[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 11/14 (2006.01)
G06F 9/445 (2006.01)

[12] 发明专利说明书

专利号 ZL 200610143830.5

[45] 授权公告日 2009 年 1 月 28 日

[11] 授权公告号 CN 100456254C

[22] 申请日 2006.11.9
[21] 申请号 200610143830.5
[30] 优先权
[32] 2005.11.10 [33] US [31] 11/271,248
[73] 专利权人 国际商业机器公司
地址 美国纽约
[72] 发明人 H·V·内里蒂特
[56] 参考文献
CN1299483A 2001.6.13
US20020133738A1 2002.9.19
CN1317742A 2001.10.17
US20020194528A1 2002.12.19
审查员 孟田革

[74] 专利代理机构 北京市中咨律师事务所
代理人 于 静 李 峥

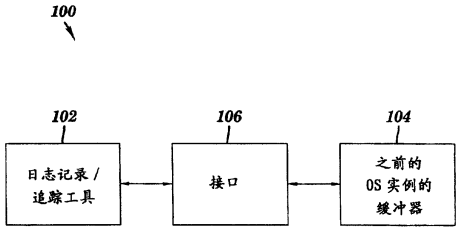
权利要求书 3 页 说明书 31 页 附图 6 页

[54] 发明名称

在系统崩溃时提取日志和追踪缓冲器的方法和系统

[57] 摘要

一种用于在操作系统发生故障之后提取缓冲器数据的方法和系统。 在所述故障之前注册应用。所述注册包括标识在所述故障之前存储了将提取的数据的缓冲器。 所述缓冲器被保留，以将驻留于所述缓冲器中的所述数据维护为从启动直到完成所述操作系统的快速重新引导均未改变。 所述快速重新引导响应于所述故障。 在所述快速重新引导期间生成内存储器文件，其指向驻留于所述缓冲器中的所述数据，并且被存储在易失性存储器中而不是永久性存储媒体中。 经由在完成所述快速重新引导之后由应用执行的、对所述内存储器文件操作的指令来提取所述数据。



1. 一种用于在计算环境中在操作系统发生故障之后提取缓冲器数据的方法，其中所述缓冲器是易失性存储器的区域，其中在所述故障之前存储所述数据，所述方法包括:

在所述故障之前注册应用，所述注册包括标识数据所驻留的缓冲器，由所述应用在所述缓冲器中存储所述数据，所述应用在所述操作系统中执行;

保留所述缓冲器，所述保留将驻留于所述缓冲器中的所述数据维护为从所述操作系统响应于所述故障而启动重新引导直到完成所述重新引导均未改变，所述重新引导是快速重新引导;

在所述重新引导期间生成内存储器文件，所述内存储器文件指向驻留于所述缓冲器中的所述数据，并且所述内存储器文件被存储在所述易失性存储器中而不是永久性存储媒体中；以及

经由对所述内存储器文件操作的指令来提取所述数据，所述指令在所述重新引导的所述完成之后由所述应用执行。

2. 根据权利要求 1 所述的方法，其中所述提取包括:

通过在所述指令中引用所述内存储器文件的标识符而获得所述数据，所述标识符标识所述内存储器文件的内容，而不标识所述易失性存储器的任何其它内容。

3. 根据权利要求 1 所述的方法，还包括:

提供单个接口，所述单个接口提供所述注册，响应于所述注册而自动提供所述保留和所述生成，以及使得经由所述注册、所述保留和所述生成的所述提取更便利，其中在所述易失性存储器的区域中存储将在所述故障之后提取的内容的任何应用能够使用所述单个接口而在所述故障之后提取所述内容。

4. 根据权利要求 1 所述的方法，其中所述标识缓冲器包括提供与所述缓冲器相关联的存储器区域信息，所述信息包括存储所述数据所需的所

述易失性存储器的总量以及所述缓冲器起始的存储器地址。

5. 根据权利要求 4 所述的方法，还包括将所述存储器区域信息存储在元数据中。

6. 根据权利要求 5 所述的方法，还包括:

在所述重新引导期间并且在所述重新引导期间启动对所述易失性存储器的分配之前，防止所述缓冲器被分配由所述操作系统的一个或多个资源的用户所使用，存储器管理模块使所述防止更便利，所述防止接收所述存储器区域信息，所述存储器管理模块驻留于所述操作系统中。

7. 根据权利要求 5 所述的方法，其中所述元数据包括标识所述应用的名称以及所述应用所需的所述易失性存储器的总量。

8. 根据权利要求 5 所述的方法，还包括响应于所述存储器区域信息的改变，实时地动态更新所述元数据。

9. 根据权利要求 1 所述的方法，其中所述生成包括使用用于所述内存储器文件的独立于平台的文件格式。

10. 一种用于在计算环境中在操作系统发生故障之后提取缓冲器数据的系统，其中所述缓冲器是易失性存储器的区域，其中在所述故障之前存储所述数据，所述系统包括:

用于在所述故障之前注册应用的装置，所述用于注册应用的装置包括用于标识数据所驻留的缓冲器的装置，由所述应用在所述缓冲器中存储所述数据，所述应用在所述操作系统中执行;

用于保留所述缓冲器的装置，所述保留将驻留于所述缓冲器中的所述数据维护为从所述操作系统响应于所述故障而启动重新引导直到完成所述重新引导均未改变，所述重新引导是快速重新引导;

用于在所述重新引导期间生成内存储器文件的装置，所述内存储器文件指向驻留于所述缓冲器中的所述数据，并且所述内存储器文件被存储在所述易失性存储器中而不是永久性存储媒体中；以及

用于经由对所述内存储器文件操作的指令来提取所述数据的装置，所述指令在所述重新引导的所述完成之后由所述应用执行。

11. 根据权利要求 10 所述的系统，其中用于提取的所述装置包括:

用于通过在所述指令中引用所述内存储器文件的标识符而获得所述数据的装置，所述标识符标识所述内存储器文件的内容，而不标识所述易失性存储器的任何其它内容。

12. 根据权利要求 10 所述的系统，还包括:

单个接口装置，所述单个接口装置用于提供所述注册，响应于所述注册而自动提供所述保留和所述生成，以及使得经由所述注册、所述保留和所述生成的所述提取更便利，其中在所述易失性存储器的区域中存储将在所述故障之后提取的内容的任何应用能够使用所述单个接口装置而在所述故障之后提取所述内容。

13. 根据权利要求 10 所述的系统，其中用于标识缓冲器的所述装置包括用于提供与所述缓冲器相关联的存储器区域信息的装置，所述信息包括存储所述数据所需的所述易失性存储器的总量以及所述缓冲器起始的存储器地址。

14. 根据权利要求 13 所述的系统，还包括用于将所述存储器区域信息存储在元数据中的装置。

15. 根据权利要求 14 所述的系统，还包括

用于在所述重新引导期间并且在所述重新引导期间启动对所述易失性存储器的分配之前，防止所述缓冲器被分配由所述操作系统的一个或多个资源的用户所使用的装置，存储器管理模块使所述防止更便利，所述防止接收所述存储器区域信息，所述存储器管理模块驻留于所述操作系统中。

16. 根据权利要求 14 所述的系统，其中所述元数据包括标识所述应用的名称以及所述应用所需的所述易失性存储器的总量。

17. 根据权利要求 14 所述的系统，还包括用于响应于所述存储器区域信息的改变，实时地动态更新所述元数据的装置。

18. 根据权利要求 10 所述的系统，其中用于生成的所述装置包括用于使用用于所述内存储器文件的独立于平台的文件格式的装置。

在系统崩溃时提取日志和追踪缓冲器的方法和系统

技术领域

本发明涉及在操作系统发生故障之后提取数据，并更具体地涉及在操作系统发生故障之后从日志和追踪缓冲器中提取数据。

背景技术

日志记录和追踪工具收集与在计算系统上运行的程序相关的关键信息（即，日志和追踪数据）。所收集的信息首先被写入存储器缓冲器，并随后被记录在永久性存储媒体（例如硬盘）上的文件中。不同于日志和追踪工具的实用工具被用来分析所收集和记录的日志和追踪数据。当发生系统崩溃时，日志和追踪数据对于问题确定是重要的，但该数据在发生系统崩溃之前可能未被完整地写出到永久性存储装置。典型地，日志和追踪数据的最重要的部分是距离崩溃时间最近时所收集的数据，但是这些数据也是最可能从永久性存储文件中丢失的部分。

传统的技术尝试在系统崩溃时解决对存储于缓冲器中的日志或追踪数据进行检索，所述技术包括：内核等级调试器、崩溃转储工具以及 Linux Kernel Messages Dump 工具（kmsgdump）。内核等级调试器是有限制的，因为它们需要重新创建系统问题，并且无法自动记录缓冲器内容。崩溃转储工具（例如 Linux Kernel Crash Dump（LKCD）和用于 Linux 的内存储器核转储系统）受到下述限制，所述限制包括时间消耗，以及当仅有相对较小部分的存储器包括了所想要的缓冲器时对整个系统存储器或大部分的存储器进行的效率很低的转储。另外，不需要的开发和维护成本与必须被构建在崩溃转储工具或从转储数据中提取想要的缓冲器的单独的实用工具之中的智能有关。因为此智能必须对于每个日志或追踪工具进行定制，

所以包括了额外的开发成本。**kmsgdump** 工具在崩溃时从系统提取消息，并将其传送到软盘，但是必须对于每个设备开发和维护定制起来昂贵并且依赖于硬件的代码。

因此，需要一种用于在系统崩溃时从缓冲器中提取日志和追踪数据的改进技术。

发明内容

在第一实施例中，本发明提供了一种用于在操作系统发生故障之后提取缓冲器数据的方法，其中在所述故障之前在所述缓冲器中存储所述数据，所述方法包括：

在所述故障之前注册应用，所述注册包括标识数据所驻留的缓冲器，由所述应用在所述缓冲器中存储所述数据，所述应用在所述操作系统中执行；

保留所述缓冲器，所述保留将驻留于所述缓冲器中的所述数据维护为从所述操作系统响应于所述故障而启动重新引导直到完成所述重新引导均未改变，所述重新引导是快速重新引导；

在所述重新引导期间生成内存储器文件，所述内存储器文件指向驻留于所述缓冲器中的所述数据，并且所述内存储器文件被存储在所述易失性存储器中而不是永久性存储媒体中；以及

经由对所述内存储器文件操作的指令来提取所述数据，所述指令在所述重新引导的所述完成之后由所述应用执行。

在第二实施例中，本发明提供了一种用于在操作系统发生故障之后提取缓冲器数据的系统，其中在所述故障之前在所述缓冲器中存储所述数据，所述系统包括：

用于在所述故障之前注册应用的装置，所述用于注册应用的装置包括用于标识数据所驻留的缓冲器的装置，由所述应用在所述缓冲器中存储所述数据，所述应用在所述操作系统中执行；

用于保留所述缓冲器的装置，所述保留将驻留于所述缓冲器中的所述

数据维护为从所述操作系统响应于所述故障而启动重新引导直到完成所述重新引导均未改变，所述重新引导是快速重新引导；

用于在所述重新引导期间生成内存储器文件的装置，所述内存储器文件指向驻留于所述缓冲器中的所述数据，并且所述内存储器文件被存储在所述易失性存储器中而不是永久性存储媒体中；以及

用于经由对所述内存储器文件操作的指令来提取所述数据的装置，所述指令在所述重新引导的所述完成之后由所述应用执行。

在第三实施例中，本发明提供了至少一个机器可读的程序存储设备，其有形地包括可被机器执行的至少一个指令程序，用以执行一种在操作系统发生故障之后提取缓冲器数据的方法，所述方法包括：

在所述故障之前注册应用，所述注册包括标识数据所驻留的缓冲器，由所述应用在所述缓冲器中存储所述数据，所述应用在所述操作系统中执行；

保留所述缓冲器，所述保留将驻留于所述缓冲器中的所述数据维护为从所述操作系统响应于所述故障而启动重新引导直到完成所述重新引导均未改变，所述重新引导是快速重新引导；

在所述重新引导期间生成内存储器文件，所述内存储器文件指向驻留于所述缓冲器中的所述数据，并且所述内存储器文件被存储在所述易失性存储器中而不是永久性存储媒体中；以及

经由对所述内存储器文件操作的指令来提取所述数据，所述指令在所述重新引导的所述完成之后由所述应用执行。

在第四实施例中，本发明提供了一种用于部署计算基础设施的方法，包括将计算机可读代码集成到计算系统中，其中所述代码与所述计算系统相结合能够执行在操作系统发生故障之后提取缓冲器数据的过程，其中在所述故障之前在所述缓冲器中存储所述数据，所述过程包括：

在所述故障之前注册应用，所述注册包括标识数据所驻留的缓冲器，由所述应用在所述缓冲器中存储所述数据，所述应用在所述操作系统中执行；

保留所述缓冲器，所述保留将驻留于所述缓冲器中的所述数据维护为从所述操作系统响应于所述故障而启动重新引导直到完成所述重新引导均未改变，所述重新引导是快速重新引导；

在所述重新引导期间生成内存储器文件，所述内存储器文件指向驻留于所述缓冲器中的所述数据，并且所述内存储器文件被存储在所述易失性存储器中而不是永久性存储媒体中；以及

经由对所述内存储器文件操作的指令来提取所述数据，所述指令在所述重新引导的所述完成之后由所述应用执行。

有利地，本发明通过提供用于在系统崩溃之后获得日志和追踪缓冲器的内容的可靠的方法和系统，增强了操作系统的可服务性，其中所述内容已在崩溃之前被保存。缓冲器内容被提取而不会由于崩溃导致丢失任何日志和追踪信息。在系统重新引导之后，本发明提供了允许写出缓冲器内容的易于使用的接口。本发明提供了新颖的基础设施，其可以被任何应用、工具或实用工具用来在系统崩溃时保留其缓冲器数据。即，这里所公开的新颖的基础设施是可被多种应用、工具和实用工具共享的公共基础设施，从而避免了对多种提取机制（每种机制具有被定制用于特定应用、工具或实用工具的编码）的需要。此外，由于本发明在内核层中实现，所以易于防止黑客对所公开方法的滥用。

附图说明

图 1 是根据本发明实施例的用于在操作系统发生故障后提取缓冲器数据的系统的框图。

图 2 是描述根据本发明实施例的提取图 1 的系统的缓冲器的数据的过程的流程图。

图 3A 描述根据本发明实施例的图 2 的元数据存储图 1 的缓冲器的存储器区域。

图 3B 描述根据本发明实施例的图 2 的存储保留模块保留图 3A 的存储器区域。

图3C描述根据本发明实施例的图2的缓冲器提取模块从图3A的存储器区域提取缓冲器。

图4描述根据本发明实施例的用于实现提取图1的系统的缓冲器的数据的计算机系统。

具体实施方式

本发明的方法和系统公开了一种单一的、公用的、高效的机制，该机制可以由操作系统提供给操作系统资源的用户（例如应用和设备驱动程序），并且可以在系统崩溃时提取日志或追踪缓冲器数据。当计算系统正运行在系统崩溃之前就运行的操作系统的第一实例之时存储所述日志或追踪缓冲器数据，并且当操作系统的第二实例正在运行时进行数据提取。由响应于系统崩溃而执行的快速重新引导来生成所述第二实例。

定义

以下定义适用于这里所公开的本发明：

操作系统：通过充当软件程序和系统硬件之间的接口而管理计算系统并使得软件程序在计算系统上的运行更便利的软件。操作系统的功能包括：管理硬件（例如随机存取存储器（RAM）、硬盘、打印机、软盘驱动器、CD-ROM驱动器、声卡和视频卡以及其它设备）、向用户提供图形界面、确保系统的安全性、以及提供网络基础设施。

内核：任何操作系统的核心部件。内核充当硬件和其它系统部件（软件程序、设备驱动程序等）之间的接口。内核在“特权”级别运行，并可以执行对硬件的大多数的允许操作。

设备驱动程序：管理计算系统中的特定硬件设备的软件。设备驱动程序在与内核相同的特权级别运行。设备驱动程序的示例包括键盘驱动程序和网卡驱动程序。

系统崩溃：操作系统（并且由此导致其它软件）发生故障或者检测到其无法继续的状态。系统崩溃的原因包括：访问无效存储器区域、数据讹误或死锁。

日志记录工具：应用或设备驱动程序在系统工作期间捕获服务成员的必要信息的机制。信息或“日志”（也称作日志数据）首先被写入作为（例如随机存取存储器中的）存储器区域的一个或多个缓冲器，并随后（例如以一个或多个文件的形式）被传送到存储媒体或设备，诸如硬盘或软盘。编写与日志记录工具不同的实用工具或工具来分析日志数据。日志记录工具的示例是 Linux 操作系统内核中的 syslog 机制。

追踪工具：追踪软件的特定部分中的代码流的机制。与通过日志记录工具收集日志数据类似，追踪工具首先将其收集的数据（也称作追踪数据）写入缓冲器，并随后将所述数据传送到存储媒体/设备上的文件。追踪工具的示例是 Linux 操作系统内核的 dprobes 追踪工具。

引导：将计算机系统引至可用状态的过程。引导包括下述步骤，诸如：验证附接于系统的设备、运行操作系统、初始化各种设备等等。

重新引导：退出或关闭当前的操作系统上下文并重新进行“引导”定义中描述的引导过程的过程。在重新引导期间，能够切换到不同的内核、不同的设备驱动程序或甚至完全不同的操作系统。

快速重新引导的特征

当与操作系统一起工作时，系统重新引导的时间是主要瓶颈。例如，在生产系统中，系统可用性是重要需求，并且软件开发者在其工作期间需要多次重新引导系统，二者都需要一种快速重新引导系统并使系统返回工作状况的方法。通过“快速重新引导”特征来提供这种方法。

如果未执行快速重新引导，则系统重新引导的典型阶段如下：

（1）当前系统被关闭。文件系统被卸载，文件被写回存储媒体，并且应用被终止；

（2）将控制移交给系统固件或者基本输入/输出系统（BIOS），以执行基本操作，诸如：检查系统上存在的设备、复位设备、清除系统存储器等等；

（3）固件或 BIOS 将控制移交给引导加载程序。该引导加载程序加载操作系统内核，并将控制移交给该内核；以及

（4）操作系统内核启用系统中的处理器、初始化设备、安装文件系统、加载应用并将系统引至可用状况（例如，操作系统准备好与用户联系）。

如上所述，当操作系统在没有快速重新引导的情况下进行重新引导时，固件或者基本输入/输出系统（BIOS）擦除系统存储器内容（即，在重新引导被完全擦除之前存在的内核中的存储器内容）。与之相反，当系统执行快速重新引导时，上述的阶段（2）和（3）被跳过。因此，快速重新引导避免了系统存储器清除过程，并且之前的内核中的存储器内容（即，在快速重新引导之前存在的内核）在快速重新引导之后仍保持可用。用于 Linux 内核的快速重新引导特征的一个示例是 kexec，在 http://www-106.ibm.com/developerworks/linux/library/l-kexec.html 以及可在 http://www.osdl.org/docs/reducing_system_reboot_time_with_kexec.pdf 找到的 Pfiffer、Andy 的“Reducing System Reboot Time With kexec”（Open Source Development Labs, Inc.）中均对其进行了描述。

本发明使用了快速重新引导特征及其从一个内核保留到下一个内核的存储器内容的副效益，以在系统崩溃之后提取（即获取）缓冲器数据。在下文中，除非另外说明，否则术语“重新引导”指代使用快速重新引导特征的重新引导。

系统概述

图 1 是根据本发明实施例的用于在操作系统发生故障后提取缓冲器数据的系统的框图。在系统 100 中，日志记录工具或追踪工具经由接口 106 从缓冲器 104 提取数据。所述日志记录工具或追踪工具驻留于例如操作系统内核（未示出）之中。将被提取的数据在系统崩溃之前被收集和保存，并在重新引导期间可用。如这里所使用的，下文的“重新引导期间”（即，“系统崩溃期间”）被定义为在从就在系统崩溃之前存在的操作系统的第一实例（即，第一上下文）到响应于所述崩溃而执行的重新引导所生成的操作系统的第二实例（即，第二上下文）的转换之后。

接口 106 由本发明提供，并包括多种模块，所述模块包括构建在操作系统内核之中的基础设施。所述基础设施允许日志记录和追踪工具快速地

和容易地访问它们各自的缓冲器，所述缓冲器的内容在系统崩溃之前被保存并被保留，从而这些内容可以在重新引导之后访问。以下相对于图 2 描述了包括所述基础设施的模块。

缓冲器提取过程

图 2 是描述根据本发明实施例的提取图 1 的系统的缓冲器的数据的过程的流程图。提取缓冲器的数据或提取缓冲器的内容在此也被简称为提取缓冲器或缓冲器提取。在步骤 200 开始缓冲器提取过程。注册模块 202 将自身注册到缓冲器追踪模块 204。在可替换实施例中，多个注册模块注册到缓冲器追踪模块 204。注册模块是在操作系统中执行的软件应用，并且包括例如被设计用于问题确定任务的应用、工具和实用工具，诸如日志记录工具和追踪工具。下文中，术语“应用”、“实用工具”和“工具”被互换地使用，并被特别地指代在系统崩溃之前在易失性存储器的区域中存储数据的软件，其中所述数据将在崩溃之后被提取出来。由注册模块 202 进行的注册包括向缓冲器追踪模块 204 请求标识出存储数据的易失性存储器中的存储器区域（即缓冲器），所述数据需要在重新引导期间可用（例如日志记录工具所需的缓冲器数据）。在注册模块 202 所标识的存储器区域中存储所述数据。即，所标识的存储器区域在系统崩溃之前存储数据，并且本发明使得该数据可以在响应于崩溃而执行的对操作系统的重新引导之后进行访问。前述的由注册模块 202 执行的标识在此也被简称为标识出将在重新引导期间保留的存储器区域。

例如，一应用在其操作期间创建和使用数据结构 1 到 10。由于该应用为这些数据结构分配存储器，因此该应用了解所分配的存储器总量以及与每一数据结构相关联的存储器地址。当此应用请求向缓冲器追踪模块 204 注册时，该应用将数据结构 1 标识为其存储器区域需要在重新引导期间保留的数据结构。注册请求通过包括分配给数据结构 1 的存储器总量以及数据结构 1 的起始存储器地址而标识该存储器区域。

通过注册模块 202 进行的注册被缓冲器追踪模块 204 所接受。对于每次注册，缓冲器追踪模块 204 记录请求过注册的注册模块以及注册中所标

识的存储器区域，并将所标识的存储器区域信息存储在元数据 206（即，一个或多个数据结构）中。如这里所使用的，存储器区域信息被定义为存储器区域所需要的存储器总量以及存储器区域的基本存储器地址，诸如该区域的起始地址。元数据 206 由缓冲器追踪模块 204 所维护，并使其在重新引导期间可用，如下所述。元数据 206 包括注册名称、注册模块 202 所需的存储器区域的大小、注册模块 202 所需的存储器区域的列表。注册名称标识了进行注册请求的注册模块 202，并在重新引导之后被用来将请求中包括的所存储的存储器区域信息关联于进行请求的注册模块。存储器区域的列表例如是把指示出将要保留的存储器区域的起始地址以及存储器区域的大小的列表的每个节点链接起来的列表。

元数据 206 的存储位置是存储器中的固定位置或者下述存储器地址，其在第一内核中存在，并被传递到响应于系统崩溃而执行的快速重新引导期间被引导的第二内核，如下所述。本领域技术人员将了解，已存在用于将地址传递给第二内核的技术。

缓冲器追踪模块 204 还允许对元数据 206 中的存储器区域信息的动态（即实时）更新，以反映注册模块 202 对存储器使用的改变。对存储器使用的这些改变是对存储器区域信息的改变，并且例如包括：用于将在重新引导期间保留的数据的存储器使用的增加、或者清除用于该数据的存储器。

在生成元数据 206 中的数据结构之后的某时，发生系统崩溃。在崩溃之后，使用快速重新引导特征来启动步骤 208 中的重新引导。图 2 中在虚线之间描述的部分包括在快速重新引导期间发生动作的模块和发生的步骤。再一次地，快速重新引导跳过了典型的由固件或 BIOS 对系统存储器的清除。在重新引导期间，存储器保留模块 210 工作，并访问元数据 206 中的存储器区域。在将元数据存储器地址传递到第二内核的前述情形中，存储器保留模块 210 接收所传递的元数据地址。在重新引导期间，但在内核开始将存储器分配给其它部件（例如，设备驱动程序和应用）使用之前，存储器保留模块 210 指示内核的存储器管理模块（未示出）来保留在元数据 206 中指示的存储器区域。该保留确保了将在重新引导期间保留的存储

器区域不会被分配给操作系统资源的任何其它用户所使用。如这里所使用的，操作系统资源的用户包括应用和设备驱动程序。该保留还确保了驻留于所指示的存储器区域中的数据从重新引导开始直到重新引导完成均被维护为未改变状态。

在由元数据 206 指示的存储器区域被存储器保留模块 210 保留之后，在重新引导期间缓冲器提取模块 212 进行工作。对于缓冲器追踪模块 204 接收的每次注册，由缓冲器提取模块 212 创建伪文件。如这里所使用的，伪文件（也称为内存储器文件）被定义为仅在易失性存储器（例如 RAM）中作为已命名单元而存储的数据位的序列，其不在非易失性存储媒体（即，永久性存储媒体）中进行任何备份存储。伪文件的名称标识或指向在易失性存储器中存储的数据。伪文件类似于典型的计算机文件，除了计算机文件在非易失性存储媒体（例如，硬盘或软盘）上存储，而伪文件仅在易失性存储器上存储。

由缓冲器提取模块 212 创建的伪文件指向数据，所述数据也是元数据 206 指示的存储器区域的内容。即，伪文件的数据是注册模块 202 请求在重新引导期间保留的存储器区域（即缓冲器）的实际内容。由于日志记录工具无法直接访问元数据 206，所以伪文件充当了允许日志记录工具访问其内容在重新引导期间被保留的缓冲器的接口。

例如，在 Linux 内核中，可以使用“proc”文件系统来创建伪文件。在此情形中，伪文件或内存储器文件不在任何后备媒体（硬盘、软盘等）上存储。由 proc 创建的伪文件可以由其在/proc 文件系统中的名称所访问。例如，proc 创建的名为 xyz 的伪文件通过/proc/xyz 来引用。

使用标准的独立于平台的文件格式（诸如 Executable and Linking Format（ELF））来创建本发明中的伪文件。使用标准的独立于平台的文件格式是有利的，因为它允许标准命令被用于对伪文件进行操作，从而避免对用于访问所保留的缓冲器的数据的定制编码或依赖于硬件的技术的需要。在步骤 214，在重新引导完成之后，使用对伪文件进行操作的指令而将由缓冲器提取模块 212 创建的伪文件写出（即，在重新引导期间保留的

日志或追踪缓冲器的内容被提取）。在重新引导完成后由注册模块 202 执行写出伪文件的指令。在一实施例中，指令通过引用伪文件名称而写出缓冲器数据，所述伪文件名称精确地指向缓冲器数据而不会标识易失性存储器的任何其它内容。伪文件数据例如由系统管理员所写出。在前述的 /proc/xyz 伪文件的情形中，任何标准的文件拷贝命令（诸如 Linux 的 cp 命令）都可以被用于写出文件。因此，通过用标准命令借助名称来访问伪文件，可以提取日志和追踪缓冲器中的数据。一旦数据被提取出来，就在步骤 216 将了解缓冲器内容的软件工具用于分析所提取的日志或追踪数据。

在步骤 214 已将缓冲器内容提取出来之后，所保留的存储器区域可选地可被存储器清除模块 218 所清除。清除所保留的存储器区域的一个示例包括删除对应于缓冲器的伪文件。一旦存储器被清除，它就可以被再次用于正常的内核存储器分配。在写出和分析伪文件数据之后以及在清除与所保留的存储器区域相关联的存储器的可选步骤之后，在步骤 220 缓冲器提取过程结束。

如上述的缓冲器提取过程所指示的，注册模块 202、缓冲器追踪模块 204、存储器保留模块 210 和缓冲器提取模块 212 共同提供可以将数据存储在易失性存储器的缓冲器中的任何应用所使用的单一接口，其中数据将在系统发生故障之后从该缓冲器中提取。该单一接口可被任何应用一起共享。此外，该接口向任何应用提供以下装置，用于：（1）注册应用，（2）保留存储将被提取数据的存储器区域，从而在系统重新引导期间不会向操作系统资源的用户分配这些存储器区域，以及（3）在重新引导期间生成指向将被提取数据的伪文件。响应于应用的注册，由接口自动提供所述保留和生成的特征。因此，该接口提供了一种易于使用的机制，用于通过使用对伪文件操作的指令，在系统出现故障时提取数据，如上所述。

附录呈现了用于实现图 2 的缓冲器提取过程的代码。以下呈现了缓冲器提取过程的示例。尽管这里呈现的示例解决了对应用所使用的缓冲器数据的保留，但是本发明还可由设备驱动程序利用，以在快速重新引导期间

保留设备阶段信息，以便使从前一阶段恢复操作更方便。

缓冲器提取的示例

图 3A-3C 描述了图 2 的缓冲器提取过程的实现的示例。图 3A 描述在图 2 的缓冲器提取过程中关注于缓冲器追踪模块 204 在系统崩溃和快速重新引导之前的操作的第一部分 300。名为 logxyz 工具 302 的日志记录工具将自身注册到缓冲器追踪模块 204。由 logxyz 使用的日志记录缓冲器被存储在系统存储器 308 的三个不同存储器区域 310、312、314。关于存储器区域 310、312、314 的大小和起始地址的信息由缓冲器追踪模块 204 从 logxyz 工具 302 所收集，并被存储在用于 logxyz 的元数据 306 中。

图 3B 描述在图 2 的缓冲器提取过程中关注于存储器保留模块 210 的操作的第二部分 320。由于系统挂起导致系统崩溃，随后启动快速重新引导。在重新引导阶段期间，存储器保留模块 210 访问用于 logxyz 的元数据 306。通过使用从对用于 logxyz 的元数据 306 的访问中收集的存储器区域信息，存储器保留模块 210 保留驻留于系统存储器 308 中的存储器区域 310、312、314。所保留的存储器区域是由元数据指示的区域。

图 3C 描述在图 2 的缓冲器提取过程中关注于缓冲器提取模块 212 的操作的第三部分 340。在重新引导阶段期间，缓冲器提取模块 212 使用所存储的用于 logxyz 的元数据 306（参见图 3B）来创建名为 logxyz-buffer 的伪文件，其数据是驻留于系统存储器 308 中的存储器区域 310、312、314 的内容。伪文件在/proc 文件系统中创建，并作为/proc/logxyz-buffer 344 而引用。通过执行简单文件复制操作而提取 logxyz 工具 302（参见图 3A）的日志缓冲器的数据。在 Linux 系统中，命令

cp /proc/logxyz-buffer /home/test/log.buf

将缓冲器内容拷贝到/home/test 目录中的名为 log.buf 的文件中。

被设计与 logxyz 工具一起使用的日志分析器实用工具分析文件 log.buf 的内容。

尽管以上呈现的实施例仅使用一种日志工具，但是本发明预期了将系统 100（参见图 1）扩大为同时执行上述操作的若干工具或实用工具。

用于缓冲器数据提取的计算机系统

图 4 描述根据本发明实施例的用于实现在图 1 的系统中在操作系统发生故障之后提取缓冲器数据的计算机系统。计算机系统 400 适当地包括处理器 402、主存储器 404、存储器控制器 408 和至少一个输入/输出（I/O）接口 410，所有这些部件经由系统总线 412 互连。主存储器 404 包括操作系统 406 和计算机程序 414。主存储器 404 是易失性存储器（例如 RAM），并包括一种包括了图 2 的流程图中描述的缓冲器提取逻辑的算法。在一实施例中，计算机程序 414 包括图 2 的逻辑的算法。作为图 4 中的计算机程序 414 的可替换位置，计算机程序 414 还可被包括在操作系统 406 中。作为一个示例，处理器 402 是基于 x86 体系结构的 Intel 处理器，而操作系统 406 是 Linux 操作系统。

处理器 402 执行计算机系统 400 的计算和控制功能，并且包括适当的中央处理单元。处理器 402 可包括单个集成电路（诸如微处理器），或可包括任何适当数量的集成电路设备和/或电路板，其协同工作来完成处理器的功能。处理器 402 适当地执行一个或多个计算机程序，包括计算机程序 414。在一实施例中，处理器 402 执行实现图 2 的流程图中描述的逻辑的算法。

I/O 接口 410 可包括用于从外部源（诸如外部设备 416）交换信息的任何系统。外部设备 416 可包括传统的外部设备，包括显示监视器、键盘、鼠标、打印机、绘图机、传真机等。计算机系统 400 可使用适合的通信信道（未示出）经由通信接口连接于一个或多个其它计算机，所述通信信道诸如调制解调器通信路径、计算机网络等。计算机网络（未示出）可包括局域网（LAN）、广域网（WAN）、内联网、和/或因特网。

I/O 接口 410 还允许计算机系统 400 存储并从辅助的存储设备 418 检索信息（例如程序指令或数据），所述存储设备 418 诸如非易失性存储设备（例如，接收 CD-ROM 盘（未示出）的 CD-ROM 驱动器）。计算机系统 400 可以存储并从其它的辅助存储设备（未示出）检索信息，所述存储设备 418 可包括直接存取存储设备（DASD）（例如硬盘或软盘）、磁光

盘驱动器、磁带驱动器或无线通信设备。

存储器控制器 408 通过使用与处理器 402 不同的处理器（未示出），而负责将所请求的信息从主存储器 404 和/或通过 I/O 接口 410 移动到处理器 402。尽管出于解释目的将存储器控制器 408 示出为单独实体，但是本领域技术人员应理解，在实践中，由存储器控制器 408 提供的功能的某些部分实际可驻留于与处理器 402、主存储器 404 和/或 I/O 接口 410 相关联的电路中。

应该理解，主存储器 404 将不必非要包含所示出的所有机制的所有部分。例如，计算机程序 414 和操作系统 406 的某些部分可被加载到指令高速缓存（未示出）中，用于处理器 402 来执行，而其它文件可被存储在磁或光盘存储设备（诸如存储设备 418）上。此外，尽管所示出的计算机程序 414 驻留于与操作系统 406 相同的存储位置中，但是应该理解，主存储器 404 可包括完全不同的存储位置。

I/O 接口 410 的终端接口允许系统管理员和计算机程序员与计算机系统 400 通信。尽管图 4 中描述的计算机系统 400 仅包含单个主处理器 402 和单个系统总线 412，但是应该理解，本发明可同等地应用于具有多个处理器和多个系统总线的计算机系统。类似地，尽管系统总线 412 是典型的硬连线的、多点总线，但是在与计算机有关的环境中支持双向通信的任何连接装置都是可用的。

根据本发明的计算机系统 400 例如是个人计算机。不过，本领域技术人员将理解，本发明的方法和装置可同等地应用于任何计算机系统，不管该计算机系统是复杂的多用户计算装置还是单个用户的设备，诸如工作站。

注意，可在本发明范围内对图 4 所述的计算机系统 400 做出多种修改、添加或删除，诸如添加高速缓冲存储器或其它外围设备。所呈现的图 4 仅例示了计算机系统 400 的某些突出特征。

重要地是要注意，尽管本发明已经（并将继续）在完全运行的计算机系统的上下文中进行描述，不过，本领域技术人员将理解，本发明的机制能够作为程序产品以多种形式分布，并且本发明可同等地被应用，而不管

实际用来执行所述分布的信号承载媒体的具体类型。信号承载媒体的示例包括可记录类型的媒体（诸如软盘和 CD-ROM）以及传输类型的媒体（诸如，包括无线通信链路的数字和模拟通信链路）。

因此，本发明公开了一种用于部署或集成计算基础设施的方法，包括将计算机可读代码集成到计算机系统 400 中，其中所述代码与计算机系统 400 相结合能够执行用于在操作系统发生故障之后提取缓冲器数据的过程。

本发明可以被包括在例如制品（例如一个或多个计算机程序产品）中，所述制品例如具有计算机可用媒体。该媒体中包括例如用于提供本发明的能力并使其更便利的计算机可读程序代码工具。所述制品可被包括为计算机系统的一部分，或者单独出售。

此外，可以提供用来执行本发明的能力的机器可读的至少一个程序存储设备，其有形地包括机器可执行的至少一个代码程序。

通过示例提供了这里描述的流程图。在不脱离本发明精神的情况下，存在对于这里所述的这些图或步骤（或操作）的变体。例如，在某些情形中，可以用不同顺序执行步骤，或者可以添加、删除或修改步骤。所有这些变体都被认为是权利要求所阐述的本发明的一部分。

尽管这里已经出于例示性目的描述了本发明的实施例，但是对于本领域技术人员来说许多修改和改变将变得明显。因而，权利要求旨在包含落在本发明真正的精神和范围内的所有这些修改和改变。

附录

Memsave.patch 代码

以下代码（**memsave.patch**）实现了本发明的易于使用的新颖的基础设施，其充当了将由任意日志或追踪工具使用的公用接口。此代码允许 Linux 内核的任意部件将自身注册到模块，以请求在重新引导期间保留指定的存储器区域。一旦发生重新引导，该区域的内容就可被写出为普通文件。此代码提供的示范说明具有为保留存储器的两个不同区域而进行的两次注册。一旦进行快速重新引导，这两个区域就被提取为文件。

---

linux-patch-hari/arch/i386/Kconfig | 6 ++

linux-patch-hari/arch/i386/kernel/Makefile | 2

linux-patch-hari/arch/i386/kernel/mdummy.c | 46 ++++++++++++++++++

linux-patch-hari/arch/i386/kernel/memsave.c | 45 ++++++++++++++++++

linux-patch-hari/arch/i386/kernel/setup.c | 6 ++

linux-patch-hari/fs/proc/Makefile | 1

linux-patch-hari/fs/proc/proc_memsave.c | 73

++++++++++++++++++++++++++++

linux-patch-hari/fs/proc/proc_misc.c | 23 ++++++++

linux-patch-hari/fs/proc/root.c | 4 +

linux-patch-hari/include/asm-i386/memsave.h | 58 ++++++++++++++++++++++++

linux-patch-hari/include/linux/proc_fs.h | 5 +

11 files changed, 269 insertions (+)

diff –puN arch/i386/Kconfig~memsave arch/i386/Kconfig

--- linux-patch/arch/i386/Kconfig~memsave 2004-08-29 11:19:10.000000000 +0530

+++ linux-patch-hari/arch/i386/Kconfig 2004-08-29 11:20:55.000000000 +0530

@@ -864,6 +864,12 @@ config REGPARM

generate incorrect output with certain kernel constructs when

-mregparm=3 is used.

+config MEMSAVE

+ bool "preserve memory across reboots"

+ depends on KEXEC

+ help

+ Allow memory to be preserved across reboots

+

```
 endmenu

diff –puN arch/i386/kernel/setup.c~memsave arch/i386/kernel/setup.c
--- linux-patch/arch/i386/kernel/setup.c~memsave          2004-08-29
11:22:24.000000000 +0530
+++ linux-patch-hari/arch/i386/kernel/setup.c       2004-08-30 22:32:02.000000000
+0530
@@ -48,6 +48,7 @@
 #include <asm/io_apic.h>
 #include <asm/ist.h>
 #include <asm/io.h>
+#include <asm/memsave.h>
 #include "setup_arch_pre.h"
 #include <bios_ebda.h>

@@ -1097,6 +1098,9 @@ static unsigned long_init setup_memory
         }
     }
 #endif
+
+    reserve_bootmem(MEMSAVE_BASE, MEMSAVE_SIZE);
+
     return max_low_pfn;
   }
#else
@@ -1358,6 +1362,8 @@ void_init setup_arch(char **cmdline_p)
#endif
     paging_init( );
```

```
+    reserve_memsave_bootmem( );
+
 #ifdef CONFIG_EARLY_PRINTK
     {
          char *s = strstr (*cmdline_p, "earlyprintk=" );
diff –puN /dev/null include/asm-i386/memsave.h
--- /dev/null     2003-01-30 15:54:37.000000000 +0530
+++ linux-patch-hari/include/asm-i386/memsave.h 2004-08-30 22:26:51.000000000
+0530
@@ -0,0 +1,58 @@
+/*
+ * Arch specific functions for the memory preserving reboot infrastructure
+ * API
+ *
+ * Created by: Hariprasad Nellitheertha
+ *
+ * Copyright (C) IBM Corporation, 2004. All rights reserved.
+ */
+
+#include <linux/bootmem.h>
+
+#define MEMSAVE_BASE 0x2000000
+#define MEMSAVE_SIZE  0x0001000 /* 1 page should be sufficient */
+#define MEMSAVE_ENTRY_SIG  0xaeaecece
+#define MEMSAVE_END_SIG 0xdeadbeef
+
+/* The memsave structure */
```

```
+
+struct memsave {
+       unsigned long sig; /* Indicates end of list */
+       char name[10];     /* Name of the interface */
+       unsigned long start; /* Start of memory region, physical address */
+       unsigned long size; /* Size of the region */
+};
+
+extern int memsave_add_entry(struct memsave *);
+
+#ifdef CONFIG_MEMSAVE
+static inline void reserve_memsave_regions(void)
+{
+
+       struct memsave *bufp;
+       int n, i = MEMSAVE_SIZE / sizeof (struct memsave);
+
+       bufp = (struct memsave *)_va (MEMSAVE_BASE);
+
+       for (n = 0; n < i; n++) {
+              if (bufp->sig != MEMSAVE_ENTRY_SIG)
+                     return;
+              /* Reserve the memory claimed by this entry */
+              reserve_bootmem (bufp->start, bufp->size);
+              (char *)bufp += sizeof (struct memsave);
+       }
+}
+
```

```
+static inline void reserve_memsave_bootmem (void)
+{
+    unsigned long *sig;
+
+    sig = (unsigned long *)_va (MEMSAVE_BASE);
+    if (*sig != MEMSAVE_ENTRY_SIG)    /* List is empty */
+            *sig = MEMSAVE_END_SIG; /* Initialize the list */
+    else                    /* There are entries. Reserve the regions */
+            reserve_memsave_regions( );
+}
+#else
+static inline void reserve_memsave_bootmem (void);
+#endif
diff –puN /dev/null arch/i386/kernel/memsave.c
--- /dev/null        2003-01-30 15:54:37.000000000 +0530

+++ linux-patch-hari/arch/i386/kernel/memsave.c 2004-08-31 00:20:00.000000000
+0530
@@ -0,0 +1,45 @@
+/*
+ * The APIs for the memory saving reboot infrastructure.
+ *
+ * Created by Hariprasad Nellitheertha
+ *
+ * Copyright (C) IBM Corporation, 2004. All rights reserved.
+ */
+
+#include <asm/errno.h>
```

```
+#include <asm/memsave.h>
+
+/*
+ * This routine adds a new memsave entry to the list. If no entry
+ * exists, a new one is initialized.
+ */
+int memsave_add_entry(struct memsave *msave)
+{
+
+	struct memsave *bufp;
+	int n, i = MEMSAVE_SIZE / sizeof (struct memsave);
+
+	bufp = (struct memsave *)_va (MEMSAVE_BASE);
+
+	/* Loop through the structure till we find an empty slot */
+	while ((i--) && (bufp->sig != MEMSAVE_END_SIG))
+		(char *)bufp += sizeof (struct memsave);
+
+	if (!i) /* No more entries accepted */
+		return -EFAULT;
+
+	/* We found an available slot. Register the entry.
+	 * We do not validate the entry. Just copy it
+	 */
+	memcpy(bufp, msave, sizeof (struct memsave));
+	bufp->sig = MEMSAVE_ENTRY_SIG;
+
+	/* Mark the next entry as available */
```

```
+       if (i > 1) {
+               (char *)bufp += sizeof (struct memsave);
+               bufp->sig = MEMSAVE_END_SIG;
+       }
+
+       return 0;
+}
diff –puN arch/ i386/kernel/Makefile~memsave arch/i386/kernel/Makefile
--- linux-patch/arch/i386/kernel/Makefile~memsave        2004-08-29
13:50:53.000000000 +0530
+++ linux-patch-hari/arch/i386/kernel/Makefile     2004-08-30 20:27:27.000000000
+0530
@@ -32,6 +32,8 @@ obj-$ (CONFIG_ACPI_SRAT) += srat.o
 obj-$ (CONFIG_HPET_TIMER)              += time_hpet.o
 obj-$ (CONFIG_EFI)                          += efi.o efi_stub.o
 obj-$ (CONFIG_EARLY_PRINTK)          += early_printk.o
+obj-$ (CONFIG-MEMSAVE)                 += memsave.o
+obj-$(CONFIG-MEMSAVE)                  += mdummy.o

  EXTRA_AFLAGS := -traditional

diff –puN fs/proc/proc_misc.c~memsave fs/proc/proc_misc.c
--- linux-patch/fs/proc/proc_misc.c~memsave        2004-08-29 20:44:57.000000000
+0530
+++ linux-patch-hari/fs/proc/proc_misc.c     2004-08-30 22:26:21.000000000 +0530
@@ -49,6 +49,7 @@
 #include <asm/io.h>
 #include <asm/tlb.h>
```

```
 #include <asm/div64.h>
+#include <asm/memsave.h>

 #define LOAD_INT(x) ((x) >> FSHIT)
 #define LOAD_FRAC(x) LOAD_INT(((x) & (FIXED_1-1)) * 100)
@@ -689,6 + 690,28 @@ void_init proc_misc_init(void)
                (size_t) high_memory – PAGE_OFFSET + PAGE_SIZE;
}
#endif
+#ifdef CONFIG_MEMSAVE
+    /* Scan through the entries and create files for registered
+    * entries
+    */
+    {
+        struct memsave *bufp;
+        int n, i = MEMSAVE_SIZE / sizeof (struct memsave);
+
+        bufp = (struct memsave *)_va (MEMSAVE_BASE);
+
+        for (n = 0; n < i; n++) {
+            if (bufp->sig != MEMSAVE_ENTRY_SIG)
+                break;
+            entry = create_proc_entry (bufp->name, s_IRUSR, NULL);
+            if (entry) {
+                entry->proc_fops = &proc_memsave_operations;
+                entry->size = bufp->size;
+            }
+            (char *)bufp += sizeof (struct memsave);
```

```
+          }
+      }
+#endif
     if (prof_on) {
        entry = create_proc_entry ( "profile" , S_IWUSR | S_IRUGO, NULL);
      if (entry) {
diff –puN include/linux/proc_fs.h~memsave include/linux/proc_fs.h
--- /linux-patch/include/linux/proc_fs.h~memsave     2004-08-29 20:59:30.000000000
+0530
+++ linux-patch-hari/include/linux/proc_fs.h          2004-08-30 23:41:17.000000000
+0530
@@ -114,6 +114,11 @@ extern struct dentry *proc_lookup(struct
   extern struct file_operations proc_kcore_operations;
   extern struct file_operations proc_kmsg_operations;
   extern struct file_operations ppc_htab_operations;
+#ifdef CONFIG_MEMSAVE
+extern struct file_operations proc_memsave_operations;
+extern void memsave_dummy_entry(void);
+extern void memsave_dummy_entry_2(void);
+#endif

 /*
  * proc_tty.c
diff –puN fs/proc/Makefile~memsave fs/proc/Makefile
--- linux-patch/fs/proc/Makefile~memsave   2004-08-29 21:03:28.000000000 +0530
+++ linux-patch-hari/fs/proc/Makefile        2004-08-29 21:04:10.000000000 +0530
@@ -12,3 +12,4 @@ proc-y += inode.o root.o base.o ge
```

```
   proc-$(CONFIG_PROC_KCORE) +=kcore.o
   proc-$(CONFIG_PROC_DEVICETREE) += proc_devtree.o
+ proc-$(CONFIG_MEMSAVE) += proc_memsave.o
diff –puN /dev/null fs/proc_memsave.c
--- /dev/null        2003-01-30 15:54:37.000000000 +0530
+++ linux-patch-hari/fs/proc/proc_memsave.c        2004-08-31 00:20:19.00000000
+0530
@@ -0,0 +1,73 @@
+/*
+ * fs/proc/proc_memsave.c Contains the routines that abstract the saved
+ * memory regions as files.
+ *
+ *Created by: Hariprasad Nellitheertha
+ *
+ *Copyright (C) IBM Corporation, 2004. All rights reserved
+ *
+ */
+
+#include <linux/mm.h>
+#include <linux/proc_fs.h>
+#include <linux/capability.h>
+#include <linux/user.h>
+#include <linux/user.h>
+#include <asm/memsave.h>
+#include <asm/uaccess.h>
+#include <asm/io.h>
+
+static int open_memsave(struct inode * inode, struct file * filp)
```

```
+{
+       return capable (CAP_SYS_RAWIO) ? 0: -EPERM;
+}
+
+static ssize_t read_memsave(struct file *, char_user *, size_t, loff_t *);
+
+struct file_operations proc_memsave_operations = {
+       .read                   = read_memsave,
+       .open           = oper_memsave,
+};
+
+/*
+ * read_memsave( )
+ *
+ * This routine provides the interface to read the saved memory regions.
+ * It uses the file pointer to identify the region based on the name. It
+ * then reads the corresponding memory region and returns it back to the
+ * user.
+ *
+ */
+static ssize_t read_memsave(
+struct file *file, char_user *buffer, size_t buflen, loff_t *fops)
+{
+       struct memsave *msave;
+       int n;
+       void *region;
+
+       /* First, get to the correct entry in the list */
```

```
+       msave = (struct memsave *)_va (MEMSAVE_BASE);
+       for (n = 0; n < (MEMSAVE_SIZE / sizeof (struct memsave)); n++) {
+               if (msave->sig != MEMSAVE_ENTRY_SIG)
+                       return -EFAULT;
+               if (!strcmp (msave->name, file->f_dentry->d_name.nema))
+                       break;    /* Found the entry */
+               (char *)msave += sizeof(struct memsave);
+       }
+
+       if (msave->sig != MEMSAVE_ENTRY_SIG)
+               return –EFAULT;
+
+       if (buflen > msave->size - *fpos)
+               buflen = msave->size - *fops;
+
+       region = _va (msave->start);
+
+       if (copy_to_user (buffer, region + *fops, buflen)) {
+               return -EFAULT;
+       }
+
+       *fops += buflen;
+
+       return buflen;
+}
diff –puN /dev/null arch/i386/kernel/mdummy.c
---   /dev/null        2003-01-30 15:54:37.000000000 +0530
+++ linux-patch-hari/arch/i386/kernel/mdummy.c     2004-08-30 23:38:01.000000000
```

```
+0530
@@ -0,0 +1,46 @@
+/*
+ *Contains code to create dummy entries for memsave.
+ *
+ *Created by Hariprasad Nellitheertha
+ *
+ *Copyright (C) IBM Corporation, 2004. All rights reserved.
+ *
+ */
+
+#include <asm/errno.h>
+#include <asm/memsave.h>
+
+struct memsave msave_dum;
+struct memsave msave_dum_2;
+
+/*
+ * Create a dummy entry to save a particular region
+ * Save 1 page starting from 144MB.
+ */
+void memsave_dummy_entry (void)
+{
+
+       int ret;
+
+       strcpy (msave_dum.name, "msave" );
+       msave_dum.start = 0x09000000;
```

```
+       msave_dum.size = 0x1000;
+
+       ret = memsave_add_entry (&msave_dum);
+
+       strcpy((char *)0xc9000000,  "harihari" );
+}
+
+void memsave_dummy_entry_2(void)
+{
+
+       int ret;
+
+       strcpy(msave_dum_2.name,  "msave2" );
+       msave_dum_2.start = 0x0a000000;
+       msave_dum_2.size = 0x1000;
+
+       ret = memsave_add_entry (&msave_dum_2);
+
+       strcpy ((char *)0xc9000000, "dummydummy" );
+}
diff –puN fs/proc/root.c~memsave fs/proc/root.c
--- linux-patch/fs/proc/root.c~memsave      2004-08-30 22:30:59.000000000 +0530
+++ linux-patch-hari/fs/proc/root.c   2004-08-30 23:38:23.000000000 +0530
@@ -52,6 +52,10 @@ void_init proc_root_init (void)
                return;
        }
        proc_misc_init( );
+#ifdef CONFIG_MEMSAVE
```

```
+       memsave_dummy_entry( );
+       memsave_dummy_entry_2 ( );
+#endif
          proc_net = proc_mkdir ( “net” , NULL);
 #ifdef CONFIG SYSVIPC
          proc_mkdir ( “sysvipc” ,  NULL);
```

安装和运行代码的指令

以下指令可被用于安装和运行以上提供的缓冲器提取代码。

1）从http://www.kernel.org/获得 Linux 版本 2.6.9-rcl 的源代码

注意：以上呈现的代码已被准备用于此版本的 Linux。

它可以稍后被更新至任何级别。

2）从http://www.xmission.com/~ebiederm/files/kexec/2.6.8.1-kexec3/

http://www.xmission.com/~ebideerm/files/kexec/kexec-tools-1.96.tgz

获得 kexec 和 kexec-tools 的补丁。

3）通过包中给出的指令来安装 kexec-tools 的用户空间实用工具。

4) 解压缩 Linux 内核源代码:

tar -xzvf linux-2.6.9-rcl.tar.gz

5）将 kexec 补丁应用于 Linux 源代码:

patch –pl <kexec-file-name

6）将 memsave.patch（在以上附录中示出）应用于代码:

patch –pl <memsave.patch

7）构建 Linux 内核并引导至该内核。

8）该补丁将创建对基础设施的两次注册，以在 kexec 引导期间保存两个存储器区域。这些存储器区域将被注册为“msave”和“msave2”。

9）加载 kexec 内核:

kexec –l <kernel-name> --append=“<command-line-options>”

10）重新引导到新的 kexec 内核:

kexec -e

11）观察到在/proc 下已经创建两个文件：

/proc/msave 和/proc/msave2

12）使用“cp”命令写出这两个文件：

cp /proc/msave file1

cp /proc/msave2 file2

为了通过此方法验证重新引导期间被保留的正确的存储器区域，所述代码将虚拟标识符串添加到所保留的存储器区域中。由 msave 表示的第一区域在开头包括“harihari”串。由 msave2 表示的第二区域包括“dummydummy”串。在重新引导之后，一旦文件已被写出（参见以上列出的指令的步骤 12），每个虚拟标识符串就可以在其相应文件中找到，从而验证该方法。

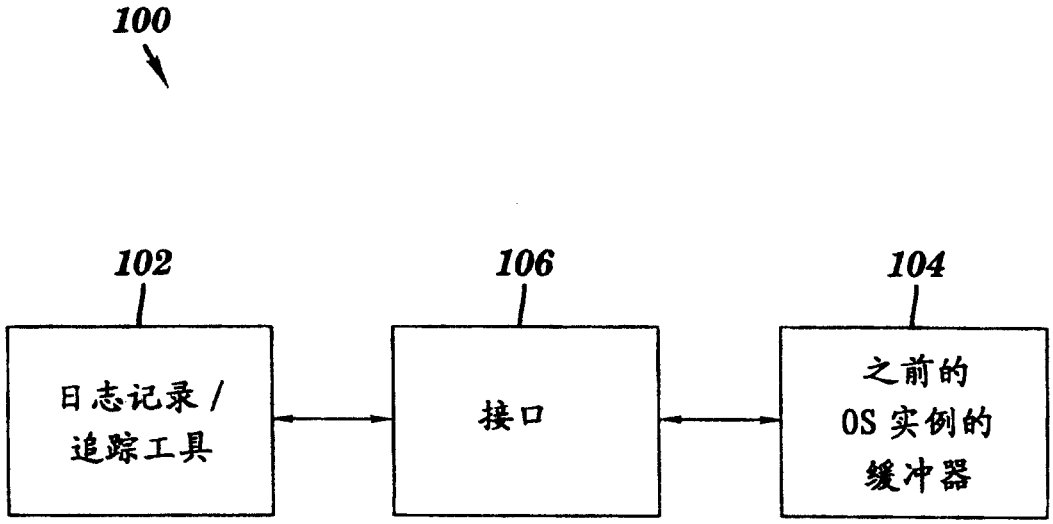


图 1

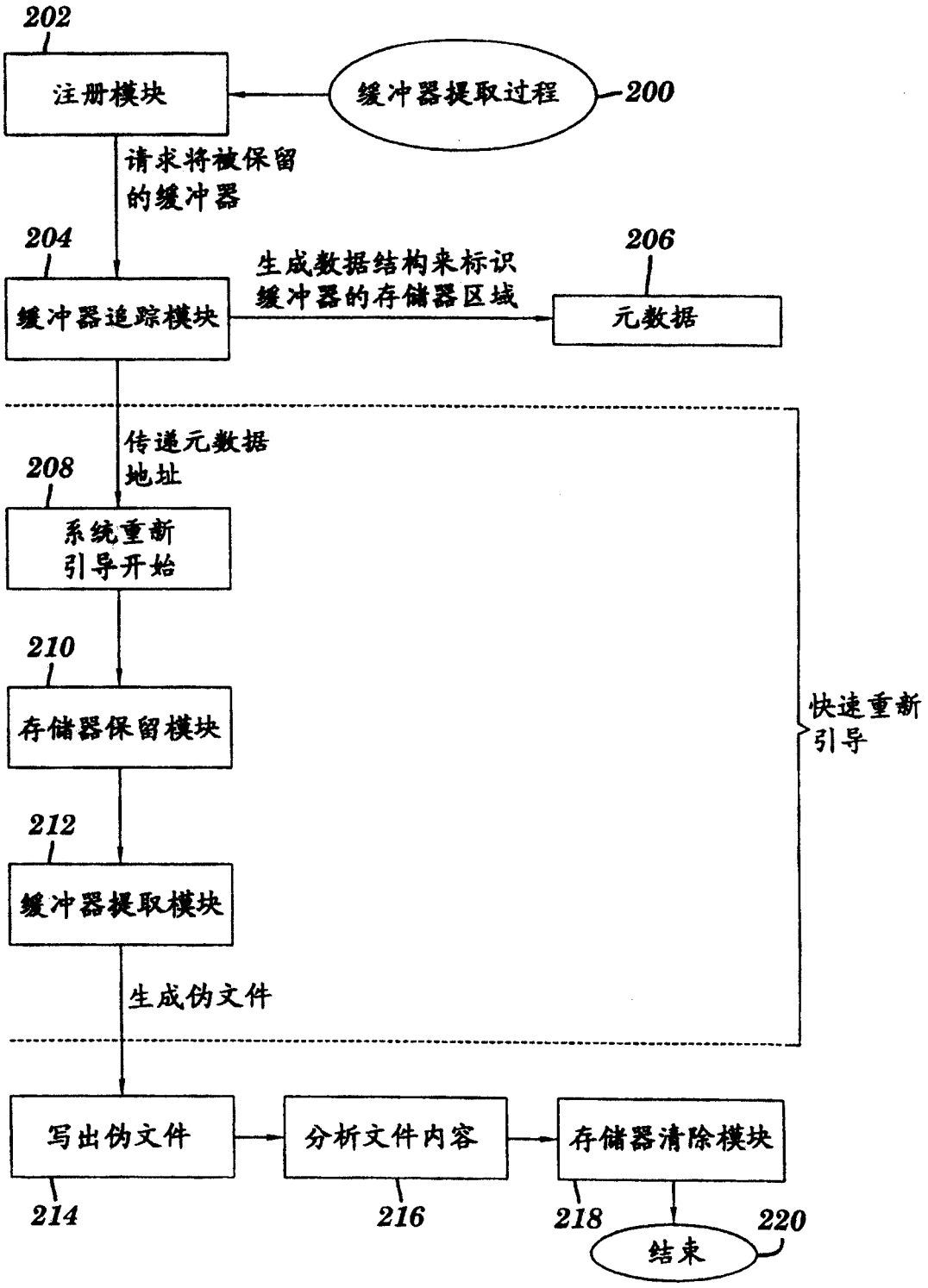

202
注册模块
缓冲器提取过程
200
请求将被保留
的缓冲器
204
缓冲器追踪模块
生成数据结构来标识
缓冲器的存储器区域
206
元数据
传递元数据
地址
208
系统重新
引导开始
210
存储器保留模块
快速重新
引导
212
缓冲器提取模块
生成伪文件
写出伪文件
分析文件内容
存储器清除模块
214
216
218
220
结束

图 2

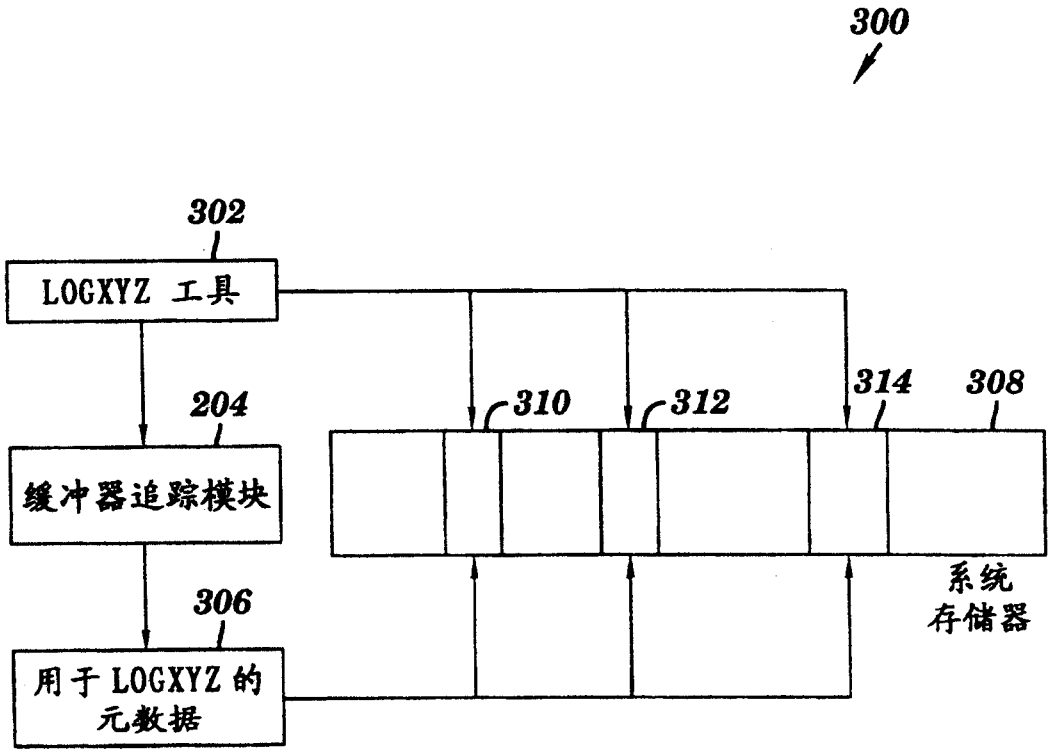

300
302
LOGXYZ 工具
204
缓冲器追踪模块
306
用于 LOGXYZ 的
元数据
310
312
314
308
系统
存储器

图 3A

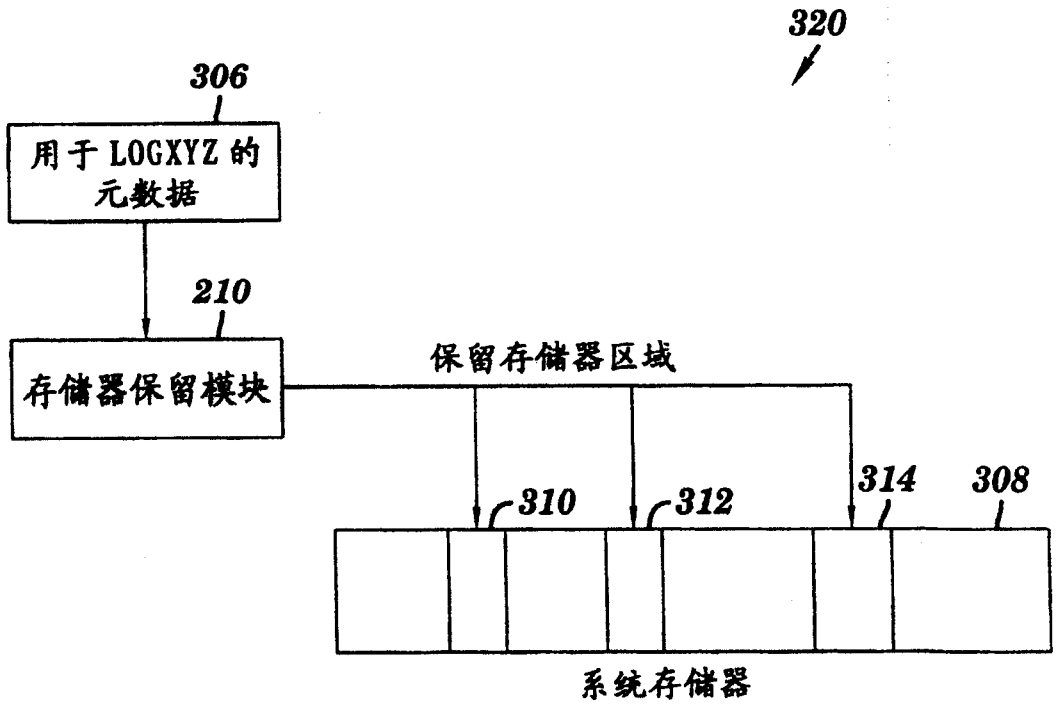

320
306
用于 LOGXYZ 的元数据
210
存储器保留模块
保留存储器区域
310
312
314
308
系统存储器

图 3B

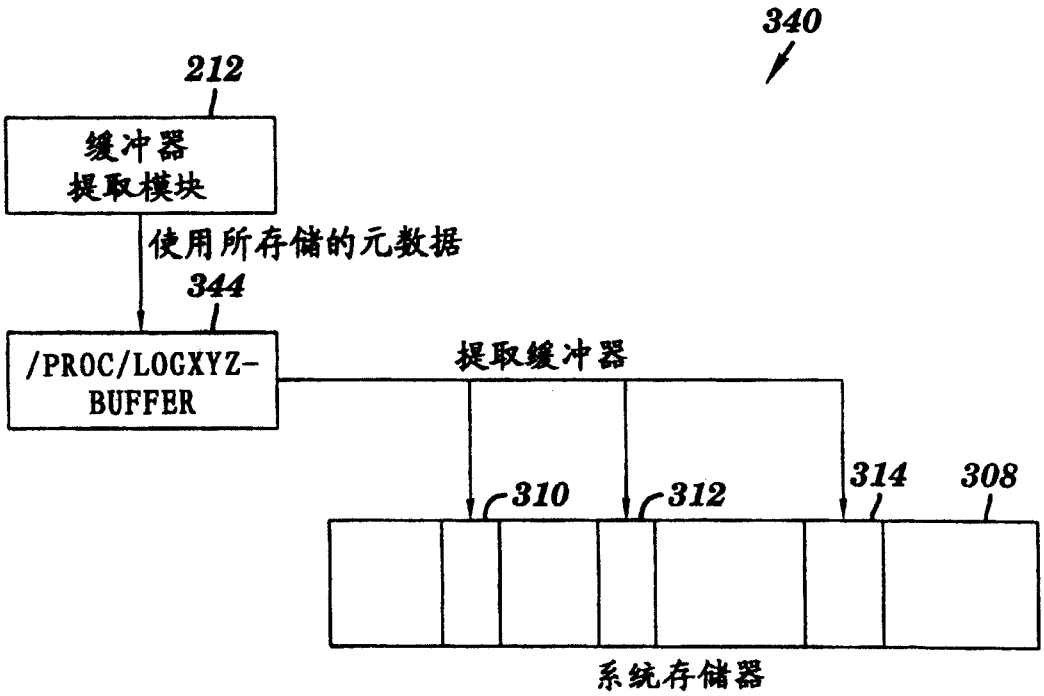

340
212
缓冲器
提取模块
使用所存储的元数据
344
/PROC/LOGXYZ-
BUFFER
提取缓冲器
310
312
314
308
系统存储器

图 3C

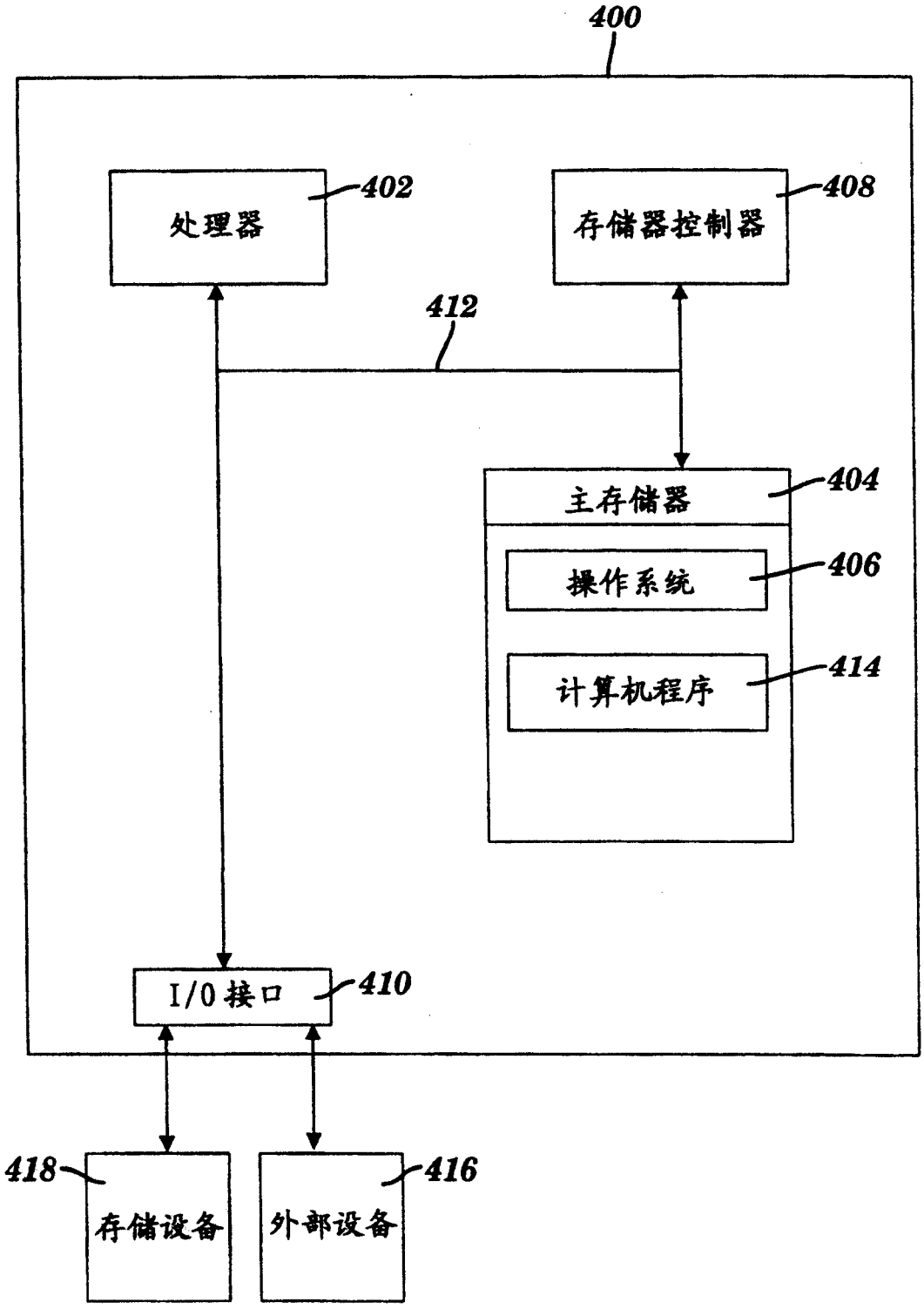

400
处理器
402
存储器控制器
408
412
主存储器
404
操作系统
406
计算机程序
414
I/O接口
410
418
存储设备
外部设备
416

图 4