[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 9/44 (2006.01)
G06F 9/455 (2006.01)

[12] 发明专利说明书

专利号 ZL 200610067054.5

[45] 授权公告日 2008 年 7 月 23 日

[11] 授权公告号 CN 100405295C

[22] 申请日 2006.3.31
[21] 申请号 200610067054.5
[30] 优先权
[32] 2005. 4. 5 [33] EP [31] 05102679.7
[73] 专利权人 国际商业机器公司
地址 美国纽约
[72] 发明人 C·奥特 U·魏甘德
[56] 参考文献
JP10240584A 1998.9.11
CN1664793A 2005.9.7
US2004061726A1 2004.4.1
审查员 周述虹

[74] 专利代理机构 北京市中咨律师事务所
代理人 于 静 李 峥

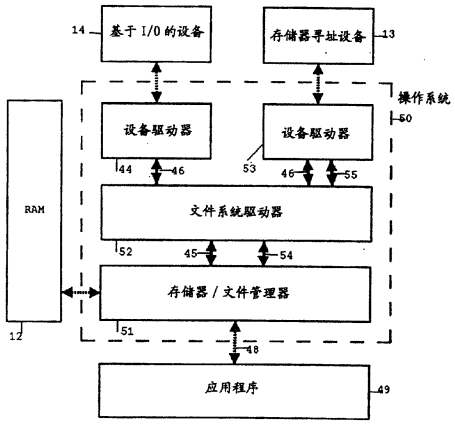
权利要求书 3 页 说明书 21 页 附图 18 页

[54] 发明名称

用于提供原地执行功能的系统和方法

[57] 摘要

本发明公开了一种操作系统，该操作系统提供了用于实现原地执行功能的新的和发明性的系统和方法。 该操作系统提供了用于实现原地执行功能的下面的新的和发明性的功能组件：存储器/文件管理器和至少一个文件系统驱动器之间的文件系统直接访问接口，其中该文件系统直接访问接口提供检索特定文件在特定偏移量处的内容的系统存储器地址的功能，其中文件位于所述存储器寻址设备上；至少一个文件系统驱动器和提供对所述至少一个存储器寻址设备的访问的至少一个设备驱动器之间的设备直接访问接口，其中该设备直接访问接口提供检索至少一个存储器寻址设备的特定块的系统存储器地址的功能；其中通过使用文件系统直接访问接口和设备直接访问接口由该存储器/文件管理器、至少一个文件系统驱动器和提供对至少一个存储器寻址设备的访问的至少一个设备驱动器实现该原地执行功能。



1. 系统（50），包括：

具有与应用程序(49)的接口(48)的存储器/文件管理器(51)，

至少一个文件系统驱动器（52），其具有与所述存储器/文件管理器（51）的文件系统 I/O 接口（45），

至少一个第一设备驱动器（44），其具有到所述文件系统驱动器（52）的设备 I/O 接口（46），其中所述至少一个第一设备驱动器（44）提供对至少一个基于 I/O 的设备（14）的访问，

至少一个第二设备驱动器（53），其具有到所述文件系统驱动器（52）的设备 I/O 接口（46），其中所述至少一个第二设备驱动器（53）提供对至少一个存储器寻址设备（13）的访问，

其中所述操作系统提供原地执行功能以访问至少一个存储器寻址设备（13），

其特征在于，

所述存储器/文件管理器（51）和所述至少一个文件系统驱动器（52）之间的文件系统直接访问接口（54），其中所述文件系统直接访问接口（54）提供检索特定文件在特定偏移量处的内容的系统存储器地址的功能，其中所述文件驻留在所述存储器寻址设备（13）上，

所述至少一个文件系统驱动器（52）和提供对所述至少一个存储器寻址设备(13)的访问的所述至少一个第二设备驱动器(53)之间的设备直接访问接口(55)，其中所述设备直接访问接口(55)提供检索至少一个存储器寻址设备（13）的特定块的系统存储器地址的功能，

其中通过使用所述文件系统直接访问接口（54）和所述设备直接访问接口（55），由所述存储器/文件管理器（51）、所述至少一个文件系统驱动器（52）和提供对所述至少一个存储器寻址

设备（13）的访问的所述至少一个第二设备驱动器（53）提供所述原地执行功能。

2. 根据权利要求1的系统，其中所述系统是操作系统的一部分或者由所述操作系统提供。

3. 根据权利要求1的系统，其中所述存储器/文件管理器(51)或是使用所述文件系统直接访问接口（54）或是使用所述文件系统 I/O 接口（45）提供对由所述文件系统驱动器（52）管理的文件的访问。

4. 根据权利要求1的系统，其中所述文件系统驱动器（52）通过检索特定文件在特定偏移量处的内容的系统存储器地址实现所述文件系统直接访问接口（54），其中所述内容驻留在所述存储器寻址设备（13）的某个块上，通过使用由所述设备驱动器提供的所述设备直接访问接口以便检索所述块的系统存储器地址，由所述设备驱动器提供对该内容的访问。

5. 根据权利要求1的系统，其中通过使用由所述文件系统驱动器（52）提供的所述文件系统直接访问接口（54）检索所述文件在所述偏移量处的内容并使用所述地址原地执行所述内容，所述存储器/文件管理器（51）实现对特定文件在特定偏移量处的原地执行访问，其中所述文件驻留在由所述文件系统驱动器（52）处理的文件系统上，并且所述文件系统驻留在所述存储器寻址设备（13）上。

6. 计算机系统，其具有根据权利要求1－5的系统。

7. 根据权利要求6的计算机系统，该计算机系统具有控制着一个或多个虚拟机的管理程序，其中至少一个虚拟机运行根据权利要求1－5的系统。

8. 根据权利要求7的计算机系统，其中以由所述管理程序给一个或多个所述虚拟机提供的共享存储器段体现所述存储器寻址设备（13）。

9. 用于由根据权利要求 1－5 的系统自动提供原地执行功能的方法，其中所述系统接受应用程序访问文件的请求，并判定是否使用原地执行功能来访问所述文件，其中所述方法包括以下步骤：

确定保持所述文件的文件系统，

确定管理所述文件系统的文件系统驱动器（52）是否提供所述文件系统直接访问接口（54），

如果所述文件系统驱动器（52）不提供所述文件系统直接访问接口（54），则不使用原地执行，

如果所述文件系统驱动器提供所述文件系统直接访问接口（54），则确定所述文件驻留在其上的设备，

确定管理所述存储器寻址设备（13）的第二设备驱动器（53）是否提供所述设备直接访问接口（55），

如果所述设备驱动器不提供所述设备直接访问接口（55），则不使用原地执行，

确定所述文件系统是否配置成允许原地执行功能，以及

使用所述文件系统直接访问接口（54）和所述设备直接访问接口（55）提供原地执行功能。

用于提供原地执行功能的系统和方法

技术领域

本发明一般地涉及操作系统，并且具体地涉及提供用于实现原地执行（execute-in-place）功能的系统和方法的操作系统。

背景技术

现有技术的计算机系统包含非易失大容量存储设备(例如硬盘驱动器)以保持程序和数据文件。这些文件的内容必须被装入 RAM（“随机访问存储器”）类的系统存储器内以便被 CPU（“中央处理单元”）访问或执行。此操作通常由操作系统代表应用程序执行。现有技术的计算机系统和操作系统支持虚拟存储器和按需调页（demand paging）。应用不直接使用系统存储器地址指定它们使用的代码和数据；而是它们使用“虚拟地址”指定存储器位置，由 CPU 电路实现的且被操作系统控制的调页机制将该存储器位置翻译成系统存储器地址。这使得操作系统可避免必须将全部程序和数据文件装入 RAM 内。而是系统存储器被分为特定大小的块（被称为“页”），并且仅在访问该特定页时，操作系统将文件内容的相应块装入各个存储器页内。此过程通常称为“按需调页”。

这种方法的一个缺点是需要 RAM 以保持程序和数据文件内容，减少了可用于其它用途的 RAM 的量。另外，通常需要一些时间将内容下载到 RAM 内。因此，一些现有技术的计算机系统提供了不同类型的非易失存储设备，CPU 可用与访问 RAM 相同的方式直接访问这些不同类型的非易失存储设备（“存储器寻址设备”）。存储器寻址设备的一个现有技术的实施例是快闪存储器卡。存储器寻址设备允许 CPU 执行该设备上存储的代码和访问该设备上存储的数据而不用首先将内容下载到 RAM 内。这种直

接执行驻留在存储器寻址设备上的代码的方法被称为“原地执行”。为了给在支持虚拟存储器的操作系统上运行的应用提供原地执行功能，该操作系统必须控制调页机制，以便将应用的地址空间的某些虚拟地址映射到该存储器寻址设备支持的地址范围内的系统存储器地址。

其它现有技术的计算机系统提供了虚拟化能力。由通常被称为“管理程序”的运行于单个计算机系统上，但是允许多个“客户”操作系统同时运行的（每个操作系统在一个单独的“虚拟机”内）软件程序实现虚拟化。对于运行在其内的操作系统来说，每个虚拟机自身看起来好像是包括CPU、RAM 和 I/O 设备的真实的计算机系统。由管理程序拦截对这些虚拟组件的访问并且将其翻译成对实际组件的访问。这允许在多个客户操作系统之间共享计算机系统的资源，以提供对系统资源的提高的总的利用率。

一些现有技术的虚拟化计算机系统的一个缺点是如果运行在同一管理程序下的多个客户同时访问相同的程序或数据，则每个客户操作系统将分别分配虚拟 RAM 以保持这些内容，因此管理程序必须在物理 RAM 内分配所述内容的多个相同的拷贝。这意味着有较少的存储器可用于其它用途，限制了能够同时有效地运行的客户的数量。因此，一些现有技术的管理程序提供了可从多个客户同时访问的物理存储器的段（“共享存储器段”）。通过将程序或数据文件存储在共享存储器段内，多个客户可同时访问所述文件而不用首先将内容下载到虚拟 RAM 内。所述共享段在客户操作系统看来就好像是物理存储器寻址设备。

数据和程序文件通常被使用标准文件系统布局存储在设备上；一些操作系统能够使用针对不同使用情况而优化的多种不同的文件系统布局。为此，现有技术操作系统通常被构造为多个组件。在一些操作系统内，存在中央文件和存储器管理组件，多个文件系统驱动器和多个 I/O 设备驱动器。因此，通过结合中央文件和存储器管理组件使用合适的文件系统驱动器和 I/O 设备驱动器对，操作系统允许在任何一个支持的 I/O 设备上使用任何一种支持的文件系统布局。但是，现有技术的操作系统不能使用已有的文件系统驱动器以允许原地执行的方式访问存储器寻址设备。以整体方式实

现支持原地执行访问存储器寻址设备。

实际上，一些现有技术的操作系统的实现根本不允许使用标准文件系统布局将数据存储在存储器寻址设备上；而是它们需要以设备专用的方式将数据布置在这种设备上。该布置具有许多缺点，对于使用 I/O 设备和存储器寻址设备的计算机系统尤其如此。支持不同文件系统布局可使得系统管理更加困难。可能需要不同的工具来格式化、管理、备份和恢复不同布局。从 I/O 设备将一组已有文件移植到存储器寻址设备可能更加困难，反之亦然。存储器寻址设备所需的特定布局可能不提供标准文件系统布局所具有的所有特征（例如，实现复杂的访问控制和特权检查）。

另一种现有技术的实现（zSeries 上用于 Linux 的 XIP2FS 文件系统）提供使用第二扩展文件系统（“ext2”）格式—Linux 操作系统提供的一种标准文件系统格式—将程序和数据存储在虚拟存储器寻址设备（由 z/VM 管理程序提供的共享存储器段）上的支持。但是，此方法仍具有大部分前面段落内所述的缺点：不能使用其它的标准 Linux 文件系统格式，并且另外 XIP2FS 不能提供 ext2 的所有特征（例如 XIP2FS 不支持写访问）。

XIP2FS 的另一个缺点是其不能被集成到操作系统的上述组件结构内；尽管 XIP2FS 使用 ext2 文件系统布局访问文件，XIP2FS 不使用 Linux ext2 文件系统驱动器这样做，而是重新实现访问 ext2 文件系统布局所需的访问逻辑。这再次使得 XIP2FS 不支持 ext2 的所有特征，因为只有整个 ext2 逻辑的子集被重新实现。作为另一个缺点，随着时间的过去 Linux 操作系统的标准 ext2 文件系统组件不断被发展并且添加新特征；例如由 Linux 内核版本 2.6 提供的 ext2 文件系统驱动器版本增加了对更快地访问非常大的目录结构以及更复杂的访问控制机制的支持。XIP2FS 不能自动地受益于对 ext2 驱动器的这种改进；所有所需的特征需要在 XIP2FS 代码内被重新实现。

发明内容

本发明的目的是提供一种由避免上述现有技术的缺点的操作系统提供

原地执行功能的方法。

根据本发明的一个方面，提供一系统，包括：

具有与应用程序的接口的存储器/文件管理器，

至少一个文件系统驱动器，其具有与所述存储器/文件管理器的文件系统 I/O 接口，

至少一个第一设备驱动器，其具有到所述文件系统驱动器的设备 I/O 接口，其中所述至少一个第一设备驱动器提供对至少一个基于 I/O 的设备的访问，

至少一个第二设备驱动器，其具有到所述文件系统驱动器的设备 I/O 接口，其中所述至少一个第二设备驱动器提供对至少一个存储器寻址设备的访问，

其中所述操作系统提供原地执行功能以访问至少一个存储器寻址设备，

其特征在于，

所述存储器/文件管理器和所述至少一个文件系统驱动器之间的文件系统直接访问接口，其中所述文件系统直接访问接口提供检索特定文件在特定偏移量处的内容的系统存储器地址的功能，其中所述文件驻留在所述存储器寻址设备上，

所述至少一个文件系统驱动器和提供对所述至少一个存储器寻址设备的访问的所述至少一个第二设备驱动器之间的设备直接访问接口，其中所述设备直接访问接口提供检索至少一个存储器寻址设备的特定块的系统存储器地址的功能，

其中通过使用所述文件系统直接访问接口和所述设备直接访问接口，由所述存储器/文件管理器、所述至少一个文件系统驱动器和提供对所述至少一个存储器寻址设备的访问的所述至少一个第二设备驱动器提供所述原地执行功能。

根据本发明的另一个方面，提供一用于由根据上述系统自动提供原地执行功能的方法，其中所述系统接受应用程序访问文件

的请求，并判定是否使用原地执行功能来访问所述文件，其中所述方法包括以下步骤：

确定保持所述文件的文件系统，

确定管理所述文件系统的文件系统驱动器是否提供所述文件系统直接访问接口，

如果所述文件系统驱动器不提供所述文件系统直接访问接口，则不使用原地执行，

如果所述文件系统驱动器提供所述文件系统直接访问接口，则确定所述文件驻留在其上的设备，

确定管理所述存储器寻址设备的第二设备驱动器是否提供所述设备直接访问接口，

如果所述设备驱动器不提供所述设备直接访问接口，则不使用原地执行，

确定所述文件系统是否配置成允许原地执行功能，以及

使用所述文件系统直接访问接口和所述设备直接访问接口提供原地执行功能。

本发明公开了一种提供了用于实现原地执行功能的新系统和方法的操作系统。

本发明基于其上的现有技术的操作系统包括具有与应用程序的接口的存储器/文件管理器，至少一个文件系统驱动器，其具有与存储器/文件管理器的文件系统 I/O 接口，至少一个设备驱动器，其具有与文件系统驱动器的设备 I/O 接口，其中所述至少一个设备驱动器提供到至少一个基于 I/O 的设备的访问，至少一个设备驱动器，其具有与文件系统驱动器的设备 I/O 接口，其中所述至少一个设备驱动器提供到至少一个存储器寻址设备的访问，其中所述操作系统提供原地执行功能以访问至少一个存储器寻址设备。

以下面的新的和发明性的功能扩展现有技术的操作系统以实现原地执行功能：

存储器/文件管理器和至少一个文件系统驱动器之间的文件系统直接

访问接口，其中该文件系统直接访问接口提供了检索特定文件在特定偏移量处的内容的系统存储器地址的功能，其中该文件驻留在所述存储器寻址设备上，

至少一个文件系统驱动器和提供到所述至少一个存储器寻址设备的访问的至少一个设备驱动器之间的设备直接访问接口，其中该设备直接访问接口提供了检索至少一个存储器寻址设备的特定块的系统存储器地址的功能，

其中通过使用文件系统直接访问接口和设备直接访问接口，由存储器/文件管理器、至少一个文件系统驱动器和提供到至少一个存储器寻址设备的访问的至少一个设备驱动器实现原地执行功能。

附图说明

从下面的详细说明中可清楚地了解本发明的上述以及另外的目的、特征和优点。

在所附权利要求中提出了本发明的新颖特征。但是，当结合附图阅读

时通过参考下文对说明性实施例的详细说明，可以最好地理解本发明本身以及其优选使用模式、另外的目的和其优点，其中：

图 1A 示出实现本发明所需的计算机系统的框图；

图 1B 示出在本发明的一些实施例中容纳有图 1A 所示的计算机系统作为客户的虚拟机环境；

图 1C 示出现有技术的操作系统提供的虚拟存储器和按需调页功能；

图 1D 示出现有技术的操作系统提供的原地执行功能；

图 1E 示出实现原地执行的现有技术操作系统的组件结构；

图 2A 示出使用本发明实现原地执行的操作系统的组件结构；

图 2B 示出根据本发明用于访问基于 I/O 的设备的图 2A 的操作系统组件中的控制流；

图 2C 示出根据本发明用于执行对存储器寻址设备的原地执行访问的图 2A 的操作系统组件中的控制流；

图 2D 是用于选择使用图 2B 和 2C 所示的两个控制流中的哪一个的图 2A 的操作系统的判定逻辑；

图 3A 示出现有技术的 Linux 操作系统内实现的设备抽象；

图 3B 示出对实现本发明的 Linux 操作系统内的设备驱动层进行的扩展；

图 3C 示出通用文件系统如何服务于地址空间操作以便从/向现有技术的 Linux 操作系统内的设备读/写一个或多个页；

图 3D 示出对实现本发明的 Linux 操作系统内的文件系统的地址空间操作进行的扩展；

图 3E 示出在现有技术的 Linux 操作系统中文件系统库函数如何执行通用文件系统的读类型文件操作；

图 3F 示出对用于实现本发明的 Linux 操作系统内的读类型操作的文件系统库函数进行的扩展；

图 3G 示出在现有技术的 Linux 操作系统中文件系统库函数如何执行通用文件系统的写类型文件操作；

图 3H 示出对用于实现本发明的 Linux 操作系统内的写类型操作的文件系统库函数进行的扩展；以及

图 3I 示出对用于实现本发明的 Linux 操作系统内的文件存储器映射的文件系统库函数进行的扩展。

具体实施方式

图 1A 示出计算机系统 10 的框图。计算机系统 10 可以是个人计算机、大型计算机或任何其它类型的计算机或数据处理系统；如下文所述，计算机系统 10 还可以是由另一计算机系统上运行的管理程序提供的虚拟机。计算机系统 10 包括中央处理单元（“CPU”）11、随机访问存储器（“RAM”）12、存储器寻址设备 13 和基于 I/O 的设备 14。在一个实施例中，存储器寻址设备 13 可以是闪速存储器卡。在另一个实施例中，存储器寻址设备 13 可以是 CPU 11 可直接访问以进行存储器操作的任何设备。在一个实施例中，基于 I/O 的设备 14 可以是硬盘驱动器。在其它实施例中，基于 I/O 的设备 14 可以是允许使用 I/O 操作向和从 RAM 12 拷贝数据的任何设备。计算机系统 10 还可包括存储器寻址设备 13 和/或基于 I/O 的设备 14 的多个实例。CPU 11、RAM 12 以及设备 13 和 14 可连接到系统总线 15。CPU 11 可直接访问 RAM 12 和存储器寻址设备 13 以进行存储器操作。CPU 11 不能直接访问基于 I/O 的设备 14 以进行存储器操作，但是可使用 I/O 操作将数据从设备 14 拷贝到 RAM 12，反之亦然。计算机系统 10 运行允许运行一个或多个应用程序的操作系统（下文将更详细地说明）；该操作系统管理和调节应用程序对计算机系统 10 的各种资源（CPU 11、RAM 12、设备 13 和 14）的访问。

在一些实施例中，计算机系统 10 可以是在另一个计算机系统上运行的管理程序仿真的虚拟机。图 1B 示出了这样的实施例，其中计算机系统 10 包含虚拟 CPU 11、虚拟 RAM 12 以及虚拟设备 13 和 14。由管理程序 21 提供所有虚拟组件，该管理程序是在本身包含 CPU、RAM 和设备的另一个计算机系统 20 上运行的软件程序。管理程序 21 或是完全用软件或是通

过使用计算机系统 20 提供的虚拟化硬件辅助功能提供虚拟组件 10－14。除了虚拟机 10 之外，可在计算机系统 20 上的管理程序 21 下同时运行其它虚拟机 22。在一个实施例中，管理程序 21 可以是运行在 IBM eServer zSeries 大型计算机上的 IBM 公司制造和销售的 z/VM 软件程序。在此实施例内，计算机系统 10 可以是 z/VM 提供的虚拟机，并且存储器寻址设备 13 可以是定义于 z/VM 之下并且虚拟机可用的不连续保存分段(“DCSS”)。DCSS 是同时可被一个或多个虚拟机使用的由 z/VM 管理的存储器段；即使得 DCSS 同时可被多个虚拟机使用，z/VM 通常仅在计算机系统 20 的真实 RAM 内保持 DCSS 内容的单个拷贝。

如图 1C 所示，CPU 11 可直接访问以进行存储器操作的所有组件构成计算机系统 10 的系统存储器地址空间 30。RAM 12 和存储器寻址设备 13 是系统存储器地址空间 30 的一部分；另外，其它组件（未示出）例如只读存储器（“ROM”）或视频卡帧缓冲器也可以是系统存储器地址空间 30 的一部分。在发生执行时，CPU 11 执行的每个程序指令和 CPU 11 执行的指令所访问的所有存储器数据必须出现在系统存储器地址空间 30 内。为了增加可用存储器表面上的大小，并且防止同时运行在相同计算机系统上的不同应用程序意外地修改彼此的存储器，操作系统为每个应用程序提供“虚拟地址空间”，并且在该虚拟地址空间内提供对系统存储器地址空间的所选择部分的访问。当 CPU 11 执行应用代码时，可以仅访问出现在应用的虚拟地址空间内的存储器。为了在虚拟地址空间和系统存储器地址空间之间转换，将虚拟地址空间和系统存储器地址空间两者分成相等大小的块，该块通常被称为“页”。

图 1C 示出了计算机系统 10 的系统存储器地址空间 30。另外示出了应用程序的虚拟地址空间 31。对于可在虚拟地址空间 31 内寻址的每个页，必须存在描述页的实际状态的页描述符。在任何时刻该页可存在于或不存在于系统存储器地址空间 30 内。如果页存在于系统存储器地址空间 30 内，页描述符将指出该事实并且将指出页在系统存储器地址空间 30 内的位置。如果页不存在于系统存储器地址空间 30 内，则页描述符将指出该事实并且

将包含附加信息，该附加信息允许操作系统定位应用程序希望通过该页访问的内容。用于虚拟地址空间的所有页描述符的集合称为页表。由操作系统保持用于虚拟地址空间的页表。

当 CPU 11 执行虚拟地址空间 31 内的应用代码时，使用用于虚拟地址空间 31 的页表将 CPU 11 访问的每个存储器地址从虚拟地址空间 31 内的虚拟地址转换到系统存储器地址空间 30 的系统存储器地址。由调页机制执行该转换，调页机制可以是硬件或软件实现或者是两者的组合。对于一些实施例，由存在于 CPU 11 内的调页单元实现调页机制。当 CPU 11 访问目前在系统存储器地址空间 30 内不具对应页的虚拟地址空间 31 内的页时，该调页机制使 CPU 11 产生页错误中断。该中断使被称为“页错误处理器”的操作系统代码将所需的内容放入系统存储器地址空间 30 内的某页内并更新页表，以便将该虚拟页映射到系统存储器地址空间 30 内的该页。然后应用可以继续运行并访问该页。此过程通常被称为“按需调页”。

在图 1C 所示的示例情况下，虚拟地址空间 31 同时保持四个页。三个页（32a－32c）包含目前执行的应用代码，一个页（32d）包含该应用访问的数据。从驻留在基于 I/O 的设备 14 上的块 34a－34c 内的应用程序文件装入页 32a－32c 内包含的应用代码。实际上应用代码页 32a 和 32b 目前存在于系统存储器地址空间 30 内，并驻留在 RAM 12 的页 33a 和 33b 内；类似的，数据页 32d 存在于系统存储器地址空间内，并驻留在 RAM 12 的页 33d 内。应用代码页 32c 目前不存在于系统存储器地址空间 30 内。一旦应用访问页 32c，操作系统的页错误处理器将在 RAM 12 内分配新页 33c（未示出），执行 I/O 操作以便将块 34c 的内容拷贝到页 33c 内，并更新页表以便将虚拟地址空间 31 的页 32c 映射到系统存储器地址空间 30 的页 33c。对于页 32a 和 32b，在图 1C 所示的时刻此过程已经完成。数据页 32d/33d 保持应用在运行时产生的内容，它不被从基于 I/O 的设备 14 装入。

如图 1C 中可见，当 CPU 11 执行驻留在基于 I/O 的设备 14 上的应用程序时，需要 RAM 12 的页以便在执行应用程序代码时保持该应用程序代码。但是，如果应用程序驻留在存储器寻址设备上，这是不需要的，使得

更多的 RAM 12 页可用于其它用途（或者可选择地，允许计算机系统 10 以较少的 RAM 总量完成其预期任务）。此过程通常被称为“原地执行”。如同图 1C，图 1D 示出了运行在虚拟地址空间内的相同的应用，但是应用现在驻留在存储器寻址设备 13 上而不是基于 I/O 的设备 14 上并且被原地执行。如图 1D 中所示，来自应用文件的应用代码驻留在存储器寻址设备 13 的块 35a－c 内。因为存储器寻址设备 13 存在于系统存储器地址空间 30 内，可分别将块 35a 和 35b 直接映射到虚拟地址空间 31 的页 32a 和 32b。类似地，一旦应用访问页 32c，操作系统的页错误处理器将简单地在虚拟地址空间 31 的页 32c 和系统存储器地址空间 30 的页 35c 之间建立另一个映射；操作系统不需要在 RAM 12 内分配任何页。但是应指出，如在图 1C 内所示的，数据页 32 被映射到 RAM 12 的页 33d。

图 1E 设想了不包含本发明的现有技术的操作系统 40 的组件结构。仅示出了实现图 1C 和 1D 内所示的按需调页和原地执行功能所需的那些组件。操作系统 40 包含存储器/文件管理器 41，该管理器处理通过接口 48 来自应用程序 49 的请求以访问驻留在基于 I/O 的设备 14 上或存储器寻址设备 13 上的文件。为了访问基于 I/O 的设备 14 上的文件，存储器/文件管理器 41 通过文件系统 I/O 接口 45 与文件系统驱动器 43 交互，文件系统驱动器 43 继而通过设备 I/O 接口 46 与设备驱动器 44 交互。操作系统 40 可包含多个版本的文件系统驱动器，每个驱动器负责处理特定的文件系统类型；每个文件系统驱动器实现相同的文件系统 I/O 接口 45。类似的，操作系统 40 可包含多个版本的设备驱动器，每个设备驱动器负责处理特定的设备类型；每个设备驱动器实现相同的设备 I/O 接口 46。

在如图 1E 所示的操作系统 40 的组件结构内，设备驱动器 44（以及所有设备驱动器）的职责是将数据从基于 I/O 的设备上的特定位置拷贝到 RAM 内，反之亦然。该设备驱动器不具有关于设备内容或者这些内容的组织方式的任何知识。通常将驻留在设备上的数据分成被称为“块”的块，以“块号”标识每个块。设备 I/O 接口 46 允许设备驱动器请求将数据从以块号 B 标识的块拷贝到以地址 A 开始的 RAM 的块内，反之亦然。

为了允许实现数据在设备上的结构化存储，操作系统 40 提供文件系统驱动器。文件系统驱动器允许访问作为“文件”的集合存储在设备上的数据，每个文件提供有序字节序列的抽象。由文件系统提供的一些方法标识每个文件，通常是“文件名”。文件系统跟踪每个文件的哪些字节存储在底层设备的哪些块内。执行该记录保持所需的信息通常被称为“文件系统元数据”，并且其本身被存储在底层设备上。文件系统元数据的特定布局因文件系统不同而不同。在如图 1E 内所示的操作系统 40 的组件结构内，文件系统驱动器 43（以及所有文件系统驱动器）职责是实现所有必要的逻辑以处理文件系统元数据布局。文件系统 I/O 接口 45 允许文件系统驱动器请求将数据从以某个文件名标识的文件 F 从文件 F 内的特定偏移量 O 开始拷贝到以地址 A 开始的 RAM 的块，反之亦然。为了处理这样的请求，文件系统驱动器将参考文件系统元数据以确定底层设备的哪个块 B 保持有对应于文件 F 内的偏移量 O 的数据，并使用处理底层设备的设备驱动器的设备 I/O 接口 46 在所述设备的块 B 和存储器的特定块之间拷贝数据。

在如图 1E 所示的操作系统 40 的组件结构内，图 1C 内示出的按需调页功能被如下实现：当应用访问目前不存在于系统存储器地址空间内的页时，操作系统的页错误处理器（通常是存储器/文件管理器 41 的一部分）将从页描述符确定该应用希望该页具有的内容。通过指出发现所述内容的文件 F 和文件 F 内的偏移量 O，页描述符标识所述内容。然后存储器/文件管理器 41 将在 RAM 12 内分配新页，并通过文件系统 I/O 接口 45 请求文件系统驱动器 43 将数据从文件 F 内的偏移量 O 拷贝到所述新页内。如上所述，文件系统驱动器 43 将参考文件系统元数据确定底层设备（在此情况下为基于 I/O 的设备 14）的哪个块 B 保持该数据，并且然后使用设备驱动器 44 的设备 I/O 接口 46 将该数据拷贝到所述新页内。一旦 I/O 操作完成，存储器/文件管理器 41 将相应地更新页表。

但是，不能使用文件系统驱动器 43 对驻留在存储器寻址设备 13 上的文件执行原地执行访问，这是因为现有技术的文件系统 I/O 接口 45 不适合于该目的。而是在如图 1E 所示的操作系统 40 的组件结构内，由 XIP 管理

器 42 处理原地执行访问，XIP 管理器 42 紧密地集成在存储器/文件管理器 41 内，并直接访问存储器寻址设备 13。因此 XIP 管理器 42 负责实际访问存储器寻址设备 13 和处理存储在所述设备上的数据的文件系统布局。不能使用 XIP 管理器 42 所支持的那些文件系统布局之外的任何其它文件系统布局访问存储器寻址设备 13 上的数据，即使操作系统 40 另外为这样的文件系统提供文件系统驱动器也是如此。

本发明除去了此限制。图 2A 示出了包含本发明的实现了对存储器寻址设备 13 的原地执行访问的操作系统 50 的组件结构。与图 1E 中所示的现有技术的操作系统 40 相比，操作系统 50 保持了到计算机系统 10 的所有硬件组件（RAM 12、基于 I/O 的设备 14、存储器寻址设备 13）的相同接口。其还使用到应用程序 49 的相同接口 48。存储器/文件管理器 51 是现有技术提供的存储器/文件管理器 41（见图 1E）的修改版本，并且文件系统驱动器 52 是现有技术提供的文件系统驱动器 43（见图 1E）的修改版本。操作系统 50 还提供了访问存储器寻址设备 13 的设备驱动器 53。应指出，操作系统 40 的某些现有技术的版本也提供类似的访问存储器寻址设备 13 的设备驱动器，但是这种驱动器仅实现了设备 I/O 接口 46（见图 1E），并且不能提供原地执行功能。此功能由 XIP 管理器 42 实现，但是该管理器直接访问存储器寻址设备 13 而不使用任何设备驱动器。如图 2A 中所示，本发明不再需要 XIP 管理器组件。而是现在将原地执行功能集成在现有的组件存储器/文件管理器 51、文件系统驱动器 52 和设备驱动器 53 内。这可通过使用两个新的接口来实现：由文件系统驱动器 52 提供的文件系统直接访问接口 54，和由设备驱动器 53 提供的设备直接访问接口 55。设备直接访问接口 55 的核心特征是提供检索存在于系统存储器地址空间内的存储器寻址设备的块 B 的系统存储器地址 A 的方法。类似的，文件系统直接访问接口 54 的核心特征是提供检索文件 F 在偏移量 O 处的内容的系统存储器地址 A 的方法，其中文件 F 驻留在存在于系统存储器地址空间内的存储器寻址设备上。下面将更详细地描述这些直接访问接口的使用。

如图 2A 所示，操作系统 50 通过接口 48 与应用程序 49 交互。接口 48

的一部分包含由应用程序 49 触发的按需调页请求，应用程序 49 访问目前没有被映射到系统存储器地址空间 30 的页的其虚拟地址空间 31 的页（应指出，虚拟地址空间 31 和系统存储器地址空间 30 如图 1C 和 1D 中所示，其中虚拟地址空间 31 对应于应用 49 的虚拟地址空间）。接口 48 的其它部分包含应用程序 49 对操作系统 50 的请求（“系统调用”）以便读、写或访问驻留在基于 I/O 的设备 14 或存储器寻址设备 13 上的文件的内容。存储器/文件管理器 51 是操作系统 50 的处理应用程序 49 通过接口 48 发出的请求的组件。为了访问驻留在基于 I/O 的设备 14 或存储器寻址设备 13 上的文件的内容，存储器/文件管理器 51 通过文件系统 I/O 接口 45 和/或文件系统直接访问接口 54 与文件系统驱动器 52 交互。操作系统 50 可包含多个版本的文件系统驱动器，每个版本负责处理特定的文件系统布局。所有文件系统驱动器实现相同的文件系统 I/O 接口 45，并且一些文件系统驱动器实现附加的文件系统直接存储接口 54，其适合于对驻留在存储器寻址设备上的文件执行原地执行访问。文件系统驱动器 52 通过设备 I/O 接口 46 和/或设备直接访问接口 55 与设备驱动器 44 和 53 交互。操作系统 50 还可包含多个版本的设备驱动器，每个驱动器负责处理特定类型的设备。所有设备驱动器实现相同的设备 I/O 接口 46，并且用于存储器寻址设备 13 的设备驱动器还附加地实现设备直接访问接口 55，其适于对驻留在存储器寻址设备 13 上的文件执行原地执行访问。

对于提供 I/O 接口 45 和直接访问接口 54 两者的文件系统驱动器，存储器/文件管理器 51 能够对由这种文件系统驱动器处理的文件执行常规访问和原地执行访问。图 2B 和 2C 分别更详细地示出执行常规和原地执行访问所需的控制流。图 2D 将详细地示出对于任何特定的文件访问选择使用两种方法中的哪一种所需的判定逻辑。

图 2B 示出了当操作系统 50 处理访问驻留在基于 I/O 的设备 14 上的文件的请求时的控制流。应指出，此控制流等同于现有技术的操作系统 40 为相应的访问所使用的控制流。依次用动作 60a－f 表示该控制流的步骤。在应用 49 已试图访问虚拟地址空间 31 的页 32c 之后，这里示出的特定动

作相应于图 1C 所示的情况。由于该页目前没有被映射到系统存储器地址空间 30 的任何页，所以由操作系统 50 的存储器/文件管理 51 组件产生并且处理页错误中断。在图 2B 中这被以动作 60a 表示。存储器/文件管理器 51 从页 32c 的页描述符读出该应用希望其内容相应于文件 F 在偏移量 O 处的内容。其确定文件 F 驻留在由文件系统驱动器 52 处理的文件系统上。其还确定文件 F 不支持原地执行（这将在下文更详细地说明）。然后其在 RAM 13 分配空闲页 33c 并确定其系统存储器地址 A，并且通过文件系统驱动器 52 的文件系统 I/O 接口 45 请求将文件 F 在偏移量 O 处的内容拷贝到系统存储器地址 A 处的页（动作 60b）。文件系统驱动器 52 确定文件 F 在偏移量 O 处的数据驻留在基于 I/O 的设备 14 的块 B 上，并且设备驱动器 44 负责访问基于 I/O 的设备 14。然后通过设备驱动器 44 的设备 I/O 接口 46 请求将基于 I/O 的设备 14 的块 B 拷贝到系统存储器地址 A 处的页（动作 60c）。设备驱动器 44 在基于 I/O 的设备 44 上实现 I/O 操作 61 以执行该拷贝。一旦 I/O 操作 61 已完成，设备驱动器 44 通过设备 I/O 接口 46 将完成报告给文件系统驱动器 52（动作 60d），其类似地通过文件系统 I/O 接口 45 将完成报告给存储器/文件管理器 51（动作 60e）。存储器/文件管理器 51 最终建立虚拟地址空间 31 的页 32c 到系统存储器地址空间 30 的地址 A 处的页 33c 的映射，页 33c 现在保持所请求的内容（动作 60f）。

图 2C 类似地示出了当操作系统 50 处理请求以便对驻留在存储器寻址设备 13 上的文件执行原地执行访问时的控制流。应指出，此流程不同于现有技术操作系统 40 为相应的访问所使用的流程，并且使用本发明所描述的新的直接访问接口。还应注意，对基于存储器的设备 13 上的文件的某些访问可能不适合于原地执行访问；在这样的情况下取而代之使用上面描述的如图 2B 给出等同的控制流执行对该文件的常规 I/O 访问。由于设备驱动器 53 像设备驱动器 44 那样也实现了设备 I/O 接口 46，所以这是可能的。

由动作 70a－f 依次表示图 2C 中所示的控制流的步骤。在应用 49 已试图访问虚拟地址空间 31 的页 32c 之后，这里所示的特定动作相应于图 1D 所示的情况。如图 2B，由存储器/文件管理器 51 产生和处理页错误中断（动

作 70a），存储器/文件管理器 51 再次从页 32c 的页描述符读出该应用希望其内容应相应于文件 F 在偏移量 O 处的内容。其再次确定文件 F 驻留在由文件系统驱动器 52 处理的文件系统上。其还确定文件 F 支持原地执行（这将在下文更详细地说明）。现在它通过文件系统驱动器 52 的文件系统直接访问接口请求检索文件 F 在偏移量 O 处的内容的系统存储器地址（动作 70b）。文件系统驱动器 52 确定文件 F 在偏移量 O 处的内容驻留在存储器寻址设备 13 的块 B 上，并且设备驱动器 53 负责访问存储器寻址设备 13。然后它通过设备驱动器 53 的设备直接访问接口 55 请求检索存储器寻址设备 13 的块 B 的系统存储器地址（动作 70c）。设备驱动器 53 确定存储器寻址设备 13 的块 B 相应于系统存储器地址空间 30 的页 35c，并通过设备直接访问接口 55 将其地址 A 返回文件系统驱动器 52（动作 70d），文件系统驱动器 52 类似地通过文件系统直接访问接口 54 将地址 A 返回存储器/文件管理器 51（动作 70e）。存储器/文件管理器 51 最终建立虚拟地址空间 31 的页 32c 到系统存储器地址空间 30 的地址 A 处的页 35c 的映射（动作 70f）。

图 2D 示出了当确定使用文件系统 I/O 接口还是使用文件系统直接访问接口访问文件 F 时操作系统 50（图 2A）内的控制流。操作系统首先确定哪个文件系统驱动器负责用于保持着文件 F 的文件系统 FS。如果该文件系统驱动器根本不支持文件系统直接访问接口，则使用文件系统 I/O 接口。否则，操作系统确定哪个设备驱动器负责用于文件系统 FS 的底层设备 D。如果该设备驱动器根本不支持设备直接访问接口，则也使用文件系统 I/O 系统接口。否则，操作系统确定文件系统 FS 是否被用户配置为对文件 F 进行原地执行访问。如果是，则使用文件系统直接访问接口，否则使用文件系统 I/O 接口。在一些实施例中，用户仅可以选择允许对 FS 上的所有文件进行原地执行访问或是对所有文件都不进行原地执行访问。在其它实施例中，用户可对每个单独的文件分别进行选择。另外，在一些实施例中，如果其它可配置的文件系统参数已被用户设置为与原地执行相兼容的值，才可以允许原地执行访问；例如必须将文件系统块的大小配置成等于或者

是系统存储器页大小的数倍的值。

注意操作系统不必为对文件 F 的每一个访问执行图 2D 所示的整个判定逻辑。而是当第一次访问文件 F 时进行一次选择，并且在与文件 F 相关联的操作系统数据结构内加以记忆；随后的访问将重新使用在所述数据结构内存储的该判定的结果。

图 3A－I 更详细地示出了 Linux 操作系统内的本发明的一个实施例。通过扩展标准操作系统 I/O 组件结构的接口而不是整个替换它，本发明将原地执行功能集成在标准操作系统 I/O 组件结构内。Linux 提供了下述与执行 I/O 操作有关的组件层：

设备驱动层，其允许访问背后的存储设备而不需要所涉及的特定硬件的知识，

文件系统层，其允许使用逻辑文件视图访问背后的存储设备而不需要关于背后的存储设备的知识，

存储器管理层，其允许执行应用而不需要该应用了解操作系统使用的虚拟存储器和动态地址转换技术。

图 3A 示出了在许多现代操作系统内实现的设备抽象。该例子示出了 Linux 内的设备驱动器。调用 make_request 函数以便提交从/向该设备读/写数据的请求。request_queue 和 do_request 函数是 Linux 内的优化的一部分。设备驱动器将按循环“发送请求→处理请求→中断处理器→发送请求”运行，直到提交给它的所有工作被完成为止。

设备驱动层允许提交读或写请求。该层的频繁的用户是文件系统的读页（多个）/写页（多个）操作，它们从/向文件传输数据。为了对所述数据寻址，使用物理块号。

如图 3B 内所示，本发明提供了对设备驱动器层的接口的扩展。在保持现有接口不动的同时，该扩展提供了可用于得到设备上的数据的直接引用的功能。可使用此引用访问设备上的数据而不需要提交请求并等待它们的完成。此扩展可选择地可由访问存储器寻址设备的设备驱动器实现。新的操作 direct_access 获得与 make_request 类似的块号，但是不传输任何

数据。而是返回对数据的引用。在此后的任何时间可使用此引用向该物理块读或写任何数据直到再次关闭/卸载该设备而不需要与设备驱动层进行其它交互。新接口的使用是可选的，对于不支持新接口的用户例如原始设备驱动器，传统的 make_request 接口保持不动。对于支持通用的文件系统，传统接口是重要的，通用的文件系统使用传统接口传输文件系统元数据（i节点、目录项等）。

图 3C 示出了 Linux 内的通用的文件系统如何可以使用设备驱动器的 make_request 函数实现地址空间操作。readpage（s）/writepage（s）函数使用 get_block 函数 [在处理多个页时重复执行]标识与目的页（多个）相关联的设备上的物理块号（多个）。如图 3A 所示的 make_request 函数用于访问数据。在 Linux 内，文件系统通常与文件系统库函数一起使用地址空间操作以执行文件操作例如 sys_read（）和 sys_write（）。这些地址空间操作和库函数的使用是可选择的，但是大部分通用文件系统使用它们。地址空间操作 readpage（）、readpages（）、writepage（）和 writepages（）被用来从/向设备驱动器读/写一个或多个存储器页的数据。为了寻址存储器页，使用逻辑文件句柄和偏移量。该寻址被文件系统转换成物理块号。

如图 3D 所示，本发明以名为“get_xip_page”的函数提供了对地址空间操作接口的扩展，get_xip_page 函数允许检索对给定存储器页之后的存储的引用。该函数使用文件句柄和偏移量以便进行寻址，通过调用 get_block 函数将其转换成物理块号，并从设备驱动器层的 direct_access 函数检索对该物理块之后的存储的引用（如图 3B 所示）。此后的任何时间可使用此引用向该物理块读或写任何数据，直到该文件被删简或文件系统被卸载而不需要与文件系统或设备驱动器层的其它交互。当被支持时新接口的使用是强制性的，出于数据完整性的原因，对于一个文件或是支持传统的 readpage（s）/wirtepage（s）接口或是新的 get_xip_page 接口。文件系统可基于每个文件自由选择。

readpage（s）/wirtepage（s）函数的主要用户是文件系统库函数。图 3E 示出了在 Linux 内文件系统库函数如何为通用文件系统执行读类型的

文件操作。

Generic_file_read 函数执行与 sys_read（）系统调用相关联的文件操作。

Generic_file_readv 函数执行与 sys_readv（）系统调用相关联的文件操作。

Generic_file_aio_read 函数执行与异步 IO 系统调用相关联的读操作。

Generic_file_sendfile 文件操作执行与 sys_sendfile（）系统调用相关联的文件操作。

所有这些函数间接地调用 generic_mapping_read，generic_mapping_read 使用 readpage（s）函数（如图 3C 内所示）从设备读一个或多个页。虽然对于文件系统这些函数的使用是可选择的，但是大多数通用文件系统使用它们而不是自己实现文件操作。

图 3G 示出了在 Linux 内文件系统库函数如何为通用文件系统执行写类型的文件操作。

Generic_file_write 函数执行与 sys_write（）系统调用相关联的文件操作。

Generic_file_writev 函数执行与 sys_writev（）系统调用相关联的文件操作。

Generic_file_aio_write 函数执行与异步 IO 系统调用相关联的写操作。

所有这些函数间接地调用 generic_file_direct_write（当使用选项 O_DIRECT 打开目标文件时）或 generic_file_buffered_write。这两个函数使用 writepage（s）函数向设备写一个或多个页。

本发明提供了对这些库函数的扩展以便使得它们在被支持时能够使用 get_xip_page 接口。图 3F 示出了对读类型操作的文件系统库函数的扩展。根据是否出现 get_xip_page 地址空间操作，使用 generic_mapping_read 函数或新的 do_xip_mapping_read 函数执行该操作。do_xip_mapping_read 函数使用 get_xip_page 地址空间操作以检索该设备上的目标数据的引用。对于数据传输，直接使用此引用而不需执行 I/O 操作。图 3H 示出了对写

类型操作的文件系统库函数的扩展。根据是否出现 get_xip_page 地址空间操作，使用 generic_file_buffered_write/generic_file_direct_write 函数或新的 generic_file_xip_write 函数执行该操作。generic_file_xip_write 函数使用 get_xip_page 地址空间操作以检索该设备上的目标数据的引用。对于数据传输，直接使用此引用而不需执行 I/O 操作。

实现 get_xip_page 地址空间操作的所有文件系统不需要其它的代码改变以便以由库函数实现的所有文件操作执行原地执行访问。图 3F、3H 和 3I 示出了扩展的库函数如何根据是否实现了 get_xip_page 地址空间操作实行它们的功能。当实现了 get_xip_page 时，所有库函数使用从 get_xip_page 检索到的引用直接执行到该存储设备的所有数据传输。图 3I 示出了对用于文件存储器映射的文件系统库函数进行的扩展。应用对其虚拟地址空间的目前未出现的一部分进行访问。使用 Linux 的标准的依赖体系结构的和核心存储器管理函数来处理引起的页错误。与常规处理不同，该文件系统已安装了用于与目标页相关联的文件的 filemap_xip_nopage 处理器。此处理器使用 get_xip_page 检索该设备上的目标数据的引用。将此引用返回 do_no_page 函数，该函数在应用的虚拟地址空间页表内创建页表表项，以允许应用直接使用该设备上的数据而不需与操作系统的其它牵连。当文件映象被选择为是私有的时（标准机制应用），页表表项稍后可经历写时拷贝机制。

因为应用二进制文件和共享库函数在 Linux 内经历文件映射，并且从而经历上述的页错误机制，所以实现了原地执行的效果。上述扩展保持 Linux 操作系统的与 I/O 相关的组件内的整个结构和层隔离不动：设备驱动器层将物理块号映射到设备，但是不对文件或其它逻辑对象起作用。文件系统执行逻辑文件和物理块号寻址之间的映射，但是不对设备的内部结构起作用。在另一方面，数据传输本身被倒转：当使用新扩展时设备驱动器不传输任何数据。在文件操作库函数中直接传输数据。此解决方法的优点包括对设备驱动器层和通用文件系统的非常小的并且仅仅是非插入的改变。任何通用文件系统可容易地受益于原地执行机制，例如减小的存储器

消耗和增加的性能。

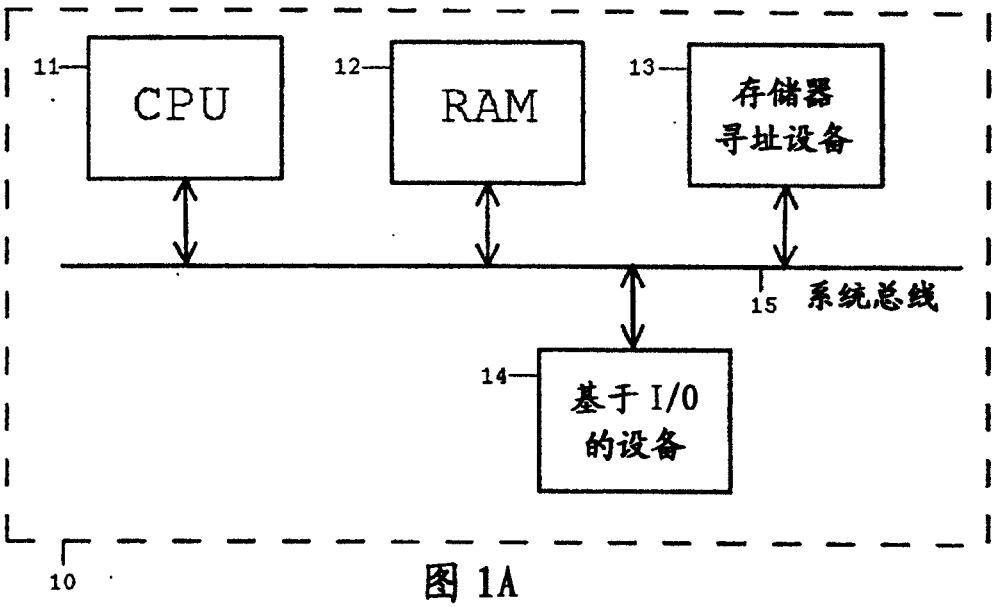


图 1A

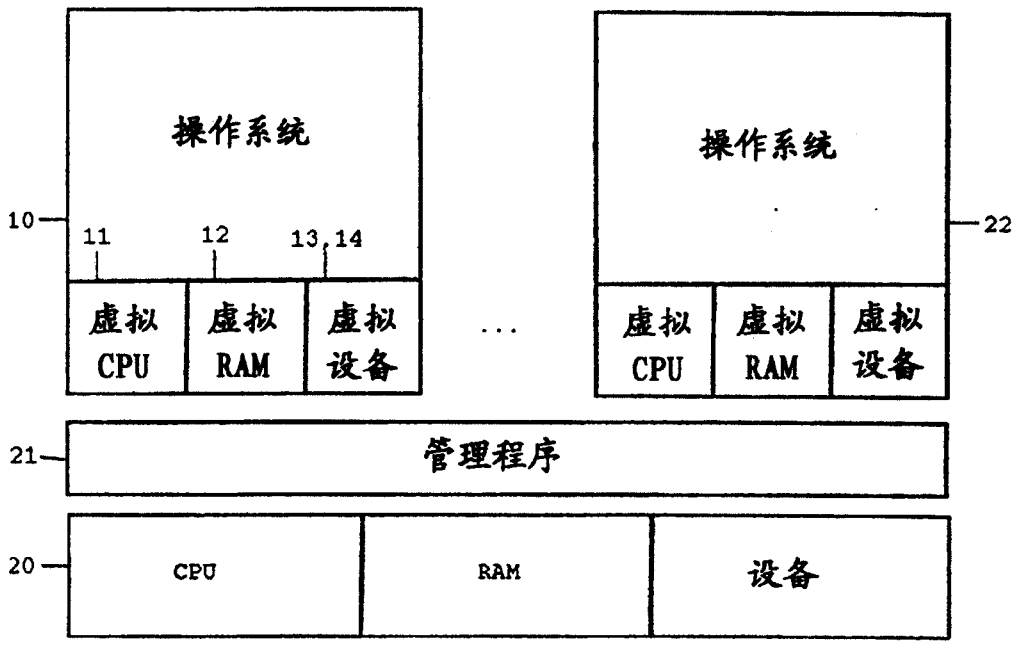

操作系统
10
11
12
13,14
虚拟 CPU
虚拟 RAM
虚拟 设备
...
操作系统
22
虚拟 CPU
虚拟 RAM
虚拟 设备
21
管理程序
20
CPU
RAM
设备

图 1B

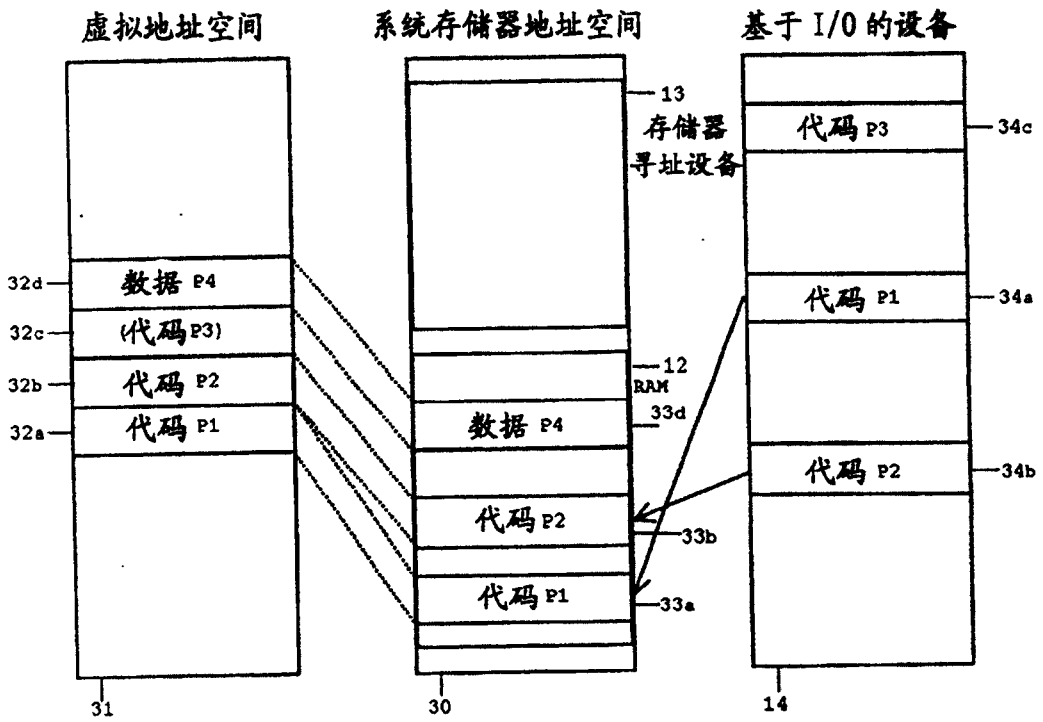

虚拟地址空间
系统存储器地址空间
基于 I/O 的设备
13
存储器
寻址设备
代码 P3
34c
32d
数据 P4
32c
(代码P3)
32b
代码 P2
32a
代码 P1
代码 P1
34a
12
RAM
33d
数据 P4
代码 P2
34b
代码 P2
33b
代码 P1
33a
31
30
14

图 1C

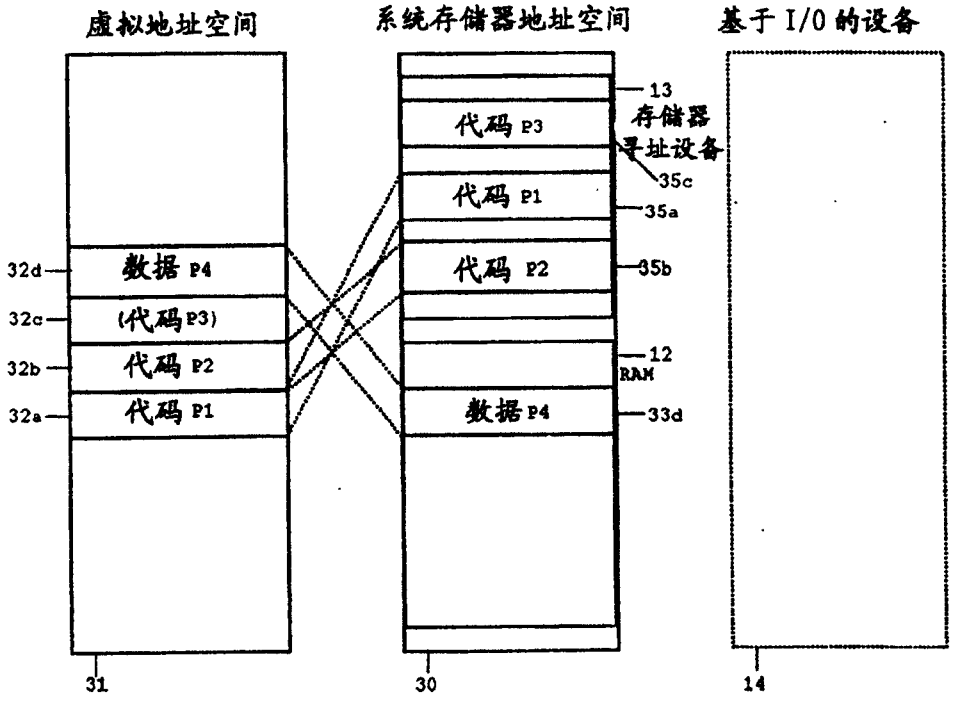

虚拟地址空间
系统存储器地址空间
基于 I/O 的设备
32d
数据 P4
32c
(代码P3)
32b
代码 P2
32a
代码 P1
31
代码 P3
代码 P1
代码 P2
数据 P4
30
13
存储器
寻址设备
35c
35a
35b
12
RAM
33d
14

图 1D

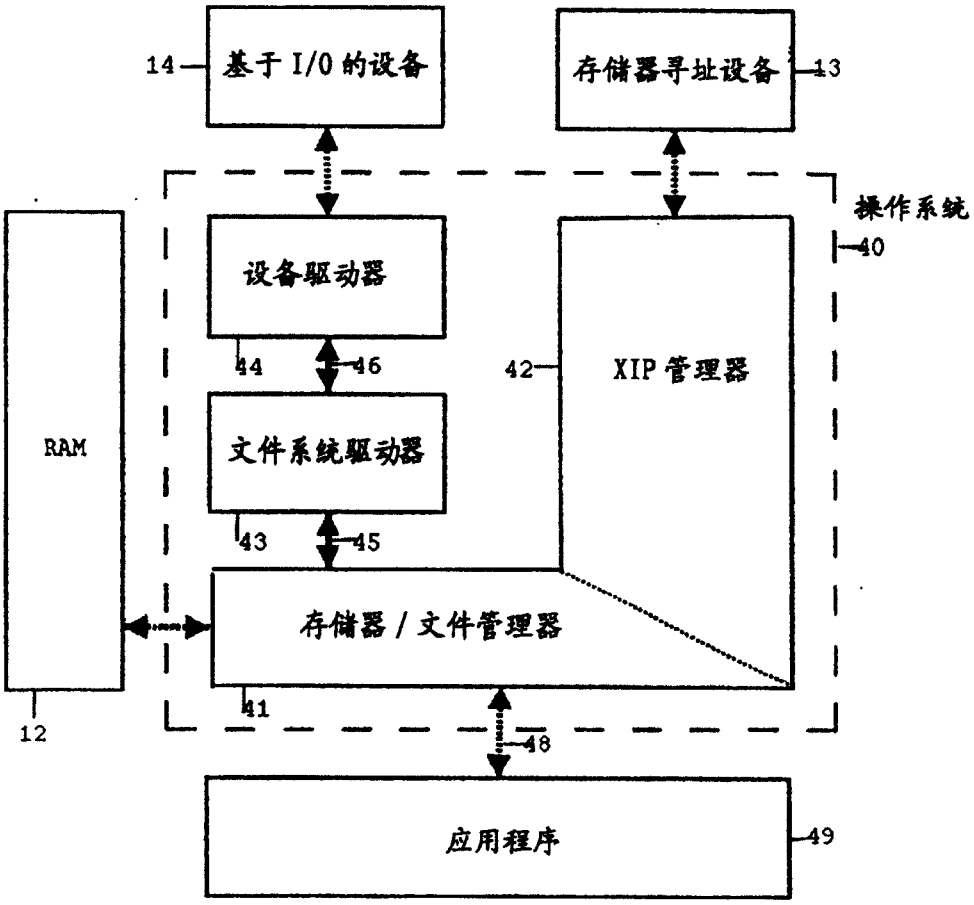

14
基于 I/O 的设备
存储器寻址设备
13
操作系统
40
设备驱动器
44
46
42
XIP 管理器
RAM
文件系统驱动器
43
45
存储器 / 文件管理器
41
12
48
应用程序
49

图 1E

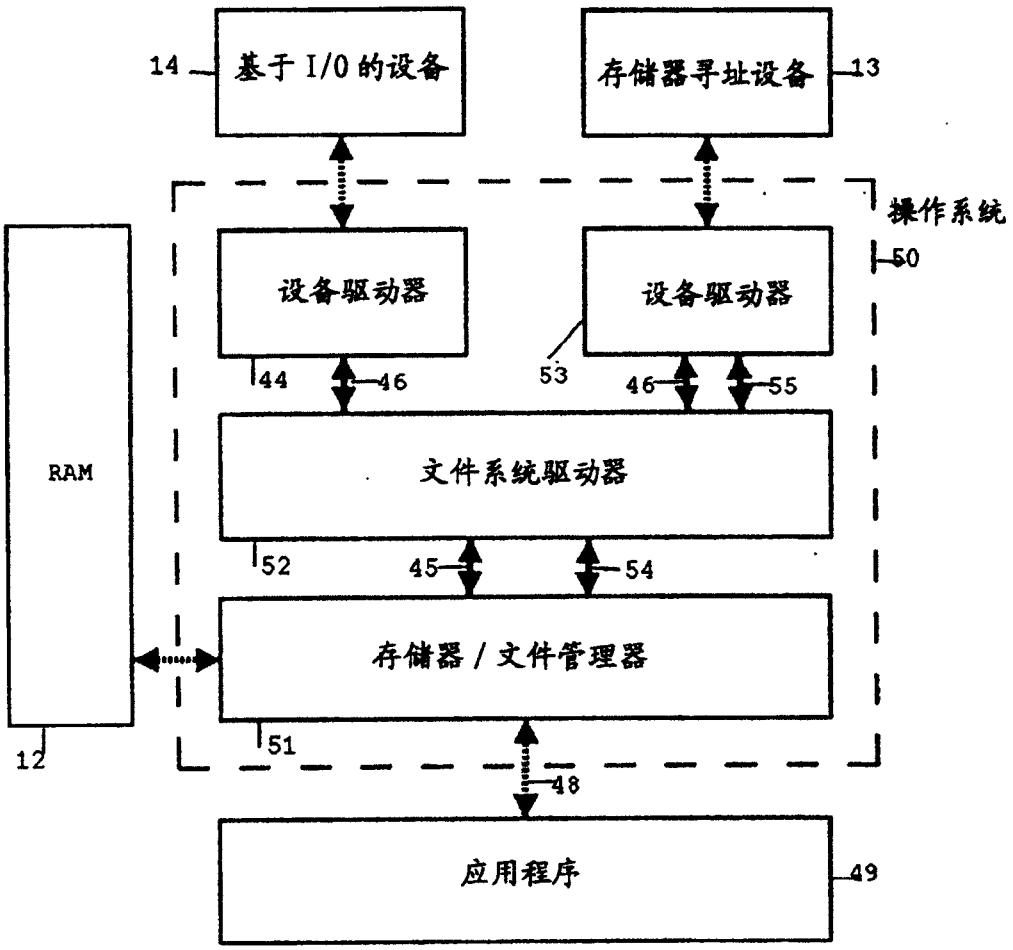

14
基于 I/O 的设备
存储器寻址设备
13
操作系统
50
设备驱动器
设备驱动器
44
46
53
46
55
RAM
文件系统驱动器
52
45
54
存储器 / 文件管理器
12
51
48
应用程序
49

图 2A

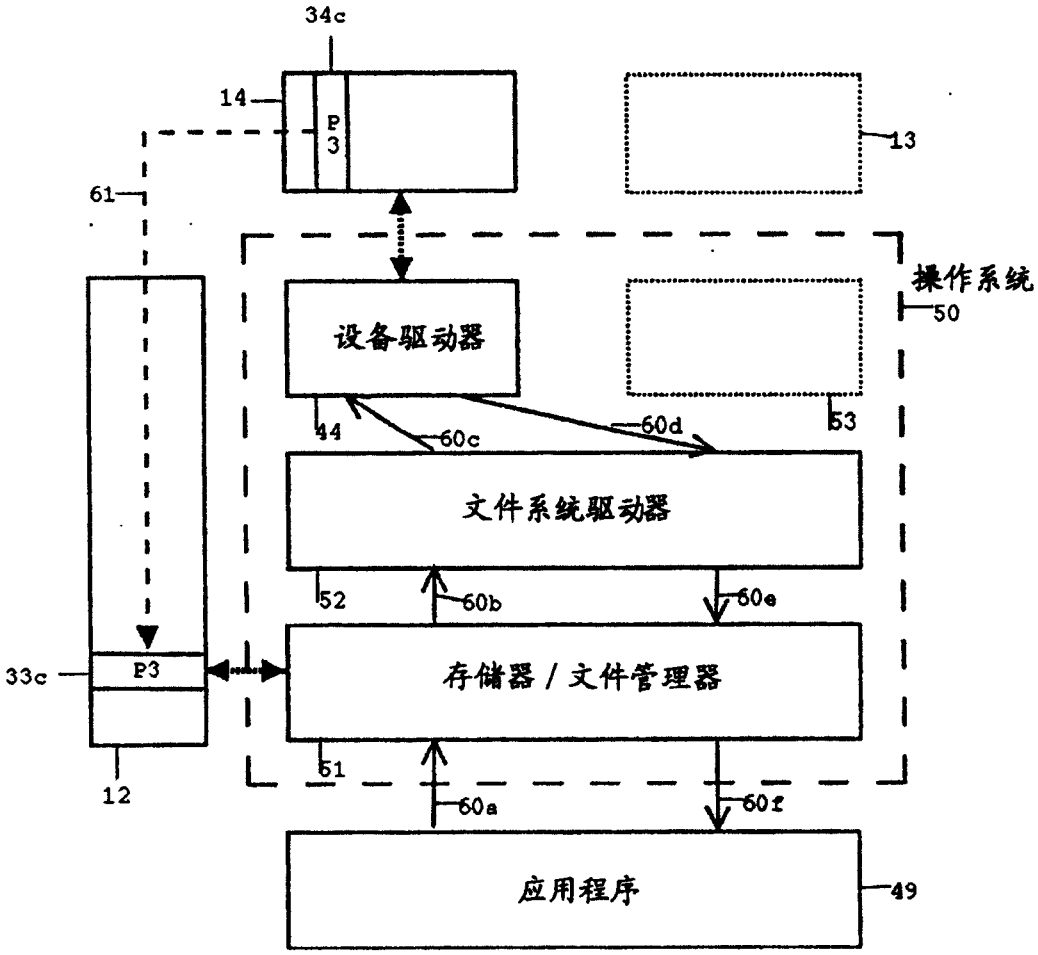

34c
14
P3
13
61
操作系统
50
设备驱动器
44
60c
60d
53
文件系统驱动器
52
60b
60e
33c
P3
存储器/文件管理器
12
51
60a
60f
应用程序
49

图 2B

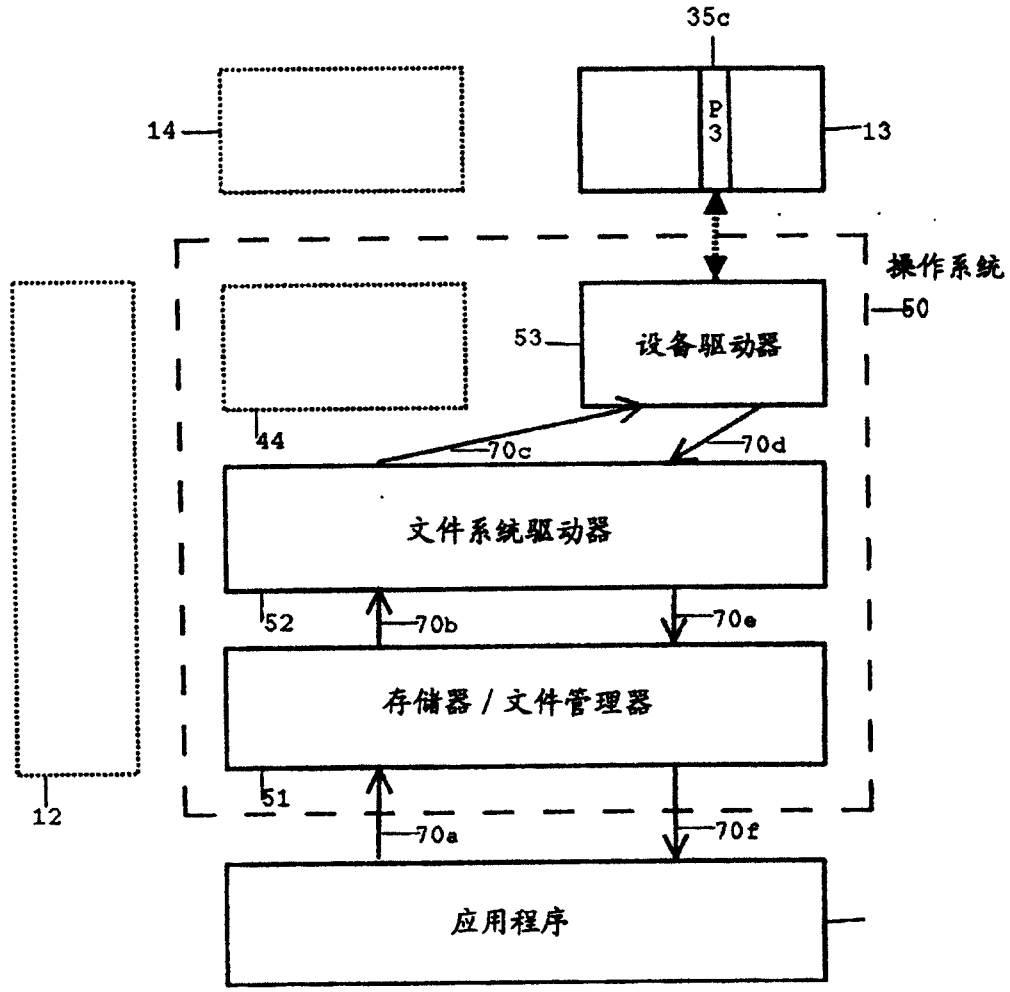

35c
14
P
3
13
操作系统
50
53
设备驱动器
44
70c
70d
文件系统驱动器
52
70b
70e
存储器 / 文件管理器
12
51
70a
70f
应用程序

图 2C

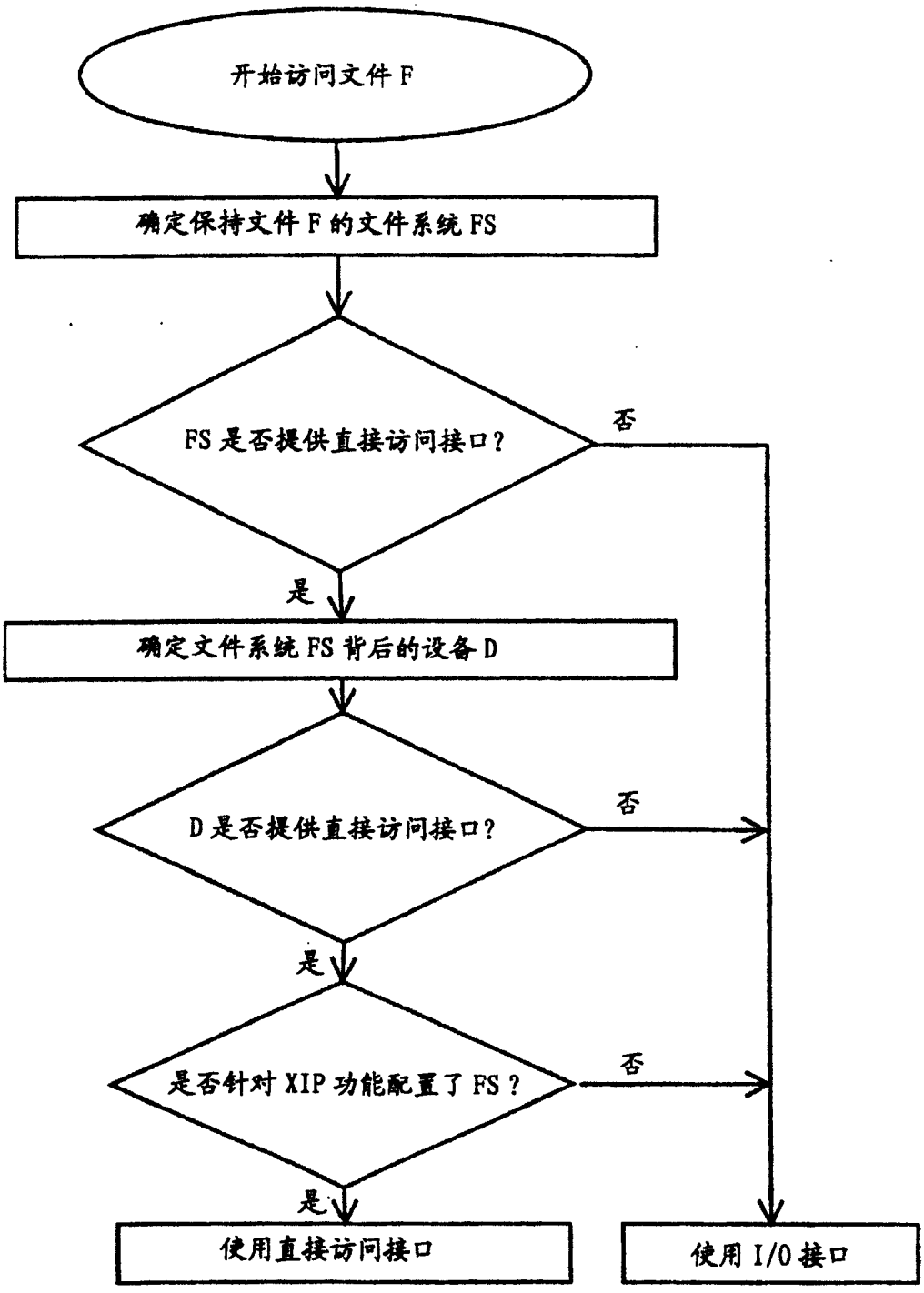

开始访问文件 F
确定保持文件 F 的文件系统 FS
FS 是否提供直接访问接口？
否
是
确定文件系统 FS 背后的设备 D
D 是否提供直接访问接口？
否
是
是否针对 XIP 功能配置了 FS？
否
是
使用直接访问接口
使用 I/O 接口

图 2D

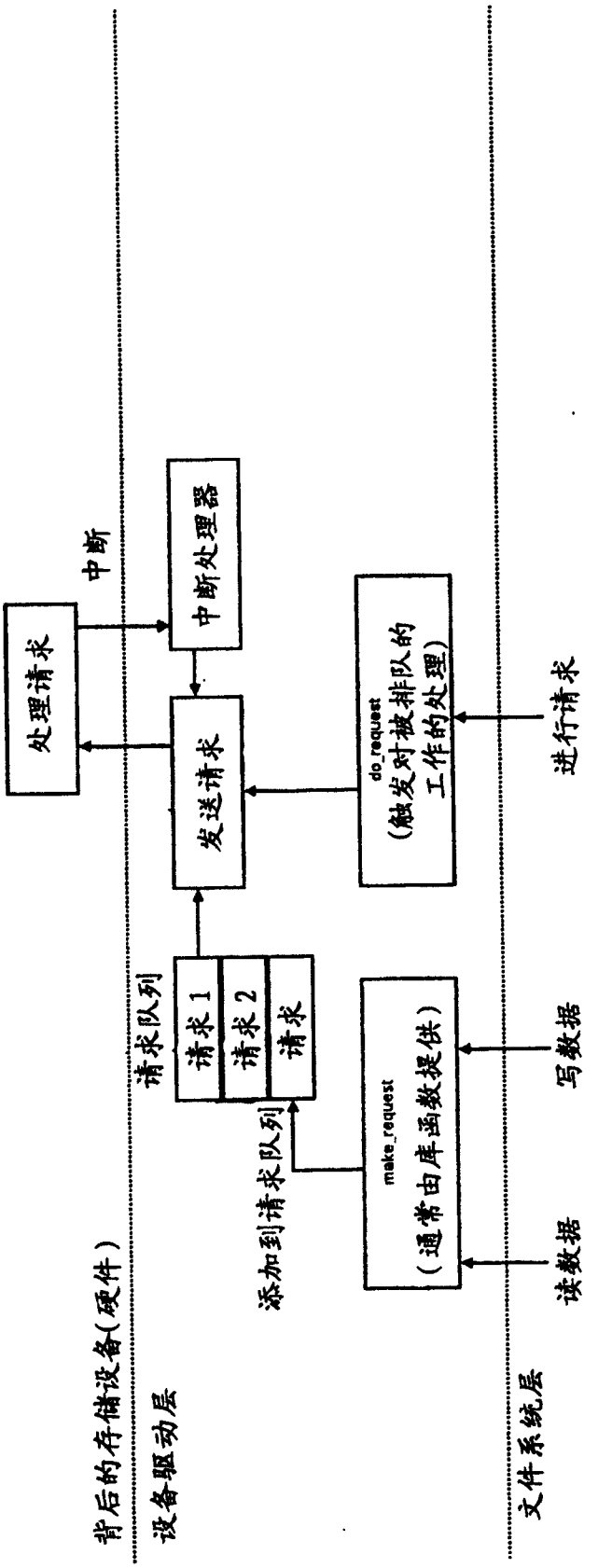

处理请求
中断
中断处理器
发送请求
do_request
(触发对被排队的工作的处理)
进行请求
请求队列
请求 1
请求 2
请求
添加到请求队列
make_request
(通常由库函数提供)
写数据
读数据
背后的存储设备(硬件)
设备驱动层
文件系统层

图 3A

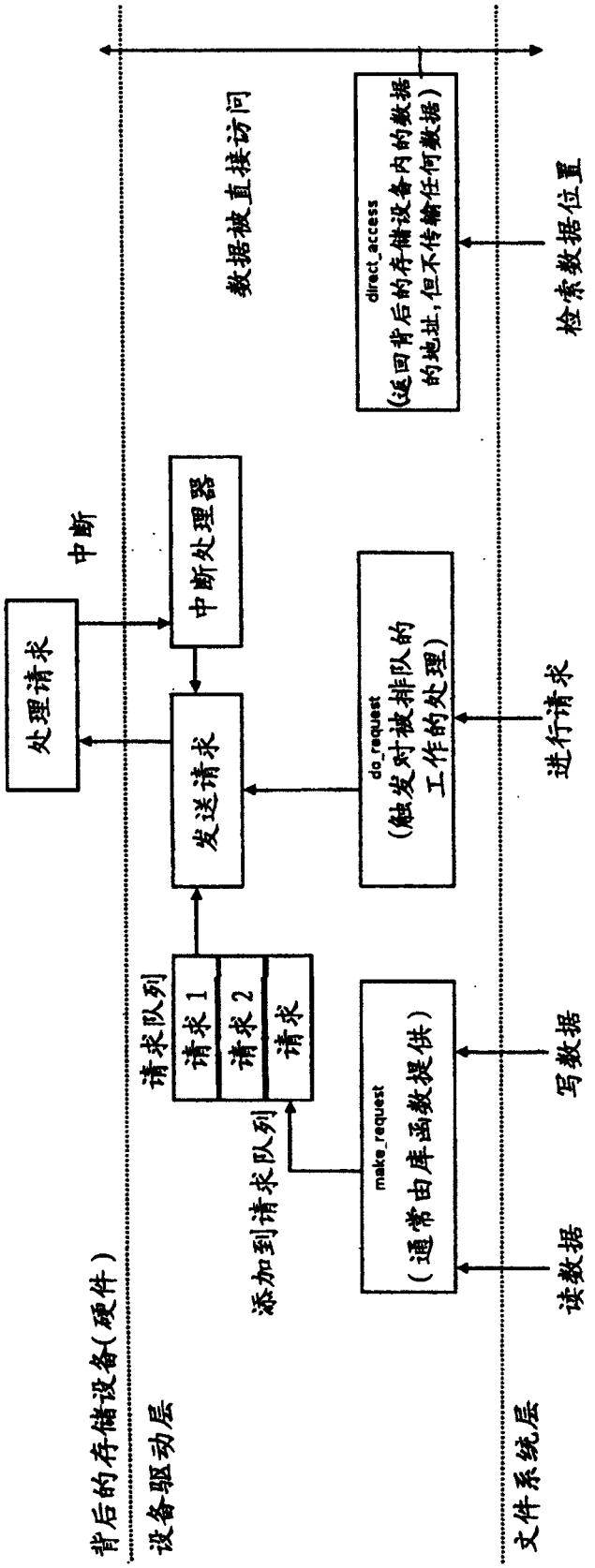

背后的存储设备(硬件)
设备驱动层
文件系统层
处理请求
中断
中断处理器
数据被直接访问
direct_access
(返回背后的存储设备内的数据的地址,但不传输任何数据)
检索数据/位置
发送请求
do_request
(触发对被排队的工作的处理)
进行请求
请求队列
请求 1
请求 2
请求
添加到请求队列
make_request
(通常由库函数提供)
写数据
读数据

图 3B

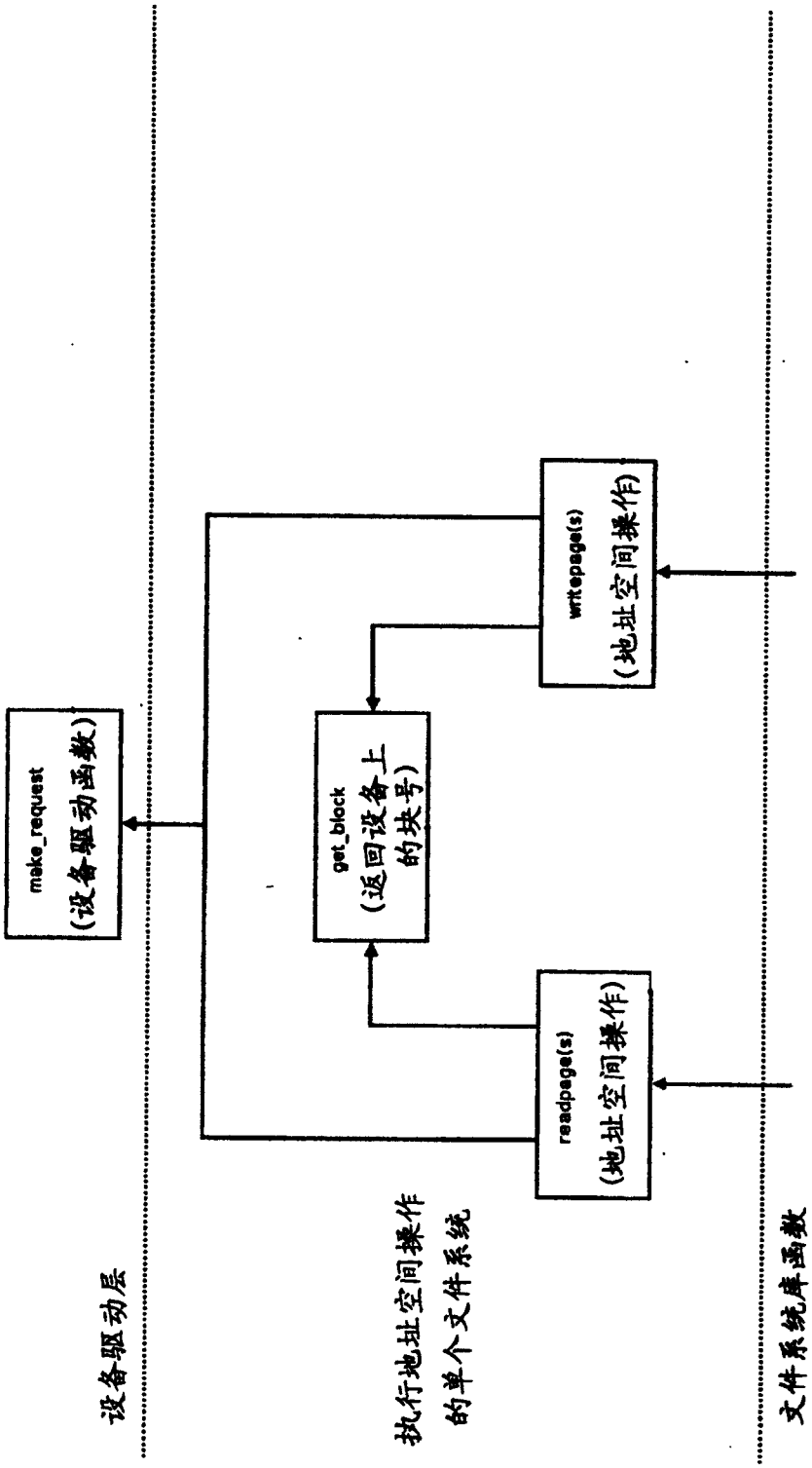

make_request
(设备驱动函数)
get_block
(返回设备上的块号)
writepage(s)
(地址空间操作)
readpage(s)
(地址空间操作)
设备驱动层
执行地址空间操作的单个文件系统
文件系统库函数

图 3C

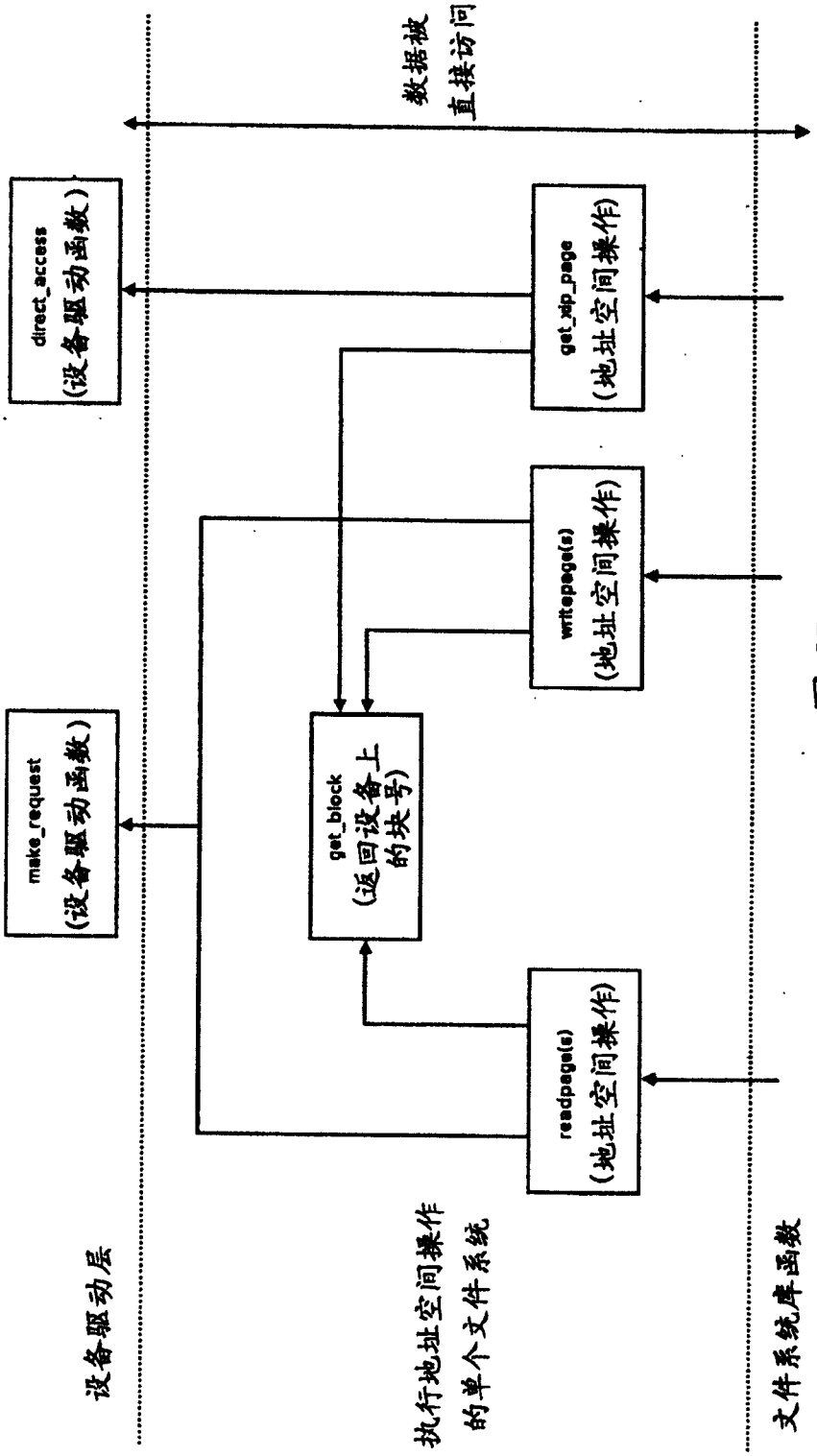

数据被直接访问
direct_access
(设备驱动函数)
get_xip_page
(地址空间操作)
writepage(s)
(地址空间操作)
make_request
(设备驱动函数)
get_block
(返回设备上的块号)
readpage(s)
(地址空间操作)
设备驱动层
执行地址空间操作的单个文件系统
文件系统库函数

图 3D

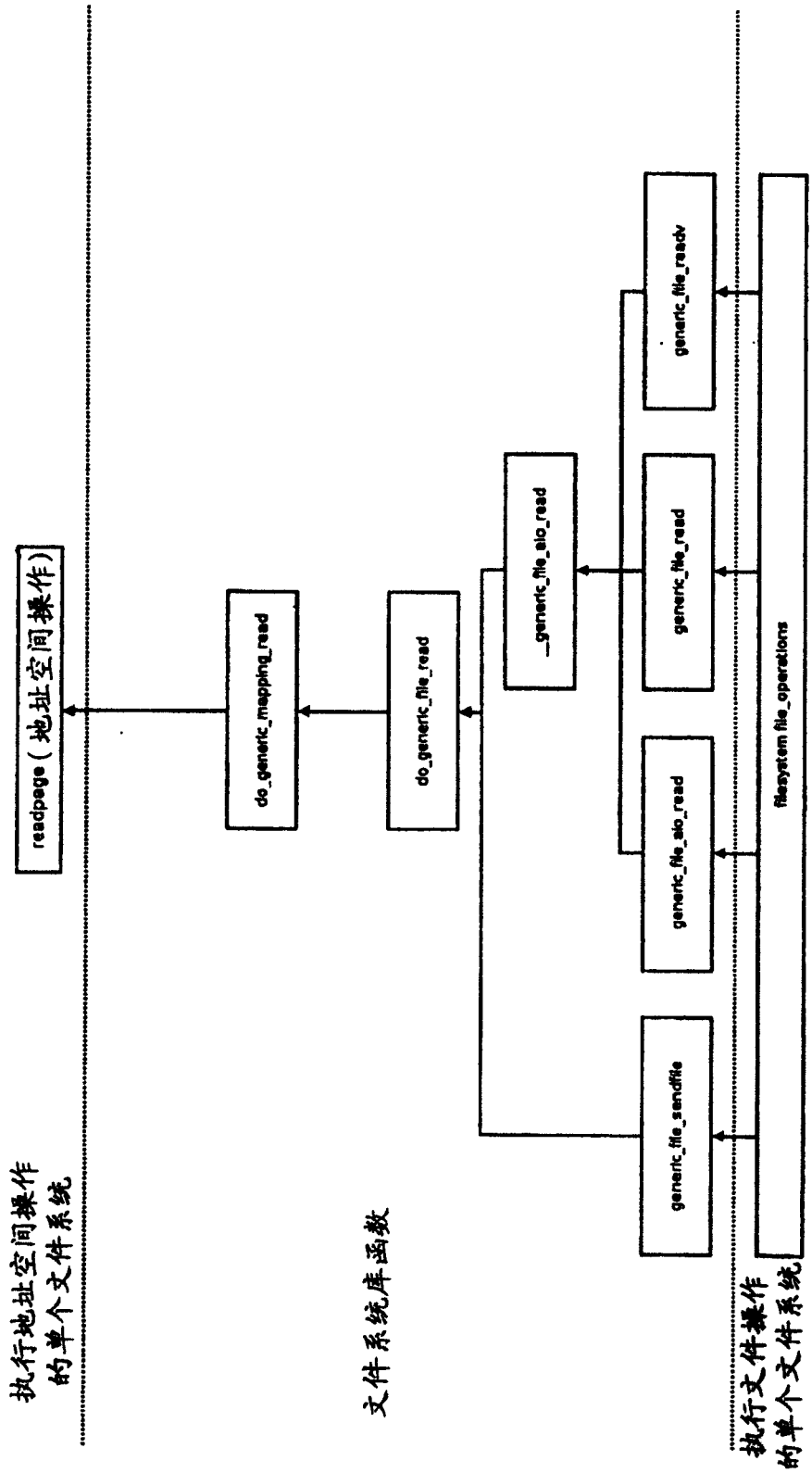

执行地址空间操作的单个文件系统
readpage（地址空间操作）
do_generic_mapping_read
文件系统库函数
do_generic_file_read
_generic_file_aio_read
generic_file_sendfile
generic_file_aio_read
generic_file_read
generic_file_readv
执行文件操作的单个文件系统
filesystem file_operations

图 3E

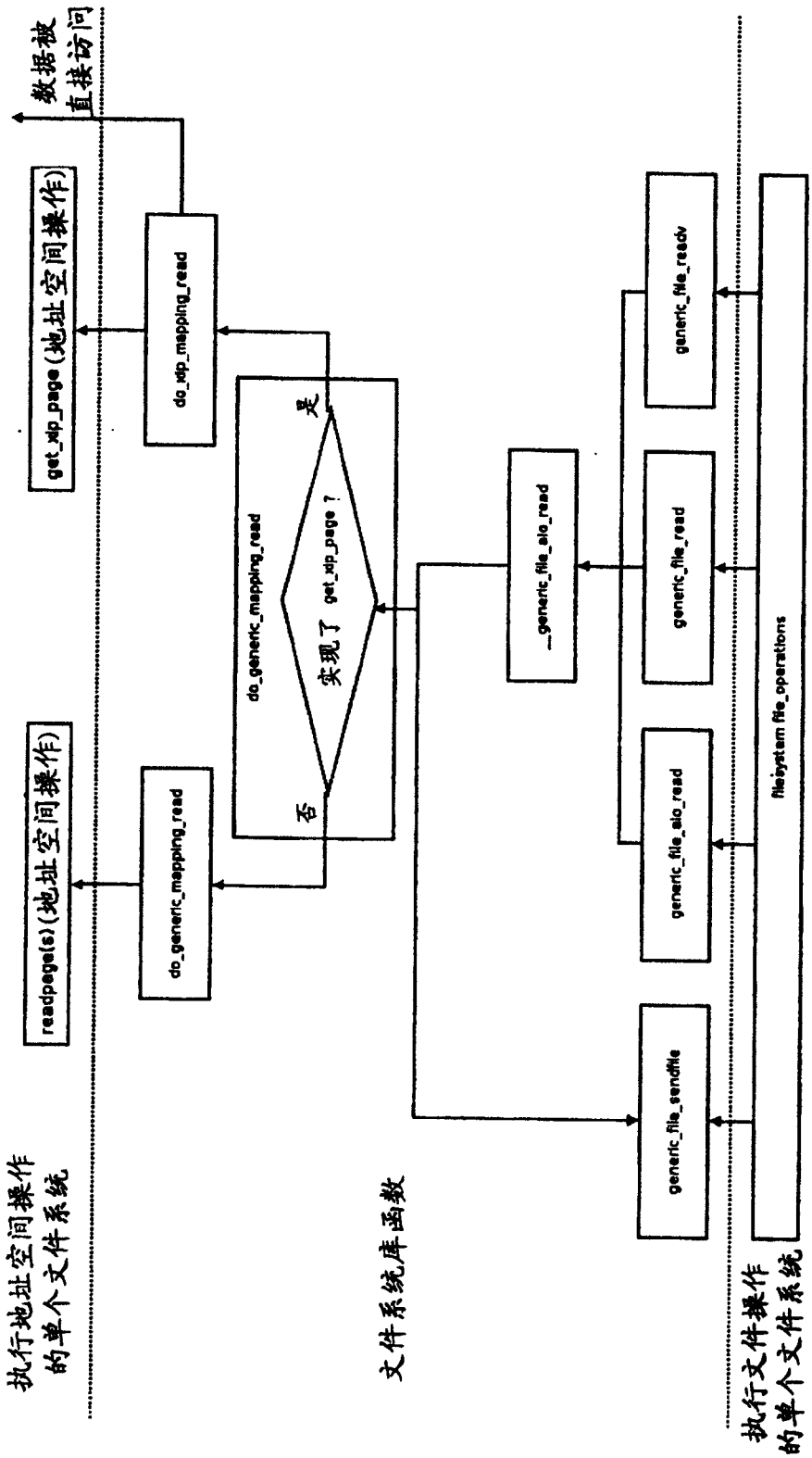

执行地址空间操作的单个文件系统
文件系统库函数
执行文件操作的单个文件系统
数据被直接访问
readpage(s)(地址空间操作)
get_xip_page(地址空间操作)
do_generic_mapping_read
do_xip_mapping_read
do_generic_mapping_read
实现了 get_xip_page ?
是
否
_generic_file_aio_read
generic_file_sendfile
generic_file_aio_read
generic_file_read
generic_file_readv
filesystem file_operations

图 3F

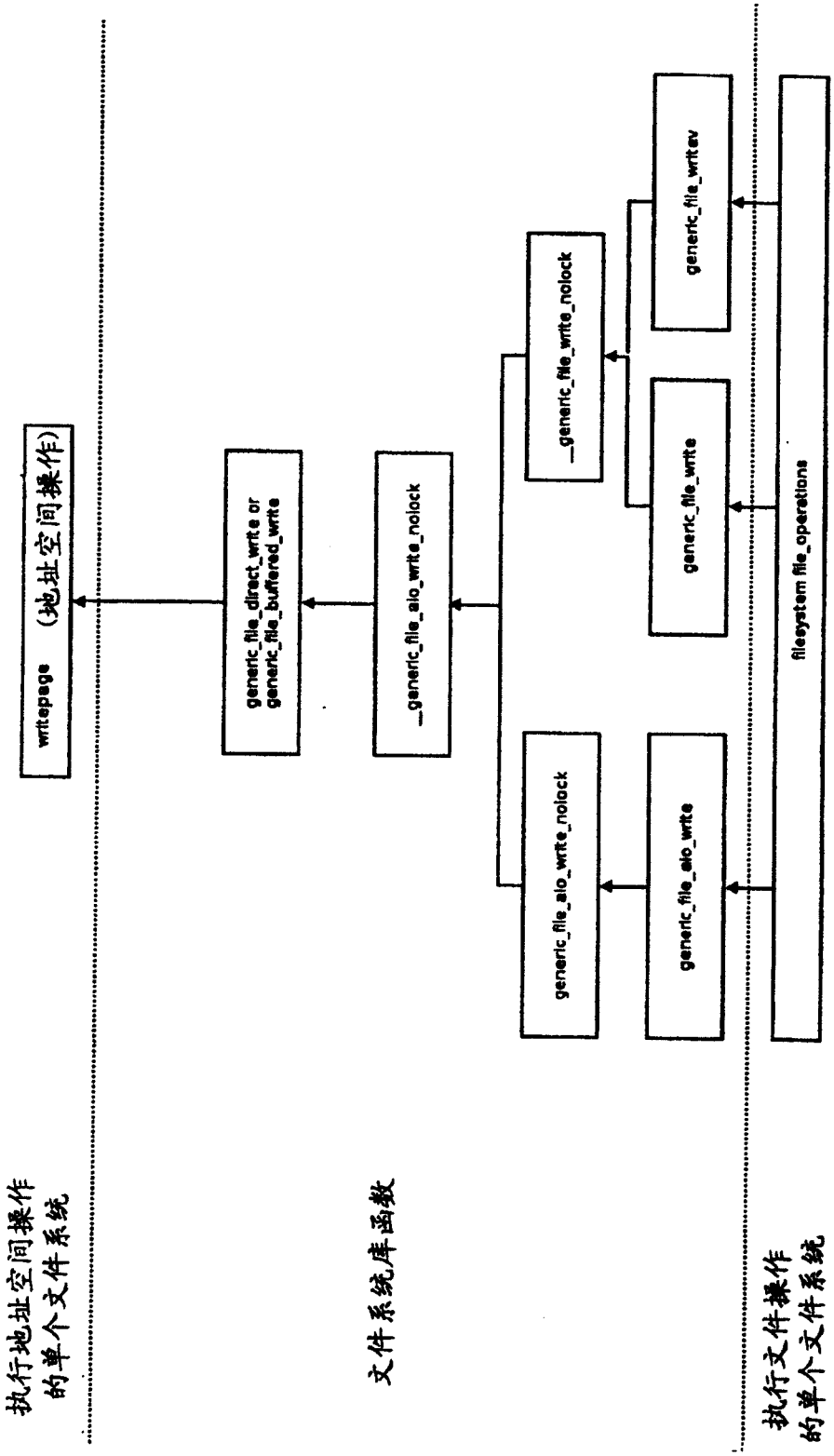

执行地址空间操作
的单个文件系统
文件系统库函数
执行文件操作
的单个文件系统
writepage (地址空间操作)
generic_file_direct_write or
generic_file_buffered_write
_generic_file_aio_write_nolock
generic_file_aio_write_nolock
generic_file_aio_write
__generic_file_write_nolock
generic_file_write
generic_file_writev
filesystem file_operations

图 3G

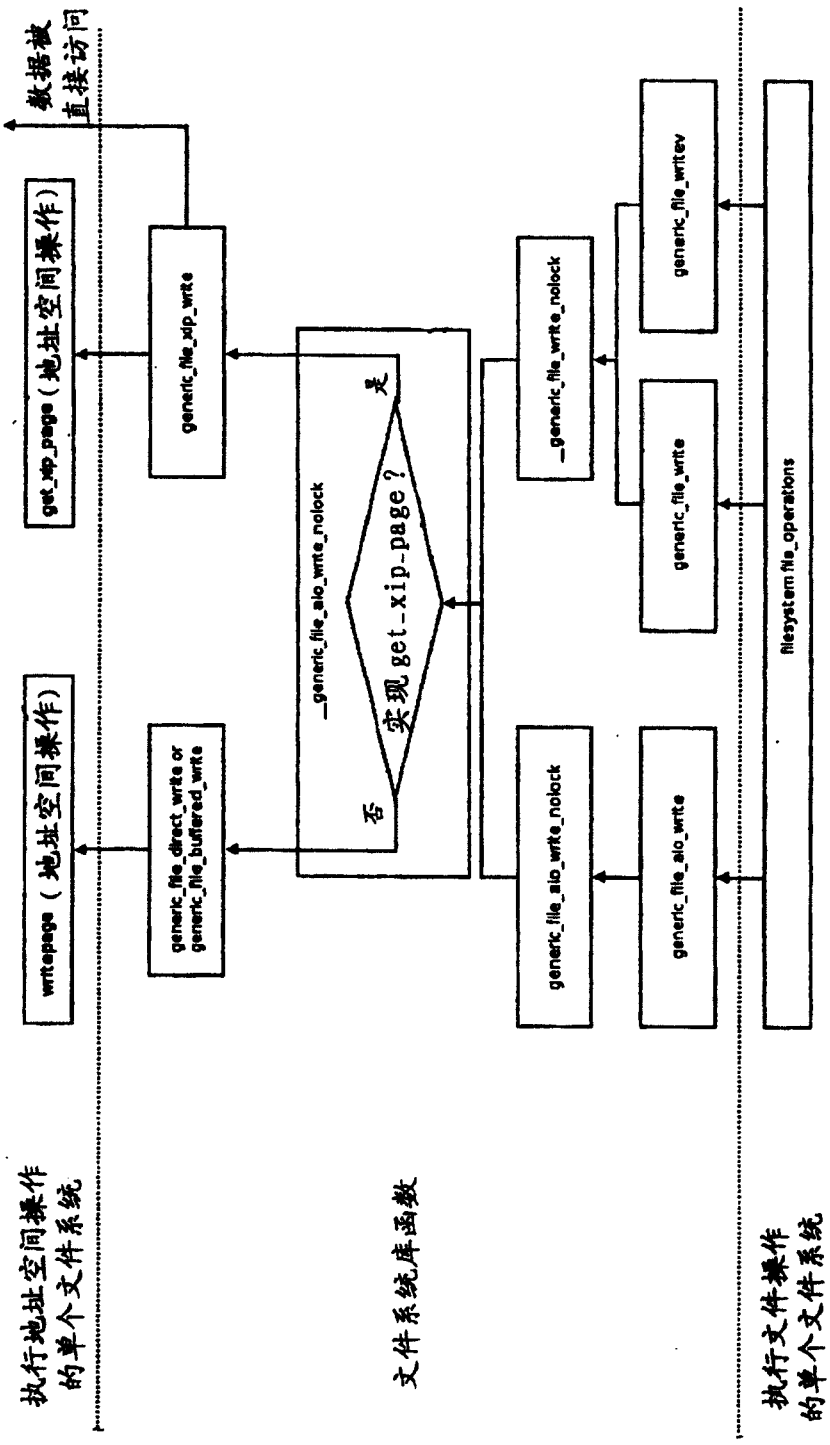

数据被直接访问
get_xip_page（地址空间操作）
generic_file_xip_write
writepage（地址空间操作）
generic_file_direct_write or generic_file_buffered_write
_generic_file_aio_write_nolock
实现get_xip_page？
是
否
generic_file_aio_write_nolock
_generic_file_write_nolock
generic_file_aio_write
generic_file_write
generic_file_writev
filesystem file_operations
执行地址空间操作的单个文件系统
文件系统库函数
执行文件操作的单个文件系统

图 3H

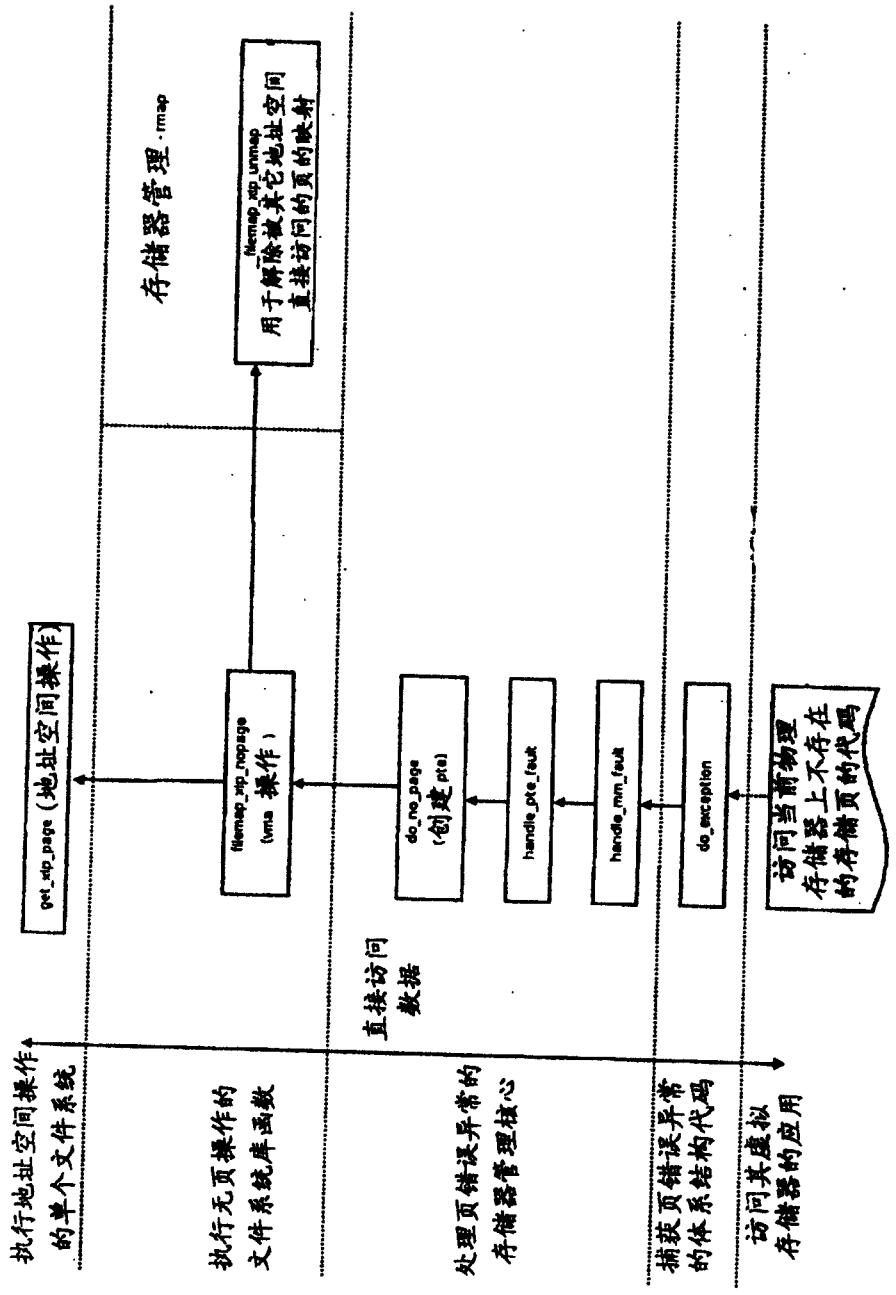

执行地址空间操作的单个文件系统
执行无页操作的文件系统库函数
处理页错误异常的存储器管理核心
捕获页错误异常的体系结构代码
访问共虚拟存储器的应用
get_xip_page（地址空间操作）
存储器管理 · rmap
filemap_xip_nopage（vma 操作）
filemap_xip_unmap 用于解除被其它地址空间直接访问的页的映射
直接访问数据
do_no_page（创建 pte）
handle_pte_fault
handle_mm_fault
do_exception
访问当前物理存储器上不存在的存储页的代码

图 3I