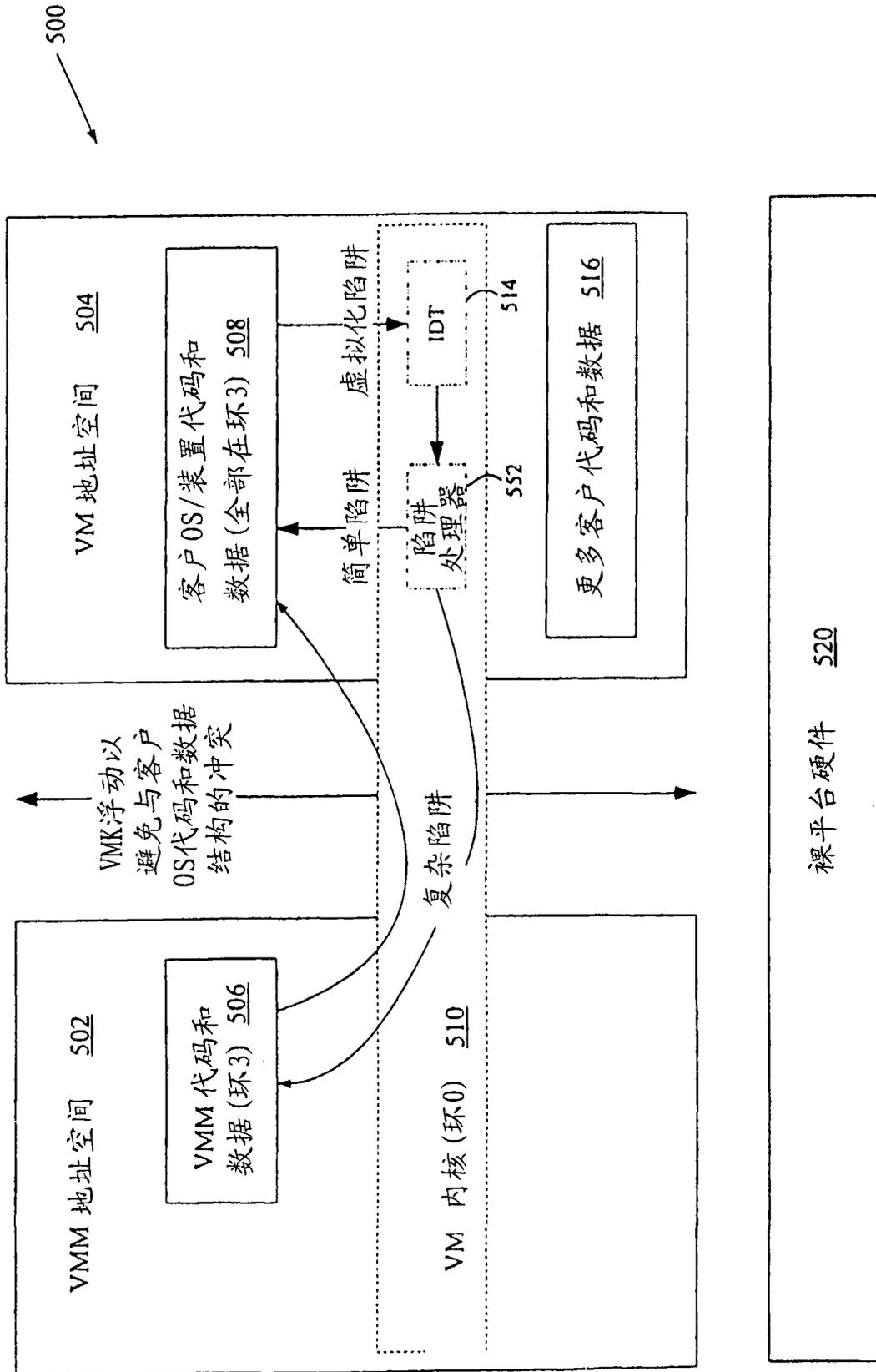


图 5

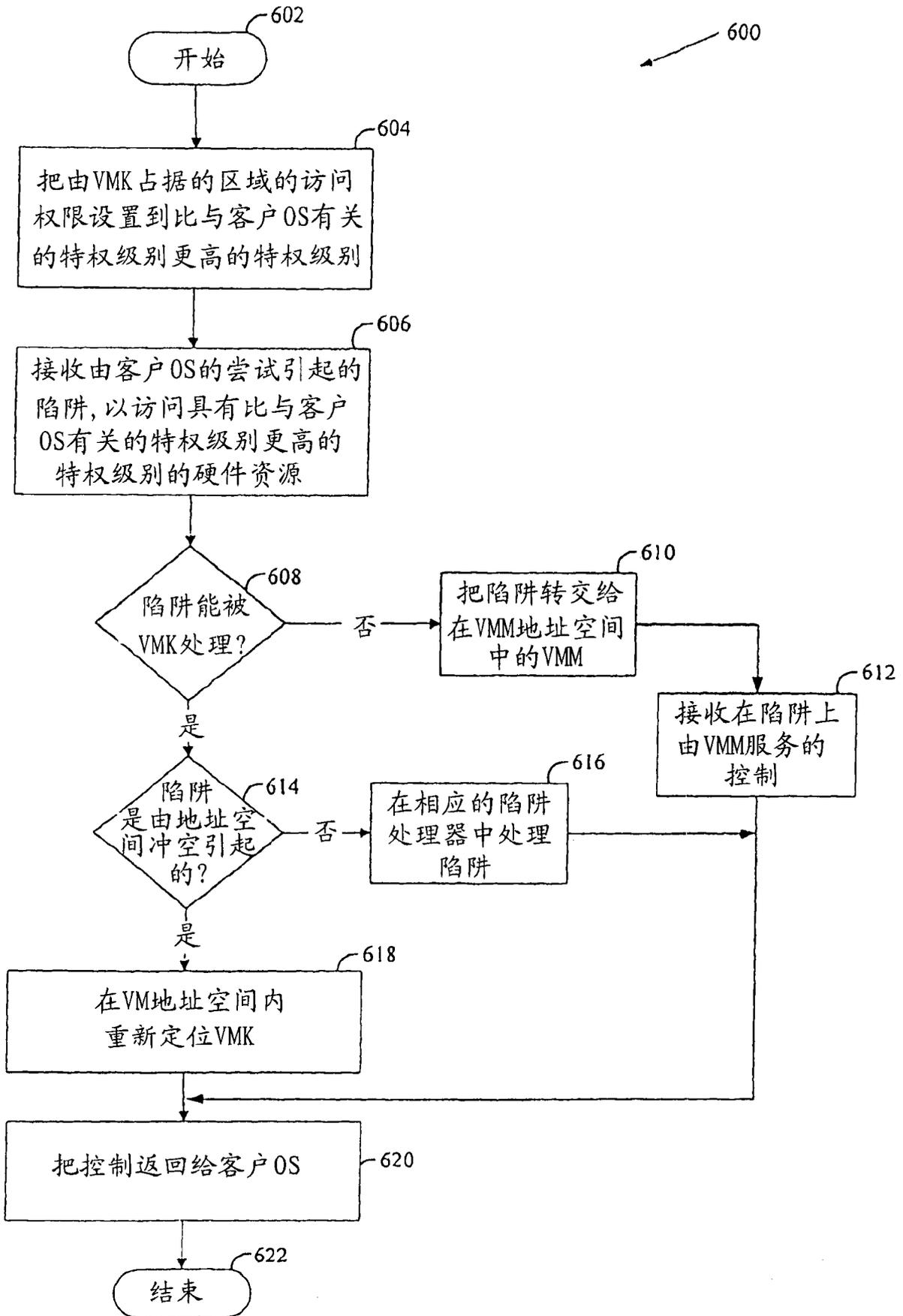


图 6

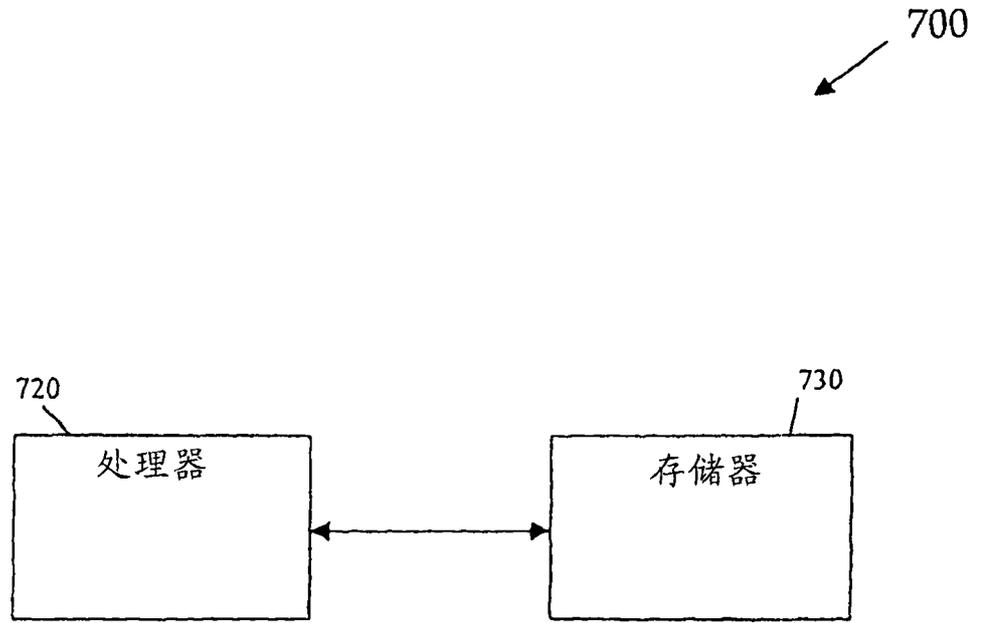


图 7