[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
*G06F 17/30 (2006.01)*

# [12] 发明专利说明书

专利号 ZL 01822945.X

[45] 授权公告日 2006 年 12 月 13 日

[11] 授权公告号 CN 1290041C

[22] 申请日 2001.11.20 [21] 申请号 01822945.X

[30] 优先权

[32] 2000.12.30 [33] US [31] 09/751,862

[86] 国际申请 PCT/US2001/044883 2001.11.20

[87] 国际公布 WO2002/054286 英 2002.7.11

[85] 进入国家阶段日期 2003.8.29

[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·达克

审查员 齐 霁

[74] 专利代理机构 中国专利代理(香港)有限公司

代理人 程天正 王 勇

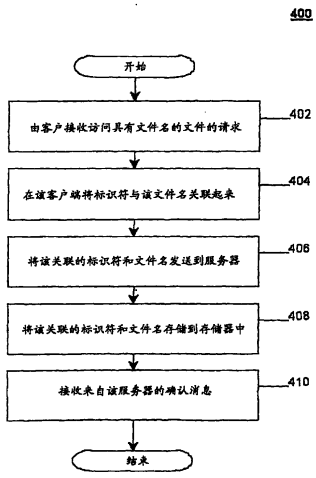
权利要求书 3 页 说明书 30 页 附图 6 页

[54] 发明名称

用于改进文件管理的方法和装置

[57] 摘要

描述一种用于在计算机和通信网络中管理文件的方法和装置。



1.一种用一个标识符来代表一文件名以改进互连系统的性能的计算机实现的方法，包括:

由文件系统接口为客户接收访问具有文件名的文件的第一请求;

由所述客户响应于所述第一请求、使用一流圆锥消息传送系统为所述文件名生成一标识符，使所述标识符代表所述文件名且包括比所述文件名更少的比特数; 和

将所述第一请求与所述标识符和所述文件名一起发送到一个服务器。

2.权利要求 1 的方法，还包括将所述标识符和文件名存储到存储器中。

3.权利要求 1 的方法，还包括接收来自所述服务器的确认消息。

4.权利要求 1 的方法，还包括:

在所述客户端接收访问所述文件的第二请求;

从所述存储器中检索与所述文件名相关联的所述标识符; 和

使用所述关联的标识符将所述第二请求发送到所述服务器。

5.权利要求 4 的方法，其中所述第一和第二请求指定一个文件操作。

6. 一种用一个标识符来代表一文件名以改进互连系统的性能的计算机实现的方法，包括:

在服务器端接收来自一客户的、访问具有文件名和标识符的文件的第一请求，所述标识符代表所述文件名且包括比所述文件名更少的比特数，使所述标识符由客户响应于所述第一请求、使用一个流圆锥消息传送系统来生成; 和

将一确认消息发送到所述客户。

7.权利要求 6 的方法，还包括:

搜索所述文件的位置信息;

将所述位置信息与所述标识符关联起来; 和

将所述位置信息和所述标识符存储到存储器中。

8.权利要求 7 的方法，还包括:

在所述服务器端接收使用所述标识符访问所述文件的第二请求; 和

使用所述标识符从所述存储器中检索所述位置信息。

9.一种用一个标识符来代表一文件名以改进互连系统的性能的计算

机实现的方法，包括:

由一个文件系统接口为客户接收具有文件名的文件请求;

使用一流圆锥消息传送系统为所述文件名生成一标识符;

将所述标识符和所述文件名发送到一服务器;

使用所述文件名来搜索位置信息; 和

存储所述位置信息与所述标识符。

10.权利要求 9 的方法，还包括将一确认消息发送到所述客户。

11.一种用一个标识符来代表一文件名以改进互连系统的性能的计算机实现的方法，包括:

通过一文件系统接口为一客户接收带有文件名的文件请求;

通过所述文件系统接口响应于所述文件请求、使用一流圆锥消息传送系统为所述文件名生成一惟一标识符，所述惟一标识符代表所述文件名且包括比所述文件名更少的比特数; 和

将所述惟一标识符和文件名发送到一文件系统管理器。

12.权利要求 11 的方法，还包括:

在所述文件系统管理器接收所述惟一标识符和所述文件名;

使用所述文件名搜索文件信息; 和

使用所述惟一标识符存储所述文件信息。

13.一种用一个标识符来代表一文件名以改进互连系统的性能的装置，包括:

用于客户的、接收对具有文件名的文件的请求的文件系统接口，所述客户响应于所述请求、使用一流圆锥消息传送系统为所述文件名生成一个惟一的识别符，所述惟一的标识符代表所述文件名且包括比所述文件名更少的比特数，并且将所述惟一标识符和所述文件名发送到一服务器; 和

与所述客户相连接的互连系统，该互连系统将所述惟一标识符和所述文件名传送到所述服务器。

14.权利要求 13 的装置，还包括接收所述惟一标识符和文件名的服务器，所述服务器用于定位所述文件的信息，并且使用所述惟一标识符来存储所述信息。

15.一种用一个标识符来代表一文件名以改进互连系统的性能的装置，包括:

用于客户的、响应于一文件请求而使用一流圆锥消息传送系统来为一个文件名生成标识符的文件接口系统，所述标识符代表所述文件名且包括比所述文件名更少的比特数；

用于使用所述文件名来定位文件信息以及使用所述标识符存储所述文件信息的服务器；和

用于在所述客户和所述服务器之间传输所述文件名和所述标识符的互连系统。

16.权利要求15的装置，其中所述客户包括一个操作系统服务模块。

17.权利要求15的装置，其中所述服务器包括一个中间服务模块。

18.权利要求15的装置，其中所述互连系统根据外围部件互连系统和$I_2O$系统进行操作。

19.一种用一个标识符来代表一文件名以改进互连系统的性能的装置，包括：

接收对具有文件名的文件的请求并且响应于所述请求、使用一流圆锥消息传送系统来为所述文件名生成一惟一标识符的文件系统接口，所述标识符代表所述文件名且包括比所述文件名更少的比特数；

使用所述文件名定位文件信息和使用所述惟一标识符存储所述文件信息的文件系统管理器；和

在所述文件系统接口和所述文件系统管理器之间传送所述惟一标识符和所述文件名的通信系统。

20.权利要求19的装置，其中所述通信系统包括：

通信介质，该通信介质包括如下组中的至少一种，该组包括双绞线、同轴电缆、光纤和射频；以及

根据一组通信协议操作的通信接口。

# 用于改进文件管理的方法和装置

## 领域

本公开内容涉及计算机和通信系统。更具体而言，它涉及到用于在计算机和通信系统中改进文件管理的方法和装置。

## 背景

计算机和通信系统在体系结构上常常是相似的。每种系统可以包括大量的独立部件，它们每个都被设计成执行确定的功能。这些部件可以通过互连系统将信息传送到其他部件。互连系统操作来管理通过通信介质，诸如金属引线、双绞线、同轴电缆、光纤、射频等等对信息的传送。在部件之间的通信典型地有助于协调单个部件的操作，从而使得它们可以作为内聚的系统来活动。这种类型的分布式系统的体系结构在性能、冗余、可伸缩性和效率上可以提供某些优点。

不过，也存在与这种类型的系统设计相关联的缺点。一个缺点是：部件在等待着来自另一个部件的信息时，它可能不得不保持空闲。空闲时间可代表系统资源的低效率使用。另一个缺点是：功能性可能采用增加部件之间传送的信息量的方式被分配给部件。这种通信中的增加会增加对该互连系统的有限资源的需求。

## 附图简述

关于本发明的实施例的主题在本说明书的结论部分被特别指出和被不同地要求权利。不过，通过参考如下的详细说明并结合附图阅读时，就会对本发明的实施例，对于组织和操作方法这二者，以及本发明的目的、特征、和优点有最好的理解，在这些附图中：

图 1 是一种适用于实践本发明的一个实施例的系统。

图 2 是根据本发明的一个实施例的客户系统的框图。

图 3 是根据本发明的一个实施例的服务器系统的框图。

图 4 由根据本发明的一个实施例的客户系统所执行的操作的方框流程图。

图 5 由根据本发明的一个实施例的服务器系统所执行的操作的方

框流程图。

图 6 说明一种根据本发明的一个实施例的软件体系结构。

详述

在以下的详细描述中，对大量的具体细节进行了阐述以便提供对本发明的实施例的彻底理解。不过，对于本领域普通技术人员而言，应该理解的是，在没有这些具体细节的情况下，本发明的实施例也是可以被实践的。在其他例子中，并没有在细节上对公知的方法、过程、部件和电路进行描述，以便不使本发明的实施例不清楚。

本发明的实施例可以通过减小分布式部件的空闲时间以及对于互连系统的带宽需求来改进分布式系统的性能。更具体而言，本发明的一个实施例可以改进用来管理文件的分布式系统的性能。因此，这样就可以减小与文件操作相关联的时延。相应地，用户就可以在更多的响应式应用和服务方面受益。

文件管理系统典型地包括客户和服务器。在本上下文中的客户可以指服务的请求者。在本上下文中的服务器可以指服务的提供者。该客户典型地通过互连系统将文件请求发送到服务器。该文件请求可以采用消息的形式，并且可以包括功能请求和文件名。在本上下文中的消息可以包括一个或者多个字母数字字符、符号或者逻辑表达式，当它们被组合时，表示例如控制字、命令、指令、信息或者数据。该消息在构成上可以变化，例如从一个位到完整的短语。该服务器将一惟一的标识符与该文件名关联起来，标识该文件的位置信息，并且将它与该惟一的标识符存储在一起。该服务器然后将该惟一标识符送回到该客户。该客户接收该惟一标识符，并且对于后续的文件请求就用它来代替该文件名。

惟一标识符被指配给文件名，以便减小对该互连系统的带宽需求。文件名例如可以包括许多字母数字字符或者符号("字符串")。每个字符或者符号被转换成一个或者多个位，典型地是每个字符 8 位(例如，1 字节)。因为文件名可以包括许多字符，所以该互连系统可能不得不传输数目相对大的位。为了减少这一问题，该服务器可以将惟一标识符指配给每个文件名，并且将该惟一标识符发送给该客户。该惟一标识符典型地在长度上要小于该文件名。例如，一个惟一标识符可

能的长度为 32 或者 64 位。于是，该客户对于后续的文件请求就可以使用该惟一标识符。

不过，让该服务器将惟一标识符指配给文件名也存在缺点。一个缺点是：在处理后续的文件请求之前，当该客户等待着该惟一标识符时，它可能不得不保持空闲。另一个缺点是：该客户和服务器可能需要传送比所需的更多的信息，由此就增加了对该互连系统的需求。

相对于第一缺点，该客户在等待着来自该服务器的惟一标识符时，它可能不得不保持空闲。为了启动该指配过程，该客户将一条消息发送到该服务器，请求将惟一标识符指配给文件名。该服务器将一个惟一标识符与该文件名关联起来，标识该文件的位置信息，并且将它与该惟一标识符存储在一起。该服务器然后就将一条消息与该惟一标识符一起送回到该客户。在等待着接收来自该服务器的惟一标识符的同时，该客户可以接收具有该相同文件名的后续的文件请求。该客户也许直到该整个指配过程完成时才能开始处理这些文件请求。因此，在这一时间段期间，该客户也许被迫要保持空闲。

相对于第二缺点，为了执行指配功能，该客户和服务器可能需要传送不必要的信息，由此就增加了对该互连系统的需求。上述的文件管理系统要求至少两条消息。该客户将第一条消息发送到该服务器，请求将惟一标识符指配给文件名。该服务器将第二条消息与该惟一标识符一起发送给该客户。每条消息要求使用该互连系统的一定量的带宽。在本上下文中的带宽可以指信息在互连系统上能够被传送的速度，并且典型地采用每秒千位(kbps)来加以测量。通过对比，本发明的一个实施例可以仅仅使用一条消息来执行该指配过程，由此潜在地就将对该互连系统的带宽需求降低达 50%之多。

本发明的实施例可以通过降低客户的空闲时间以及对该互连系统的带宽需求来改进分布式系统的性能。本发明的一个实施例在客户端将一惟一标识符指配给文件名，并将该惟一标识符发送到该服务器。这样就可以减小客户的空闲时间，因为该客户在不必等待来自该服务器的消息的情况下就可以开始处理后续的文件请求。这样也可以减小带宽需求，因为该服务器不必为完成该指配过程而将消息送回到该服务器，或者可选地，可以发送比以前的文件管理系统所需要的消息更短的消息。

值得指出的是：在本说明书对"一个实施例"或者"实施例"的任何参考在本上下文中意思是:结合该实施例所描述的特定的特征、结构、或者特性可能被包括在本发明的至少一个实施例中。在本说明书的各个地方所出现的短语"在一个实施例中"未必都要指向相同的实施例。

现在详细地参考这些附图，其中相同的部件自始至终由相同的附图标记来指定，图 1 说明一种适用于实践本发明的一个实施例的系统 100。如图 1 所示，系统 100 包括由互连系统 104 所连接的客户 102 和服务器 106。在此所使用的术语"客户"可以指信息的任何请求者。在此所使用的术语"服务器"可以指信息的任何提供者。

图 2 是根据本发明的一个实施例的客户系统的框图。图 2 说明可以代表客户 102 的客户 200。如图 2 所示，客户 200 包括处理器 202、存储器 204 和接口 208，它们都由连接 210 所连接。存储器 204 可以存储程序指令和数据。术语"程序指令"包括计算机代码段，该计算机代码段包括预定的计算机语言的字、值和符号，当它们以按照预定的方式或者语法的组合被放置时，使处理器执行确定的功能。计算机语言的例子包括 C、C++和汇编语言。处理器 202 执行该程序指令，并且处理在存储器 204 中所存储的数据。接口 208 协调数据从客户 200 到另一个设备的传输。连接 210 传送处理器 202、存储器 204、和接口 208 之间的数据。

处理器 202 可以是能够提供本发明的各实施例所需要的速度和功能性的任何类型的处理器。例如，处理器 202 可以是由英特尔公司（Intel）、摩托罗拉（Motorola）、康柏（Compaq）或者太阳微系统（Sun）制造的处理器系列中的处理器。在本发明的一个实施例中，处理器 202 可以是用于管理输入/输出(I/O)设备，诸如硬盘驱动器、键盘、打印机、网络接口卡等等的专用处理器。这种处理器典型地被称为 I/O 处理器(IOP)。

在本发明的一个实施例中，存储器 204 包括机器可读介质，并且可以包括能够存储适用于由处理器加以执行的指令的任何介质。这种介质的一些例子包括但不限于，只读存储器(ROM)、随机存取存储器(RAM)、可编程 ROM、可擦除可编程 ROM、电可擦除可编程 ROM、动态 RAM、磁盘(例如，软盘和硬盘驱动器)、光盘(例如，CD-ROM)和可以存储数字信息的任何其他介质。在本发明的一个实施例中，该指令采用

压缩和/或加密的格式存储在该介质上。如此处所使用的，短语"适用于由处理器加以执行"意思是包括采用压缩和/或加密格式所存储的指令，以及必须被编译或者在由该处理器执行之前由安装程序安装的指令。此外，客户 200 通过各种 I/O 控制器可以包含机器可读存储设备的各种组合，它们可由处理器 202 加以访问并且能够存储计算机程序指令和数据的组合。

存储器 204 可以存储和允许由处理器 202 执行的程序指令和数据，以便实现客户（诸如客户 106 和客户 200）的功能。在本发明的一个实施例中，存储器 204 包括一组程序指令，该组程序指令在此共同地被称为文件系统接口 206。

文件系统接口 206 可以是这样的一个接口，它操作来提供对系统 100 的一个或者多个文件的访问。在本上下文中的接口可以指一种定义的协议，通过该协议，一个软件模块可以访问另一个软件模块的功能性。在本上下文中的文件是指在存储器诸如存储器 204 或者硬盘驱动器中所存储的数据的离散集。文件系统接口 206 可以接收请求以便对文件执行某些操作，诸如建立、打开、查找、读、写、重命名、删除、复制、移动等等。该请求例如可以源自主机 OS 或者应用程序。主机 OS 可以包括用于系统的 OS。例如，如果本发明的实施例是作为个人计算机的一部分加以实现的，则该主机 OS 可以包括微软公司所出售的 OS，例如像 Microsoft Windows®95、98、2000 和 NT。

在本发明的一个实施例中，文件系统接口 206 作为由智能 I/O 规范 ($I_2O$) 所定义的操作系统服务模块 (OSM) 进行操作，智能 I/O 规范 ($I_2O$) 由 $I_2O$ 特殊兴趣组 (SIG) ($I_2O$ SIG) 开发，版本为 1.5，于 1997 年 4 月采用，并且可从"www.i20sig.org" ("$I_2O$ 规范") 中获得，不过本发明在范围上并不限制于这一方面。

作为背景，该 $I_2O$ 规范定义了用于独立于受控的专用设备和主机操作系统 (OS) 这二者的智能 I/O 的标准体系结构。在本上下文中的智能 I/O 是指将处理低级中断的功能从中央处理单元 (CPU) 或者其他处理器移到特别设计成对 I/O 功能提供处理的 I/O 处理器 (IOP)。这样就可以改进 I/O 性能以及允许该 CPU 或者其他处理器给其他任务提供处理功能性。在本上下文中的中断是指对访问 I/O 设备，诸如硬盘驱动器、软盘驱动器、打印机、监视器、键盘、网络接口卡 (NIC) 等等的请求。

该 $I_2O$ 规范描述 OSM、中间服务模块(ISM)和硬件设备模块(HDM)。该 OSM 可以是一个驱动器，该驱动器作为主机 OS 和 ISM 之间的接口进行操作。在本上下文中的驱动器是指一组程序指令，该组程序指令管理特定部件、设备或者软件模块的操作。该 ISM 可以作为在该 OSM 和硬件设备模块(HDM)之间的接口进行操作。该 ISM 可以执行用于 I/O 管理功能的特定功能性、网络协议或例如像后台归档这样的对等功能性。该 HDM 可以是用于操作来控制特定 I/O 设备的驱动器。

该 $I_2O$ 规范定义一种通信模型，该通信模型包括消息传递系统。该 OSM、ISM 和 HDM 通过采用消息的形式经消息层传递信息来通信和协调操作。在本上下文中的消息层可以管理和调度请求，提供用于交付消息的一组应用编程接口(API)，以及提供用于处理消息的一组支持例程。

在本发明的一个实施例中，文件系统接口 206 作为根据 $I_2O$ 规范的 OSM 进行操作。在本发明的一个实施例中，文件系统接口 206 通过主机 OS 接收来自应用程序的文件请求，根据该 $I_2O$ 规范将该请求翻译成消息，并且将它发送到文件系统管理器(在以下描述)以便处理。在本上下文中的应用程序是指这样的程序，该程序为专门任务提供预定的一组函数，典型地具有用户界面以便有利于处理用户和该计算机系统之间的命令和指令。应用程序的例子可以包括字处理程序、扩展页、数据库或者因特网浏览器。

接口 208 可以包括任何适合于使用例如希望的一组通信协议、服务和操作过程来控制在计算机或者网络设备之间的通信信号的技术。在本发明的一个实施例中，接口 208 可以例如根据 PCI 规范和 $I_2O$ 规范进行操作。在本发明的另一个实施例中，接口 208 可以根据如下协议进行操作，该协议分别是由因特网工程任务组(IETF)标准 7 的、于 1981 年 9 月采用的请求评注(RFC)793 所定义的传输控制协议(TCP)，和 IETF 标准 5 的、1981 年 9 月采用的请求评注(RFC)791 所定义的网际协议(IP)，这二者都可从"www.ietf.org"中获得。尽管接口 208 可以根据上述协议进行操作，但是应该理解的是，接口 208 可以使用任何适合于使用例如希望的一组通信协议、服务和操作过程来控制计算机或者网络设备之间的通信信号的技术进行操作，并且仍然落在本发明的范围之内。

接口 208 还包括用于将接口 208 连接到适合的通信介质的连接器。接口 208 可以通过任何合适的介质诸如铜线、双绞线、同轴电缆、光纤、射频等等接收通信信号。在本发明的一个实施例中，该连接器适合与总线一起使用，以便承载遵从 PCI 规范的信号。

图 3 是根据本发明的一个实施例的服务器系统的框图。图 3 说明根据本发明的一个实施例、代表服务器 106 的服务器 300。如图 3 所示，服务器 300 包括处理器 302、存储器 304 和接口 308，它们都由连接 310 加以连接。图 3 的元件 302、304、308 和 310 在结构和操作上与参考图 2 所描述的对应元件 202、204、208 和 210 是相似的。尽管所展示的服务器 300 带有处理器 302，但是应该理解的是，服务器 300 在没有处理器 302 的情况下可以通过使用可用于服务器 300 的另一个处理器(例如，处理器 202)来进行操作，并且仍然落在本发明的范围之中。例如，如果本发明的该实施例被并入到个人计算机中，其中该客户和服务器由 PCI 总线加以连接并且都共享单一的处理器，则这种配置可以出现。

在本发明的一个实施例中，存储器 304 包含文件系统管理器 306 的程序指令。文件系统管理器 306 执行文件管理，并提供对包含多个文件的存储介质(未示出)的访问。文件系统 306 对从文件系统接口 206 所接收的文件请求作出响应而执行文件操作，诸如建立、打开、查找、读、写、重命名、删除、复制、移动等等。文件系统接口 206 的一个例子包括根据 $I_2O$ 规范操作的 ISM，不过本发明的范围并不限制于这一方面。

将参考图 4 和 5，更详细地对系统 100、200 和 300 的操作进行描述。尽管在此所展示的图 4 和 5 包括特定的操作顺序，但是应该理解的是，操作的顺序仅仅提供在此所描述的通用功能是如何实现的例子。此外，操作的顺序不一定必须要按照所展示的次序加以执行，除非另有指示。

图 4 是由根据本发明的一个实施例的客户执行的操作的方框流程图。在本发明的该实施例中，文件系统接口 206 作为客户 106 的一部分进行操作。不过，应该理解的是，文件系统接口 206，可以由位于计算机或者网络系统中的任何地方的任何设备，或者设备的组合加以实现，并仍然落在本发明的范围之内。

如图 4 所示，在块 402，客户接收访问具有文件名的文件的请求。在块 404，该客户将该文件名与标识符关联起来。在块 406，该客户将该关联的标识符和文件名发送到服务器。在块 408，该客户将该关联的标识符和文件名存储到存储器中。在块 410，该客户接收来自该服务器的确认消息。

一旦该客户将标识符指配给文件，则该标识符就被用于对该文件的以后的请求。该客户在该客户端接收访问该文件的第二个请求。该客户从存储器中检索与该文件名相关联的标识符。该客户使用该关联的标识符将第二个请求发送到该服务器。

图 5 是由根据本发明的一个实施例的服务器加以执行的操作的方框流程图。在本发明的该实施例中，文件系统管理器 306 作为客户 106 的一部分进行操作。不过，应该理解的是，这一功能性可以由位于计算机或者网络系统中任何地方的任何设备，或者设备的组合加以实现，并仍然落在本发明的范围之内。

如 5 图所示，在块 502，服务器接收文件的文件名和关联的标识符。在块 504，该服务器将确认消息发送给该客户。在块 506，该服务器搜索该文件的位置信息。在块 508，该服务器将该位置信息和该标识符关联起来。该服务器将该关联的位置信息和标识符存储到存储器中。

一旦该服务器使用该标识符索引到了文件的位置信息，该服务器就能够对后续的文件请求使用该标识符来访问该位置信息。该服务器接收具有该标识符的访问文件的第二个请求。该服务器使用该标识符从存储器检索该位置信息。

通过例子，对系统 100、200 和 300 的操作，以及在图 4 和 5 中所展示的流程图可以有更好的理解。应用程序发送请求以便将信息从文件名为"测试文件一"的文件中读到系统 100 的主机 OS。在本上下文中的惟一标识符是指一系列字母数字字符，当它们被组合时，相对于该客户、服务器和/或系统所使用的其他字、值或者二进制串，它们代表对该客户、服务器和/或系统惟一的字、值或者二进制串。主机 OS 将该文件请求传递给文件系统接口 206。文件系统接口 206 生成惟一标识符"A123"，并将它指配给文件名"测试文件一"。在本发明的该实施例中，该惟一标识符"A123"例如可以是十六进制的 32 位数。文件系统接

口 206 建立一条消息"标识(测试文件一，A123)"，并且将它放置到出站消息队列，以便通过连接 104 传输到文件系统管理器 306。在本上下文中的该出站消息队列是指诸如先进先出(FIFO)这样的队列，该队列用于保持消息直到该互连系统能够传输该消息为止。文件系统接口 206 将"测试文件一"与"A123"一起存储在存储器 204 中的查找表中。

文件系统管理器 306 通过连接 104 接收该消息。文件系统管理器 306 解析该消息，并且调用函数"identify(测试文件一，A123)"。在本上下文中的术语“解析”是指将单个的字符或者字符的子集从表示例如命令、控制字、文件名、数据、函数调用、子例程名、标志等等的消息中分离开。在本上下文中术语"调用"是指被发送到该处理器的命令开始执行与给定的函数或者子例程相关联的程序指令。这一函数接受作为输入的"测试文件一"和"A123"，并且通知文件系统管理器 306: 该文件名"测试文件一"将在后续的文件请求中作为"A123"被加以引用。这可以通过以下方式来实现: 将该文件名和惟一标识符一起作为对应的或者链接的项存储来更新存储器中的查找表，即，通过搜索一个可以找到另一个。文件系统管理器 306 搜索文件"A123"的位置信息，它典型地位于诸如硬盘驱动器这样的存储设备上。位置信息的例子可以包括寻址信息、设备、柱面号和磁道号，不过，本发明在范围上并不限制于这一方面。文件系统管理器 306 将该位置信息与标识符"A123"关联起来，并且将该位置信息与标识符"A123"存储在存储器 304 中的查找表中。文件系统管理器 306 向文件系统接口 206 发送确认消息: 该文件名标识符和位置信息已经被接收。在本上下文中的确认消息是指一条短消息，指示前一条消息已被接收。该确认消息可以包括单个位、字符、字或者短语，如特定系统所希望的。

当请求对文件名"测试文件一"进行操作时，由文件系统管理器 306 所接收的后续的文件请求然后将使用标识符"A123"。例如，文件系统接口 206 接收第二个请求来执行对"测试文件一"的"delete(删除)"操作。文件系统接口 206 从存储器中检索"测试文件一"的先前关联的标识符"A123"。文件系统接口 206 将消息"delete('A123')"发送到文件系统管理器 306。文件系统管理器 306 接收该消息"delete('A123')"，并且使用该标识符"A123"检索文件名"测试文件一"的位置信息。文件系统管理器 306 然后使用所检索的位置信息执行所请求的文件操作。

在本发明的一个实施例中，文件系统接口 206 可以作为根据 $I_2O$ 规范的 OSM 以及作为诸如可从"www.kernel.org"("Linux kernel")中获得的 Linux OS 版本 2.3.99 pre-3 内核这样的特定主机 OS 加以实现。在本发明的这一实施例中，该 OSM 还可以包括流森林（stream forest）OSM、流树（stream tree）OSM、以及类的说明。在本上下文中的类可以指特定的接口定义。在本上下文中的树可以指被称为圆锥（cone）的存储对象的集合。在本上下文中的对象可以指类的实例。在本上下文中的圆锥可以指例如支持读、写和加锁能力的存储对象。在本上下文中的森林可以指树的集合。

在本发明的这一实施例中，流森林 OSM 可以被用于建立文件系统集合的模型。流森林 OSM 可以例如提供文件命名操作，诸如建立名字、删除名字、或者重命名特定的森林。此外，打开和关闭操作也可以被支持。树 OSM 可以被用于建立特定文件系统的模型。文件可以被建立成圆锥的模型。流森林 OSM 例如还可以用来支持文件操作，诸如命名文件、重命名它、删除它、给它加锁以防改变、读取它或者写入它。

该 OSM 可以被设计成与 ISM 进行通信。该 ISM 还可以包括流森林 ISM 和流树 ISM。流森林 ISM 可以支持该 OSM 的文件系统命名能力。在本发明的一个实施例中，流树 ISM 可以支持长度为 $2^8$ 个字符的流圆锥标识符。流树 ISM 还可以支持 $2^{16}$ 个打开的流圆锥，以及 $2^{32}$ 个可能的流圆锥。流树 ISM 对于所有包含的流圆锥可以支持 $2^{64}$ 个字节。应该理解的是，这些值并未将本发明的范围限制在这一方面。

在操作中，流树 OSM、流森林 OSM、流树 ISM 和流森林 ISM 都可以使用根据以上所述的类规范的消息传送方案进行通信。这种消息传送方案将被称为流式消息传送方案，并且以下还要详细地加以论述。主机 OS 可以使用由流森林 OSM 和流树 OSM 所提供的功能性。该主机 OS 例如可以使用该流森林 OSM 来建立文件系统的编组的模型，并且使用该流树 OSM 来建立特定文件系统的模型。该流森林 OSM 可以使用该流森林 ISM 来管理 IRTOS 环境之中的文件系统的编组。该流树 OSM 可以使用该流树 ISM 来管理在 $I_2O$ 实时操作系统(IRTOS)环境之中的文件系统。该流森林 OSM 和流树 OSM 可以使用该流消息传送方案分别与该流森林 OSM 和流树 ISM 进行通信。还将参考图 7 对本发明的该实施例进行描述。

图 6 说明一种根据本发明的一个实施例的软件体系结构。如图 6 所示，系统 600 可以包括具有流森林 OSM 602、Linux OS 604、Linux 虚拟文件系统(VFS)606、流树 OSM 608、和一个或者多个 Linux $I_2O$ 驱动器 610 的文件系统接口。Linux $I_2O$ 驱动器 610 可以根据 Linux 内核和 $I_2O$ 规范进行操作。Linux $I_2O$ 驱动器 610 可以包括 PCI $I_2O$ 驱动器 612。PCI $I_2O$ 驱动器 612 可以根据 PCI 规范和 $I_2O$ 规范进行操作。

该文件系统接口通过用于根据 PCI 规范传送信号的总线 614 与文件系统管理器进行通信。该文件系统管理器可以包括流森林 ISM 616、流树 ISM 618、廉价磁盘冗余阵列(RAID)ISM 620、小型计算机系统接口(SCSI)HDM 622 和 IRTOS 624。

在本发明的这一实施例中，在系统 600 中所展示的模块可以使用 C 编程语言加以实现，不过本发明的范围并不限制于这一方面。此外，在本发明的这一实施例中，在系统 600 中展示的模块支持长度为 255 个字符的标识符。

流树 OSM 608 可以被配置成提供对典型文件系统的访问。流树 OSM 608 可以被配置成支持一个或者多个 Linux VFS 所要求的函数，如由 Linux 规范所定义的。流树 OSM 608 可以被配置成支持流树类消息，如在以下将更加详细地加以论述的。流树 OSM 608 可以被配置成使用例如 Linux 内核与 Linux OS 604 一起操作。流树 OSM 608 可以支持流树 ISM，诸如流树 ISM 618。

流森林 OSM 602 可以提供函数 loctl()kernel interface（内核接口）。流森林 OSM 602 可以用于使用函数名诸如"IOCTL_SF_TREECREATE"在流森林中建立流树。这一函数可以接收例如用 C 语言按如下定义的输入缓冲器作为输入:

```
struct ik_sf_treecreate{
    U32 SizeInBlocks;
    char name[256];
}。
```

流森林 OSM 602 还可以用于使用函数名诸如"IOCTL_SF_TREERENAME"来重命名已有的流树。这一函数可以接收例如用 C 语言按如下定义的输入缓冲器作为输入:

```
struct ik_sf_treerename {
```

```
    char IdentifierBeforeRename [256];
    char IdentifierAfterRename [256];
}。
```

流森林 OSM 602 还可以用于使用函数名诸如"IOCTL_SF_TREEERASE"来删除已有的流树。这一函数可以接收例如用 C 语言按如下定义的输入缓冲器作为输入:

```
struct ik_sf_treeerase {
    char IdentifierToErase[256];
}。
```

流森林 OSM 602 可以使用一个或者多个如下的数据结构。此处被称为"超级块（super block）"的数据结构可以被用于代表文件系统的初始"i 节点（inode）"。在本上下文中的"i 节点"可以指该文件系统的文件 节点。超级块操作列表可以是函数的编组，该函数被用于操纵该超级块。i 节点记录数据结构可以被用于代表该文件系统的每个文件节点。i 节点操作列表可以是函数的编组，该函数被用于操纵 i 节点。文件记录数据结构可以被用于代表文件系统中的抽象文件。文件操作列表可以被用于支持文件操作。与操作列表操作组合在一起的所有这些数据结构可以被用于代表文件系统。

流树类消息可以向几个该数据结构发出。流树类消息可以例如对于 i 节点操作建立、查找、建立目录("mkdir")、删除目录("rmdir")、和重命名目录而被发出。流树类消息可以对于各种文件操作，诸如打开、读、写、查找、和关闭而被发出。

流森林 OSM 602 可以使用标识符"i2Ofs"将所包含的文件系统注册到例如 Linux 内核中。例如，对于根树圆锥容器，流森林 OSM 602 可以使用 NULL 标识符。例如，对于当前的树圆锥容器，流森林 OSM 602 可以使用标识符'.'。例如，对于父树圆锥容器，流森林 OSM 602 可以使用标识符'..'。

系统 600 可以包含对于一个或者多个如下模块的系统描述:

1.内核的 MODULE_AUTHOR 描述;

2."I2O Filesystem Offload Driver（I2O 文件系统卸载驱动器）"的 MODULE_DESCRIPTION;

3.module_init( )，它可以注册该文件系统;

4.module_exit( )，它可以除去该文件系统的注册;

5.DECLARE_FSTYPE 模块，它可以具有参数 i2ofs_type i2ofs、读超级操作以及 0;

6.OSM 句柄管理器模块，它可以具有第一句柄零，它按照每使用的句柄增加 1;

a.内部软件接口 AcquireHandle( )，它可以检索未用过的句柄;和

b.内部软件接口 ReleaseHandle(int)，它可以将当前使用的句柄释放到空闲池中。

系统 600 还可以提供函数 VFS i2ofs_read_super(struct super_block *sb, void *options, int silent)。这一函数可以接收如表 1 中所定义的输入.

表 1

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| Struct super_block* | Sb | 这一输入可以指定该超级块应该被设置的位置。 |
| Void* | Options | 这一输入可以指定通过 Mount 所传递的选项。 |
| int | Silent | 这一输入可以指定 mount 操作是否是寂静的。 |

这一函数将该超级块结构中的值设置为 0。以下变量可以按如下被设置:

sb->s_blocksize=512。

sb->s_blocksize_bits=10。

sb->S_s_magic=I2OFS_SUPER_MAGIC

sb->s_op=在该 OSM 中所定义的 super_operations 结构。

该函数可以使用被称为 get_empty_inode( )的模块建立新的 i 节点。

该 i 节点可以被设置为零，各种关联的变量具有如下的设置:

inode i_uid =20。

inode i_gid =20。

i 节点操作可以被设置成描述该 i 节点操作列表的指针。

inode i_fop 可以被设置成描述该文件操作列表的点。

inode I_mode 可以被设置成 S_IFDIR|S_IRUGO|S_IXUGO。

i 节点可以被插入到 i 节点哈希表中。

Sb->s_root 可以被设置成该根 i 节点的 d_alloc_root( )。

VFS 606 中的每个文件系统可以具有至少一个超级块，该超级块包含有关该文件系统的足够信息以启动该文件系统的活动的。该超级块可以用 C 结构 struct super_block 如下地加以实现:

```
struct super_block {
    struct list_head s_list;
    kdev_t s_dev;
    unsigned long s_blocksize;
    unsigned long s_blocksize_bits;
    unsigned char s_lock;
    unsigned char s_dirt;
    struct file_system_type *s_type;
    struct super_operations *s_op;
    struct dquot_operations *dq_op;
    unsigned long s_flags;
    unsigned long s_magic;
    struct dentry *s_root;
    wait_queue_head_t s_wait;
    struct inode *s_ibasket;
    short int s_ibasket_count;
    short int s_ibasket_max;
    struct list_head s_dirty;
    struct list_head s_files;
    struct block_device *s_bdev;
}。
```

变量 super_block->s_op 可以包含如下 C 函数，并且提供对该超级块的访问。

```
struct super_operations {
 void(*read_inode) (struct inode *);
 void(*write_inode) (struct inode *)
```

```
  void(*put_inode) (struct inode *);
  void(*delete_inode) (struct inode *);
  void(*put_super) (struct super_block *);
  void(*write_super) (struct super_block *);
  void(*statfs)(struct super_block*, s6truct statfs *);
  int(*remount_fs) (struct super_block *, int *, char *);
  void(*clear_inode) (struct inode);
  void(*umount_begin) (struct super_block *);
}。
```

i 节点接口的例子可以如下:

```
struct inode {
    struct list_head i_hash;
    struct list_head i_list;
    struct list_head i_dentry;
    unsigned long i_ino;
    unsigned int i_count;
    kdev_t i_dev;
    umode_t i_mode;
    nlink_t i_nlink;
    uid_t i_uid;
    gid_t i_gid;
    kdev_t i_rdev;
    loff_t I_size;
    time_t I_atime;
    time_t I_mtime;
    time_t I_ctime;
    unsigned long I_blksize;
    unsigned long I_blocks;
    unsigned long I_version;
    struct semaphore I_sem;
    struct semaphore I_zombie;
    struct inode_operations *I_op;
    struct file_operations *I_fop;
    struct super_block *I_sb;
    wait_queue_head_t I_wait;
    struct file_lock *I_flock;
    struct address_space *I_mapping;
    struct address_space I_data;
```

```
    struct dquot *I_dquot[MAXQUOTAS];
    struct pipe_inode_info *I_pipe;
    Struct block_dev ice I_bdev;
    Unsigned long I_state;
    Unsigned int I_flags;
    Unsigned char I_sock;
    Atomic_t I_writecount;
    Unsigned int I_attr_flags;
    __u32 I_generation;
  }。
```

变量 inode->i_op 可以包含如下 C 函数，并提供对该超级块的方法访问，如下：

```
  struct inode_operations {
   struct file_operations *default_file_ops;
   int (*create) (struct inode *, const char *, int, int, struct inode
**);
   struct dentry * (*lookup) (struct inode *,struct dentry *)
   int (*link) (struct dentry *, struct inode *, struct dentry *);
   int (*unlink) (struct inode *, struct dentry *);
   int (*symlink) (struct inode *, struct dentry *, const char *);
   int (*mkdir) (struct inode *, struct dentry *, int);
   int (*rmdir) (struct inode *, struct dentry *);
   int (*mknod) (struct inode *, struct dentry *, int, int);
   int (*rename)(struct inode *, struct dentry *, struct inode *,struct
   dentry *);
   int (*readlink) (struct dentry *, char *, int);
   struct dentry * (*follow_link) (struct dentry *, struct dentry *,
   unsigned int);
   void (*truncate) (struct inode *);
   int (*permission) (struct inode *, int);
   int (*revalidate) (struct dentry *);
   int (*setattr) (struct dentry *, struct iattr *);
   int (*getattr) (struct dentry *, struct iattr *);
  }。
```

系统 600 还提供函数 create(struct inode *, const char *, int, int, struct inode **)。这一函数可以接受在表 2 中所阐述的输入。

表 2

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| struct inode* | ParentDirectory | 这一输入可以指定建立操作的输入目录。 |
| const char* | NewName | 这一输入可以指定新的文件系统对象的名字。 |
| int | NewSize | 这一输入可以指定该对象的新的大小。 |
| Int | NewMode | 这一输入可以指定该对象的新的模式。 |
| Struct inode** | NewInode | 这一输入可以指定所建立的对象的新的 i 节点。 |

这一函数可以使用 get_empty_inode ( ) 来建立新的 i 节点。该函数可以通过经由 dentry 关系将 const char *变量附加到该新的 i 节点上而初始化该新的 i 节点。该函数可以使用所指定的大小来初始化新的 i 节点结构。该函数可以使用所指定的模式初始化该新的 i 节点结构。该函数可以将 StreamConeCreate 消息发送到该流树 ISM。该消息可以按照如下加以配置:

1. HANDLE 可以被从该 OSM 句柄管理器中加以检索;

2. TYPE 可以被设置为 1，指示一个文件; 和

3. SGL 可以被设置成 NewName。

NewInode 的参考解除可以使用该新的 i 节点结构加以设置。

系统 600 还可以提供函数 struct dentry *lookup (struct inode *, struct dentry *)。这一函数可以接受在表 3 中所阐述的输入。

表 3

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| struct inode* | ParentDirectory | 这一输入可以指定查找操作的输入目录。 |

这一函数可以将 StreamConeIdentify 消息发送到流树 ISM 618。该消息可以按如下配置:

1. PARENTHANDLE 可以被设置成 ParentDirectory inode 局部数据之内所标识的句柄;

2. CHILDHANDLE 可以被设置成从该 OSM 句柄管理器中所检索的句柄;

3. ATTRIBUTES 可以被设置成 STREAMCONECREATE_QUERY; 和

4. SGL 可以被设置成 Name。

该函数可以使用 get_empty_inode ()建立新的 i 节点。该函数可以通过经由 dentry 关系将 const char *变量附加到该新的 i 节点来初始化它。该函数可以将 StreamConeGetInformation 消息发送到该流树 ISM。该消息可以按如下配置:

1. HANDLE 可以按如上被设置成 CHILDHANDLE; 和

2. SGL 可以被设置成局部数据结构类型的信息结果块。该函数可以根据该信息结果块设置该 i 节点的值。该函数可以将该 NewINODE 变量的参考解除设置到已经被建立的 i 节点。

系统 600 可以提供函数 mkdir(struct inode *, struct dentry *, int)。这一函数可以接受在表 4 中所阐述的输入。

表 4

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| struct inode* | ParentDirectory | 这一输入可以指定 mkdir 操作的输入目录。 |
| Struct dentry* | Name | 这一输入可以指定要建立的对象的名字。 |
| int | Mode | 这一输入可以指定该新的目录的模式。 |

该函数可以发送 StreamConeCreate 消息以便建立目录。该消息可以按如下配置:

1. HANDLE 可以被设置成在 i 节点结构输入 ParentDirectory 中所

标识的句柄;

2.TYPE 可以被设置成 2; 和

3.SGL 可以被设置成包含该实际名字的输入 Name。

系统 600 可以提供函数 rmdir (struct inode *, struct dentry *)。这一函数可以接受表 5 所阐述的输入。

表 5

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| struct inode* | ParentDirectory | 这一输入可以指定查找操作的输入目录 。 |
| Struct dentry* | Name | 这一输入可以指定要删除的对象的名字。 |
| int | Length | 这一输入可以指定 Name 的字符长度。 |

这一函数可以发送 StreamConeErase 消息以便删除目录。该消息可以按如下配置:

1.HANDLE 可以被设置成在该 i 节点结构 输入 ParentDirectory 中所标识的句柄; 和

2.SGL 可以被设置成该输入 Name。

系统 600 还可以提供函数 rename (struct inode *,struct dentry *, struct inode *, struct dentry *)。这一函数可以接受表 6 所阐述的输入。

表 6

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| struct inode* | OldDir | 这一输入可以指定要重命名的文件的目录。 |
| Struct dentry* | OldName | 这一输入可以指定要重命名的对象的名字。 |
| struct inode* | NewDir | 这一输入可以指定该对象的新的目录。 |

| | | |
|---|---|---|
| struct dentry* | NewName | 这一输入可以指定该对象的新的名字。 |

这一函数可以将 StreamConeIdentify 消息发送到该 ISM。该消息可以按如下配置:

1. PARENTHANDLE 可以根据 OldDir i 节点内部存储被设置用于 OSM 句柄;

2. CHILDHANDLE 可以被设置成从该 OSM 句柄管理器中检索的新的句柄;

3. ATTRIBUTES 可以被设置成 STREAMCONECREATE_QUERY; 和

4. SGL 可以被设置成 OldName。

该函数可以将 StreamConeRename 消息发送到该 ISM。该消息可以按如下配置:

1. HANDLE 可以被设置成 StreamConeIdentify 的 CHILDHANDLE;

2. NEWPARENT 可以为 OSM 句柄被设置到 NewDir i 节点内部存储; 和

3. SGL 可以被设置成 NewName。

该函数可以将 StreamConeClose 消息发送到该 ISM。该消息可以按如下配置: HANDLE 可以根据 StreamConeRename 消息被设置成以前的 HANDLE。

系统 600 还可以提供函数 truncate (struct inode *)。这一函数可以接受表 7 所阐述的输入。

表 7

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| struct inode* | File | 这一输入可以指定要截取的文件。 |

该函数可以将 StreamConeResize 消息发送到该 ISM。该消息可以按如下配置:

1. HANDLE 可以被从该 i 节点内部 OSM 句柄存储中检索到; 和

2. SIZE 可以被从该 i 节点变量 i_size 中检索到。

VFS 606 中的每个文件可以具有至少一个超级块，该超级块可以包含关于该文件系统的足够信息以启动该文件系统的活动。这一超级块的细节用 C 结构 struct super_block 给出，诸如前面所描述的超级块。方法变量，f_op，可以提供对文件操作的访问。struct super_operate *s_op 函数可以提供对根 i 节点的访问，它对于该 OSM 可能是需要的。

```
struct file {
    struct list_head f_lst;
    struct dentry *f_dentry;
    struct file_operations *f_op;
    atomic_t f_count;
    unsigned int f_flags;
    mode_t f_mode;
    loff_t f_pos;
    unsigned long f_reada;
    unsigned long f_ramax;
    unsigned long f_raend;
    unsigned long f_ralen;
    unsigned long f_rawin;
    struct fown_struct f_owner;
    unsigned int f_uid;
    unsigned int f_gid;
    int f_error;
    unsigned long f_version;
}。
```

系统 600 可以提供以下描述的一个或者多个文件操作。变量 f->s_op 可以包含以下的 C 函数，并且可以提供对该超级块的访问。

```
struct file_operations {
 loff_t (*llseek) (struct file *, off_t, int)
 ssize_t (*read) (struct file *, char *, size_t, loff t *);
 ssize_t (*write) (struct file *, const char *, size_t, loff_t *);
 int (*readdir) (struct file *, void *, filldir_t);
 u_int (*poll) (struct file *, struct poll_table_struct *);
 int (*ioctl) (struct inode *, struct file *, unsigned int,
 unsigned long);
 int (*mmap) (struct file *, struct vm_area_struct *);
```

```
int (*open) (struct inode *, struct file *);
int (*release) (struct inode *, struct file *);
int (*fsync) (struct inode *, struct dentry *);
int (*fasynch) (int, struct file *, int);
int (*lock) (struct file *, int, struct file_lock *);
ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff_t *);
ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff_t *);
}。
```

系统 600 可以提供函数 llseek (struct file * ,off_t, int)。这一函数可以接受表 8 中所阐述的输入。

表 8

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| struct file* | File | 这一输入可以指定要查找的文件指针。 |
| off_t | Offset | 这一输入可以指定从源到所查找的偏移。 |
| int | Origin | 这一输入可以指定源。 |

这一函数可以将 StreamConeSeek 消息发送到该 ISM。该消息可以按如下配置:

1. HANDLE 可以为 OSM 句柄根据节点内部存储而加以设置; 和

2. NEWPOSITION 可以被设置成根据 Offset 和 Origin 的计算而获得的位置。

系统 600 可以提供函数 read (struct file *, char *, size_t, loff_t)。这一函数可以接受表 9 所阐述的输入。

表 9

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| struct file* | File | 这一输入可以指定要读 |

| | | |
|---|---|---|
| | | 取的文件指针。 |
| char* | Buffer | 这一输入可以指定要读入到的缓冲器。 |
| Size_t | Size | 这一输入可以指定要读取的字节大小。 |
| loff_t* | SeekChange | 这一输入指定读取了多少数据。 |

这一函数可以将 StreamConeRead 消息发送到该 ISM。该消息可以按如下配置:

1. HANDLE 可以为 OSM 句柄根据节点内部存储而加以设置; 和

2. SGL 可以被设置成 Buffer 和 Size。

系统 600 可以提供函数 write (struct file *, const char *, size_t, loff_t *)。这一函数可以接受表 10 所阐述的输入。

表 10

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| struct file* | File | 这一输入可以指定要读取的文件指针。 |
| Const char* | Buffer | 这一输入可以指定要写入的缓冲器。 |
| Size_t | Size | 这一输入可以指定要读取的字节大小。 |
| Loff_t | SeekChange | 这一输入指定写入了多少数据。 |

这一函数可以将 StreamConeWrite 消息发送到该 ISM。该消息可以按如下配置:

1. HANDLE 可以为 OSM 句柄根据节点内部存储而加以设置; 和

2. SGL 可以被设置成 Buffer 和 Size。

系统 600 可以提供函数 readdir (struct file *, void *, filldir_t)。这一函数可以接受表 11 所阐述的输入。

表 11

| 变量类型 | 变量标识符 | 描述 |
| --- | --- | --- |
| struct file * | File | 这一输入可以指定要读取的文件指针。 |

这一函数可以将 StreamConeEnumerate 消息发送到该 ISM。该消息可以按如下配置:

1. HANDLE 可以为 OSM 句柄根据节点内部存储而加以设置;

2. ENUMERATOR 被设置为 Count;

3. SGL 可以被设置成 Entry 的文件名缓冲器;

4. SGL 的大小可以被设置成 255。

系统 600 还可以提供如下的 dentry 接口:

```
struct dentry {
    int d_count;
    unsigned int d_flags;
    struct inode * d_inode;
    struct dentry *d_parent;
    struct dentry *d_mounts;
    struct dentry *d_covers;
    struct list_head d_ash;
    struct list_head d_lru;
    struct list_head d_child;
    struct list_head d_subdirs;
    struct list_head d_alias;
    struct qstr d_name;
    unsigned long d_time;
    struct dentry_operations *d_op;
    struct super_block *d_sb;
    unsigned long d_reftime;
    void *d_fsdata;
    unsigned char d_iname[DNAME_INLINE_LEN];
}。
```

流树 ISM 618 可以支持一个或者多个流树类消息，如此处所定义的。流树 618 可以运行并支持来自流树 OSM，诸如流树 OSM 608 的消息。流森林 ISM 616 可以支持流森林类消息，如此处所定义的。流森

林 ISM 616 可以运行并支持来自流森林 OSM，诸如流森林 OSM 602 的消息。

流森林 ISM 616 可以包括用户界面。该用户界面可以包括用户屏幕，该用户屏幕可以在初始化时显示一个或者多个流树。用户可能能够建立新的流树。当建立新的流树时，可以向该用户展示一个屏幕，来询问大小和标识。如果流树被建立了，则流树类对象就可以被建立。用户还可能能够删除流树。该用户界面可以提供用于对删除流树进行确认。流森林消息可以被处理，并且树对象被适当地建立 。

系统 600 可以提供消息 StreamConeCreate。这一消息可以被用于建立新的流圆锥，该新的流圆锥将存储信息，并且可以接受表 12 中所阐述的输入。

表 12

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| U32 | HANDLE | 这一输入可以指定应该被关闭的句柄。 |
| U32 | TYPE | 这一输入可以指定要建立的类型。1<br>指示文件，2 指示目录。 |
| SGL | SGL | 指定该标识符。 |

系统 600 还可以提供消息 StreamConeEnumerate。这一消息可以被用于列出在流圆锥容器中的一个或者多个流圆锥，并且可以接受表 13 中所阐述的输入。

表 13

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| U32 | HANDLE | 这一输入可以指定应该被关闭的句柄。 |
| U32<br>* | ENUMERATOR | 这一输入可以指定基于零的索引，指示哪个条目应该被枚举到该 SGL 中。 |

| | | |
|---|---|---|
| SGL | SGL | 指定所枚举的流圆锥标识符的位置。 |

系统 600 可以从该句柄管理器中检索涉及到 HANDLE 的 i 节点。当枚举型量（enumberator）被设置为 0 时，"ext2fs"库调用 dir_iterate( )可以与涉及到作为父的句柄的 i 节点一起使用以便在该"ext2"文件系统上生成流圆锥标识符的列表。对应于 ENUMERATOR 索引的列表条目可以被拷贝到 SGL 中。

系统 600 可以提供消息 StreamConeErase。这一消息被用于删除流圆锥标识符，并且可以接受在表 14 中所阐述的输入。

表 14

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| U32 | HANDLE | 这一输入可以指定应该被关闭的句柄。 |
| SGL | SGL | 这一输入可以指定应该被删除的标识符的名字。 |

系统 600 可以提供消息 StreamConeGetInformation。这一消息可以被用于检索关于该流圆锥的信息，并且可以接受表 15 中所阐述的输入。

表 15

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| U32 | HANDLE | 这一输入可以指定信息应该被从其检索的句柄。 |
| SGL | SGL | 这一输入可以指定应该被删除的标识符的名字。 |

系统 600 可以提供消息 StreamConeIdentify。这一消息可以被用于将句柄标识符映射成字符串标识符，并且可以接受表 16 所阐述的输入。

表 16

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| U32 | PARENTHANDLE | 这一输入可以指定父句柄流圆锥容器。 |
| U32 | CHILDHANDLE | 这一输入可以指定应该被映射成流圆锥标识符的子句柄。 |
| U32 | ATTRIBUTES | 这一输入可以指定所标识的流圆锥的属性。 |
| SGL | SGL | 这一输入可以指定该标识符的名字。 |

系统 600 可以提供消息 streamConeLock。这一消息可以被用于加锁文件的字节范围，并且可以接受表 17 中所阐述的输入。

表 17

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| U32 | HANDLE | 这一输入可以指定父句柄流圆锥容器。 |
| U64 | INDEX | 这一输入可以指定应该被加锁的字节索引。 |
| U64 | SIZE | 这一输入可以指定应该被加锁的字节大小。 |

系统 600 可以提供消息 StreamConeRead。这一消息可以被用于从流圆锥中读取块，并且可以接受表 18 中所阐述的输入。

表 18

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| U32 | HANDLE | 这一输入可以指定父句柄流圆锥容器。 |
| U64 | SIZE | 这一输入可以指定要读 |

| | | |
|---|---|---|
| | | 取的块的大小。 |
| SGL | SGL | 这一输入可以指定该文件应该被存储到存储器中的哪个地方。 |

系统 600 还可以提供消息 StreamConeRelease。这一消息可以被用于关闭所指定的句柄的标识，并且可以接受表 19 中所阐述的输入。

表 19

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| U32 | HANDLE | 这一输入可以指定应该被关闭的句柄。 |

这一函数可以不链接来自该 i 节点标识符的 HANDLE。

系统 600 可以提供消息 StreamConeRename。这一消息可以被用于重命名流圆锥，并且可以接受在表 20 中所阐述的输入。

表 20

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| U32 | HANDLE | 这一输入可以指定应该被重命名的句柄。 |
| U32 | NEWPARENT | 这一输入可以指定当前或者新的父句柄。 |
| SGL | SGL | 这一输入可以指定该新的 SGL 的名字。 |

系统 600 还可以提供消息 StreamConeResize。这一消息被用于调整流圆锥的大小，并且可以接受表 21 中所阐述的输入。

表 21

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| U32 | HANDLE | 这一输入可以指定应该被重命名的句柄。 |
| U64 | SIZE | 这一输入可以指定该流圆锥的新的大小。 |

系统 600 可以提供消息 StreamConeSeek。这一消息被用于改变流圆锥的位置，并且可以接受表 22 中所阐述的输入。

表 22

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| U32 | HANDLE | 这一输入可以指定应该被重命名的句柄。 |
| U64 | NEWPOSITION | 这一输入可以指定该流圆锥的新位置。 |

系统 600 可以提供消息 StreamConeGetInformation。这一消息可以被用于设置关于该流圆锥的信息，并且可以接受表 23 中所阐述的输入。

表 23

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| U32 | HANDLE | 这一输入可以指定应该有信息要被设置的句柄。 |
| SGL | SGL | 这一输入可以指定信息被设置的块。 |

系统 600 可以提供消息 StreamConeUnlock。这一消息可以被用于解锁以前设置的字节加锁范围，并且可以接受表 24 中所阐述的输入。

表 24

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| U32 | HANDLE | 这一输入可以指定应该被解锁的句柄。 |
| U64 | INDEX | 这一输入可以指定要加以解锁的字节范围的开始字节。 |
| U64 | BYTECOUNT | 这一输入可以指定加以解锁的字节计数。 |

系统 600 可以提供消息 StreamConeWrite。这一消息可以被用于将块写入到流圆锥，并且可以接受表 25 中所阐述的输入。

表 25

| 变量类型 | 变量标识符 | 描述 |
|---|---|---|
| U32 | HANDLE | 这一输入可以指定应该被写入的句柄。 |
| U64 | BYTECOUNT | 这一输入可以指定应该被写入的字节计数。 |
| SGL | SGL | 这一输入可以指定要被写入的块。 |

尽管如在此所描述的那样对本发明的实施例的某些特征进行了说明，但是现在，许多修改、替换、改变以及等价内容对于本领域普通技术人员而言是会想到的。因此，应该理解的是，所附的权利要求书旨在涵盖落在本发明的实施例的实际精神之中的所有这种修改和改变。

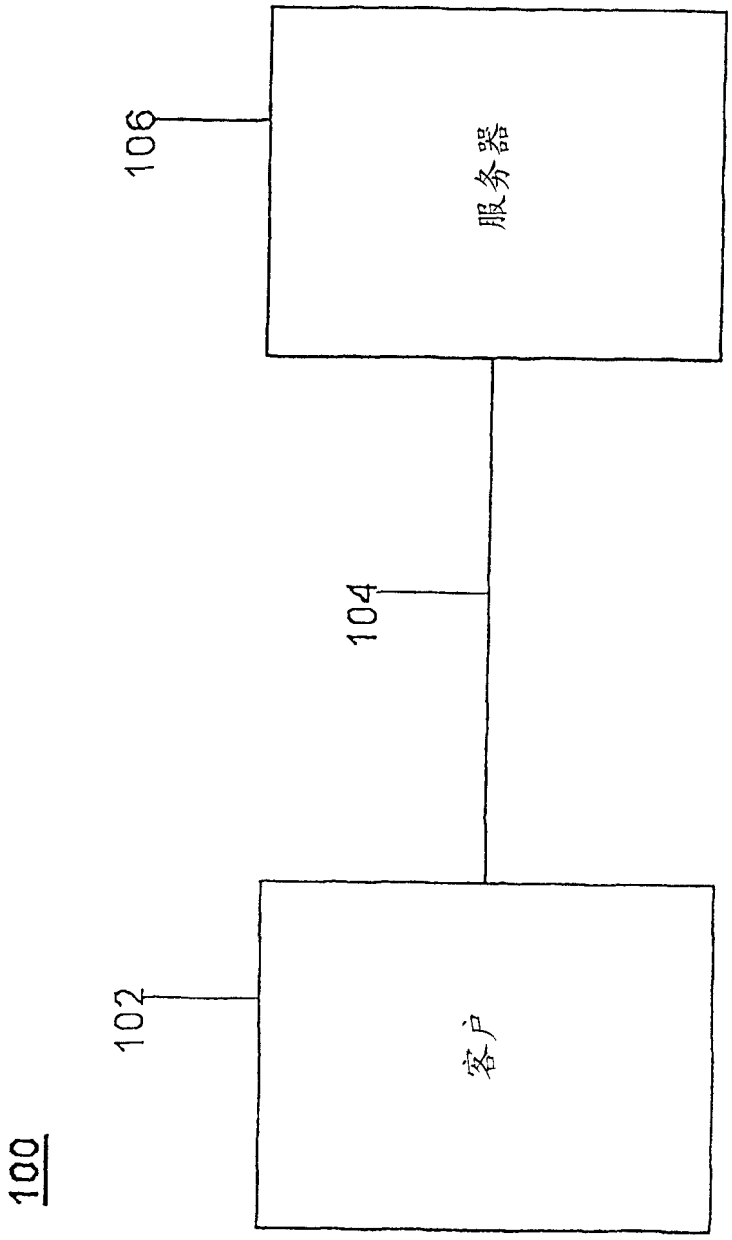



图 1

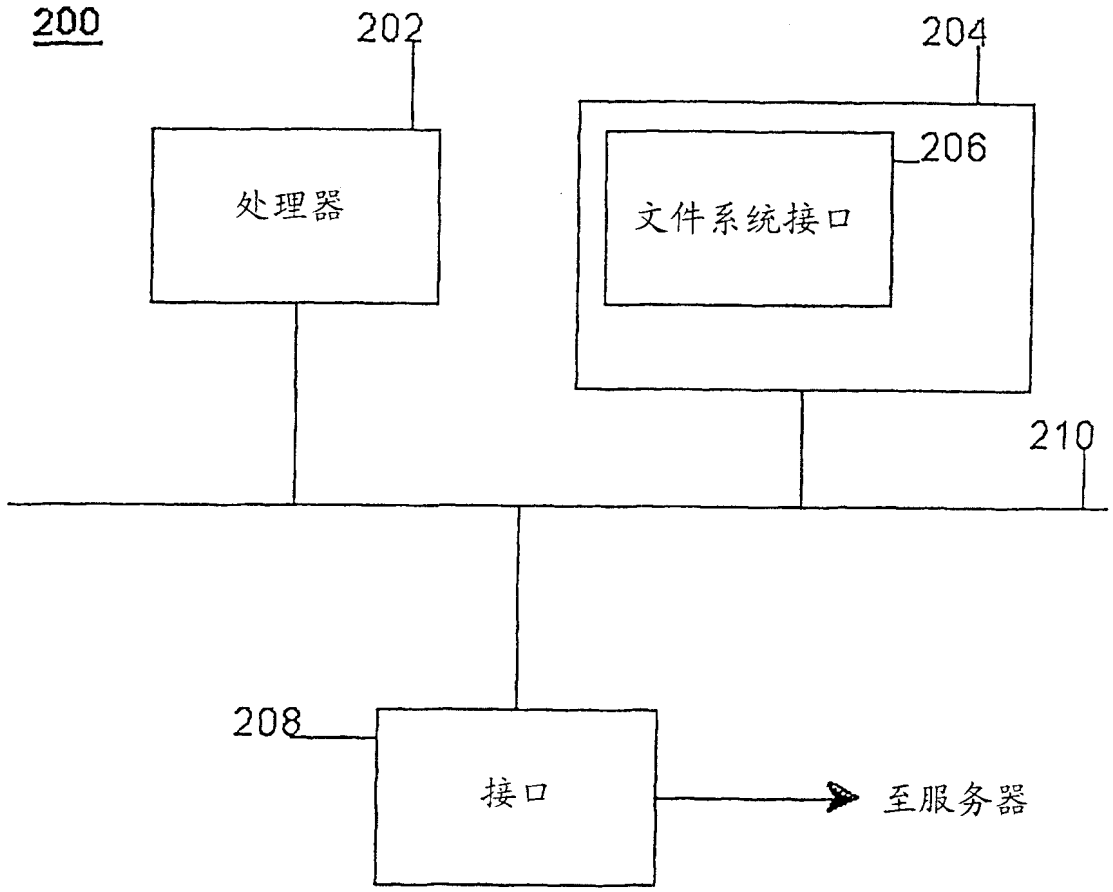



图 2

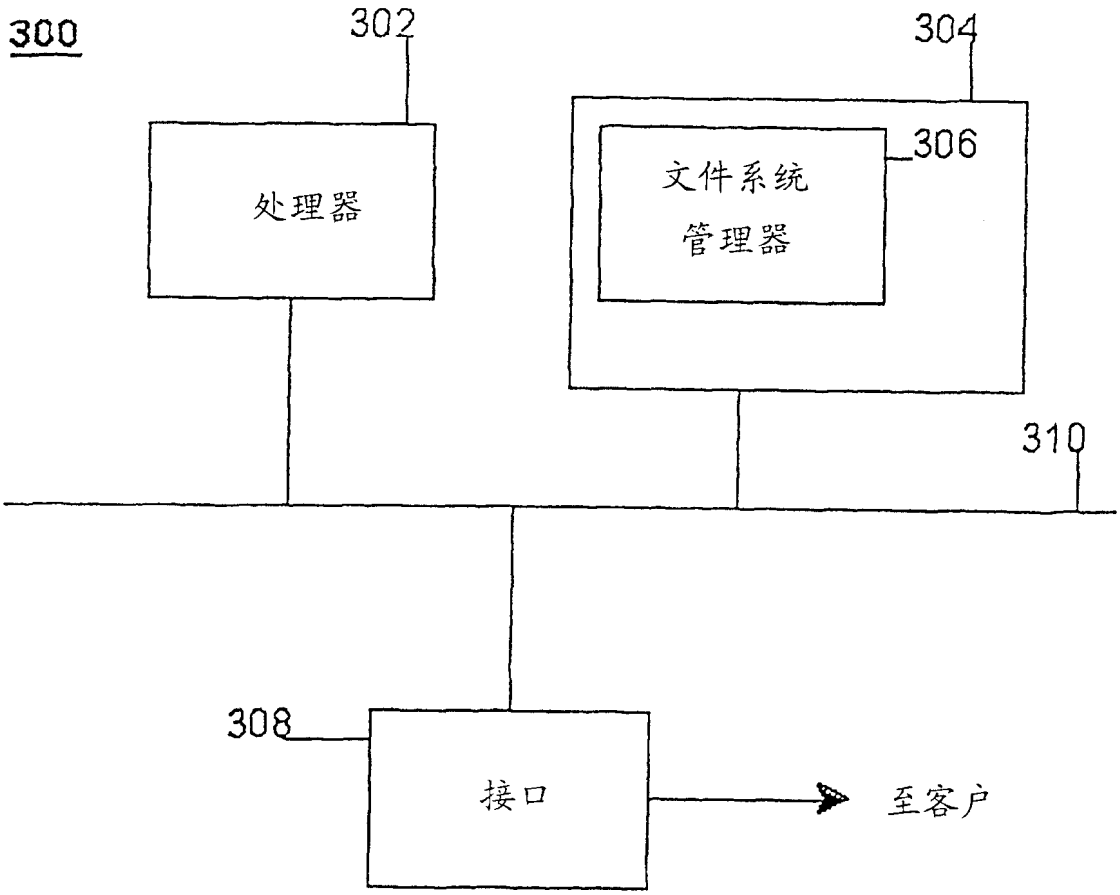



图 3

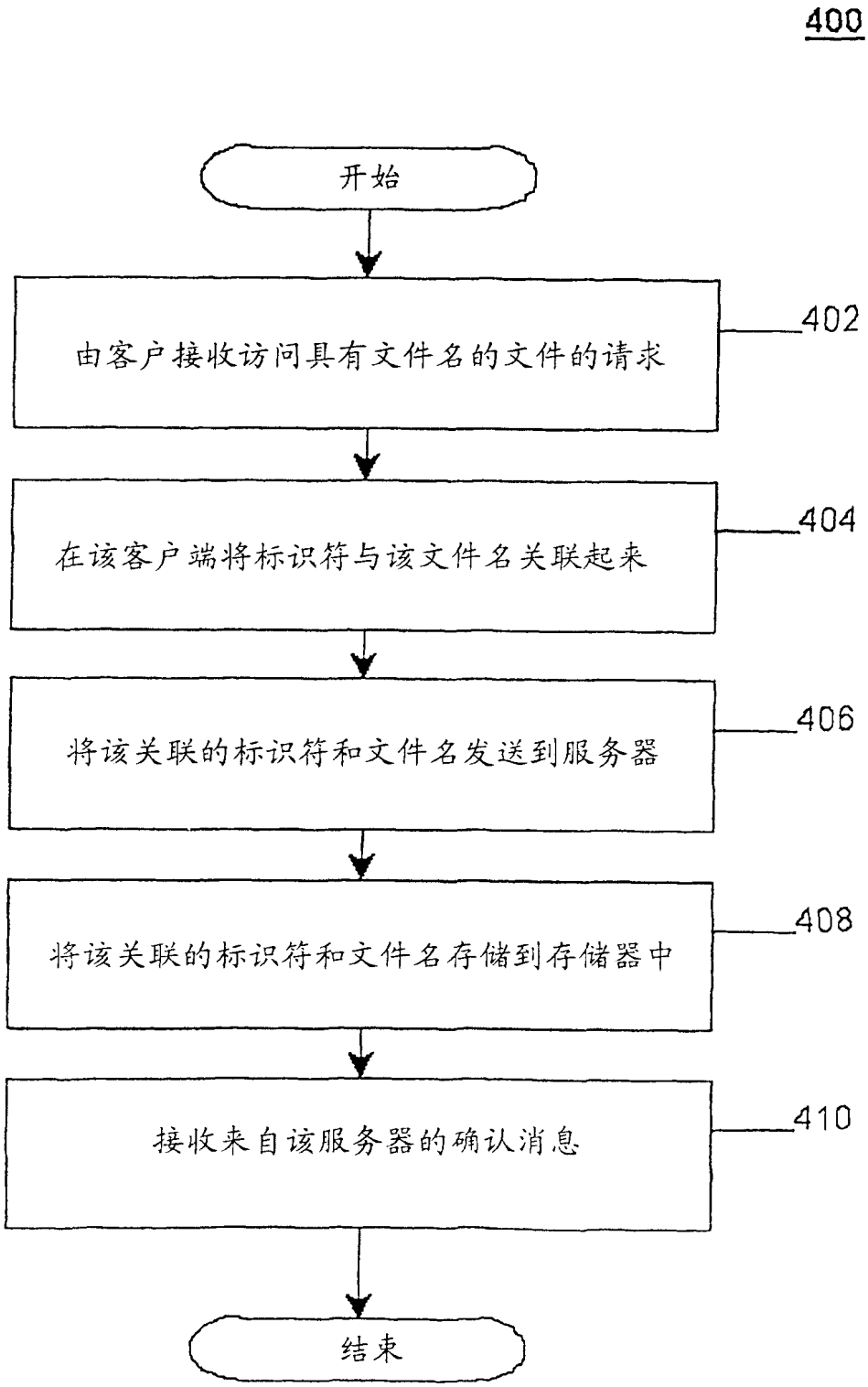

400
开始
由客户接收访问具有文件名的文件的请求
402
在该客户端将标识符与该文件名关联起来
404
将该关联的标识符和文件名发送到服务器
406
将该关联的标识符和文件名存储到存储器中
408
接收来自该服务器的确认消息
410
结束

图 4

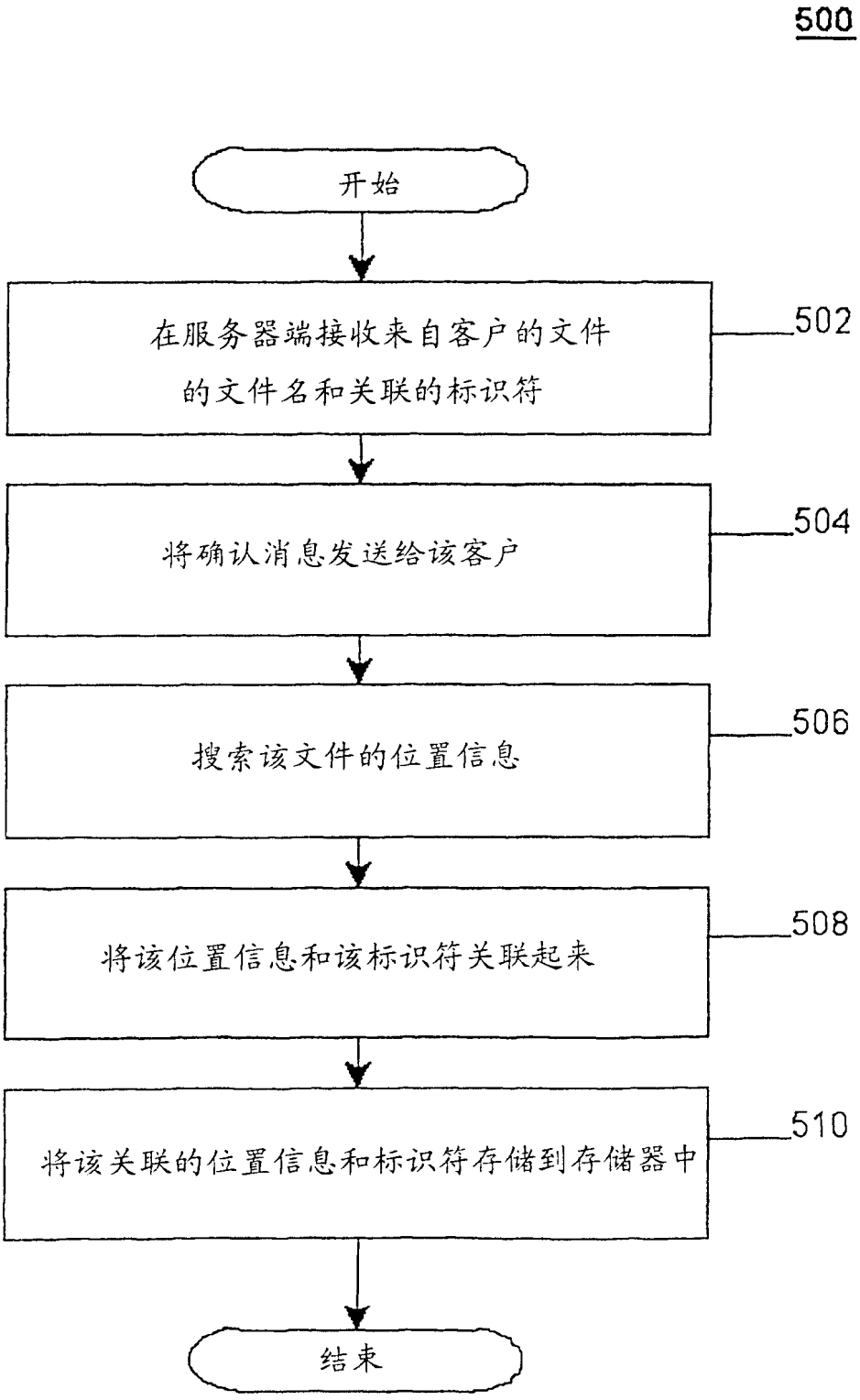

500
开始
在服务器端接收来自客户的文件的文件名和关联的标识符
502
将确认消息发送给该客户
504
搜索该文件的位置信息
506
将该位置信息和该标识符关联起来
508
将该关联的位置信息和标识符存储到存储器中
510
结束

图 5

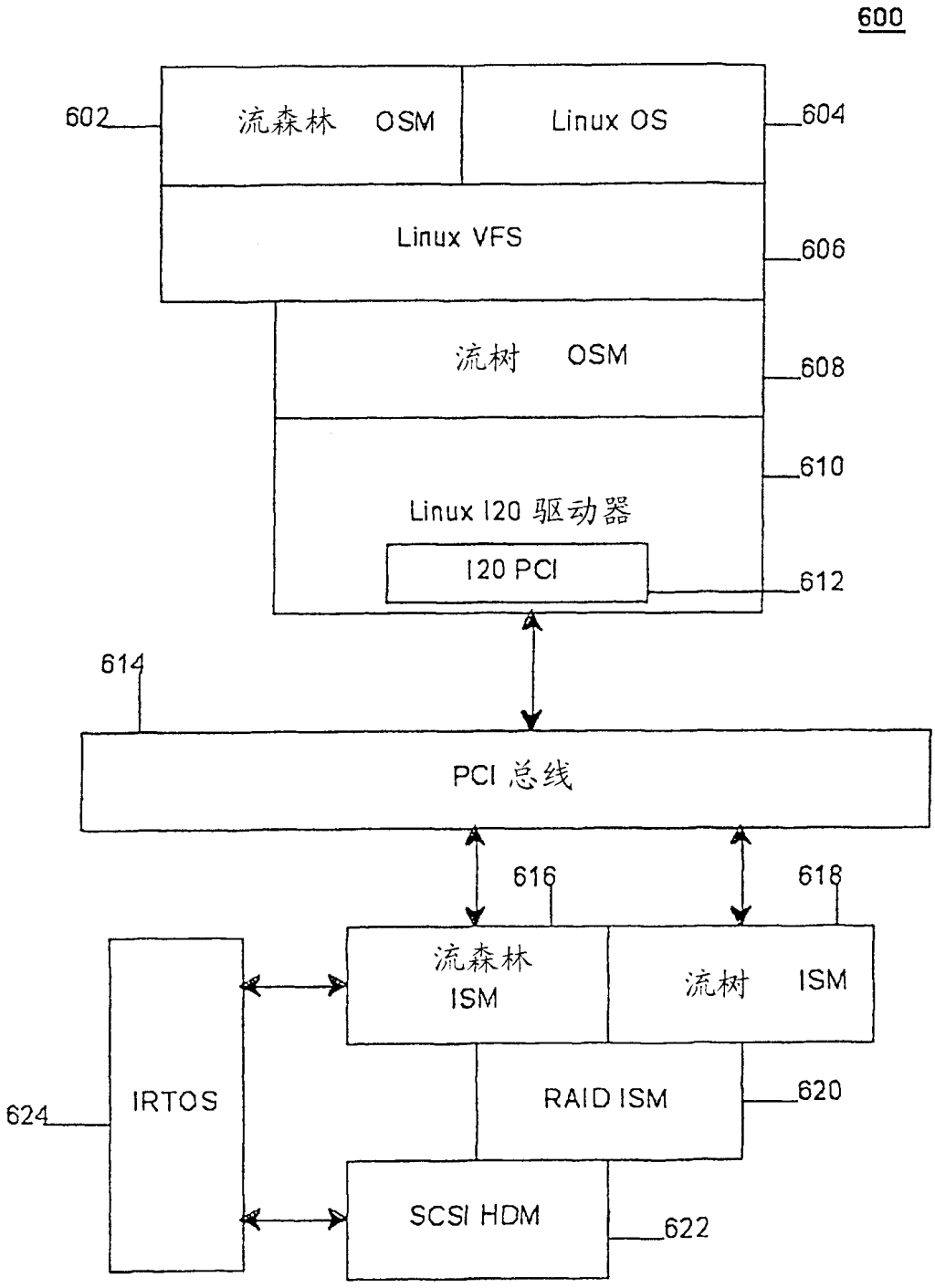

600
602
流森林 OSM
Linux OS
604
Linux VFS
606
流树 OSM
608
610
Linux I20 驱动器
I20 PCI
612
614
PCI 总线
616
618
流森林 ISM
流树 ISM
IRTOS
624
RAID ISM
620
SCSI HDM
622

图 6