(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5019669号
(P5019669)
(45)発行日　平成24年9月5日(2012.9.5)　(24)登録日　平成24年6月22日(2012.6.22)

(51)Int.Cl.　F I
G06F 12/00 (2006.01)　G06F 12/00 545F

請求項の数 64 (全 31 頁)

(21)出願番号	特願2000-569312 (P2000-569312)
(86)(22)出願日	平成11年8月20日 (1999.8.20)
(65)公表番号	特表2002-524793 (P2002-524793A)
(43)公表日	平成14年8月6日 (2002.8.6)
(86)国際出願番号	PCT/US1999/019243
(87)国際公開番号	WO2000/014632
(87)国際公開日	平成12年3月16日 (2000.3.16)
審査請求日	平成18年8月18日 (2006.8.18)
審判番号	不服2010-2333 (P2010-2333/J1)
審判請求日	平成22年2月3日 (2010.2.3)
(31)優先権主張番号	60/099,425
(32)優先日	平成10年9月8日 (1998.9.8)
(33)優先権主張国	米国 (US)
(31)優先権主張番号	09/323,333
(32)優先日	平成11年6月1日 (1999.6.1)
(33)優先権主張国	米国 (US)

(73)特許権者　506379910
デイライト　ケミカル　インフォメーション　システムズインコーポレイテッド
ＤＡＹＬＩＧＨＴ　ＣＨＥＭＩＣＡＬ　ＩＮＦＯＲＭＡＴＩＯＮ　ＳＹＳＴＥＭＳ，ＩＮＣ．
アメリカ合衆国　９２６５６　カリフォルニア州　アリソ　ビエホ　バンティス　１２０　スイート　５５０
(74)代理人　100068755
弁理士　恩田　博宣
(74)代理人　100105957
弁理士　恩田　誠
(74)代理人　100142907
弁理士　本田　淳

最終頁に続く

(54)【発明の名称】仮想ネットワーク・ファイル・サーバ

(57)【特許請求の範囲】
【請求項１】
ディジタル・コンピュータ・ネットワークを経由する標準ネットワーク・ファイル・システム・プロトコルにおいてコンピュータのオペレーティング・システムによって行われる遠隔ファイル・システム要求に応じてネットワーク・ファイル・システムの行動をエミュレートするための方法であって、
ａ）前記ネットワークを経由して前記要求を受信するステップと、
ｂ）前記要求を標準ネットワーク・ファイル・システム・プロトコルに従ってデコードするステップと、
ｃ）前記要求に応じて、プラグイン関数を利用してリアルタイムに仮想ファイルを構成するコンテンツ・データを生成するステップであって、前記要求に応じて複数のプラグイン関数から前記プラグイン関数を選択すること、選択された前記プラグイン関数を呼び出して前記コンテンツ・データを生成することを含む、コンテンツ・データを生成するステップと、
ｄ）前記ネットワークを経由して、前記標準ネットワーク・ファイル・システム・プロトコルに従って前記仮想ファイルを前記コンピュータのオペレーティング・システムに対する応答として送信するステップと、
を含む方法。
【請求項２】
請求項１に記載の方法において、前記ステップｃ）において、前記仮想ファイルの前記

コンテンツ・データがアルゴリズム的に生成される方法。
【請求項３】
請求項１に記載の方法において、前記ステップｃ）において、前記仮想ファイルの前記コンテンツ・データが、格納されている物理ファイルを暗号解読することによって生成される方法。
【請求項４】
請求項１に記載の方法において、前記ステップｃ）において、前記仮想ファイルの前記コンテンツ・データが、格納されている物理ファイルを解凍することによって生成される方法。
【請求項５】
請求項１に記載の方法において、前記ステップｂ）、ｃ）およびｄ）において、前記標準ネットワーク・ファイル・システム・プロトコルがＮＦＳを含む方法。
【請求項６】
請求項１に記載の方法において、前記ステップｂ）、ｃ）およびｄ）において、前記標準ネットワーク・ファイル・システム・プロトコルがＳＭＢを含む方法。
【請求項７】
請求項１に記載の方法において、前記ステップｂ）、ｃ）およびｄ）において、前記標準ネットワーク・ファイル・システム・プロトコルがＣＩＦＳを含む方法。
【請求項８】
請求項１に記載の方法において、前記ステップｂ）、ｃ）およびｄ）において、前記標準ネットワーク・ファイル・システム・プロトコルがＡｐｐｌｅｓｈａｒｅを含む方法。
【請求項９】
請求項１に記載の方法において、前記ステップａ）において、前記要求がファイル読取り要求を含み、
前記ステップｃ）において、前記ファイル読取り要求に応じて前記仮想ファイルのコンテンツ・データが、データベース・システムに問い合わせることによって生成されるようになっている方法。
【請求項１０】
請求項９に記載の方法において、前記ステップｃ）において、前記データベース・システムが関係データベース・システムである方法。
【請求項１１】
請求項９に記載の方法において、前記ステップｃ）において、前記データベース・システムがオブジェクト指向データベース・システムである方法。
【請求項１２】
請求項９に記載の方法において、前記ステップａ）において、前記要求がファイル書込み要求を含み、
前記ステップｃ）が、前記要求に応じてデータベース・システムの中にデータを挿入するステップをさらに含む方法。
【請求項１３】
請求項１２に記載の方法において、前記ステップｃ）において、前記データベース・システムが関係データベース・システムである方法。
【請求項１４】
請求項１２に記載の方法において、前記ステップｃ）において、前記データベース・システムがオブジェクト指向データベース・システムである方法。
【請求項１５】
請求項１に記載の方法において、前記ステップａ）において、前記要求がファイル読取り要求を含み、
前記ステップｃ）において、前記要求に対して応答して生成される前記仮想ファイルのコンテンツが、格納されている物理ファイルのファイル・フォーマットを第２のファイル・フォーマットに変換することによって生成されるようになっている方法。

【請求項１６】
請求項１５に記載の方法において、前記ステップｃ）において、前記ファイルのフォーマットがワードプロセッシングのファイル・フォーマットである方法。
【請求項１７】
請求項１５に記載の方法において、前記ステップｃ）において、前記ファイルのフォーマットがデータベースのファイル・フォーマットである方法。
【請求項１８】
請求項１５に記載の方法において、前記ステップｃ）において、前記ファイルのフォーマットが画像またはグラフィックスのファイル・フォーマットである方法。
【請求項１９】
請求項１５に記載の方法において、前記ステップｃ）において、前記ファイルのフォーマットが分子構造のファイル・フォーマットである方法。
【請求項２０】
請求項１５に記載の方法において、前記ステップｃ）において、前記ファイルのフォーマットが生物情報学的シーケンス・ファイル・フォーマットである方法。
【請求項２１】
請求項１５に記載の方法において、前記ステップｃ）において、前記ファイルのフォーマットが生物情報学的データベースのファイル・フォーマットである方法。
【請求項２２】
請求項１５に記載の方法において、前記ステップｃ）において、前記ファイルのフォーマットがバイナリの実行可能なファイル・フォーマットである方法。
【請求項２３】
請求項１５に記載の方法において、前記ステップａ）において、前記要求がファイル書込み要求を含み、
前記ステップｃ）において、前記仮想ファイルの前記コンテンツ・データを異なるフォーマットに変換するステップをさらに含む方法。
【請求項２４】
請求項２３に記載の方法において、前記ステップｃ）において、前記仮想ファイルのコンテンツ・データが、第１のワードプロセッシングのファイル・フォーマットから第２のワードプロセッシングのファイル・フォーマットへ変換されるようになっている方法。
【請求項２５】
請求項２３に記載の方法において、前記ステップｃ）において、前記仮想ファイルのコンテンツ・データが、第１のデータベースのファイル・フォーマットから第２のデータベースのファイル・フォーマットへ変換されるようになっている方法。
【請求項２６】
請求項２３に記載の方法において、前記ステップｃ）において、前記仮想ファイルのコンテンツ・データが、第１のグラフィックスのファイル・フォーマットから第２のグラフィックスのファイル・フォーマットへ変換されるようになっている方法。
【請求項２７】
請求項２３に記載の方法において、前記ステップｃ）において、前記仮想ファイルのコンテンツ・データが第１の分子構造のファイル・フォーマットから第２の分子構造のファイル・フォーマットへ変換されるようになっている方法。
【請求項２８】
請求項２３に記載の方法において、前記ステップｃ）において、前記仮想ファイルのコンテンツ・データが、第１の生物情報学的シーケンス・ファイル・フォーマットから第２の生物情報学的シーケンス・ファイル・フォーマットへ変換されるようになっている方法。
【請求項２９】
請求項２３に記載の方法において、前記ステップｃ）において、前記仮想ファイルのコンテンツ・データが、第１の生物情報学的データベースのファイル・フォーマットから第

２の生物情報学的データベースのファイル・フォーマットへ変換されるようになっている方法。
【請求項３０】
請求項２３に記載の方法において、前記ステップｃ）において、前記仮想ファイルのコンテンツ・データが、第１のバイナリの実行可能なファイル・フォーマットから第２のバイナリの実行可能なファイル・フォーマットへ変換されるようになっている方法。
【請求項３１】
請求項１に記載の方法において、前記ステップａ）、ｂ）およびｃ）において、前記要求がファイルのコンテンツを読み取ることである方法。
【請求項３２】
請求項１に記載の方法において、前記ステップａ）、ｂ）およびｃ）において、前記要求がファイルに対してデータを書き込むことである方法。
【請求項３３】
請求項１に記載の方法において、前記ステップａ）、ｂ）およびｃ）において、前記要求がファイルの長さを求めることである方法。
【請求項３４】
請求項１に記載の方法において、前記ステップａ）、ｂ）およびｃ）において、前記要求がファイルの属性を求めることである方法。
【請求項３５】
請求項１に記載の方法において、前記ステップａ）、ｂ）およびｃ）において、前記要求がファイルの属性を変更することである方法。
【請求項３６】
請求項１に記載の方法において、前記ステップａ）、ｂ）およびｃ）において、前記要求がファイルを生成することである方法。
【請求項３７】
請求項１に記載の方法において、前記ステップａ）、ｂ）およびｃ）において、前記要求がファイルを削除することである方法。
【請求項３８】
請求項１に記載の方法において、前記ステップａ）、ｂ）およびｃ）において、前記要求がファイルの名称を変更することである方法。
【請求項３９】
請求項１に記載の方法において、前記ステップａ）、ｂ）およびｃ）において、前記要求がディレクトリを生成することである方法。
【請求項４０】
請求項１に記載の方法において、前記ステップａ）、ｂ）およびｃ）において、前記要求がディレクトリを削除することである方法。
【請求項４１】
請求項１に記載の方法において、前記ステップａ）、ｂ）およびｃ）において、前記要求が１つのディレクトリの中のファイル名を探索することである方法。
【請求項４２】
請求項１に記載の方法において、前記ステップａ）、ｂ）およびｃ）において、前記要求が１つのディレクトリの中のファイルのリストを求めることである方法。
【請求項４３】
請求項１に記載の方法において、前記ステップｃ）において、前記プラグイン関数が、格納されている物理ファイルを読み取ることによって前記コンテンツ・データを生成するようになっている方法。
【請求項４４】
請求項１に記載の方法において、前記ステップｃ）において、前記プラグイン関数が前記コンテンツ・データをアルゴリズム的に生成するようになっている方法。
【請求項４５】

請求項１に記載の方法において、前記ステップｃにおいて、前記プラグイン関数が、格納されている物理ファイルを暗号解読することによって前記コンテンツ・データを生成するようになっている方法。

【請求項４６】

請求項１に記載の方法において、前記ステップｃ）において、前記プラグイン関数が、格納されている物理ファイルを解凍することによって前記コンテンツ・データを生成するようになっている方法。

【請求項４７】

請求項１に記載の方法において、前記ステップｃ）において、前記プラグイン関数が、前記要求を行っている前記ユーザ、マシンおよびオペレーティング・システムによって変わる前記コンテンツ・データを生成するようになっている方法。

【請求項４８】

請求項１に記載の方法において、前記ステップｃ）において、前記プラグイン関数が、データベース・システムに問い合わせることによって前記コンテンツ・データを生成するようになっている方法。

【請求項４９】

請求項４８に記載の方法において、前記ステップｃ）において、前記データベース・システムが関係データベース・システムである方法。

【請求項５０】

請求項４８に記載の方法において、前記ステップｃ）において、前記データベース・システムがオブジェクト指向データベースである方法。

【請求項５１】

請求項１に記載の方法において、前記ステップｃ）において、前記プラグイン関数が、格納されている物理ファイルの前記ファイル・フォーマットを第２のファイル・フォーマットに変換することによって前記コンテンツ・データを生成するようになっている方法。

【請求項５２】

請求項５１に記載の方法において、前記ファイル・フォーマットがワードプロセッシングのファイル・フォーマットである方法。

【請求項５３】

請求項５１に記載の方法において、前記ファイル・フォーマットがデータベースのファイル・フォーマットである方法。

【請求項５４】

請求項５１に記載の方法において、前記ファイル・フォーマットが画像またはグラフィックスのファイル・フォーマットである方法。

【請求項５５】

請求項５１に記載の方法において、前記ファイル・フォーマットが分子構造のファイル・フォーマットである方法。

【請求項５６】

請求項５１に記載の方法において、前記ファイル・フォーマットが生物情報学的シーケンス・ファイル・フォーマットである方法。

【請求項５７】

請求項５１に記載の方法において、前記ファイル・フォーマットが生物情報学的データベースのファイル・フォーマット方法。

【請求項５８】

請求項５１に記載の方法において、前記ファイル・フォーマットがバイナリの実行可能なファイル・フォーマットである方法。

【請求項５９】

請求項１に記載の方法において、前記ステップａ）において、前記要求がファイル読取り要求を含み、

前記ステップｃ）において、前記仮想ファイルの前記コンテンツ・データが、データの

非実行可能表現から変換されたバイナリの実行可能な表現を含む方法。
【請求項６０】
請求項５９に記載の方法において、前記ステップｃ）において、前記バイナリの実行可能な表現が、前記要求を行っている前記マシンの前記アーキテクチャおよびオペレーティング・システムによって変わるようになっている方法。
【請求項６１】
請求項１に記載の方法において、前記ステップｃ）において、前記プラグイン関数によって生成された前記コンテンツ・データが、他のいくつかの非実行可能表現から変換されたバイナリの実行可能な表現である方法。
【請求項６２】
請求項６１に記載の方法において、前記ステップｃ）において、前記バイナリの実行可能表現が、前記要求を行っている前記マシンの前記アーキテクチャおよびオペレーティング・システムに依存している方法。
【請求項６３】
請求項１に記載の方法において、前記ステップｃ）において、前記コンテンツ・データがデータを呼び出すことによって生成されるようになっている方法。
【請求項６４】
請求項１に記載の方法において、前記ステップｃ）において、前記コンテンツ・データが、データを変更することによって生成されるようになっている方法。
【発明の詳細な説明】
【０００１】
（関連出願の相互参照）
適用無し。
（連邦政府により後援された研究／開発）
適用無し。
（発明の背景）
本発明は、概して、データベースを管理するためのシステムおよび方法に関し、特に、従来のプロトコル手段によって遠隔マシンの中に格納されているデータを必要とするファイル・システム要求を受信し、それに応答することがさらにできるファイル・システムを提供するためのシステムおよび方法に関する。
【０００２】
好むと好まざるとにかかわらず、「ファイル」の概念は、コンピュータ・サイエンスにおいて共通である。データ記憶の命名されたユニットとしてのファイル、およびファイルのフォーマット、ファイルの中の情報の構成および構造の概念は、プログラマおよびコンピュータ・ユーザによって等しく理解されている。これらの理由のために、ファイルは１９５０年代以来、プログラムとコンピュータとの間の通信の主な事実上の標準の方法となっているが、ファイル・フォーマットの種類が非常に多いこと、表示の細分性、同時アクセス、そして共同アクセスなどの問題がないわけではない。
【０００３】
１９７０年代以来、ネットワーク上でコンピュータを互いに接続するための機能によって、異なるコンピュータ間でファイルを共有したいという要望が発生してきた。初期の試みは、「ｕｕｃｐ」または「ｏｆｔｐ」などのプロトコルを使用して、１つのマシンから別のマシンへファイル全体を転送することだけができるものであった。１９８０年代半ばに分散形ファイル・システムが導入され、それによって遠隔マシン上でファイルに対してそれらがあたかも局所ディスク上にあるかのようにアクセスすることができるようになった。これまで、これらの標準のうちの最もポピュラーなものは、ＳＵＮ Ｍｉｃｒｏｓｙｓｔｅｍｓの Ｎｅｔｗｏｒｋ Ｆｉｌｅ Ｓｙｓｔｅｍ（ＮＦＳ）であった。他の重要な標準としては、ＭｉｃｒｏｓｏｆｔのＬＡＮ Ｍａｎａｇｅｒ、ＳＭＢおよびＣＩＦＳネットワーク・ファイル・システム、およびＡｐｐｌｅのＡｐｐｌｅＳｈａｒｅネットワーク・ファイル・システムなどがある。もっと最近の１９９０年代初期に、Ｗｏｒｌｄ Ｗｉ

ｄｅ Ｗｅｂ（WWW）が導入され、それによってファイル全体をインターネット上の任意のホストからハイパーテキスト転送プロトコル（ＨＴＴＰ）を使用して読むことができるようになった。ＨＴＴＰによって導入された技術革新のうち、ｈｔｂｉｎまたはｃｇｉ－ｂｉｎ WWWページの概念があり、それは遠隔サーバによってオンザフライで生成されたファイルである。これは、ＭＩＭＥ型（ＭａｃｉｎｔｏｓｈのＭａｃＯＳのファイル・システムに似たファイル・タイピング・システム）と組み合わされて、ソフトウェアおよびコンピュータ産業の重要な部分を根本から変えた。

【０００４】

（１）ＮＦＳ ネットワーク・ファイル・システムの概要

このセクションでは、Ｎｅｔｗｏｒｋ Ｆｉｌｅ Ｓｙｓｔｅｍ（ネットワーク・ファイル・システム）（ＮＦＳ）プロトコルについて記述する。これは仮想ネットワーク・ファイル・サーバによって使用されるプロトコルの１つであり、ＳＵＮ Ｍｉｃｒｏｓｙｓｔｅｍｓによって１９８５年に導入された。ＮＦＳはクライアント・サーバのアーキテクチャに基づいており、遠隔ファイル・システムに対する透過的なアクセスを提供するための手段を提供する。ファイル・サーバは一組のファイルをエクスポートするマシンである。クライアントは、そのようなファイルにアクセスするマシンである。クライアントとサーバとは同期要求として働く「リモート・プロシージャ・コール」を経由して通信する。クライアント上のアプリケーションが遠隔ファイルにアクセスしようとする時、カーネルが、応答を受信するまでサーバおよびクライアント・ブロックに対して要求を送信する。サーバは、入りクライアントの要求を待ち、それらを処理し、応答をクライアントに対して送り返す。

【０００５】

（２）ユーザの展望

ＮＦＳサーバは１つまたはそれ以上のファイル・システムをエクスポートする。エクスポートされる各ファイル・システムは、１つのパーティション全体か、あるいはそのサブツリーのいずれであってもよい。サーバは、通常、“／ｅｔｃ／ｅｘｐｏｒｔｓ”ファイルの中のエントリを通じて、どのクライアントがエクスポートされた各システムに対してアクセスできるか、その許可されているアクセスが読取り専用であるか、あるいは読取り／書込み可能であるかどうかを指定することができる。

【０００６】

次に、クライアント・マシンは、そのようなファイル・システムまたはそのサブツリーをそれぞれの既存のファイル階層の中に、それらをあたかも局所ファイル・システム上にマウントするかのように、その中の任意のディレクトリにマウントする。クライアントは、サーバがそれを読取り／書込み用としてエクスポートした場合であっても、そのディレクトリを読取り専用としてマウントすることができる。ＮＦＳは２種類のマウント、すなわち、「ハード」および「ソフト」をサポートする。これは、サーバが要求に対して応答しない場合におけるクライアントの行動に影響する。ファイル・システムがハード・マウント型であった場合、クライアントは応答が受信されるまで再試行を継続する。ソフトマウント型のファイル・システムの場合、クライアントはしばらくしてから諦め、そしてエラーを返す。「マウント」が成功すると、クライアントは、局所ファイルに対して適用されるのと同じ動作を使用して遠隔ファイル・システムの中のファイルにアクセスすることができる。

【０００７】

（３）プロトコルの設計目標

元々のＮＦＳ設計には、「ＮＦＳは、ＵＮＩＸに制限されるべきではない」という目標があった。任意のオペレーティング・システムがＮＦＳのサーバまたはクライアントを実装できることが必要である。そのプロトコルは特定のハードウェアに依存すべきではない。サーバまたはクライアントのクラッシュから単純に回復するメカニズムがなければならない。アプリケーションは、特殊なパス名またはライブラリを使用せずに、再コンパイルを行わずに遠隔ファイルに透過的にアクセスすることができる必要がある。ＵＮＩＸのファ

イル・システムのセマンティックスは、ＵＮＩＸのクライアントに対して維持されなければならない。ＮＦＳの性能は、局所ディスクの性能に匹敵するものでなければならない。その実装はトランスポート独立でなければならない。

【０００８】

ＮＦＳプロトコルの単独の最も重要な特性は、サーバがステートレスであり、そのクライアントが正しく動作することに関して何の情報も維持する必要がないことである。各要求は他と完全に独立であり、それを処理するために必要な情報のすべてを含む。サーバは、クライアントからの過去の要求の記録を、キャッシングまたは統計情報収集の目的のためにオプションとしてそうすること以外は維持する必要はない。

【０００９】

例えば、ＮＦＳプロトコルは、ファイルを開くか閉じるかに対する要求を提供しない。というのは、それはそのサーバが覚えておかなければならない状態情報を含むことになるからである。同じ理由のために、「読取り」および「書込み」の要求は、局所ファイル上での「読取り」および「書込み」の動作と違って、ファイル記述からオフセットを得るパラメータとして開始オフセットを渡す。

【００１０】

ステートレス・プロトコルによって、クラッシュからの回復が単純になる。クライアントがクラッシュした時は、回復は不要である。それはリブートする時にそのファイル・システムを単純に再マウントし、サーバはそのことを知らず、あるいは気にかけない。サーバがクラッシュすると、クライアントは、要求がタイムアウトになっていることを知り、単純にそれらを再送信する。サーバがリブートした後、最終的に答えるまで要求の再送を継続する。クライアントは、サーバがクラッシュしてリブートしたかどうか、あるいは単純に遅かったのかどうかを知る方法を持たない。しかし、状態のあるプロトコルは、クラッシュ‐回復のメカニズムを必要とする。サーバはクライアントのクラッシュを検出し、そのクラッシュに対して維持されていた状態を捨てなければならない。サーバがリブートする時、サーバはクライアントに対して通知し、従って、クライアントがそれぞれの状態をサーバ上に再構築できるようにしなければならない。

【００１１】

（４）ＮＦＳネットワーク・ファイル・システムのプロトコル・スタック

ＮＦＳプロトコル・スタックは、ファイル・システムの動作がネットワーク・プロトコル上でのパケットに変換される方法を定義するいくつかのコンポーネントまたは層から構成されている。そのプロトコル・スタックの最低レベルにはネットワーク・トランスポート層がある。従来はＮＦＳの下で、これはＵＤＰ（Ｕｎｒｅｌｉａｂｌｅ Ｄａｔａｇｒａｍ Ｐｒｏｔｏｃｏｌ）（信頼性のないデータグラム・プロトコル）インターネット・トランスポートから構成されている。しかし、現代の実装はＴＣＰ（Ｔｒａｎｓｍｉｓｓｉｏｎ Ｃｏｎｔｒｏｌ Ｐｒｏｔｏｃｏｌ）（伝送制御プロトコル）インターネット・プロトコルもサポートする。ＮＦＳプロトコル・スタックの次の層は、ＳＵＮ ＭｉｃｒｏｓｙｓｔｅｍｓのＸＤＲ（Ｅｘｔｅｎｄｅｄ Ｄａｔａ Ｒｅｐｒｅｓｅｎｔａｔｉｏｎ）（拡張データ表現）であり、それはネットワーク上で送信するためのデータの符号化のマシン独立の方法を提供する。次の層はＳＵＮ ＭｉｃｒｏｓｙｓｔｅｍｓのＲＰＣ（Ｒｅｍｏｔｅ Ｐｒｏｃｅｄｕｒｅ Ｃａｌｌ）（リモート・プロシージャ・コール）プロトコルであり、それはクライアントとサーバとの間のすべての対話のためのＸＤＲパケットのフォーマットを定義する。この上の次の層は３つのコンポーネント、すなわち、ＮＦＳ、ＭＯＵＮＴおよびＰＯＲＴＭＡＰプロトコルから構成されている。これらのピアー・プロトコルは、ＲＰＣを経由してそれぞれリモートＮＦＳ、ＭＯＵＮＴおよびＰＯＲＴＭＡＰのｄａｅｍｏｎｓ（ｎｆｓｄ、ｍｏｕｎｔおよびｐｏｒｔｍａｐｐｅｒ）に対してそれぞれ交信するためのＡＰＩレベルのインターフェースを定義する。最後に、最高の層が論理プロトコルであり、それはＰＯＲＴＭＡＰのｄａｅｍｏｎ（ＭＯＵＮＴおよびＮＦＳのｄａｅｍｏｎｓのポートを得るため）、ＭＯＵＮＴのｄａｅｍｏｎ（エクスポートされるファイル・システムのｒｏｏｔのファイル・ハンドルを得るため）および最後に、ＮＦ

Ｓのｄａｅｍｏｎに対する要求の順序を定義する（ＭＯＵＮＴのｄａｅｍｏｎまたは前のＮＦＳ応答からのファイル・ハンドルを使用して）。

【００１２】

さらに、現在２つのバージョンのＮＦＳおよびＭＯＵＮＴプロトコルがあることに言及する必要がある。元々の公開実装は、ＮＦＳバージョン２およびＭＯＵＮＴバージョン１プロトコルから構成されていた。しかし、これらは最近ＮＦＳバージョン３およびＭＯＵＮＴバージョン３として改訂されており、性能が改善され、そして２Ｇバイトより大きいファイル・システムに対してサポートする。

【００１３】

（５）層１：ＵＤＰ／ＩＰおよびＴＣＰ／ＩＰプロトコル

ＮＦＳプロトコル・スタックの最低レベルには、トランスポートとして使用されるインターネット・プロトコルがある。元々の実装は、本来的に信頼性のないＵＤＰプロトコルを使用していた。これは、コネクションレスのトランスポート・メカニズムであり、ネットワーク上のソケット間で任意のサイズのデータ・パケットを送信する。信頼性は低いが、そのプロトコル・スタックのＲＰＣ層は回答されない要求を追跡し続け、応答が受信されるまでそれらを定期的に再送信することによって、信頼性の高いデータ・グラム・サービスを実装する。ＵＤＰは、その実装が信頼性の高いコネクション・オリエントのＴＣＰの性能の利点を提供したので、最初は使用されていた。しかし、実装がさらに改善されてこの差はもはや存在しない。ほとんどのＮＦＳの実装に対しては、ＵＤＰがまだデフォールトであるが、多くの実装がＴＣＰ／ＩＰを代わりのものとしてサポートし、最近のＷｅｂＮＦＳの仕様はトランスポートとしてＴＣＰ／ＩＰをサポートすることを要求している。

【００１４】

ＴＣＰ／ＩＰを使用している時、データ転送はパケットの中に整理され、そのパケットのサイズをサーバが決定できるようにし、従って、完全な要求または応答が受信されたことを検出することができる。

【００１５】

（６）層２：拡張型データ表現（ＸＤＲ）

ＸＤＲ標準はネットワーク上でのデータ送信のためのマシン独立の表示を定義する。それはいくつかの基本のデータ型（ｉｎｔ、ｃｈａｒおよびｓｔｒｉｎｇなど）および複雑なデータ型（固定長および可変長の配列、構造体およびユニオンなど）を作るための規則を定義する。この標準は、バイトの順序、ワードのサイズおよび文字列のフォーマットなどの問題を扱い、それらはネットワーク接続のいずれかの端にある異質のコンピュータおよびオペレーティング・システム間で互換性がない可能性がある。

【００１６】

（７）層３：リモート・プロシージャ・コール（ＲＰＣ）プロトコル

ＳＵＮのＲＰＣプロトコルはクライアントとサーバとの間の通信のフォーマットを規定する。クライアントは、ＲＰＣ要求をサーバに対して送信し、サーバはそれらを処理し、その結果をＲＰＣの応答の中で返す。このプロトコルはメッセージ・フォーマット、送信および認証の問題に対処し、それはサービスの特定のアプリケーションには依存しない。ＳＵＮのＲＰＣは同期要求を使用する。クライアントがＲＰＣ要求を行うと、クライアントは応答を受け取るまでブロックする。これによってローカル・プロシージャ・コールの行動に似たＲＰＣの行動となる。

【００１７】

ＲＰＣはＸＤＲ符号化を使用して要求および応答のパケットのフォーマットを規定する。ＲＰＣの要求パケットは、送信ＩＤ、そのパケットが要求であるということ、そのパケットが意図されているプログラムの識別子およびプログラムのバージョン、実行されるべきプログラムの内部のプロシージャ、クライアントの認証情報（もしあれば）、およびプロシージャ固有のパラメータを含む。ＲＰＣの応答パケットは、それが応答している要求の送信ＩＤ、そのパケットが応答であるということ、その動作が実行されたかどうか、サーバの認証情報（もしあれば）およびプロシージャ固有の結果を含む。ユニークな送信ＩＤ

によって、クライアントは応答が到着した要求を識別することができ、そしてサーバは複製の要求（クライアントからの再送信によって発生した）を検出することができる。プログラム識別子およびプログラム・バージョンによって、単独のアプリケーション（またはソケット）が複数のプログラム要求にサービスし、そして同時に複数のプロトコル・バージョンをサポートすることができる。

【００１８】

ＲＰＣはサーバに対して呼出し側を識別するために５つの認証メカニズムを使用する。それらは、ＡＵＴＨ＿ＮＵＬＬ（認証なし）、ＡＵＴＨ＿ＵＮＩＸ（ＵＮＩＸスタイルの信用証明、クライアントのマシン名、ユーザＩＤおよび１つまたはそれ以上のグループＩＤを含む）、ＡＵＴＨ＿ＳＨＯＲＴ（前のＡＵＴＨ＿ＵＮＩＸ要求からのクッキー）、ＡＵＴＨ＿ＤＥＳ（データ暗号化標準の認証）およびＡＵＴＨ＿ＫＥＲＢ（Ｋｅｒｂｅｒｏｓ認証）である。ＡＵＴＨ＿ＳＨＯＲＴの概念は、クライアントがＡＵＴＨ＿ＵＮＩＸ信用証明を使用して認証されると、サーバはそのクライアントが将来のＲＰＣ要求において使用することができる短いトークンまたはクッキーを発生する。このＡＵＴＨ＿ＳＨＯＲＴは、既知のクライアントを非常に迅速に、暗号化することができ、高速の認証を提供する。

【００１９】

（８）層４Ａ： ｐｏｒｔｍａｐ（ｒｐｃｂｉｎｄ）プロトコル

ＮＦＳプロトコル・スタックの第１のサーバ・プロセス（ｄａｅｍｏｎ）は、ＲＰＣのｐｏｒｔｍａｐのｄａｅｍｏｎ（ｒｐｃｂｉｎｄとしても知られている）である。このサーバ・プロセスは、ソケットのコネクションを生成するためのＢＳＤスタイルのポート番号に対してプログラムの識別情報およびプログラムのバージョン番号をマップするディレクトリ・サービスを提供する。ＲＰＣの要求は、遠隔マシン上の特定のサービス（例えば、ＮＦＳバージョン３）を見つけるために、または局所マシン上でサービスを登録（および登録解除）するためにサーバに対して送信される。このポート・マッピングのサービスは、そのｐｏｒｔｍａｐのｄａｅｍｏｎのポート（普通、ポート１１１）だけが前もってクライアントによって知られている必要があることを意味する。次に、クライアントはこのサーバに問い合わせてｍｏｕｎｔのｄａｅｍｏｎおよびＮＦＳのｄａｅｍｏｎが実行中であるかどうかを知り、実行中の場合、それぞれのポート番号を問い合わせる。サーバは、通常、自分がスタートアップする時にｐｏｒｔｍａｐのｄａｅｍｏｎと交信して、自分が要求を待っているポートの番号をそれに知らせ、またサーバが自分自身の登録を解除するために遮断している時にｐｏｒｔｍａｐのｄａｅｍｏｎと交信する。

【００２０】

（９）層４Ｂ： ｍｏｕｎｔプロトコル

ＮＦＳプロトコル・スタックの次のサーバ・プロセス（ｄａｅｍｏｎ）は、ｍｏｕｎｔのｄａｅｍｏｎである。ＭＯＵＮＴプロトコルは、ＮＦＳプロトコルとは別であるが、それに関連付けられている。それはオペレーティング・システム固有のサービス、例えば、サーバのパス名の探索、ユーザのアイデンティティの検証、およびアクセス許可のチエックなどを提供する。ｍｏｕｎｔプロトコルは、ＮＦＳプロトコルから別のものとしてキープされ、ＮＦＳプロトコルを変更することなしに新しいアクセス・チエックおよび検証方法を実装し易くしている。また、ｍｏｕｎｔはディレクトリのパス名を必要とするが、ＮＦＳプロトコルはオペレーティング・システム依存のディレクトリ構文とは無関係である。ＮＦＳのクライアントは、ＭＯＵＮＴプロトコルを使用して最初のファイル・ハンドルを取得しなければならない。そのハンドルによってクライアントは遠隔ファイル・システムの中に入ることができる。ｍｏｕｎｔのｄａｅｍｏｎは、現在エクスポートされているファイル・システムのリストを知るために問い合わせることもできる

【００２１】

（１０）層４Ｃ： ＮＦＳプロトコル

ＮＦＳプロトコル・スタックの主要な、そして最終のサーバ・プロセス（ｄａｅｍｏｎ）は、ＮＦＳのｄａｅｍｏｎそのものである。そのステートレス・サーバは、読取り、書込

みおよび削除などのすべてのファイル操作要求を処理することを担当する。そのプロトコルの第１の最初の公開バージョンはＮＦＳバージョン２であった。それは１９８５年にＳｕｎＯＳ　２．０においてリリースされたもであり、すべてのＮＦＳ実装によってサポートされている。１９９３年に、高機能版のプロトコルＮＦＳバージョン３が発表され、現在ほとんどの実装によってサポートされている。（興味深いことに、この明細書の執筆時には、現在のＬｉｎｕｘ　ＮＦＳサーバおよびカーネルの実装はＮＦＳのバージョン２だけをサポートしている）。ＮＦＳのバージョン３は、４Ｇバイトより大きいファイル・システムに対する性能を増加させ、そしてサポートを可能にするいくつかの小変更を提供する。ＮＦＳｖ２プロトコルにおけるそのプロシージャのすべてが、その操作が完了し、そしてその要求に関連付けられているデータがすべて安定な記憶装置に対して引き渡された後でのみ制御がクライアントに対して戻る時に同期的であると仮定されている。ＮＦＳｖ３においては、この条件はＷＲＩＴＥ要求に対しては緩和されており、クライアントとサーバがＣＯＭＭＩＴ要求の使用をネゴシエートし、書込みをさらに高速に完了することができるようにしている。さらに、ＮＦＳｖ３はほとんどの操作の後にファイル属性を返し、そしてディレクトリを読み取っている時、ＮＦＳｖ２を使用している時に要求される多くのＧＥＴＡＴＴＲコールを不要にしている。

【００２２】

（１１）ＮＦＳのネットワーク・ファイル・システム・プロトコルの仕様

ＮＦＳｖ２プロトコルはＮＦＳサーバによってエクスポートされる１５個のプロシージャ（操作または方法）を規定している。ＲＰＣのプロシージャ番号は、シーケンシャルではない。というのは、２つの操作はバージョン２プロトコルにおいては決して実装または廃棄されないからである。これらはＲＯＯＴ（ｐｒｏｃｎｏ＝３）およびＷＲＩＴＥＣＡＣＨＥ（ｐｒｏｃｎｏ＝７）のプロシージャである。

【００２３】

```
／＊　ＮＦＳバージョン２プロトコルのＡＰＩ　＊／
attrstat      GETATTR(fhandle);      /*   proc=1    */
attrstat      SETATTR(sattrargs);     /*   proc=2    */
```

【００２４】

```
diropres      LOOKUP(diropargs);     /*   proc= 4    */
readlinkres   READLINK(fhandle);    /*   proc=5    */
```

【００２５】

```
readres       READ(readargs);        /*   proc=6    */
attrstat       WRITE(writeargs);      /*   proc=8    */
```

【００２６】

```
diropres      CREATE(createargs);     /*   proc=9     */
stat          REMOVE(diropargs);     /*   proc=10   */
```

【００２７】

```
stat          RENAME(renameargs);   /*   proc=11   */
stat          LINK(linkargs);         /*   proc=12   */
```

【００２８】

```
stat          SYMLINK(symlinkargs);  /*   proc=13    */
diropres      MKDIR(createargs);       /*   proc=14   */
```

【００２９】

```
stat          RMDIR(diropargs);        /*   proc=15   */
readdirres    READDIR(readdirargs);   /*   proc=16   */
```

【００３０】

```
statfsres      STATFS(fhandle);         /*   pr oc=17   */
```

ＮＦＳｖ３プロトコルはＮＦＳサーバによってエクスポートされる２１個のプロシージャを規定している。これらのプロシージャのほとんどは、バージョン２のものと構文が同じである。しかし、ほとんどの操作の後、ファイル属性が返され、いくつかのフィールドが

大きくなっているので、引数および結果の正確な型は僅かに異なっている。

【００３１】

```
／＊　ＮＦＳバージョン３プロトコルのＡＰＩ　＊／
GETATTR3res       GETATTR(GETATTR3args);          /*proc=1*/
SETATTR3res       SETATTR(SETATTR3args);          /*proc=2*/
```

【００３２】

```
LOOKUP3res        LOOKUP(LOOKUP3args);           /*proc=3*/
ACCESS3res        ACCESS(ACCESS3args);            /*proc=4*/
```

【００３３】

```
READLINK3res     READLINK(READLINK 3args);       /*proc=5*/
READ3res          READ(READ3args);                /*proc=6*/
```

【００３４】

```
WRITE3res         WRITE(WRITE3args);              /*proc=7*/
CREATE3res        CREATE(CREATE3args);            /*proc=8*/
```

【００３５】

```
MKDIR3res         MKDIR(MKDIR3args);              /*proc=9*/
SYMLINK3res       SYMLINK(SYMLINK3args);          /*proc=10*/
```

【００３６】

```
MKNOD3res         MKNOD(MKNOD3args);              /*proc=11*/
REMOVE3res        REMOVE(REMOVE3args);            /*proc=12*/
```

【００３７】

```
RMDIR3res          RMDIR(RMDIR3args);             /*proc=13*/
RENAME3res        RENAME(RESNAMEargs);            /*proc=14*/
```

【００３８】

```
LINK3res           LINK(LINK3args);                /*proc=15*/
READDIR3res       READDIR(READDIR3a rgs);         /*proc=16*/
```

【００３９】

```
READDIRPLU3res   READDIRPLUS(READDIRPLUS3args); /*proc=17*/
FSSTAT3res         FSSTAT(FSSTAT3args);           /*proc=18*/
```

【００４０】

```
FSINFO3res         FSINFO(FSINFO3args);           /*proc=19*/
PATHCONF 3res     PATHCONF(PATHCONF3args);        /*proc=20*/
COMMIT3res        COMMIT3args);                   /*proc=21*/
```

【００４１】

（１２）他の遠隔ファイル・システムプロトコル

上記のＮＦＳプロトコルの詳細内容は本発明の「好適な実施形態」に対する背景を提供する。しかし、その仮想ネットワーク・ファイル・サーバの発明は、他の共通のネットワーク・ファイル・システム・プロトコルをカバーするように容易に拡張することができ、その「好適な実施形態」は、本発明の一例、またはアプリケーションに過ぎない。次のパラグラフではＮＦＳと他のポピュラーなネットワーク・ファイル・システム・プロトコル、Ｍｉｃｒｏｓｏｆｔのサーバ・メッセージ・ブロック（ＳＭＢ）との間の類似性について説明する。

【００４２】

ＳＭＢプロトコルは現在、共通インターネット・ファイル・システム（ＣＩＦＳ）として改訂されつつあり、それは次の数年にわたって重要な標準プロトコルとなる可能性がある。

【００４３】

Ｍｉｃｒｏｓｏｆｔのサーバ・メッセージ・ブロック（ＳＭＢ）は、ＭＳ‐Ｎｅｔ、ＬＡＮ　ＭａｎａｇｅｒおよびＷｉｎｄｏｗｓ　Ｎｅｔｗｏｒｋｉｎｇによって使用されているファイル共有プロトコルである。このプロトコルは、ＭｉｃｒｏｓｏｆｔのＷｉｎｄｏ

ｗｓ　９ｘ、Ｗｉｎｄｏｗｓ　ＮＴおよびＯＳ／２のオペレーティング・システムのネイティブのファイル共有プロトコルである。ＮＦＳプロトコル・スタックの層２および３の中で使用されているＳＵＮのＸＤＲおよびＲＰＣ層の代わりに、ＳＭＢがその中間層として使用されているＮｅｔＢＩＯＳである。ＮｅｔＢＩＯＳは、ＩＢＭ　ＰＣのネットワーク・ブロードバンドＬＡＮに対する高級プログラミング言語のインターフェースとして始まったが、トークン・リング、ＴＣＰ／ＩＰ、ＩＰＸ／ＳＰＸなどの底流にあるトランスポート・メカニズムのいくつかの上の「書込みプロトコル」として進化した。現在好まれているトランスポートは、ＴＣＰ／ＩＰおよびＵＤＰ／ＩＰ（インターネットのＲＦＣ１００１および１００２において記述されている）であり、層１をＮＦＳとＳＭＢとの間で同じものにしている。ｐｏｒｔｍａｐのｄａｅｍｏｎと交信する代わりに、ＳＭＢは遠隔ファイル・サーバを見つけるためにＮｅｔＢＩＯＳのネーム・サーバ（ＭｉｃｒｏｓｏｆｔのＷＩＮＳなど）に対して要求をブロードキャストする。ｐｏｒｔｍａｐのｄａｅｍｏｎと同様に、ネーム・サーバは名前付きのファイル・システムをサポートしているサーバのＩＰアドレスを回答する。次に、ＳＢＭのクライアントはＮｅｔＢＩＯＳのセッション・マネージャを使用してこのホスト上のファイル・サービスと交信し、ＭＯＵＮＴのｄａｅｍｏｎと交信した後、ＴＣＰ／ＩＰ上でＮＦＳに似たコネクションを生成する。次に、ＴＣＰ／ＩＰをまったく同様に使用して、マシン間の送信されるメッセージのフォーマットを除くすべてにおいてＮＦＳに対してパケットが送信および受信される。これらのメッセージを正しく解釈して応答することによって、仮想ファイル・サーバはネットワーク上でのＷｉｎｄｏｗｓベースのＰＣに対して仮想ＳＭＢファイル・システムを提供することができる。

【００４４】

（１３）生物学的シーケンスのデータベース・マネジメント

前記のフレームワークを念頭において、蛋白質および核酸シーケンスのデータベースの効率的な記憶が生物情報学（ｂｉｏｉｎｆｏｒｍａｔｉｃｓ）における主要な課題の１つである。問題は４つの項目、すなわち、データベースのサイズ、データのフォーマッティング、データのサブセットおよびデータの完全性の相互作用から生じる。

【００４５】

最も明らかな問題点は、現在のデータベースのサイズが非常に大きいことである。現在のデータベースのサイズは数百万の核酸および数十万の蛋白質のシーケンスを表すためのデータの１０～１００Ｇバイトの範囲にある。この問題は、約１８カ月で倍になるこれらのデータベースの成長の速度によって倍加される。実際、科学者達がヒト・ゲノムのプロジェクトの最終段階に入っている状態で、この速度は近い将来において減少するどころか、増加すると予想されている。次の問題点はデータ表現の問題点である。

【００４６】

ほとんどの生物情報学のサイトは、Ｂｌａｓｔ、ＦＡＳＴＡおよびＧＣＧなどのプログラムを含んでいるデータベースのサーチ用ソフトウェアをいくつか維持している。不幸なことに、この多様性の結果、ほとんどの生物情報学的サイトは、元のフラットなファイル、ＦＡＳＴＡフォーマット、ＧＣＧ／ＰＩＲフォーマット、Ｂｌａｓｔ圧縮フォーマットおよびインデックス、およびＳＲＳのインデックスなどの複数のファイル・フォーマットで主要なデータベースを維持している。追加の各表示は、通常、そのデータベースのために数十ギガバイトの追加のファイル記憶装置を必要とする。次の問題は、データベースのサブセットおよびスーパーセットの問題である。各スタティック・データベースの他に、生物情報学的サイトはすべての蛋白質シーケンス（ｐｒｏｔｅｉｎ＝ｓｗｉｓｓｐｒｏｔ＋ｇｅｎｐｅｐｔ＋ｐｉｒ＋ｐｄｂまたはｓｗｉｓｓｐｒｏｔ＝ｓｗｉｓｓｍａｉｎ＋ｓｗｉｓｓｎｅｗ）およびすべての核酸シーケンス（ｎｕｃｌｅｉｃ＝ｅｍｂｌ＋ｇｅｎｂａｎｋ）などの複合データベース（またはスーパーセット）を維持していることが多い。

【００４７】

スーパーセットのいくつかのフォームを、複合型仮想データベースとして複数のデータベースを処理するデータベース探索ソフトウェアによって処理することができる。しかし、

これは、データベース間の重複しているエントリを消去するあらかじめ定義された非冗長性データベースよりずっと性能が悪い。同様に、センシブルなデータのサブセッティングを実行することができるパッケージは非常に少なく、従って、ほとんどのサイトは、すべてのイースト・シーケンス、すべてのヒトＥＳＴシーケンス、すべての蛋白質キナーゼなどのサブセット・データベースも独立に維持している。最後に、いくつかの機関にとって、頻繁に更新されるシーケンスのデータベースの利用可能性が保証されていることが不可欠であると考えられる。従って、これらのサイトは複製のデータベースを維持し、１つを更新および修正することができるようにし、一方、他のデータベースによって通常のサービスを提供することができる。この方法で、自動更新ができなくなるか、あるいはデータベースのフォーマットまたは編成が変わった場合でも、「ライブ」のデータベースは壊されない。

【００４８】

この制約は、ほとんどの競合している生物情報学的サイトが利用可能性の高い数百ギガバイトの記憶装置を必要とすることを意味する。実際、これらの需要は非常に大きいので、多くのサイト（ほとんどの大学のサイトを含む）は、潜在的な開示の問題がある場合でも、インターネット上での生物情報学的リソースにアクセスすることが制限されている。

【００４９】

（発明の概要）

本発明は、現代の技術における上記の欠点に対処し、それらを緩和するために特に設計されている。これに関して、本発明は、仮想ファイル・サーバに関しており、標準プロトコル手段を使用することによってデータベースを効率的に管理する方法を提供し、ディスク空間（すなわち、メモリ）が少なくて済み、通信データを受け渡す新しい方法をさらに提供する。本発明は生物学的シーケンスのデータベース管理の上記の問題に対処するのに特によく適している。

【００５０】

仮想ファイル・サーバは、本質的に、遠隔ファイル・システムからのファイルの内容をファイル・システムの要求に応じて生成して返すことができるプロセスを含む。

【００５１】

仮想ファイル・サーバは、遠隔ファイル・システムをシミュレートし、ネットワーク・インターフェースを経由してローカル・エリア・ネットワーク上で要求を行うマシンに対して「仮想の」ファイルおよびディレクトリを提供する。これに関して、仮想ネットワーク・ファイル・システムは、そのような仮想ネットワーク・ファイル・システムが物理記憶媒体（すなわち、ハード・ディスク）上でファイルの検索および記憶を行っているかのようにネットワークからのファイル・システム要求を受信し、それに応答するように動作する。動作において、その仮想ネットワーク・ファイル・サーバが、例えば、ファイルの読取り要求を受信すると、仮想ネットワーク・ファイル・サーバは、その指定された「仮想の」ファイルの内容を生成し、その内容はそのファイル名および環境からアルゴリズム的に生成されるか、あるいは暗号化、圧縮解凍等によって格納されている物理的なファイルを転送することなどによって生成することができる。

【００５２】

クライアントのアプリケーションに対して、仮想ファイル・サーバは、適切なフォーマットで適切なファイルを階層的に含んでいる普通のディレクトリのように見える。有利なこととして、仮想ネットワーク・ファイル・サーバは、クライアントのオペレーティング・システムに関与せず、遠隔マシンにアクセスするためにそのネイティブのメカニズムを使用する。さらに、ＮＦＳまたはＳＭＢなどの標準プロトコルを使用することによって、仮想ファイル・サーバは、特殊化されたネットワーク・ソフトウェアが、そのクライアントに対して書き込まれることを必要とせず、従って、既存のソフトウェアがそのファイル・サーバと修正なしで動作することができる。例えば、ＮＦＳのクライアント・ソフトウェアは、ＵＮＩＸによって分配され、ＭｉｃｒｏｓｏｆｔのＷｉｎｄｏｗｓ、ＡｐｐｌｅのＭａｃｉｎｔｏｓｈおよびＶＡＸ／ＶＭＳなどの実質的にすべてのオペレーティング・シ

ステムに対して利用できる。同様に、ＭｉｃｒｏｓｏｆｔのＳＭＢクライアントは、ＭｉｃｒｏｓｏｆｔのＷｉｎｄｏｗｓ　ＮＴ、Ｗｉｎｄｏｗｓ　９５およびＷｉｎｄｏｗｓ　９８の中に含まれている。

【００５３】

データ管理の観点から、仮想ファイル・サーバは、単独のフォーマットで内部的に主題のデータベースを維持することができる。ファイル要求時に、仮想ファイル・サーバは、サブセットおよびスーパーセットの動作を実行することができ、次に適切な再フォーマット化を行うことができる。データベースの１つのフォーマットだけが維持されているので、サーバ上のキャッシングは遥かに効率的である。多くのシーケンス・データベース・フォーマット変換を非常に効率的に実装することができ（例えば、有限の状態マシンを使用して）、その結果として性能損失が無視できる。実際に、サーバは非常に効率的な圧縮されたフォーマットでデータを内部的に自由に表示し、例えば、重複シーケンスの除去、残差のビットごとの符号化またはハフマン符号化、および親の中の場所に対する参照として別のサブシーケンスであるシーケンスを表すことができる。

【００５４】

もう１つの重要なアプリケーションは、個々のシーケンス・データベースのエントリを個々のシーケンス・ファイルとしてエクスポートすることができることである。これによって、生物情報学的アルゴリズムに対して、問合わせシーケンスを、それらをデータベースから先ず最初に抽出する（「取り出す」）ことなしに指定することができる。

【００５５】

また、仮想ファイル・サーバのアーキテクチャは、構造的データベースの記憶管理および外部計算化学の応用の統合化に対しても適用することができる。１つの大きな応用はＢｒｏｏｋｈａｖｅｎ　Ｐｒｏｔｅｉｎ　Ｄａｔａｂａｎｋ（ブルックヘブン蛋白質データバンク）、ＰＤＢ（およびルートガー大学（Ｒｕｔｇｅｒｓ　Ｕｎｉｖｅｒｓｉｔｙ）の核酸構造データベースＮＤＢも）の記憶装置およびメンテナンスにある。現在この「データベース」は、ＡＳＣＩＩテキスト・ファイルとして格納されている約７０００個のファイルの集合として維持されている。これらのデータ・ファイルを、圧縮および表示における冗長性の削減の両方によって、より効率的に内部的に表すことができる。

【００５６】

最後に、仮想ファイル・サーバは、計算化学サービスを提供するための便利なメカニズムを提供する。例えば、仮想ファイル・サーバは、Ｓｙｂｙ１　Ｍｏｌ２、ＸＰＬＯＲ　ＰＤＢおよび他のファイル・フォーマットをエクスポートすることによって、ファイルのフォーマット変換を実行することができる。計算的には、サーバは各ＰＤＢファイルの中でのＤＳＳＰまたはＳｔｒｉｄｅ二次構造割当てを提供することができ、アルファ炭素専用ファイルからのバックボーンおよび／またはサイドチェインの座標を再構成し、結晶学的対称性を生成し、代表的なＮＭＲモデルを選択し、あるいはプロパティの計算を実行することもできる。

本発明のこれらおよび他の特徴は、図面を参照することによってさらに明らかになるだろう。

【００５７】

（発明の詳細な説明）

添付図面に関して以下に説明される詳細な説明は、本発明のこの好適な実施形態の説明を意図しており、本発明を構築あるいは利用することができる唯一の形式を表すことが意図されているわけではない。この記述は示されている実施形態に関して本発明を構築し、動作させるための機能およびシーケンスを示している。しかし、これと同じか、あるいは等価な機能およびシーケンスを異なった実施形態によって実現できること、そしてそれらも本発明の範囲内に含まれることが意図されていることを理解されたい。

【００５８】

本発明は、仮想ネットワーク・ファイル・サーバに関しており、遠隔ファイル・システムを含むコンピュータのオペレーティング・システムに対する要求に応じて指定されたファ

イルの内容を生成し、その内容をその要求がなされたコンピュータに対して、その要求がなされたコンピュータとそのような要求を受信するサーバ・マシンとの間でネットワーク・インターフェースを経由して返す。この例に関して実際には、オペレーティング・システムを経由してそのような要求を受信するサーバ・マシンは、そのような要求を仮想ネットワーク・ファイル・サーバに対して渡し、後者はユーザ・プロセスとして実行している。

【００５９】

図１に示されているように、コンピュータ上で実行されているユーザのアプリケーションまたはプロセス１０が論理ファイル・システムおよびディレクトリ３０にアクセスするために、そのマシン上にオペレーティング・システム２０に対して要求を行う。その要求の中で指定されている論理ファイル名（およびディレクトリ）は、そのオペレーティング・システムのファイル・サーバによって、局所記憶装置４０上に格納されているのではなく、遠隔ファイル・システムに属しているとして理解される。次に、オペレーティング・システム２０は、その従来の遠隔ファイル・システムのマッピングを使用して要求されたファイルに対するファイル・サーバのネットワーク上の場所を知る。次に、オペレーティング・システム２０は、ネットワーク・ファイル・サーバのネットワーク・インターフェース５０を経由して、そのネットワーク・ファイル・サーバに対するＴＣＰ／ＩＰ要求を発生する。ネットワーク・ファイル・サーバは同じマシン上にあってもよく（その場合はネットワーク・インターフェースはサーバ・プロセスに対する要求パケットを単純にコピーする）、あるいは遠隔マシン上にあってもよい（その場合はネットワーク・インターフェース５０がその要求パケットをローカル・エリア・ネットワーク６０上で送信する）。サーバ・マシンは、そのパケットをそのネットワーク・インターフェース７０経由で受け取り、オペレーティング・システム８０は、その要求をユーザ・プロセスとして動作している仮想ネットワーク・ファイル・サーバ９０に対して渡す。

【００６０】

仮想ネットワーク・ファイル・サーバ９０は、ファイル・システムをエミュレートし、ローカル・ネットワーク上でのマシンの「仮想」ファイルおよびディレクトリをエミュレートする。ファイル読取り要求を受け取ると、それは指定された「仮想」ファイルの内容を生成する。そのファイル内容はそのファイル名および環境からアルゴリズム的に生成されるか、あるいは暗号化によって格納された物理ファイルを変換することによってのいずれかで生成することができる。トリビアル（重要でない）変換と、非トリビアル（重要な）変換との間の区別が行われる。いくつかの既存のファイル・サーバを、ファイルを遠隔マシンに対してエクスポートする前に、ライン・ターミネーション・キャラクタを変換するように構成することができる。ここで説明されるメカニズムは、データを異なるフォーマットに、例えば、画像ファイル・フォーマット間で、あるいは共通の表示間で生物情報学的データベースに変換するために適用される。そのような変換機能をさらに有利に利用して１つの特定のワードプロセッシング・フォーマットまたはプログラムで存在しているテキスト・ファイルを別のタイプのワードプロセッシング・フォーマットに迅速に変換することができる。同様に、そのような変換特性によって、１つの暗号化されたファイルを別のファイルに、あるいは特定のタイプの圧縮されたフォーマットで格納されているデータを、別のフォーマットに迅速に変換することができる。このことに関して、仮想ファイルの内容が一度生成されると、その同じものをネットワーク上でその要求しているユーザ・プロセスに対して応答パケットとして送り返すことができる。

【００６１】

従来の技術の仮想ファイル・システムはすべてクライアントのオペレーティング・システム２０を、ファイルが局所ホスト上の論理ファイル・システム３０によって変換されるように修正されることを要求した。例えば、フス（Ｈｓｕ）に対する「Ｃｏｍｐｕｔｅｒ Ｓｙｓｔｅｍ Ｉｎｃｌｕｄｉｎｇ ａ Ｔｒａｎｓｐａｒｅｎｔ ａｎｄ Ｓｅｃｕｒｅ Ｆｉｌｅ Ｔｒａｎｓｌａｔｉｏｎ」（透過的で安全なファイル変換メカニズムを含んでいるコンピュータ・システム）と題する米国特許第５，５８４，０２３号の内容を参

照されたい。１つの大きな違いは、本発明の仮想ネットワーク・ファイル・システムは、クライアントのオペレーティング・システムを関与させず、そのネイティブのメカニズムを使用して遠隔マシンにアクセスすることである。代わりに、仮想ネットワーク・ファイル・システムは、あたかも物理記憶媒体上のファイルを読取りおよび格納しているかのように、ネットワークからのファイル・システムに対する要求を受け取り、それに応答する。

【００６２】

サーバ・プロセス１００は、１つのマシン上でステップ１１０において開始されると、ステップ１２０においてサーバを初期化し、連続のループ１３０に入ってステップ１４０において遠隔マシンまたはシステムに対して向けられたファイル・システム要求を待ち、その要求をステップ１５０においてデコードし、その応答をステップ１６０において決定し、すなわち、その要求されたデータを生成するか、あるいはその要求されたアクションを実行し、ステップ１７０においてその応答を符号化し、そして次にステップ１８０においてその結果を要求しているプロセスに対して図２に示されているように送信する（すなわち、送り返す）。「要求をデコードする」のステップ１５０および「応答を符号化する」のステップ１７０は、ネイティブのファイル・システムの要求と、仮想ネットワーク・ファイル・サーバによって使用されている内部のデータ構造およびルーチンとの間でそれぞれマッピングを実行する。これによって、仮想ファイル・サーバがＵＮＩＸのＮＦＳ、ＭｉｃｒｏｓｏｆｔのＬａｎＭａｎａｇｅｒ、ＡｐｐｌｅＳｈａｒｅなどの複数の遠隔ファイル・システム・プロトコルをサポートすることができる。

【００６３】

第１に、現在の実装は、複数のスレッド（またはプロセス）を使用し、複数のファイル・システム要求が同時に実施されるようにし、そして仮想サーバに対して仮想ファイル・システムの内容を、ファイル・システム要求を受信せずに更新し続ける処理を実行させることができる。

【００６４】

図３に示されている、前のフローチャートの１つの変形版１９０は、仮想ファイル・システムの内容および行動が仮想ネットワーク・ファイル・サーバそのものから隔離されている実装である。この方法で、本発明の仮想ネットワーク・ファイル・サーバによって、複数の使用（または実装）を実行時に決定することができる。これはプラグインと呼ばれることが多い動的共有オブジェクトの技術を使用して実現される。これらのプラグインは、通常、ＵＮＩＸマシン上での共有ライブラリ、ＭｉｃｒｏｓｏｆｔのＷｉｎｄｏｗｓでのＤＬＬ、およびＡｐｐｌｅ　Ｍａｃｉｎｔｏｓｈ上でのＣｏｄｅＦｒａｇｍｅｎｔｓの形を取る。サーバが最初にステップ２００からスタートアップすると、それはステップ２１０において仮想ファイル・システムの中で「プラグイン」を見つけ、動的にリンクする。そのプラグインは仮想ファイル・システムのユーザ定義行動を表し、符号化する。クライアント・マシンに対してこれらのプラグインは図４に概念的に示されているように、遠隔ファイル・システム上の独立のファイルまたはディレクトリを形成する。

【００６５】

要求が受信されるたびに、ネットワーク・ファイル・サーバは、ステップ２２０において初期化し、ロードされたどのプラグインがその要求を扱い、ディスパッチし、デコードし、そしてその要求を適切に待つかをステップ２３０、２４０において決定する。ステップ２６０においてそのような要求がファイル・システムに対して向けられていない範囲に対して、その後、サーバはステップ２７０において応答を発生し、その結果のデータを要求しているシステムのネイティブ・フォーマットに変換し、ステップ２８０においてそれをその要求しているマシンに対して送り返す。しかし、仮想ネットワーク・ファイル・サーバはステップ２９０において選択的にプラグインを選定し、ステップ３００において呼び出すことによって、ステップ２６０においてそのようなファイル・システムの「ハウスキーピング」要求を有利に処理することもでき、その後、ステップ３２０においてそれから応答を符号化し、従って、トリビアル機能あるいは普通の機能を実施するタスクの他のプ

ラグインを解放する。この方法で、仮想ネットワーク・ファイル・サーバは、プラグインからその遠隔ファイル・システムの符号化および解読の複雑性を隠し、また、その動作を単純化する。

【００６６】

各仮想ファイル・システムのプラグイン動作は、図４に概念的に示されている遠隔ファイル・システム要求の単純化された組を処理することである。そのような要求は、ファイルからのデータの読取り（ＲＥＡＤ）３４０、ファイルへのデータの書込み（ＷＲＩＴＥ）３６０、ファイル上のセキュリティ保護および許可（ＣＨＭＯＤ）３８０、新しいファイルの作成（ＣＲＥＡＴＥ）４００、およびディレクトリまたはフォルダの内容の決定（ＲＥＡＤＤＩＲ）４２０を含む。各プラグインに対して、仮想ネットワーク・ファイル・サーバは、そのプラグインの内部に各遠隔ファイル・システム要求に対するルーチンのテーブルを維持する。仮想ネットワーク・ファイル・サーバが要求をデコードし、どのプラグインがそれに応答する必要があるかを決定すると、それはその適切なプラグインの「機能ディスパッチ」テーブルの内部の要求された機能を探索し、その後、その機能を呼び出す。

【００６７】

この方法で、ファイル・システムの正確な行動を、仮想ネットワーク・ファイル・サーバが設計されるか、あるいは実装される時点で規定する必要はない。しかし、仮想ネットワーク・ファイル・サーバおよび非トリビアル変換、例えば、データベース変換または計算を提供する実装は、新しく、実際的であり、商用の利点があるものである。

【００６８】

仮想ファイル・システムのプラグインの必要なタスクを拡張するために、図５はＲＥＡＤ機能３４０に対する単純化されたフローチャートを表し、ステップ４６０においてファイル名をデコードし、ステップ４８０において仮想ファイルの内容を生成し、そしてステップ５００において仮想ファイルの内容を返す。プラグインの読取り機能は、ファイル識別子、そのファイルの中のロケーションおよび読み取られるべきデータの量においてステップ４６０において呼び出される。そのプラグインはそのファイルの内容が何であるべきかを主としてそのサーバの開始時に、そのデータを要求しているユーザおよび／またはマシンまたは他のファクタに基づいてステップ４８０において自由に選定する。例えば、そのプラグインはその要求されたファイル“ｄａｔｅ．ｔｘｔ”の内容が常に現在の時刻および日付、すなわち、“Ｗｅｄ　Ａｕｇ　１２　１１：２８：４３　ＭＤＴ　１９９８”であることを宣言することができる。次の日の同じファイルの内容または丁度数秒後の内容の読取りが異なる結果を生成すること、およびファイルの内容が物理記憶媒体上に存在していないが、ホストのＣＰＵの時計からアルゴリズム的に生成されていることに留意されたい。この例を続けると、プラグインに対して渡されたネットワークの読取り要求が、そのファイルの５番目および６番目および７番目の文字を要求していた場合、そのプラグインは“Ａｕｇ”を返すことになる。次に、このデータがステップ５００においてネットワーク・ファイル・サーバに対して送り返され、ネットワーク・ファイル・サーバは実遠隔ファイル・サーバからの応答をエミュレートするパケットの中にそのデータを符号化して送り返すタスクを実行する。

【００６９】

（１）仮想ＮＦＳサーバの実行

仮想ファイル・サーバのプロトタイプは現在ＮＦＳプロトコルのバージョン２および３およびＭＯＵＮＴプロトコルのバージョン１および３を、ＵＤＰ／ＩＰトランスポート・プロトコル上に実装している。以下に説明されるこの実装は、仮想ファイル・サーバのこの好適な実施形態であるが、いずれにしても本発明の範囲をＮＦＳプロトコルまたはＵＮＩＸオペレーティング・システムに対して制限するものではない。そのように、例題の「実施形態」は「ｖｎｆｓｄ」のＵＮＩＸバージョンを記述する次のセクションにおいては、仮想ＮＦＳサーバまたは「ｖｎｆｓｄ」と呼ばれる。

【００７０】

仮想ＮＦＳサーバ（ｖｎｆｓｄ）は、単純に“ｖｎｆｓｄ”とタイプすることによって、ＵＮＩＸのコマンド・ラインから開始することができる。そのサーバは実行するためにスーパユーザの特権を必要としない。従って、「ｒｏｏｔ」から開始される必要はない。

【００７１】

“ｖｎｆｓｄ”は、次のオプションのコマンド・ラインの引数も受け付ける。

１．）－ｐｏｒｔ　＜ｎ＞は、省略時ポート２１０６９の代わりに指定されたポートを使用する。

【００７２】

２．）－ｄａｅｍｏｎ　バックグラウンドにおいてｄａｅｍｏｎとして実行し、親プロセスから離れる。

３．）－ｄｅｂｕｇ　各ＲＰＣ要求パケットに対してデバッグ情報をｓｔｄｏｕｔに表示する。

【００７３】

４．）－ｎｏｒｅｇ．　ＲＰＣ　ｐｏｒｔｍａｐｐｅｒ　ｄａｅｍｏｎでサーバを登録しようとしない。

システムの“／ｅｔｃ／ｒｃ．＊”スクリプトのネットワーキング・セクションに対してｖｎｆｓｄコマンドを追加することによって、マシンがリブートされる時に自動的にｖｎｆｓｄをスタートさせるのが普通の慣習である。唯一の条件は、従来のｍｏｕｎｔおよびＮＦＳのｄａｅｍｏｎがスタートした後で“ｖｎｆｓｄ”がスタートされることである。

【００７４】

２）仮想ＮＦＳサーバの終了

現在、ｖｎｆｓｄのｄａｅｍｏｎを停止させるための唯一の方法は、ＳＩＧＩＮＴの割込み信号をそれに送信する方法である。そのプログラムがフォアグラウンドにおいて実行している場合、これは制御しているＵＮＩＸのシェルから“＾Ｃ”（あるいは同様なプロセス制御文字をタイプすることによって行うことができる。プログラムがｄａｅｍｏｎとして実行している場合、それはコマンド“ｋｉｌｌ　－ＩＮＴ＜ｐｉｄ＞”を使用して停止させることができる。ここでＰＩＤは、実行しているｄａｅｍｏｎのプロセスｉｄである。

【００７５】

ファイル・システムを現在マウントしているクライアントは、サーバを停止させる前に該当のディレクトリをアンマウントする必要がある。さもなければ、そのクライアント・マシンは、ｖｎｆｓファイル・システムに次にアクセスする時にハングする可能性がある。ｍｏｕｎｔコマンドが「ソフト」を指定した場合、そのクライアント・プロセッサは、何秒か後にタイムアウトするが、システム性能が大きく劣化する可能性がある。

【００７６】

仮想ＮＦＳのｄａｅｍｏｎは、停止させられず、割込みが掛かるだけでなければならない。ｄａｅｍｏｎは、その割込み信号を捉え、適切なファイルをすべて優美に閉じてフラッシュし、自分自身をＲＰＣのｐｏｒｔｍａｐのｄａｅｍｏｎから登録解除して停止する。

【００７７】

（３）仮想ＮＦＳファイル・システムのマウント

理想的には、次に、ＵＮＩＸクライアントは、次のコマンドによって仮想ファイル・システムをマウントする。

ｍｏｕｎｔ　－ｔ　ｎｆｓ　－ｏ　ｐｏｒｔ＝２１０６９、ｍｏｕｎｔｐｏｒｔ＝２１０６９　ｓｅｒｖｅｒ：／／ｖｎｆｓ

【００７８】

このコマンドは局所カーネルに遠隔マシン“ｓｅｒｖｅｒ”から局所マウント・ポイント“／ｖｎｆｓ”上にＮＦＳを直接マウントするよう指示する。ｍｏｕｎｔオプションが指定されると、遠隔にエクスポートされたファイルは、そのサーバ、上記の例の中のｒｏｏｔ・ディレクトリによって解釈されない。局所マウント・ポイント／ｖｎｆｓは、局所ファイル・システム上の既存のディレクトリでなければならない。さらに、“ｓｏｆｔ”マ

ウント・オプションも含めて、遠隔ｖｎｆｓのｄａｅｍｏｎが故障した場合に、そのマシンをハングさせるのではなく、クライアントがＮＦＳの要求をタイムアウトすることができる。

【００７９】

不幸なことに、ｍｏｕｎｔコマンドに対して“ｍｏｕｎｔｐｏｒｔ”のオプションを現在サポートしているオペレーティング・システムはほとんどない。そのような場合、適切なＵＮＩＸのコマンドは次のようになる。

【００８０】

ｍｏｕｎｔ －ｔ ｎｆｓ ｏ ｐｏｒｔ＝２１０６９，ｓｏｆｔ ｓｅｒｖｅｒ；／ｖｎｆｓ／ｖｎｆｓ

遠隔サーバ上のディレクトリのパスが異なっている理由は、ｍｏｕｎｔｐｏｒｔオプションがない場合、ＭＯＵＮＴプロトコル要求が遠隔“ｓｅｒｖｅｒ”上での従来のマウントに進むことである。次に、ベンダ供給のｍｏｕｎｔが、有効なマウント・ポイントを求めて”／ｅｔｃ：ｅｘｐｏｒｔｓ”の中のエクスポート・リストをチエックする。そのサーバが現在ファイル・システムをエクスポートしている場合、それらのどのパスでも十分である。上記の比較的良好な解決策は空のディレクトリ”／ｖｎｆｓ”を遠隔マシン上に単純に生成し、［ｖｎｆｓｄのマウント・ポイント］そしてこれを“／ｅｔｃ／ｅｘｐｏｒｔ”ファイルに対して追加することである。

【００８１】

代わりに、そのサーバを指定しているクライアント・マシン上の“／ｅｔｃ／ｆｓｔａｂ”に対して１つのエントリを追加することができ、遠隔ディレクトリおよびｍｏｕｎｔオプションを局所ＮＦＳに対して追加することができる。ｖｎｆｓのｄａｅｍｏｎに対する代表的なｆｓｔａｂエントリは、次のようになる。

【００８２】

ｓｅｒｖｅｒ：／ｖｅｆｓ ／ｖｎｆｓ ｎｆｓ ｐｏｒｔ＝２１０６９，ｓｏｆｔ，ｎｏａｕｔｏ ０ ０

“ｎｏａｕｔｏ”オプションが上記のように指定された場合、仮想ＮＦＳサーバはクライアントがブートされる時に自動的にマウントされない。このラインが／ｅｔｃ／ｆｓｔａｂに対して挿入されると、ｖｎｆｓのｄａｅｍｏｎは、ずっと単純なコマンドｍｏｕｎｔ ／ｖｎｆｓによって手動でマウントすることができる。

【００８３】

（４）仮想ＮＦＳファイル・システムのアンマウント

クライアント・マシンが仮想ＮＦＳサーバに対するアクセスを終了すると、次のＵＮＩＸコマンドを使用してクライアントのファイル・システムから／ｖｎｆｓディレクトリをアンマウントすることができる。

【００８４】

ｕｍｏｕｎｔ／ｖｎｆｓ

すべてのプロセスが仮想ファイル・システム上で開いていたファイルを閉じていなければならず、そしてどのプロセスも現在のワーキング・ディレクトリとしてマウント・ポイントのサブディレクトリを有していてはならない。ファイル・システムが現在使用中である場合、ｕｍｏｕｎｔコマンドは失敗する。サーバが終了される前に、クライアント・マシンがファイル・システムを正常にアンマウントしなかった場合、それらがその仮想ファイル・システムに対して次にアクセスしようとした時に応答を待ち続けてハングする危険性がある。

【００８５】

（５）ＲＰＣ Ｐｏｒｔｍａｐｐｅｒ Ｄａｅｍｏｎとの相互作用

ｖｎｆｓｄのｄａｅｍｏｎによって実行される最初のタスクは、それが現在、従来のＮＦＳおよびｍｏｕｎｔのｄａｅｍｏｎを実行しているかどうかを知るために局所マシン上のｐｏｒｔｍａｐのｄａｅｍｏｎと交信することである。デフォールトによって、これらのどれもが現在実行中でないことをｐｏｒｔｍａｐのｄａｅｍｏｎが示していた場合、それ

はｐｏｒｔｍａｐのｄａｅｍｏｎと一緒に自分自身を実ＮＦＳおよびｍｏｕｎｔのｄａｅｍｏｎとして局所マシン上に登録する。

【００８６】

サーバが終了する際、サーバはｐｏｒｔｍａｐのｄａｅｍｏｎともう一度交信して、自分がまだそのシステム上の登録されたＮＦＳおよびＭＯＵＮＴのｄａｅｍｏｎであるかどうかを知る。そうであった場合、それは自分自身の登録を解除し、他のプログラムがＮＦＳおよびＭＯＵＮＴサービスがもはやそのサーバ・マシンによっては提供されないことに気付くようにする。

【００８７】

ＲＰＣのｐｏｒｔｍａｐｐｅｒとのこの相互作用は、そのサーバが最初にスタートする時に“－ｎｏｒｅｇ”コマンド・ライン・オプションを使用してオフにすることができる。これはそのサーバ・マシン上で従来のＮＦＳが実行中であることが前もって知られている時に、スタートアップおよび遮断の性能を僅かに改善するはずである。

【００８８】

従来のＭＯＵＮＴのｄａｅｍｏｎが局所マシン上で稼働している可能性があり、クライアントの“ｍｏｕｎｔ（１）”コマンドが”ｍｏｕｎｔｐｏｒｔ”オプションをサポートしない可能性があるので、クライアントからの最初の要求は“ｆｏｒｅｉｇｎ”ファイル・ハンドルを伴うＮＦＳ要求とすることができる。仮想ＮＦＳのｄａｅｍｏｎは、任意の未登録のファイル・ハンドルをそのシステムのｒｏｏｔのファイル・ハンドルとして扱うことによって、この動作モードをサポートする。

【００８９】

（６）総称仮想ファイル・システムのアプリケーション

仮想ファイル・システムのアーキテクチャに対するいくつかの総称アプリケーションもある。これらは以下に説明されるような、圧縮、据置き削除、レビジョン管理およびＮＦＳＷｅｂなどである。

【００９０】

（ａ）“Ｍａｋｅｆｉｌｅ”ファイル・システム

仮想ファイル・システム・アーキテクチャの１つの興味深いバリエーションは、Ｍａｋｅｆｉｌｅファイル・システムである。ＵＮＩＸのｍａｋｅ（１）ユーティリティは、大型のアプリケーションをコンパイルしてリンクするタスクを単純化するプログラム開発ツールである。”ｍａｋｅ”プログラムは、アプリケーションを生成するために必要なコマンドおよびその依存性（すなわち、順序付け情報）のリストをＭａｋｅｆｉｌｅの中に格納する。このアプリケーションによってファイルがどのように生成されるかを指定するための広く受け入れられているメカニズムを使用して、任意の外部プログラムを仮想ファイル・サーバの中に統合化することができる。Ｍａｋｅｆｉｌｅファイル・システムの各ディレクトリは、どのファイルがエクスポートされるか、そしてどのプログラムがそれらの内容を生成することを実行するかを仮想ＮＦＳのｄａｅｍｏｎに知らせるＭａｋｅｆｉｌｅを含んでいる。

【００９１】

（ｂ）圧縮されたファイル・システム

経済的な制約のためにほとんどのパーソナル・コンピュータ・システム上ではポピュラーであるが、ディスクの自動圧縮および自動解凍のソフトウェアは、ＵＮＩＸおよびＶＭＳのシステムにおいては稀である。現在のワークステーションのディスクがずっと大型であり、プロセッサがずっと高速であることは、理論的にはオンザフライでの圧縮および解凍の利点が何倍にもなるはずである。実際に、“Ｓｔａｃｋｅｒ”および”ＤｏｕｂｌｅＳｐａｃｅ”などのＰＣシステムのソフトウェアは、圧縮および解凍のコストがファイルＩ／Ｏの削減より小さいので性能が向上したことを示している。従って、本発明を“ｚｌｉｂ”、“ｇｚｉｐ”または”ｂｚｉｐ”ベースのファイル・システムを提供するために適合させることがさらに考えられる。

【００９２】

（ｃ）据置き削除
ファイル・システムは、クライアントが“ｒｍ”または“ｄｅｌ”コマンドを発行した時点でサーバからファイルを削除または取り除く必要はない。代わりに、そのファイルを“ごみ箱”または“リサイクル”のエリアに、それが自動的に適切な期間経過後に削除されるまで一時的に移動しておくことができる。これによって普通のユーザの誤操作を、その“ごみ箱”からそのファイルを復元することによって訂正することができる。そのようなごみ箱のエリアを圧縮して格納し、そしてディスクの自由空間があるしきい値以下に落ち入った時に自動的に不要部分の整理を行うことができる。
【００９３】
（ｄ）レビジョン管理
ファイル・システムは、特定のファイルの複数のレビジョンまたはバージョンを格納するために使用されることが多い。世代番号に似たものが、ＶＭＳファイル・システムの中のファイル名に対して付加される。一組のファイルの内容が同様であるか、あるいは既存のファイルの修正されたものであることをファイル・システムが知っている場合、仮想ネットワーク・ファイル・サーバは、ファイル全体の内容ではなく、ファイル間の差（または編集）を維持することができる。これによって定期的に変更されるドキュメントの前のバージョンを維持するための大きなディスク空間が必要になるのを避けることができる。
【００９４】
（ｅ）ＮＦＳＷｅｂ
ＷＷＷのクライアントが修正されたＮＦＳプロトコル上でＮＦＳサーバにアクセスすることができる提案されているＷｅｂＮＦＳプロトコルと違って、ＮＦＳＷｅｂは局所ファイル・システム上のファイルを経由してＷＷＷのページがアクセスされるようにすることができる。Ａｎｄｒｅｗファイル・システム（ＡＦＳ）と同様に、“／ｈｔｔｐ／ｗｗｗ・ｍｉｃｒｏｓｏｆｔ．ｃｏｍ／ｉｎｄｅｘ．ｔｈｍ”などのファイルに対するアクセスの結果、適切なＨＴＴＰ要求となり、その返される内容がファイルの内容としてエクスポートされる。
【００９５】
（７）実装の機能の詳細例
この分野に熟達した人によって認識されるように、上記の仮想ファイル・サーバは本質的に開発者が実行時にｖｎｆｓｄのｄａｅｍｏｎによってリンクされる共有のライブラリとして仮想ファイル・システムを実装することができるようにする。ｖｎｆｓｄサーバがすべての低レベルのＴＣＰ／ＩＰベースのソケット、コネクション管理およびＳｕｎ　ＭｉｃｒｏｓｙｓｔｅｍｓのＸＤＲ、ＲＰＣ、ＭＯＵＮＴおよびＮＦＳのｗｉｒｅプロトコルの面倒を見る。このファイル・システムのライブラリは、約１２のサブルーチンのエントリ・ポイントのずっと単純な機能的ＡＰＩを実装するだけで済む。これサブルーチンの多くはファイル・システムのタイプに依存するオプションである。これによって単独ファイルの読取り専用ファイル・システムを、最小限の２つの関数を使用して実装すること、そして複数ファイルの読取り専用ファイル・システムを最小限９個の関数を使用して実装することができる。
【００９６】
前のパラグラフで述べたように、２種類のファイル・システムがある。それらは単独のファイルだけを記述する単独ファイルのファイル・システムおよび、任意の数のファイルおよびサブディレクトリを記述する複数ファイルのファイル・システムである。この２つの間の主な相違点は、ｒｏｏｔのｉｎｏｄｅ、ゼロのｉｎｏｄｅが前者の場合は普通のファイルであり、後者の場合はディレクトリであることである。単独ファイルのファイル・システムは、２つのサブルーチンだけによってプラグインを定義することができる仮想ＮＦＳのＡＰＩの単純化であるが、ｐｌｕｇｉｎｆｓのダイナミック・リンキングを使用しなければならず、ｖｎｆｓｄサーバに対してスタティックにリンクすることができない。
【００９７】
仮想ファイル・システムの開発者は、ファイル・システムの内容を記述するこれらの必要

なサブルーチンの実装を単純に作成する。すべてのサブルーチンが実装される場合、そのファイル・システムのオブジェクトは、ｖｎｆｓｄディレクトリの中にスタティックにリンクされるか、あるいは提供されている”ｐｌｕｇｉｎｆｓ”ファイル・システムによって動的共有オブジェクトとして実行時にリンクされるかのいずれかが可能である。すべての関数が実装されない場合、ｐｌｕｇｉｎｆｓが使用されなければならず、そして欠落しているエントリ・ポイントに対して省略時実装を提供することになる。プロダクションの環境においては、動的にリンクされるｐｌｕｇｉｎｆｓのソリューションが好ましいが、スタティックなリンキングが開発およびデバッグのために役立つ。

【００９８】

ｐｌｕｇｉｎｆｓは共有ライブラリの名前の形式が“＊ｆｓ．ｓｏ”であること、そしてｖｎｆｓｄのプラグインのディレクトリの中に置かれていることが必要である。

【００９９】

（８）ＡＰＩの機能の概要

仮想ネットワーク・ファイル・システムを現在定義している１２個の関数が以下にリストされる。各関数のプロトタイプの前に、実装が必要であるかどうか、特定のタイプのファイル・システムに対して適用可能かどうかを示すキーがある。

【０１００】

‘Ｄ’は、その関数が複数ファイルのファイル・システムに対して必要であることを示す。

‘ｄ’は、その関数が複数ファイルのファイル・システムに対して適用できることを示している。

【０１０１】

‘Ｆ’は、その関数が単独ファイルのファイル・システムに対して必要であることを示している。

‘ｆ’は、その関数が単独ファイルのファイル・システムに対して適用できることを示している。

【０１０２】

```
Df      char* mxINodeInitialize( void );
Df      int mxI NodeIsValid( long inode );
```

【０１０３】

```
Df      int mxINodeType ( long inode );
DF     unsigned long mxINodeFileLen( long inode );
```

【０１０４】

```
DF     int mxINodeFileData( long inode, unsigned char *ptr,
        unsigned long off, int len );
```

【０１０５】

```
D       long  mxINodeDirEntry( long inode, unsigned char
        *ptr, int len );
d       int mxINodeDirSubdirs( long inode );
```

【０１０６】

```
D       int mxINodeDirNext( long inode, int cookie );
D       int mxINodeDirName( long inode, int cookie, unsigned
```

【０１０７】

```
         char *ptr);
D       int mxINodeDirNameLen( long inode, int cookie );
D       long mxINodeDirINode( long inode, int cookie );
```

【０１０８】

```
Df      void mxINodeCleanUp( void );
```

上記のリストから分かるように、単独ファイルのファイル・システムは、通常、２～６個の関数が実装される必要があり、複数ファイルのファイル・システムは、通常、９～１２

個の関数が実装される必要がある。ファイル・システムのプラグインがロードされる時、それは初期化され（ｍｘＩＮｏｄｅＩｎｉｔｉａｌｉｚｅが存在する場合）、そして次にｍｘＩＮｏｄｅＴｙｐｅが、そのファイル・システムのタイプを決定するために呼び出される。ｍｘＩＮｏｄｅＴｙｐｅが存在しなかった場合、そのファイル・システムは単独ファイルのシステムであると仮定される。次に、プラグイン・マネージャが必要な関数のすべてが存在しているかどうかをチエックする。そうでなかった場合、ｍｘＩＮｏｄｅＣｌｅａｎＵｐ関数が呼び出され（存在していた場合）、そしてそのモジュールがアンロードされる。

【０１０９】

仮想ファイル・システムのＡＰＩは、ｉｎｏｄｅおよびディレクトリ・クッキーの概念を使用する。ｉｎｏｄｅは符号付きの３２ビットの長さであり、それはそのファイル・システム上のすべてのファイルおよびディレクトリを識別するために使用される。下位２８ビットだけに意味があり、任意の単独ファイル・システムが最大２６，８００万個の個々に命名されたファイルまたはサブディレクトリをエクスポートすることができる。複数ファイルのファイル・システムは、ｉｎｏｄｅ　０をそのファイル・システムのｒｏｏｔディレクトリとしてマップしなければならず、単独ファイルのファイル・システムはｉｎｏｄｅ　０を仮想ファイルとしてマップしなければならない。ｉｎｏｄｅは１つのファイル・システムの内部でユニークに命名される必要はない。これによっていくつかのファイル（またはディレクトリ）が同じ内容を有することが可能である。

【０１１０】

ディレクトリ・クッキーはディレクトリのエントリのリストの中の論理的な位置を概念的に表す符号付きの３２ビットの整数である。ディレクトリ・クッキーは自分自身のディレクトリに対してだけユニークであればよい。ゼロクッキーは、第１のディレクトリ・エントリの前の位置または最後のエントリの後の位置のいずれかを表している特殊な意味を有する。ゼロクッキーは、“ｍｘＩＮｏｄｅＤｉｒＮｅｘｔ”関数に対してのみ渡されるか、あるいはその関数からのみ返される。ｐｌｕｇｉｎｆｓおよびＵＮＩＸのオペレーティング・システムはディレクトリ・クッキーに追加の制約を課す。決まりによって、クッキー１は局所名“．”を持たなければならず、そして現在のディレクトリを表し、そしてクッキー２は局所名“．．”を持たなければならず、それは現在のディレクトリの親［現在のディレクトリがｒｏｏｔディレクトリである時はそのｒｏｏｔディレクトリ］を持っていなければならない。クッキーはシーケンシャルである必要はなく、そしてゼロクッキーまで任意の順序で“ｍｘＩＮｏｄｅＤｉｒＮｅｘｔ”によって返すことができる。

【０１１１】

（９）サーバのスタートアップおよび遮断の機能

仮想ＮＦＳのＡＰＩは、単独の初期化関数および単独のクリーンアップ関数を含んでいる。これらの関数は両方とも、両方のタイプの仮想ファイル・システムにおいてオプションである。

【０１１２】

ｍｘＩＮｏｄｅＩｎｉｔｉａｌｉｚｅ

ｃｈａｒ＊　ｍｘＩＮｏｄｅＩｎｉｔｉａｌｉｚｅ（ｖｏｉｄ）；

ｍｘＩＮｏｄｅＩｎｉｔｉａｌｉｚｅ関数は、プラグインが必要なデータ構造をそれぞれの初期状態に初期化することができるようにするために使用される。このオプションのルーチンには引数がなく、ヌルで終了されたＣの文字列を返す。この文字列がそのファイル・システムを命名するために使用される。この名前はスタティックにリンクする時には無視されるが、プラグインのファイル・システムを使用している時に、そのファイル・システムを含んでいるディレクトリ名として使用される。慣習的には、これらの名前は“ｆｓ．ｓｏ”サフィックスの前のシステムのライブラリ名のプリフィックスである。例えば、“ｄｅｍｏｆｓ．ｓｏ”の中のデモンストレーション用ファイル・システムは、文字列“ｄｅｍｏ”を返す。この関数は他のすべての関数の前に、ｖｎｆｓｄサーバが最初にスタートアップする時に一度だけしか呼び出されない。

【０１１３】
ｍｘＩＮｏｄｅＣｌｅａｎＵｐ
ｖｏｉｄ　ｍｘＩＮｏｄｅＣｌｅａｎＵｐ（ｖｏｉｄ）；
ｍｘＩＮｏｄｅＣｌｅａｎＵｐ関数は、ｖｎｆｓｄサーバが終了する前に、任意の割り付けられていたメモリまたはシステム資源をプラグインが割当て解除できるようにするために使用される。この関数は引数を取らず、結果を何も返さない。この関数は一度だけしか呼び出されず、そしてその後、ふたたび呼び出される他の関数は何もない。
【０１１４】
（１０）一般の関数
仮想ＮＦＳのＡＰＩは、ｉｎｏｄｅ引数のファイル・システムのタイプとは無関係な２つの関数を含んでいる。従って、これらの関数は、ファイルまたはディレクトリのいずれかである可能性があるｉｎｏｄｅ引数を有している。
【０１１５】
ｍｘＩＮｏｄｅＶａｌｉｄ
ｉｎｔ　ｍｘＩＮｏｄｅＶａｌｉｄ（ｌｏｎｇ　ｉｎｏｄｅ）；
この関数はｉｎｏｄｅの値の有効性をチエックするために使用される。それはそのｉｎｏｄｅを指定している引数として単独のｌｏｎｇを取り、そのｉｎｏｄｅがファイル・システムの内部で有効である場合にはゼロでない整数を返し、そうでない場合はゼロを返す。この戻り値はＣのプログラミング言語におけるＢｏｏｌｅａｎの値の従来の符号化を使用する。ゼロのｉｎｏｄｅは常に有効でなければならない。この関数によって無効であるとしてフラグが立てられているｉｎｏｄｅで他の関数は呼び出されないと仮定するのが安全である。
【０１１６】
この関数は、複数ファイルのシステムに対して実装されなければならない。単独ファイルのファイル・システムは、省略時の行動はゼロのｉｎｏｄｅが唯一の有効なｉｎｏｄｅであることである。
【０１１７】
ｍｘＩＮｏｄｅＴｙｐｅ
ｉｎｔ　ｍｘＩＮｏｄｅＴｙｐｅ（ｌｏｎｇ　ｉｎｏｄｅ）；
この関数は、ファイル・システムのｉｎｏｄｅの「型」を決定するために使用される。それは引数としてそのｉｎｏｄｅを指定している符号付きのｌｏｎｇを取り、インクルード・ファイル“ｆｉｌｅｓｙｓ．ｈ”の中にマクロとして定義されている３つの整数のうちの１つを返す。これらの値は、正規のファイルであるｉｎｏｄｅに対してはＭＸ＿ＩＮＯＤＥＴＹＰＥ＿ＦＩＬＥであり、ディレクトリであるｉｎｏｄｅに対してはＭＸ＿ＩＮＯＤＥ＿ＤＩＲＥＣＴＲＹであり、ｉｎｏｄｅが無効である場合はＭＸ＿ＩＮＯＤＥ＿ＮＯＴＦＯＵＮＤである。
【０１１８】
この関数はゼロのｉｎｏｄｅが常にＭＸ＿ＩＮＯＤＥ＿ＤＩＲＥＣＴＯＲＹを返さなければならない時、複数ファイルのファイル・システムに対して実装されなければならない。単独ファイルのファイル・システムに対して実装された場合、その省略時（そして必須の）実装は、ゼロのｉｎｏｄｅに対してはＭＸ＿ＩＮＯＤＥ＿ＦＩＬＥを返すことである。
【０１１９】
（１１）ファイル関数
仮想ＮＦＳのＡＰＩは、仮想ファイル・システムの内部の正規のファイルにアクセスするための２つの関数を含む。従って、これらの関数はｉｎｏｄｅの引数を有し、それらはファイルでなければならない。
【０１２０】
ｍｘＩＮｏｄｅＦｉｌｅＬｅｎ
ｕｎｓｉｇｎｅｄ　ｌｏｎｇ　ｍｘＩＮｏｄｅＦｉｌｅＬｅｎ（ｌｏｎｇ　ｉｎｏｄｅ）；

ｍｘＩＮｏｄｅＦｉｌｅＬｅｎ関数はファイルの長さを求めるために使用される。この関数はそのファイルのｉｎｏｄｅを表している単独のｌｏｎｇ引数を有し、バイト単位でそのファイルの長さを示している符号なしのｌｏｎｇを返す。この制限は現在各仮想ファイルを４Ｇバイトの最大サイズに制限する。無効であるか、あるいはディレクトリであると決定されるｉｎｏｄｅはゼロの値を返す必要がある。このｍｘＩＮｏｄｅＦｉｌｅＤａｔａ関数によって、そのファイルの指定されたビットのすべてを読むことができなければならない。ｍｘＩＮｏｄｅＦｉｌｅＬｅｎがｍｘＩＮｏｄｅＦｉｌｅＤａｔａによって読み取ることができる大きさより大きい値を返す場合、ほとんどのクライアントは、その仮想ファイルの終りを超えて連続的に読もうとしてループすることになる。

【０１２１】

この関数は両方のタイプのファイル・システムに対して実装されなければならない。

ｍｘＩＮｏｄｅＦｉｌｅＤａｔａ

ｉｎｔ ｍｘＩＮｏｄｅＦｉｌｅＤａｔａ（ｌｏｎｇ ｉｎｏｄｅ，ｕｎｓｉｇｎｅｄ ｃｈａｒ ＊ｐｔｒ，ｕｎｓｉｇｎｅｄ ｌｏｎｇ ｏｆｆ，ｉｎｔ ｌｅｎ）；

【０１２２】

ｍｘＩＮｏｄｅＦｉｌｅＤａｔａ関数は、正規のファイルの内容を呼び出すために使用される。この関数は４つの引数を取る。それらはそのファイルのｉｎｏｄｅを表しているｌｏｎｇ、そのデータを受け取るための符号なし文字のバッファに対するポインタ、読み取るバイトの数を表している整数をそこから読むファイルの内部のオフセットを表している符号なしｌｏｎｇである。この関数は、ｉｎｔｅｇｅｒを返す。それはそのバッファの中に実際に置かれているバイトの数である。そのｉｎｏｄｅが無効であるか、あるいはディレクトリであった場合、この関数はゼロを返し、そのバッファを変更せずにおく必要がある。同様に、そのオフセットがファイルの長さより大きいか、あるいはそれに等しかった場合、この関数はゼロを返し、そのバッファを変更しないままにする。この関数は要求されたバイト数以上のバイトでバッファを決して埋めてはならない。この関数は、そのファイルの中に残っている数以上の文字をバッファの中に決して置いてはならない。その戻り値がゼロより大きかった場合、この値＋そのファイルのオフセットの値は、そのファイルの長さより小さい。

この関数は両方のタイプのファイル・システムに対して実装されなければならない。

【０１２３】

（１２）ディレクトリ関数

仮想ＮＦＳのＡＰＩは、仮想ファイル・システムの中のディレクトリ（またはフォルダ）にアクセスするための６つの関数を含む。従って、これらの関数は、ｉｎｏｄｅの引数を有し、それらはディレクトリでなければならない。これらの関数のうちの４つがディレクトリ・クッキーの引数も有している。これらの関数は、ディレクトリの内容を縦覧するために使用される。

【０１２４】

ｍｘＩＮｏｄｅＤｉｒＥｎｔｒｙ

ｌｏｎｇ ｍｘＩＮｏｄｅＤｉｒＥｎｔｒｙ（ｌｏｎｇ ｉｎｏｄｅ，ｕｎｓｉｇｎｅｄ ｃｈａｒ ＊ｐｔｒ，ｉｎｔ ｌｅｎ）；

ｍｘＩＮｏｄｅＤｉｒＥｎｔｒｙ関数は、ディレクトリのエントリを名前で見つけるために使用される。この関数は３つの引数を取る。それらはそのディレクトリのｉｎｏｄｅを表しているｌｏｎｇ、そのファイル名を保持している文字列に対するポインタおよびそのファイル名の引数の長さを保持しているｉｎｔｅｇｅｒである。この関数は、その見つかったディレクトリ・エントリ（ファイルまたはサブディレクトリ）のｉｎｏｄｅ、またはＭＸ＿ＩＮＯＤＥ＿ＩＮＶＡＬＩＤの値のいずれかを返す。ＭＸ＿ＩＮＯＤＥ＿ＩＮＶＡＬＩＤの値は、そのｉｎｏｄｅの引数が有効なディレクトリのｉｎｏｄｅでない場合、あるいはそのファイル名が見つからない場合に返される必要がある。ファイル名のパラメータは、ヌルで終了されておらず（されていない場合があり）、そしてその実装は指定されたバイト数以上に検査する必要はない。すべての実装はディレクトリ・エントリ“．”お

よび“．．”を認識しなければならない。それらはその引数のｉｎｏｄｅまたはその親のｉｎｏｄｅをそれぞれ返す。実装はディレクトリのクッキーに対応しない「隠された」ファイル名を認識することが許される。

【０１２５】

この関数は複数ファイルのファイル・システムに対して必要である。

ｍｘＩＮｏｄｅＤｉｒＳｕｂｄｉｒｓ

ｉｎｔ　ｍｘＩＮｏｄｅＤｉｒＳｕｂｄｉｒｓ（ｌｏｎｇ　ｉｎｏｄｅ）；

ｍｘＩＮｏｄｅＤｉｒＳｕｂｄｉｒｓ関数は、１つのディレクトリの中のサブディレクトリの数を効率的に求めるために使用される。この関数は検査されるべきディレクトリのｉｎｏｄｅを保持している単独のｌｏｎｇ引数を取り、サブディレクトリのエントリの数を含んでいるｉｎｔｅｇｅｒを返す。引数が有効なディレクトリのｉｎｏｄｅでない場合、あるいはその実装がサブディレクトリの数をレポートしたくない場合、値ゼロを返す必要がある。

【０１２６】

この関数は、オプションであり、複数ファイルのファイル・システムに対してのみ呼び出される。この関数がそのプラグインによって実装されていない場合、省略時の行動は常にゼロを返す。

【０１２７】

ｍｘＩＮｏｄｅＤｉｒＮｅｘｔ

ｉｎｔ　ｍｘＩＮｏｄｅＤｉｒＮｅｘｔ（ｌｏｎｇ　ｉｎｏｄｅ，ｉｎｔ　ｃｏｏｋｉｅ）；

ｍｘＩＮｏｄｅＤｉｒＮｅｘｔ関数は、ディレクトリの内容を縦覧するために使用される。この関数は、２つの引数、すなわち、ディレクトリのｉｎｏｄｅを保持しているｌｏｎｇと、現在のディレクトリのクッキーを保持しているｉｎｔｅｇｅｒとを取り、シーケンスの中の次のクッキーを表しているｉｎｔｅｇｅｒを返す。ｉｎｏｄｅの引数が有効なディレクトリのｉｎｏｄｅでない場合、この関数は値ゼロを返す必要がある。ｖｎｆｓｄサーバは、ゼロクッキーでこの関数を呼び出すことによって、そのシーケンスを開始する。この関数はそのシーケンスの終りを示しているゼロクッキーを返すまで、返されたクッキーで繰返し呼び出される。この関数はゼロクッキーか、あるいは同じディレクトリｉｎｏｄｅでの以前の呼出しによって返されたクッキーのいずれかによってのみ呼び出される必要がある。しかし（プロトコルのために同時並行性のために）、非ゼロのクッキーは、この関数に対する前の呼出しの結果であることは決して保証することはできない。この関数は、指定されたディレクトリに対して認識されていないクッキーの値を使用して提示された場合、値ゼロを返す必要がある。

【０１２８】

実装はクッキーのそれ自身の内部表示を自由に選択することができる。そのシーケンスは、連続的または単調である必要はないが、シーケンスの中で同じ値が処理されてはならない。クッキーのシーケンスは、値１（現在のディレクトリ“．”を表している）および値２（親ディレクトリ“．．”を表している）も含む必要があるが、これらの値は、そのシーケンスの中のどの位置に現れてもよい。

【０１２９】

この関数は複数ファイルのファイル・システムに対して必要である。

ｍｘＩＮｏｄｅＤｉｒＮａｍｅ

ｉｎｔ　ｍｘＩＮｏｄｅＤｉｒＮａｍｅ（ｌｏｎｇ　ｉｎｏｄｅ，ｉｎｔ　ｃｏｏｋｉｅ，ｕｎｓｉｇｎｅｄ　ｃｈａｒ　＊ｐｔｒ）；

ｍｘＩＮｏｄｅＤｉｒＮａｍｅ関数は、ディレクトリの縦覧の間に見つかったディレクトリ・エントリの名前を見つけるために使用される。この関数は３つの引数、すなわち、そのディレクトリのｉｎｏｄｅを保持しているｌｏｎｇ、ディレクトリのクッキーを保持しているｉｎｔｅｇｅｒ、およびそのファイル名を返すためのバッファに対するポインタを取る。この関数は、そのファイル名の長さ（指定されたバッファの中に置かれているバイ

トの数）を含んでいるｉｎｔｅｇｅｒを返す。そのｉｎｏｄｅの引数が有効なディレクトリでないか、あるいはそのクッキーが与えられたディレクトリに対する認められたクッキーでない場合、この関数は値ゼロを返し、バッファを変更されないままにして置く必要がある。２５６バイトを超える数のバイトが指定されたバッファに対して書き込まれてはならない。それはファイル名の最大サイズを制限する。その文字列は、ヌルで終了されていることが許されるが、その戻り値は、その末端のゼロのバイトを含んでいてはならない。このルーチンはゼロクッキーで呼び出されてはならない。

【０１３０】

有効なディレクトリｉｎｏｄｅの場合、クッキー１は結果としてバッファの第１バイトを‘．’に設定して１の値を返す必要があり、そしてクッキー２は結果としてバッファの最初の２バイトを“．．”に設定して値２を返す必要がある。この関数は、複数ファイルのファイル・システムの場合に必要である。

【０１３１】

ｍｘＩＮｏｄｅＤｉｒＮａｍｅＬｅｎ

ｉｎｔ ｍｘＩＮｏｄｅＤｉｒＮａｍｅＬｅｎ（ｌｏｎｇ ｉｎｏｄｅ，ｉｎｔ ｃｏｏｋｉｅ）；

ｍｘＩＮｏｄｅＤｉｒＮａｍｅＬｅｎ関数は、ディレクトリの縦覧の間に見つかったディレクトリ・エントリのファイル名の長さを求めるために使用される。この関数は、ｍｘＩＮｏｄｅＤｉｒＮａｍｅと似ているが、それを呼び出す前に、そのファイル名の長さを求めるために使用される。この関数は２つの引数、すなわち、そのディレクトリのｉｎｏｄｅを保持しているｌｏｎｇおよび、ディレクトリのクッキーを保持しているｉｎｔｅｇｅｒを取り、そのファイル名の長さを含んでいるｉｎｔｅｇｅｒを返す。ｉｎｏｄｅの引数が有効なディレクトリでない場合、あるいはクッキーが与えられたディレクトリの有効なクッキーでない場合、この関数は値ゼロを返す必要がある。この関数は、同じｉｎｏｄｅおよびクッキーの引数の場合はｍｘＩＮｏｄｅＤｉｒＮａｍｅと同じ値を返す。このルーチンはゼロクッキーで呼び出されてはならない。

【０１３２】

有効なディレクトリのｉｎｏｄｅの場合、クッキーの値１が値１を返す必要があり、そしてクッキーの値２が値２を返す必要がある。この関数は複数ファイルのファイル・システムに対して必要である。

【０１３３】

ｍｘＩＮｏｄｅＤｉｒＩＮｏｄｅ

ｌｏｎｇ ｍｘＩＮｏｄｅＤｉｒＩＮｏｄｅ（ｌｏｎｇ ｉｎｏｄｅ，ｉｎｔ ｃｏｏｋｉｅ）；

ｍｘＩＮｏｄｅＤｉｒＩｎｏｄｅ関数は、ディレクトリの縦覧の間に見つかったディレクトリ・エントリのｉｎｏｄｅを見つけるために使用される。この関数は２つの引数、すなわち、そのディレクトリのｉｎｏｄｅを保持しているｌｏｎｇおよびディレクトリのクッキーを保持しているｉｎｔｅｇｅｒを取り、そのファイル名の長さを含んでいるｉｎｔｅｇｅｒを返す。ｉｎｏｄｅの引数が有効なディレクトリでない場合、あるいはそのクッキーがその与えられたディレクトリに対する有効なクッキーでない場合、この関数は値ＭＸ＿ＩＮＯＤＥ＿ＩＮＶＡＬＩＤを返す必要がある。

【０１３４】

有効なディレクトリのｉｎｏｄｅおよびクッキーの値１に対して、この関数は、そのディレクトリのｉｎｏｄｅの引数を返す必要がある。ゼロのｉｎｏｄｅおよびクッキーの値２に対しては、この関数はゼロを返す必要がある。有効な非ゼロ・ディレクトリｉｎｏｄｅおよびクッキーの値２に対しては、この関数はこのディレクトリの親のｉｎｏｄｅを返す必要がある。この関数は複数ファイルのファイル・システムに対して必要である。

【０１３５】

内容を生成し、それを遠隔ファイル・システム要求に応じて転送することができる効率的なデータベース管理を提供するアーキテクチャを備えている仮想ネットワーク・ファイル

・サーバが上記のように提供された。この分野に熟達した人によって、これらおよび他の変更および追加は明らかであり、限定されるものではないが、計算化学およびオブジェクト・データ管理などの各種アプリケーションにおいて使用するために本発明を適合させるように実装できることを理解することができるだろう。本発明は、従来の技術が許す範囲でできるだけ広く、必要な場合は、本明細書の範囲において定義されるべきであることを理解されたい。

【図面の簡単な説明】

【図１】　論理ファイルまたはディレクトリに対してアクセスするための通信フォーマットのブロック図であり、ディレクトリが局所物理記憶媒体上または遠隔マシン上に置かれている。

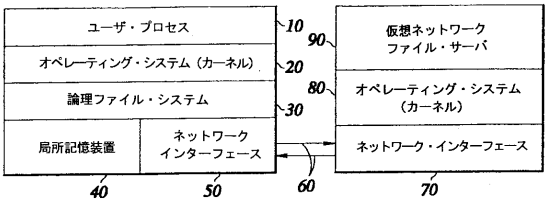
【図２】　仮想ネットワーク・ファイル・サーバがファイル・システム要求を処理して遠隔ファイル・システムに対して返す連続ループのプログラムのフローチャートである。

【図３】　図２のフローチャートを示しており、その中でそのようなプログラムは本発明の仮想ファイル・システムを仮想ネットワーク・ファイル・サーバそのものから隔離することをさらに提供する。

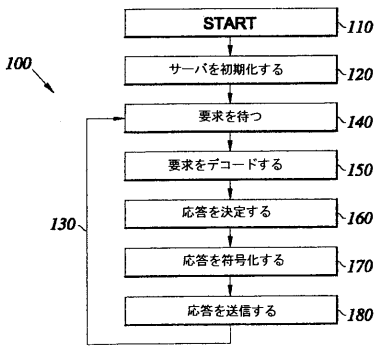
【図４】　遠隔ファイル・システム要求を処理するためにネットワーク・ファイル・サーバによって転送される独立のファイルまたはディレクトリに対して本発明のサーバによって変換される指定された要求の図である。

【図５】　仮想ネットワーク・ファイル・サーバに対して仮想ファイルの内容を生成して返すための仮想ファイル・システム「プラグイン」の読取り機能のためのフローチャートを示している。

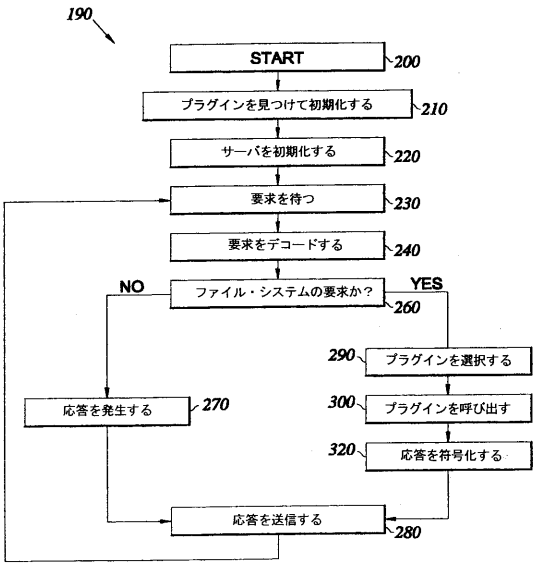
【図１】



【図２】



【図３】



【図４】

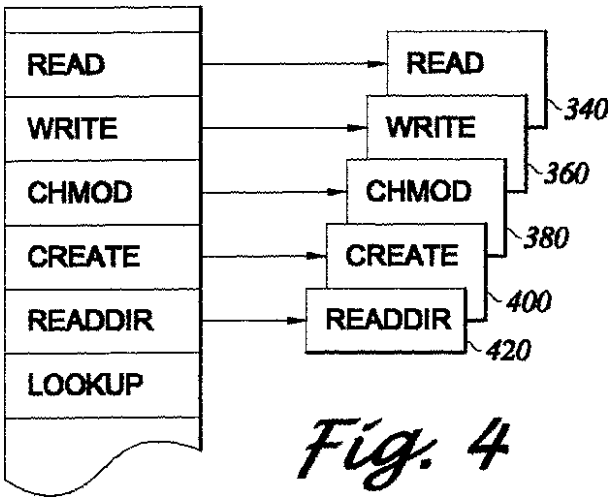
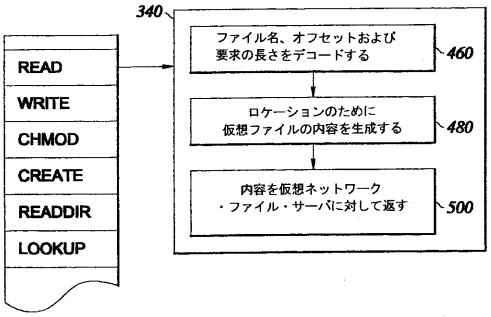

READ
WRITE
CHMOD
CREATE
READDIR
LOOKUP
READ
WRITE
CHMOD
CREATE
READDIR
340
360
380
400
420

Fig. 4

【図５】


READ
WRITE
CHMOD
CREATE
READDIR
LOOKUP
340
ファイル名、オフセットおよび
要求の長さをデコードする
460
ロケーションのために
仮想ファイルの内容を生成する
480
内容を仮想ネットワーク
・ファイル・サーバに対して返す
500

フロントページの続き

(72)発明者 セイル、ロジャー エイ.
アメリカ合衆国 ８７５０１ ニューメキシコ州 サンタフェ カミーノ デローラ １１２８

合議体
審判長 清水 稔
審判官 稲葉 和生
審判官 衣川 裕史

(56)参考文献 特開平８－１３７７２８（ＪＰ，Ａ）
特開平７－６５０３２（ＪＰ，Ａ）
特開平７－１３８４４（ＪＰ，Ａ）
Ａｌｂｅｒｔ Ｄ. Ａｌｅｘａｎｄｒｏｖ ｅｔ ａｌ.，Ｕｆｏ：Ａ Ｐｅｒｓｏｎａｌ Ｇｌｏｂａｌ Ｆｉｌｅ Ｓｙｓｔｅｍ Ｂａｓｅｄ ｏｎ Ｕｓｅｒ－Ｌｅｖｅｌ Ｅｘｔｅｎｓｉｏｎｓ ｔｏ ｔｈｅ Ｏｐｅｒａｔｉｎｇ Ｓｙｓｔｅｍ，ＡＣＭ Ｔｒａｎｓａｃｔｉｏｎｓ ｏｎ Ｃｏｍｐｕｔｅｒ Ｓｙｓｔｅｍｓ，米国，ＡＣＭ，１９９８年8月，Ｖｏｌ. １６，Ｎｏ. ３，ｐ. ２０７－２３３

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F12/00
G06F13/00