(19)日本国特許庁(JP)　(12)公表特許公報(A)　(11)特許出願公表番号
特表2004-533668
(P2004-533668A)
(43)公表日　平成16年11月4日(2004.11.4)

(51)Int.Cl.[7]	F I			テーマコード（参考）
G06F 12/00	G06F	12/00	520E	5B082
G06F 13/00	G06F	12/00	545F	
	G06F	13/00	520D	

審査請求 未請求　予備審査請求 有　（全 88 頁）

(21)出願番号	特願2002-555317(P2002-555317)
(86)(22)出願日	平成13年11月20日(2001.11.20)
(85)翻訳文提出日	平成15年6月30日(2003.6.30)
(86)国際出願番号	PCT/US2001/044883
(87)国際公開番号	WO2002/054286
(87)国際公開日	平成14年7月11日(2002.7.11)
(31)優先権主張番号	09/751,862
(32)優先日	平成12年12月30日(2000.12.30)
(33)優先権主張国	米国(US)

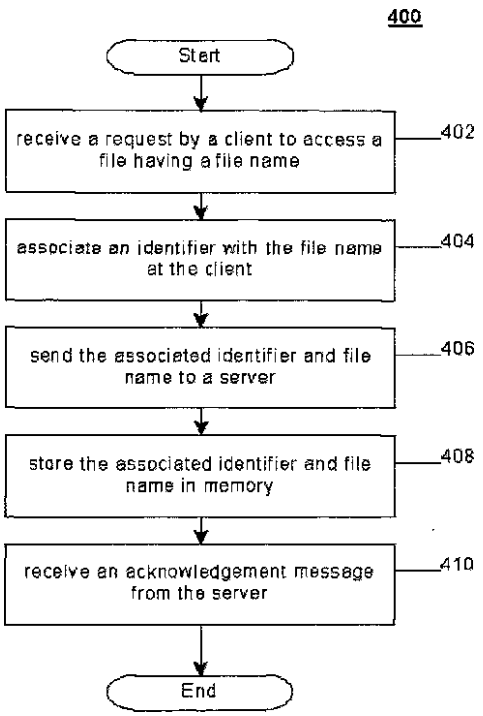
(71)出願人　591003943
インテル・コーポレーション
アメリカ合衆国　９５０５２　カリフォルニア州・サンタクララ・ミッション　カレッジ　ブーレバード・２２００
(74)代理人　100064621
弁理士　山川　政樹
(72)発明者　デイク，スティーヴン
アメリカ合衆国・８５２４８・アリゾナ州・チャンドラー・ウエスト　バートレット　コート・１９７２

Fターム(参考)　5B082 EA07　HA08

最終頁に続く

(54)【発明の名称】ファイル管理を改良するための方法および装置

(57)【要約】
コンピュータ・システムや通信ネットワーク内でファイルを管理するための方法および装置について記述する。



【特許請求の範囲】
【請求項１】
ファイル名を有するファイルにアクセスするためにクライアントで第１の要求を受け取ること、
前記クライアントで識別子を前記ファイル名に関連付けること、および、
前記識別子と前記ファイル名とともに前記第１の要求をサーバに送ること
を含むファイルを管理する方法。
【請求項２】
メモリ内に前記識別子とファイル名を格納することをさらに含む請求項１に記載の方法。
【請求項３】
前記サーバから肯定応答メッセージを受け取ることをさらに含む請求項１に記載の方法。
【請求項４】
前記ファイルにアクセスするために前記クライアントで第２の要求を受け取ること、
前記メモリから前記ファイル名に関連付けられた前記識別子を検索すること、および、
前記関連付けられた識別子を使用して前記第２の要求を前記サーバに送ることをさらに含む請求項１に記載の方法。
【請求項５】
前記第１と第２の要求がファイル・オペレーションを指定する請求項４に記載の方法。
【請求項６】
ファイル名と識別子を有するファイルにアクセスするためにサーバで第１の要求を受け取ること、および、
肯定応答メッセージを前記クライアントに送ること
を含むファイルを管理する方法。
【請求項７】
前記ファイルのための場所情報をサーチすること、
前記場所情報を前記識別子に関連付けること、および、
メモリ内に前記場所情報および前記識別子を格納することをさらに含む請求項６に記載の方法。
【請求項８】
前記識別子を使用して前記ファイルにアクセスするために前記サーバで第２の要求を受け取ること、および、
前記識別子を使用して前記メモリから前記場所情報を検索することをさらに含む請求項７に記載の方法。
【請求項９】
クライアントがファイル名を有するファイル要求を受け取ること、
識別子を前記ファイル名に関連付けること、
前記識別子と前記ファイル名をサーバに送ること、
前記ファイル名を使用して場所情報をサーチすること、および、
前記識別子とともに前記場所情報を格納すること
を含むファイルを管理する方法。
【請求項１０】
肯定応答メッセージを前記クライアントに送ることをさらに含む請求項９に記載の方法。
【請求項１１】
ファイル・システム・インターフェースによりファイル名とともにファイル要求を受け取ること、
前記ファイル・システム・インターフェースにより一意の識別子を前記ファイル名に割り当てること、および、
前記一意の識別子とファイル名をファイル・システム・マネージャに送ること
を含むファイル・オペレーションを管理する方法。
【請求項１２】

前記ファイル・システム・マネージャで前記一意の識別子と前記ファイル名を受け取ること、
前記ファイル名を使用してファイル情報をサーチすること、および、
前記一意の識別子を使用して前記ファイル情報を格納することをさらに含む請求項１１に記載の方法。
【請求項１３】
記憶媒体を備え、
プロセッサによって実行された場合、その結果として、ファイル名を有するファイルにアクセスするためにクライアントで第１の要求を受け取り、前記クライアントで識別子を前記ファイル名に関連付け、前記識別子と前記ファイル名とともに前記第１の要求をサーバに送る格納された命令を前記記憶媒体が有する物品。
【請求項１４】
格納された命令が、プロセッサによって実行された場合、その結果としてさらに、メモリ内に前記識別子とファイル名を格納する請求項１３に記載の物品。
【請求項１５】
格納された命令が、プロセッサによって実行された場合、その結果としてさらに、前記サーバから肯定応答メッセージを受け取る請求項１３に記載の物品。
【請求項１６】
格納された命令が、プロセッサによって実行された場合、その結果としてさらに、前記ファイルにアクセスするために前記クライアントで第２の要求を受け取り、前記メモリから前記ファイル名に関連付けられた前記識別子を検索し、前記関連付けられた識別子を使用して前記第２の要求を前記サーバに送る請求項１３に記載の物品。
【請求項１７】
記憶媒体を備え、
プロセッサによって実行された場合、その結果として、クライアントでファイル名を有するファイル要求を受け取り、識別子を前記ファイル名に関連付け、前記識別子と前記ファイル名をサーバに送り、前記ファイル名を使用して場所情報をサーチし、前記識別子とともに前記場所情報を格納する格納された命令を前記記憶媒体が有する物品。
【請求項１８】
格納された命令が、プロセッサによって実行された場合、その結果としてさらに、前記クライアントに肯定応答メッセージを送る請求項１７に記載の物品。
【請求項１９】
ファイル名を有するファイルのための要求を受け取り、一意の識別子を前記ファイル名に割り当て、前記一意の識別子と前記ファイル名をサーバに送るクライアントと、
前記一意の識別子と前記ファイル名を前記サーバに通信するために前記クライアントに接続された相互接続システムと
を備えたファイル管理を実施する装置。
【請求項２０】
前記一意の識別子とファイル名を受け取るためのサーバをさらに備え、前記サーバが、前記ファイルのための情報を位置付け、前記一意の識別子を使用して前記情報を格納する請求項１９に記載の装置。
【請求項２１】
識別子をファイル名に割り当てるためのクライアントと、
前記ファイル名を使用してファイル情報を位置付け、前記識別子を使用して前記ファイル情報を格納するためのサーバと、
前記クライアントと前記サーバの間で前記ファイル名と前記識別子をトランスポートするための相互接続システムと
を備えたファイル管理を実施する装置。
【請求項２２】
前記クライアントが、オペレーティング・システム・サービス・モジュールを備えた請求

項２１に記載の装置。
【請求項２３】
前記サーバが、中間サービス・モジュールを備えた請求項２１に記載の装置。
【請求項２４】
前記相互接続システムが、周辺構成要素相互接続システムおよびI$_2$Oシステムに従って動作する請求項２１に記載の装置。
【請求項２５】
ファイル名を有するファイルのための要求を受け取り、一意の識別子を前記ファイル名に割り当てるためのファイル・システム・インターフェースと、
前記ファイル名を使用してファイル情報を位置付け、前記一意の識別子を使用して前記ファイル情報を格納するためのファイル・システム・マネージャと、
前記ファイル・システム・インターフェースと前記ファイル・システム・マネージャの間で前記一意の識別子と前記ファイル名を通信するための通信システムと
を備えたファイル管理を実施する装置。
【請求項２６】
前記通信システムが、
ツイストペア線、同軸ケーブル、光ファイバ、無線周波数からなる群のうちの少なくとも１つを有する通信媒体と、
１組の通信プロトコルに従って動作する通信インターフェースとを備えた請求項２５に記載の装置。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、コンピュータ・システムと通信システムに関する。より詳細には、本発明は、コンピュータ・システムや通信システム内でのファイル管理を改良するための方法および装置に関する。
【背景技術】
【０００２】
コンピュータ・システムや通信システムは、アーキテクチャが似ていることが多い。各システムは、それぞれ一定の機能を実施するように設計されているいくつかの別個の構成要素を備えている。構成要素は、相互接続システムを介して他の構成要素に情報を通信できる。相互接続システムは、金属のリード線、ツイストペア線、同軸ケーブル、光ファイバ、無線周波数など通信媒体を介して情報の転送を管理するように動作する。構成要素間の通信は、一般に、個々の構成要素の動作が互いに協調しあい、その結果、それらはまとまりのあるシステムとして動作することができる。このタイプの分散型システム・アーキテクチャは、性能、冗長性、スケーラビリティ、効率の点で一定の利点を提供できる。
【０００３】
しかし、このタイプのシステム設計に関連していくつかの欠点がある。１つの欠点は、ある構成要素が、別の構成要素からの情報を待っているときアイドル状態のままでなければならないことである。アイドル時間はシステム・リソースの非効率的な使用を表す。別の欠点は、構成要素間で通信される情報量を増加するような仕方で、構成要素に機能が割り当てられることがあることである。通信におけるこの増加により、相互接続システムの有限なリソースに対する要求が増加する。
【０００４】
本発明の実施態様とみなされる主題については、本明細書の結論部分で特に指摘し明確に特許請求している。しかし、本発明の実施態様は、オペレーションの構成および方法のどちらに関しても、その目的、特徴、利点と共に添付図面と合わせて以下の詳細な説明を参照すれば、最も良く理解されるであろう。
【発明の開示】
【０００５】

以下の詳細な説明では、本発明の実施態様を完全に理解するために、かずかずの特定の詳細事項を記載している。しかし、これらの特定の詳細事項がなくとも本発明の実施態様を実践できることは、当業者なら理解するであろう。他の例では、本発明の実施態様を不明瞭にしないために、周知の方法、手順、構成要素、回路について詳細に記述していない。

【０００６】

本発明の実施態様は、分散された構成要素のアイドル時間および相互接続システムに対する帯域幅需要を減少させることにより、分散型システムの性能を改善することができる。より具体的には、本発明の一実施態様では、ファイル管理に使用される分散型システムの性能を改善することができる。したがって、これは、ファイル・オペレーションに関連する遅延を減少できる。したがって、ユーザが、より応答性の良いアプリケーションおよびサービスという点で利益を得ることができる。

【０００７】

ファイル管理システムは、一般に、クライアントとサーバを備える。この状況におけるクライアントはサービスの要求側を指すことがある。この状況におけるサーバはサービスのプロバイダを指すことがある。クライアントは、一般に、相互接続システムを介してサーバにファイル要求を送る。ファイル要求はメッセージの形式とすることができ、機能要求とファイル名を有する。この状況におけるメッセージは、組み合わされたとき、たとえば制御語、コマンド、命令、情報またはデータを表す１つまたは複数の英数字、記号または論理式を有することができる。メッセージは、合成されて、たとえば単一のビットからフレーズ全体にまで変わる可能性がある。サーバは、一意の識別子をファイル名に関連付け、ファイルのための場所情報を識別し、それを一意の識別子とともに格納する。次いで、サーバは、一意の識別子をクライアントに送り戻す。クライアントは、一意の識別子を受け取り、後続のファイル要求のためにファイル名の代わりにそれを使用する。

【０００８】

相互接続システムに対する帯域幅需要を減少するために一意の識別子がファイル名に割り当てられる。ファイル名は、たとえば、いくつかの英数字または記号（「文字列」）を有することができる。それぞれの文字または記号が、１つまたは複数のビット、一般に１文字毎に８ビット（すなわち、１バイト）に変換される。１つのファイル名が多くの文字を有するので、相互接続システムは、比較的多数のビットをトランスポートしなければならない。この問題を軽減するために、サーバは、一意の識別子をそれぞれのファイル名に割り当て、その一意の識別子をクライアントに送ることができる。一意の識別子は、一般に長さがファイル名より短い。たとえば、一意の識別子は、３２または６４ビットの長さでよい。次いで、クライアントは、後続のファイル要求のためにその一意の識別子を使用できる。

【０００９】

しかし、サーバが一意の識別子をファイル名に割り当てることについては、いくつかの欠点がある。１つの欠点は、後続のファイル要求を処理するのに先だってクライアントが一意の識別子を待っている間は、そのクライアントはアイドル状態のままでなければならないことである。別の欠点は、クライアントとサーバが、必要以上に多くの情報を通信する必要があり、それにより相互接続システムに対する要求が増加することがあることである。

【００１０】

第１の欠点に関して、クライアントは、サーバから一意の識別子を待っている時はアイドル状態のままでなければならない。割当てプロセスを開始するために、ファイル名への一意の識別子の割当てを要求してクライアントはサーバにメッセージを送る。サーバは、一意の識別子をファイル名に関連付け、ファイルのための場所情報を識別し、それを一意の識別子とともに格納する。次いで、サーバは、一意の識別子を有するメッセージをクライアントに送り戻す。サーバから一意の識別子を受け取るのを待っている間、クライアントは、同じファイル名を有する後続のファイル要求を受け取ることができる。クライアントは、全割当てプロセスが完了するまで、これらのファイル要求の処理を開始できないこと

がある。その結果、クライアントは、この時間中アイドル状態のままであることを強いられることがある。

【００１１】

第２の欠点について、クライアントとサーバは、割当て機能を実行するために不必要な情報を通信する必要があり、それにより相互接続システムに対する要求が増加する場合がある。上述したファイル管理システムは少なくとも２つのメッセージを必要とする。クライアントは、ファイル名への一意の識別子の割当てを要求するサーバに第１のメッセージを送る。サーバは、第２のメッセージを一意の識別子を有するクライアントに送る。それぞれのメッセージが、相互接続システムから一定量の帯域幅の使用を必要とする。この状況における帯域幅とは、相互接続システムを介して情報を転送できる速度を指すことがあり、一般に、キロビット／秒（ｋｂｐｓ）で測定される。これと対称的に、本発明の一実施態様では、ただ１つのメッセージを使用して割当てプロセスを実施し、それにより、恐らく相互接続システムに対する帯域幅需要を５０％も減少することができるようになる。

【００１２】

本発明の実施態様では、クライアントのためのアイドル時間を減少させ、かつ相互接続システムに対する帯域幅需要を減少させることにより、分散型システムの性能を改善することができる。本発明の一実施態様では、クライアントで一意の識別子をファイル名に割り当て、その一意の識別子をサーバに送る。このことにより、クライアントがサーバからのメッセージを待つ必要がなく後続のファイル要求の処理を開始できるので、クライアントのアイドル時間を減少させることができる。このことによりまた、サーバが割当てプロセスを完了するためにメッセージをサーバに送り戻す必要がないか、あるいは以前のファイル管理システムが必要としていたメッセージより短いメッセージを送ることができるので、帯域幅需要を減少できる。

【発明を実施するための最良の形態】

【００１３】

本明細書において「一実施形態」または「実施形態」について記載している場合、この状況においては、本発明の少なくとも１つの実施形態内にその実施形態と合わせて記述した特定の機能、構造、または特性を有することができることを意味するということに留意されたい。本明細書のさまざまな箇所に現れる「一実施形態では」というフレーズは、必ずしもすべて同じ実施形態を指すものではない。

【００１４】

次に、全体を通じて同様の部分を同様の参照番号で表した図面を詳細に参照していくが、図１に、本発明の一実施形態を実践するのに適切なシステム１００が示されている。図１に示されるように、システム１００には、相互接続システム１０４によって接続されたクライアント１０２とサーバ１０６が備えられている。本明細書で使用する用語「クライアント」は、情報の任意の要求側を指し、本明細書で使用する用語「サーバ」は、情報の任意のプロバイダを指すことがある。

【００１５】

図２は、本発明の一実施形態によるクライアント・システムを示すブロック図である。図２は、クライアント１０２であってよいクライアント２００を例示する。図２に示されるように、クライアント２００には、プロセッサ２０２、メモリ２０４、インターフェース２０８が備えられ、すべて接続２１０によって接続されている。メモリ２０４は、プログラム命令とデータを格納できる。用語「プログラム命令」は、定義済み方式またはシンタックスに従って組み合わせられた場合に、プロセッサに一定の機能を実施させる、定義済みコンピュータ言語からの語、値、記号を含むコンピュータ・コード・セグメントを有する。コンピュータ言語の例には、Ｃ、Ｃ＋＋、アセンブリがある。プロセッサ２０２は、メモリ２０４内に格納されたプログラム命令を実行し、データを処理する。インターフェース２０８は、クライアント２００から別のデバイスへのデータのトランスポートを調整する。接続２１０が、プロセッサ２０２、メモリ２０４、インターフェース２０８の間でデータをトランスポートする。

【００１６】
プロセッサ２０２は、本発明のさまざまな実施形態に望ましい速度と機能を提供することが可能な任意のタイプのプロセッサでよい。たとえば、プロセッサ２０２は、インテル社、モトローラ、コンパック、またはサン・マイクロシステムズ製のプロセッサ・ファミリのプロセッサなどである。本発明の一実施形態では、プロセッサ２０２は、ハード・ドライブ、キーボード、プリンタ、ネットワーク・インターフェース・カードなどの入出力（Ｉ／Ｏ）デバイスを管理するための専用プロセッサでもよい。そのプロセッサは、一般に、Ｉ／ＯプロセッサＩＯＰ）と呼ばれる。

【００１７】
本発明の一実施形態では、メモリ２０４は、機械読取り可能媒体であり、プロセッサによって実行されるよう適合された命令を格納することが可能な任意の媒体を含むことができる。このような媒体の例には、読取専用メモリ（ＲＯＭ）、ランダムアクセス・メモリ（ＲＡＭ）、プログラマブルＲＯＭ、消去可能なプログラマブルＲＯＭ、電子的に消去可能なプログラマブルＲＯＭ、ダイナミックＲＡＭ、磁気ディスク（たとえば、フロッピ・ディスクおよびハード・ドライブ）、光学ディスク（たとえば、ＣＤ－ＲＯＭ）、ディジタル情報を格納できる他の任意の媒体を含むが、それらに限定されるものではない。本発明の一実施形態では、命令は、圧縮されたかつ／または暗号化されたフォーマットで媒体に格納される。本明細書で使用するフレーズ「プロセッサによって実行されるよう適合された」は、圧縮されたかつ／または暗号化されたフォーマットで格納された命令を含むと共に、プロセッサによって実行される前にインストーラによってコンパイルまたはインストールすべき命令をも含むことを意味する。さらに、クライアント２００が、プロセッサ２０２からアクセス可能でありかつコンピュータ・プログラム命令およびデータの組み合わせを格納することが可能な、さまざまなＩ／Ｏコントローラを通じて機械読取り可能記憶装置デバイスのさまざまな組み合わせを備えることができる。

【００１８】
メモリ２０４が、クライアント１０６やクライアント２００などのクライアントの機能を実施するためのプログラム命令とデータを格納し、かつプロセッサ２０２による実行を可能にしている。本発明の一実施形態では、メモリ２０４は、本明細書では全体としてファイル・システム・インターフェース２０６と呼ばれる１組のプログラム命令を有する。

【００１９】
ファイル・システム・インターフェース２０６は、システム１００のために１つまたは複数のファイルへのアクセスを提供するよう動作するインターフェースである。この状況におけるインターフェースは定義済みプロトコルを指すことがあり、それにより、１つのソフトウェア・モジュールが、別のソフトウェア・モジュールからの機能にアクセスできる。この状況におけるファイルは、メモリ２０４やハード・ドライブなどのメモリ内に格納された個別の１組のデータを指す。ファイル・システム・インターフェース２０６は、作成する、開ける、シークする、読み取る、書き込む、名称変更する、削除する、コピーする、移動するなどの、ファイルのための一定のオペレーションを実施するための要求を受け取ることができる。要求は、たとえば、ホストＯＳまたはアプリケーション・プログラムから発信できる。ホストＯＳはシステムのためのＯＳからなる。たとえば、本発明の一実施形態がパーソナル・コンピュータの一部として実施される場合は、ホストＯＳは、たとえばＭｉｃｒｏｓｏｆｔ　Ｗｉｎｄｏｗｓ（登録商標）９５、９８、２０００、ＮＴなどの、マイクロソフト社より販売されているＯＳを備えることがある。

【００２０】
本発明の一実施形態では、ファイル・システム・インターフェース２０６が、I_2O分科会（ＳＩＧ）（I_2OＳＩＧ）によって開発された、「ｗｗｗ／ｉ２０ｓｉｇ．ｏｒｇ」（「I_2O規格」）から使用可能なインテリジェントＩ／Ｏ規格（I_2O）よって定義されているオペレーティング・システム・サービス・モジュール（ＯＳＭ）の、１９９７年４月に採用されたバージョン１．５にとして動作するが、本発明はこの点に範囲が限定されるものではない。

【００２１】
背景として、I_2O規格は、制御されている特定のデバイスとホスト・オペレーティング・システム（ＯＳ）の両方から独立したインテリジェントＩ／Ｏのための標準アーキテクチャを定義する。この状況におけるインテリジェントＩ／Ｏは、低レベルの割込みを処理する機能を、中央処理ユニット（ＣＰＵ）または他のプロセッサからＩ／Ｏ機能のための処理を提供するために特に設計されたＩ／ＯプロセッサへＩＯＰ）へ移動することを指す。このことにより、Ｉ／Ｏ性能を改良でき、ＣＰＵまたは他のプロセッサが他のタスクのための機能の処理を提供できるようになる。この状況における割込みは、ハード・ドライブ、フロッピ・ディスク・ドライブ、プリンタ、モニタ、キーボード、ネットワーク・インターフェース・カード（ＮＩＣ）などのＩ／Ｏデバイスにアクセスするための要求を指す。

【００２２】
I_2O規格は、ＯＳＭ、中間サービス・モジュール（ＩＳＭ）、ハードウェア・デバイス・モジュール（ＨＤＭ）を記述する。ＯＳＭは、ホストＯＳとＩＳＭの間をインターフェースとして動作するドライバである。この状況におけるドライバは、特定の構成要素、デバイス、またはソフトウェア・モジュールのオペレーションを管理する１組のプログラム命令を指す。ＩＳＭは、ＯＳＭとハードウェア・デバイス・モジュール（ＨＤＭ）の間のインターフェースとして動作できる。ＩＳＭは、たとえばバックグラウンド・アーカイブなどの、Ｉ／Ｏ管理機能、ネットワーク・プロトコル、またはピアツーピア機能のための特定の機能を実行できる。ＨＤＭは、特定のＩ／Ｏデバイスを制御するよう動作するドライバよい。

【００２３】
I_2O規格は、メッセージ受渡しシステムを備えた通信モデルを定義する。ＯＳＭ、ＩＳＭ、ＨＤＭは通信し、メッセージ・レイヤを通じてメッセージの形式で情報を渡すことにより、オペレーションを調整する。この状況におけるメッセージ・レイヤは、要求を管理しディスパッチし、メッセージを配信するために１組のアプリケーション・プログラミング・インターフェース（ＡＰＩ）を提供し、メッセージを処理するために１組のサポート・ルーチンを提供する。

【００２４】
本発明の一実施形態では、ファイル・システム・インターフェース２０６が、I_2O規格に従ってＯＳＭとして動作する。本発明の一実施形態では、ファイル・システム・インターフェース２０６が、ホストＯＳを介してアプリケーション・プログラムからファイル要求を受け取り、I_2O規格に従ってその要求をメッセージに翻訳し、処理するためにそれをファイル・システム・マネージャ（以下に記述する）に送る。この状況におけるアプリケーション・プログラムは、一般に、ユーザとコンピュータ・システムの間のコマンドと命令の処理を促進するためのユーザ・インターフェースを有する専用タスクのための所定の組の機能を提供するプログラムを指す。アプリケーション・プログラムの例には、ワード・プロセッサ、スプレッド・シート、データベース、またはインターネット・ブラウザがある。

【００２５】
インターフェース２０８は、たとえば、一組の所望の通信プロトコル、サービス、オペレーティング手順を使用してコンピュータまたはネットワーク・デバイス間の通信信号を制御するための適切な技術を有する。本発明の一実施形態では、インターフェース２０８が、たとえば、ＰＣＩ規格およびI_2O規格に従って動作できる。本発明の別の実施形態では、インターフェース２０８が、１９８１年９月に採用された、インターネット技術タスク・フォース（ＩＥＴＦ）標準７、コメント要求（ＲＦＣ）７９３によって定義された伝送制御プロトコル（ＴＣＰ）、および１９８１年９月に採用された、ＩＥＴＦ標準５、ＲＦＣ７９１によって定義されたインターネット・プロトコル（ＩＰ）（両方とも「ｗｗｗ．ｉｅｔｆ．ｏｒｇ」から使用可能）に従って動作できる。インターフェース２０８は上述したプロトコルに従って動作できるが、インターフェース２０８は、たとえば、一組の

所望の通信プロトコル、サービス、オペレーティング手順を使用してコンピュータまたはネットワーク・デバイス間の通信信号を制御するための任意の適切な技術とともに動作でき、かつなお本発明の範囲内に包含されることが理解できるであろう。

【００２６】

また、インターフェース２０８はインターフェース２０８を適切な通信媒体に接続するためのコネクタを備えることもできる。インターフェース２０８は、銅リード線、ツイストペア線、同軸ケーブル、光ファイバ、無線周波数などの任意の適切な媒体を介して通信信号を受け取ることができる。本発明の一実施形態では、コネクタは、ＰＣＩ規格に準拠する信号を運ぶためのバスとともに使用するのに適している。

【００２７】

図３は、本発明の一実施形態によるサーバ・システムを示すブロック図である。図３は、サーバ１０６を表す本発明の一実施形態によるサーバ３００を例示する。図３に示されるように、サーバ３００は、すべて接続３１０によって接続されているプロセッサ３０２、メモリ３０４、インターフェース３０８を備える。図３の素子、３０２、３０４、３０８、３１０は、図２を参照しながら記述した対応する素子、２０２、２０４、２０８、２１０と構造およびオペレーションの点で同様である。サーバ３００がプロセッサ３０２とともに示されているが、サーバ３００は、プロセッサ３０２なしでも、サーバ３００で使用可能な別のプロセッサ（たとえば、プロセッサ２０２）を使用して動作でき、かつなお本発明の範囲内に包含されることが理解できるであろう。たとえば、本発明の実施形態が、クライアントとサーバがＰＣＩバスによって接続され、両方が単一のプロセッサを共用しているパーソナル・コンピュータ内に組み込まれた場合にこのような構成が生じることがある。

【００２８】

本発明の一実施形態では、メモリ３０４が、ファイル・システム・マネージャ３０６のためのプログラム命令を有する。ファイル・システム・マネージャ３０６は、ファイル管理を実施し、複数のファイルを有する記憶媒体（図示せず）へのアクセスを提供する。ファイル・システム３０６は、ファイル・システム・インターフェース２０６から受け取ったファイル要求に応答して、作成する、開ける、シークする、読み取る、書き込む、名称変更する、削除する、コピーする、移動するなどのファイル・オペレーションを実施する。ファイル・システム・インターフェース２０６の一例が、I_2O規格によるＩＳＭオペレーティングを含むが、本発明の範囲がこの点に限定されるものではない。

【００２９】

システム、１００、２００、３００のオペレーションを、図４、５を参照しながらより詳細に記述する。本明細書に提示する図４、５は特定のオペレーションのシーケンスを有するが、そのオペレーションのシーケンスは、本明細書に記述する一般的な機能をどのように実施できるかという例を提供するためにすぎないことが理解できるであろう。さらに、別段の表示がない場合は、そのオペレーションのシーケンスを必ずしも提示された順序で実行する必要はない。

【００３０】

図４は、本発明の一実施形態によるクライアントによって実施されるオペレーションを示すブロック流れ図である。本発明のこの実施形態では、ファイル・システム・インターフェース２０６はクライアント１０６の一部として動作する。しかし、ファイル・システム・インターフェース２０６は、コンピュータまたはネットワーク・システム内のどこかに置かれた任意のデバイス、またはデバイスの組み合わせに実装でき、かつなお本発明の範囲内に包含されることが理解できるであろう。

【００３１】

図４に示されるように、クライアントが、ブロック４０２でファイル名を有するファイルにアクセスするための要求を受け取る。クライアントは、ブロック４０４でファイル名を識別子に関連付ける。クライアントは、ブロック４０６で関連付けられた識別子とファイル名をサーバに送る。クライアントは、ブロック４０８で関連付けられた識別子とファイ

ル名をメモリ内に格納する。クライアントは、ブロック４１０でサーバから肯定応答メッセージを受け取る。

【００３２】

ひとたびクライアントがファイルに識別子を割り当てると、識別子は、そのファイルの将来の要求のために使用される。クライアントは、ファイルにアクセスするためにクライアントで第２の要求を受け取る。クライアントは、メモリからファイル名に関連付けられた識別子を検索する。クライアントは、関連付けられた識別子を使用してサーバに第２の要求を送る。

【００３３】

図５は、本発明の一実施形態によるサーバによって実施されるオペレーションを示すブロック流れ図である。本発明のこの実施形態では、ファイル・システム・マネージャ３０６はクライアント１０６の一部として動作する。しかし、この機能は、コンピュータまたはネットワーク・システム内のどこかに置かれた任意のデバイス、またはデバイスの組み合わせに実装でき、かつなお本発明の範囲内に包含されることが理解できるであろう。

【００３４】

図５に示されるように、サーバが、ブロック５０２でファイルのためのファイル名と関連付けられた識別子を受け取る。サーバは、ブロック５０４で肯定応答メッセージをクライアントに送る。サーバは、ブロック５０６でファイルのための場所情報をサーチする。サーバは、ブロック５０８で場所情報を識別子に関連付ける。サーバは、関連付けられた場所情報と識別子をメモリ内に格納する。

【００３５】

ひとたびサーバが識別子を使用してファイルのための場所情報に索引を付けると、サーバは、識別子を使用して、後続のファイル要求のために場所情報にアクセスできる。サーバは、識別子を有するファイルにアクセスするために第２の要求を受け取る。サーバは、識別子を使用してメモリから場所情報を検索する。

【００３６】

システム１００、２００、３００のオペレーションと、図４、５に示されたフロー・ダイアグラムを例にとると、より良く理解できるであろう。アプリケーション・プログラムが、ファイル名「ｔｅｓｔ　ｆｉｌｅ　ｏｎｅ」を有するファイルからシステム１００のホストＯＳに情報を読み取るための要求を送る。この状況における一意の識別子は、クライアント、サーバおよび／またはシステムによって使用される他の語、値、または２進列に関して、クライアント、サーバおよび／またはシステムに対して、組み合わされたとき一意の語、値、または２進列を表す一連の英数字を指す。ホストＯＳは、ファイル要求をファイル・システム・インターフェース２０６に渡す。ファイル・システム・インターフェース２０６が、一意の識別子「Ａ１２３」を生成し、それをファイル名「ｔｅｓｔ　ｆｉｌｅ　ｏｎｅ」に割り当てる。本発明のこの実施形態では、一意の識別子「Ａ１２３」は、たとえば１６進数の３２ビット数である。ファイル・システム・インターフェース２０６が、メッセージ「識別する（ｔｅｓｔ　ｆｉｌｅ　ｏｎｅ、Ａ１２３）」を作成し、それを接続１０４を介してファイル・システム・マネージャ３０６にトランスポートするためのアウトバウンド・メッセージ待ち行列内に入れる。この状況におけるアウトバウンド・メッセージ待ち行列は、相互接続システムがメッセージをトランスポートできるようになるまで、メッセージを保持するのに使用される先入れ先出し法（ＦＩＦＯ）などの待ち行列を指す。ファイル・システム・インターフェース２０６は、「Ａ１２３」とともに「ｔｅｓｔ　ｆｉｌｅ　ｏｎｅ」をメモリ２０４内のルックアップ・テーブル内に格納する。

【００３７】

ファイル・システム・マネージャ３０６が接続１０４を介してメッセージを受け取る。ファイル・システム・マネージャ３０６は、メッセージをパーズし、機能「識別する（ｔｅｓｔ　ｆｉｌｅ　ｏｎｅ、Ａ１２３）」を起動する。この状況における用語「パーズする」は、たとえば、コマンド、制御語、ファイル名、データ、関数呼出し、サブルーチン名

、フラグなどを表すメッセージから個々の文字またはサブセットの文字を分離することを指す。この状況における用語「起動する」は、所与の機能またはサブルーチンに関連付けられたプログラム命令の実行を開始するためにプロセッサに送られるコマンドを指す。この機能は、入力として「ｔｅｓｔ　ｆｉｌｅ　ｏｎｅ」および「Ａ１２３」をとり、ファイル・システム・マネージャ３０６にファイル名「ｔｅｓｔ　ｆｉｌｅ　ｏｎｅ」が後続のファイル要求で「Ａ１２３」として参照されることを通知する。このことは、ファイル名と一意の識別子を、対応するまたはリンクされた項としてともに格納することにより、メモリ内のルックアップ・テーブルを更新することによって達成できるであろう。つまり、一方をサーチすることにより他方を見つけることができるであろう。ファイル・システム・マネージャ３０６は、一般にハード・ドライブなどの記憶装置デバイスに置かれているファイル「Ａ１２３」のための場所情報をサーチする。場所情報の例には、アドレス指定情報、デバイス、シリンダ番号、トラック番号があるが、本発明はこの点に範囲が限定されるものではない。ファイル・システム・マネージャ３０６が、場所情報を識別子「Ａ１２３」に関連付け、メモリ３０４内のルックアップ・テーブル内に識別子「Ａ１２３」とともに場所情報を格納する。ファイル・システム・マネージャ３０６は、ファイル名識別子と場所情報を受け取ったファイル・システム・インターフェース２０６に肯定応答メッセージを送る。この状況における肯定応答メッセージは、以前のメッセージが受け取ったことを示す短いメッセージを指す。肯定応答メッセージは、特定のシステムが所望する通りに、単一のビット、文字、語、またはフレーズを有することができる。

【００３８】

次いで、ファイル・システム・マネージャ３０６によって受け取られた後続のファイル要求が、ファイル名「ｔｅｓｔ　ｆｉｌｅ　ｏｎｅ」のためのオペレーションを要求する場合、識別子「Ａ１２３」を使用する。たとえば、ファイル・システム・インターフェース２０６が、「ｔｅｓｔ　ｆｉｌｅ　ｏｎｅ」に対して「削除する」オペレーションを実施するために第２の要求を受け取る。ファイル・システム・インターフェース２０６は、メモリから「ｔｅｓｔ　ｆｉｌｅ　ｏｎｅ」のために以前に関連付けられた識別子「Ａ１２３」を検索する。ファイル・システム・インターフェース２０６は、メッセージ「削除する」（　「Ａ１２３」）をファイル・システム・マネージャ３０６に送る。ファイル・システム・マネージャ３０６は、そのメッセージ「削除する」（　「Ａ１２３」）を受け取り、識別子「Ａ１２３」を使用してファイル名「ｔｅｓｔ　ｆｉｌｅ　ｏｎｅ」のための場所情報を検索する。次いで、ファイル・システム・マネージャ３０６が、検索された場所情報を使用して要求されたファイル・オペレーションを実施する。

【００３９】

本発明の一実施形態では、ファイル・システム・インターフェース２０６を、「ｗｗｗ．ｋｅｒｎｅｌ．ｏｒｇ．」（「Ｌｉｎｕｘ　Ｋｅｒｎｅｌ」）より使用可能なＬｉｎｕｘ ＯＳバージョン２．３．９９ｐｒｅ３カーネルなどの、I_2O規格および特定のホストＯＳに従ってＯＳＭとして実施できる。本発明のこの実施形態では、ＯＳＭは、ストリーム・フォレストＯＳＭ、ストリーム・ツリーＯＳＭ、クラス規格をさらに有することができる。この状況におけるクラスは、特定のインターフェース定義を指すことがある。この状況におけるツリーは、コーンと呼ばれるストレージ・オブジェクトの集まりを指す。この状況におけるオブジェクトはあるクラスのインスタンスを指す。この状況におけるコーンは、たとえば、読取り、書込み、ロックのケイパビリティをサポートするストレージ・オブジェクトを指す。この状況におけるフォレストはツリーの集まりを指す。

【００４０】

本発明のこの実施形態では、ストリーム・フォレストＯＳＭを使用して、ファイル・システムの集まりをモデル化できる。ストリーム・フォレストＯＳＭは、たとえば、名称を作成し、名称を削除し、または特定のフォレストを名称変更するなどのファイル命名オペレーションを提供できる。さらに、開閉オペレーションもサポートできる。ツリーＯＳＭを使用して、特定のファイル・システムをモデル化できる。ファイルは、コーンとしてモデル化できる。ストリーム・フォレストＯＳＭは、たとえば、ファイルに名称を付け、それ

を名称変更し、それを削除し、変更できないようそれをロックし、それを読み取り、またはそれを書き込むなどのファイル・オペレーションをサポートするよう機能する。

【００４１】

ＯＳＭは、ＩＳＭと通信するよう設計できる。ＩＳＭは、ストリーム・フォレストＩＳＭおよびストリーム・ツリーＩＳＭをさらに含むこともある。ストリーム・フォレストＩＳＭは、ＯＳＭのファイル・システム命名ケイパビリティをサポートできる。本発明の一実施形態では、ストリーム・ツリーＩＳＭは、2^8文字の長さを有するストリーム・コーン識別子をサポートできる。ストリーム・ツリーＩＳＭはまた、2^{16}オープン・ストリーム・コーンと2^{32}の可能なストリーム・コーンもサポートできる。ツリーＩＳＭは、含まれるすべてのストリーム・コーンのための2^{64}バイトをサポートできる。これらの値により本発明の範囲がこの点に限定されるものではないことが理解できるであろう。

【００４２】

オペレーションでは、ストリーム・ツリーＯＳＭ、ストリーム・フォレストＯＳＭ、ストリーム・ツリーＩＳＭ、ストリーム・フォレストＩＳＭはすべて、以下に説明するクラス規格に従ってメッセージング・スキームを使用して通信できる。このメッセージング・スキームは、ストリーミング・メッセージング・スキームと呼ばれ、以下にさらに詳細に説明する。ホストＯＳが、ストリーム・フォレストＯＳＭおよびストリーム・ツリーＯＳＭによって提供される機能を使用できる。ホストＯＳは、たとえば、ストリーム・フォレストＯＳＭを使用して１群のファイル・システムをモデル化し、ストリーム・ツリーＯＳＭを使用して特定のファイル・システムをモデル化できる。ストリーム・フォレストＯＳＭは、ストリーム・フォレストＩＳＭを使用して、ＩＲＴＯＳ環境内で複数の群のファイル・システムを管理できる。ストリーム・ツリーＯＳＭは、ストリーム・ツリーＩＳＭを使用して、I_2Oリアルタイム・オペレーティング・システム（ＩＲＴＯＳ）環境内でファイル・システムを管理できる。ストリーム・フォレストＯＳＭとストリーム・ツリーＯＳＭは、ストリーム・メッセージング・スキームを使用して、それぞれ、ストリーム・フォレストＯＳＭとストリーム・ツリーＩＳＭと通信できる。本発明のこの実施形態については、図７を参照しながらさらに記述する。

【００４３】

図６は、本発明の一実施形態によるソフトウェア・アーキテクチャを示す図である。図６に示されるように、システム６００は、ストリーム・フォレストＯＳＭ６０２、ＬｉｎｕｘＯＳ６０４、Ｌｉｎｕｘ仮想ファイル・システム（ＶＦＳ）６０６、ストリーム・ツリーＯＳＭ６０８、１つまたは複数のＬｉｎｕｘI_2Oドライバ６１０を含むファイル・システム・インターフェースを備えることができる。ＬｉｎｕｘI_2Oドライバ６１０が、ＬｉｎｕｘカーネルとI_2O規格に従って動作できる。ＬｉｎｕｘI_2Oドライバ６１０は$PCI I_2O$ドライバ６１２を含むことができる。$PCI I_2O$ドライバ６１２はＰＣＩ規格およびI_2O規格に従って動作できる。

【００４４】

ファイル・システム・インターフェースは、ＰＣＩ規格に従って信号を通信するバス６１４を介してファイル・システム・マネージャと通信する。ファイル・システム・マネージャは、ストリーム・フォレストＩＳＭ６１６、ストリーム・ツリーＩＳＭ６１８、レイド（ＲＡＩＤ：Ｒｅｄｕｎｄａｎｔ Ａｒｒａｙ ｏｆ Ｉｎｅｘｐｅｎｓｉｖｅ Ｄｉｓｋｓ）ＩＳＭ６２０、スカジ（ＳＣＳＩ：Ｓｍａｌｌ Ｃｏｍｐｕｔｅｒ Ｓｙｓｔｅｍ Ｉｎｔｅｒｆａｃｅ）ＨＤＭ６２２、ＩＲＴＯＳ６２４を有することができる。

【００４５】

本発明のこの実施形態では、システム６００内に示されるモジュールは、Ｃプログラミング言語を使用して実装できるが、本発明の範囲がこの点に限定されるものではない。さらに、本発明のこの実施形態では、システム６００内に示されるモジュールが、２５５文字の長さを有する識別子をサポートする。

【００４６】

ストリーム・ツリーＯＳＭ６０８を、代表的なファイル・システムへのアクセスを提供す

るよう構成できる。ストリーム・ツリーＯＳＭ６０８を、Ｌｉｎｕｘ規格によって定義される機能に必要な１つまたは複数のＬｉｎｕｘＶＦＳをサポートするよう構成できる。ストリーム・ツリーＯＳＭ６０８を、ストリーム・ツリー・クラス・メッセージをサポートするよう構成できるが、これについては以下により詳細に説明する。ストリーム・ツリーＯＳＭ６０８を、たとえば、Ｌｉｎｕｘカーネルを使用してＬｉｎｕｘＯＳ６０４とともに動作するよう構成できる。ストリーム・ツリーＯＳＭ６０８が、ストリーム・ツリーＩＳＭ６１８などのストリーム・ツリーＩＳＭをサポートできる。

【００４７】

ストリーム・フォレストＯＳＭ６０２は、機能ｌｏｃｔｌ（）Ｋｅｒｎｅｌ　Ｉｎｔｅｒｆａｃｅを提供できる。ストリーム・フォレストＯＳＭ６０２が、「ＩＯＣＴＬ＿ＳＦ＿ＴＲＥＥＣＲＥＡＴＥ」などの機能名を使用して、ストリーム・フォレスト内でストリーム・ツリーを作成するよう機能できる。この機能は、以下に示すように、たとえばＣ言語で定義されている入力バッファを、入力として受け入れることができる。

【００４８】

【表１】

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

また、ストリーム・フォレストＯＳＭ６０２が「ＩＯＣＴＬ＿ＳＦ＿ＴＲＥＥＲＥＮＡＭＥ」などの機能名を使用して、既存ストリーム・ツリーを名称変更するよう機能できる。この機能は、以下に示すように、たとえばＣ言語で定義されている入力バッファを、入力として受け入れることができる。

【００４９】

【表２】

```
struct ik_sf_treerename {
        char    IdentifierBeforeRename [256];
        char    IdentifierAfterRename [256];
}.
```

また、ストリーム・フォレストＯＳＭ６０２が、「ＩＯＣＴＬ＿ＳＦ＿ＴＲＥＥＥＲＡＳＥ」などの機能名を使用して、既存ストリーム・ツリーを消去するよう機能できる。この機能は、以下に示すように、たとえばＣ言語で定義されている入力バッファを、入力として受け入れることができる。

【００５０】

【表３】

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

ストリーム・フォレストＯＳＭ６０２が、以下のデータ構造のうちの１つまたは複数を使用できる。本明細書で「スーパー・ブロック」と呼ぶデータ構造を使用して、ファイル・システムの初期「ｉノード」を表すことができる。この状況における「ｉノード」は、ファイル・システムのファイル・ノードを指すことがある。スーパー・ブロック・オペレーション・リストは、スーパー・ブロックを操作するために使用される１群の機能であり得る。ｉノード・レコード・データ構造を使用して、ファイル・システムのそれぞれのファイル・ノードを表すことができる。ｉノード・オペレーション・リストは、ｉノードを操作するために使用される１群の機能である。ファイル・レコード・データ構造を使用して、ファイル・システム内の抽象化ファイルを表すことができる。ファイル・オペレーション・リストを使用して、ファイル・オペレーションをサポートできる。オペレーション・

リストと組み合せたこれらのデータ構造のオペレーションのすべてを使用して、ファイル・システムを表すことができる。

【００５１】

データ構造のいくつかのために、ストリーム・ツリー・クラス・メッセージを発行できる。たとえば、作成する、ルックアップする、ディレクトリを作成する（「ｍｋｄｉｒ」）、ディレクトリを除去する（「ｒｍｄｉｒ」）、ディレクトリを名称変更するのｉノード・オペレーションのために、ストリーム・ツリー・クラス・メッセージを発行できる。開ける、読み取る、書き込む、シークする、閉じるなどのさまざまなファイル・オペレーションのために、ストリーム・ツリー・クラス・メッセージを発行できる。

【００５２】

ストリーム・フォレストＯＳＭ６０２が、たとえば識別子「ｉ２ｏｆｓ」とともに含まれているファイル・システムをＬｉｎｕｘカーネル内に登録できる。ストリーム・フォレストＯＳＭ６０２が、たとえばルート・ツリー・コーン・コンテナのためのＮＵＬＬ識別子を使用できる。ストリーム・フォレストＯＳＭ６０２が、たとえば現在のツリー・コーン・コンテナのための識別子「．」を使用できる。ストリーム・フォレストＯＳＭ６０２が、たとえば親ツリー・コーン・コンテナのための識別子「．．」を使用できる。

【００５３】

システム６００は、以下のモジュールのうちの１つまたは複数のためのシステム記述を有することができる。

１．　カーネルのためのＭＯＤＵＬＥ＿ＡＵＴＨＯＲ記述

２．　「Ｉ２Ｏ　Ｆｉｌｅｓｙｓｔｅｍ　Ｏｆｆｌｏａｄ　Ｄｒｉｖｅｒ」のＭＯＤＵＬＥ＿ＤＥＳＣＲＩＰＴＩＯＮ

３．　ファイル・システムを登録できるモジュール＿ｉｎｉｔ（）

４．　ファイル・システムの登録を削除できるモジュール＿ｅｘｉｔ（）

５．　ｉ２ｏｆｓ＿ｔｙｐｅｉ２ｏｆｓ、ｒｅａｄ　ｓｕｐｅｒ　ｏｐｅｒａｔｉｏｎ、０のパラメータを有することのできるＤＥＣＬＡＲＥ＿ＦＳＴＹＰＥモジュール

６．　使用されるハンドル毎に１つずつ増加するゼロの第１のハンドルを有することのできるＯＳＭハンドル・マネージャ・モジュール

ａ．　未使用のハンドルを検索できる内部ソフトウェア・インターフェースＡｃｑｕｉｒｅＨａｎｄｌｅ（）

ｂ．　現在使用されているハンドルをフリー・プール内にリリースできる内部ソフトウェア・インターフェースＲｅｌｅａｓｅＨａｎｄｌｅ（ｉｎｔ）

【００５４】

また、システム６００は機能ＶＦＳｉ２ｏｆｓ＿ｒｅａｄ＿ｓｕｐｅｒ（ｓｔｒｕｃｔ　ｓｕｐｅｒ＿ｂｌｏｃｋ＊ｓｂ，ｖｏｉｄ＊ｏｐｔｉｏｎｓ，ｉｎｔ　ｓｉｌｅｎｔ）も提供できる。この機能は、表１で定義された入力を受け入れることができる。

【００５５】

【表４】

表１

変数タイプ	変数識別子	記述
Struct super_block*	Sb	この入力は、スーパー・ブロックを設定すべき場所を指定できる。
Void*	Options	この入力は、マウントにより渡されるオプションを指定できる。
int	Silent	この入力は、マウント・オペレーションがサイレントであるかどうかを指定できる。

この機能は、スーパー・ブロック構造内の値を０に設定する。以下の変数を、以下の通り

に設定できる。

sb-〉s_blocksize=512

sb-〉s_blocksize_bits=10.

sb-〉S_s_magic=I2OFS_SUPER_MAGIC

sb-〉s_op=OSM内で定義されるsuper_operations構造

機能は、ｇｅｔ＿ｅｍｐｔｙ＿ｉｎｏｄｅ（）と呼ばれるモジュールを使用して新しいｉノードを作成できる。ｉノードは、以下の設定を有する関連付けられたさまざまな変数とともに、ゼロに設定できる。

inode i_uid=20.

inode i_gid=20.

iノード・オペレーションを、iノード・オペレーション・リストを記述するポインタに設定できる。

iノードi_fopをファイル・オペレーション・リストを記述するポインタに設定できる。

iノードI_modeをS_IFDIR|S-IRUGO|S_IXUGOに設定できる。

iノードをiノード・ハッシュ・テーブル内に挿入できる。

Sb-＞s_rootをルートiノードのd_alloc_root()に設定できる。

ＶＦＳ６０６内のそれぞれのファイル・システムが、少なくとも１つのスーパー・ブロックを有することができ、これにはファイル・システムのアクティビティを開始するためのファイル・システムについての十分な情報が含まれる。このスーパー・ブロックは、以下の通りにＣ構造ストラクトｓｕｐｅｒ＿ｂｌｏｃｋ内に実装できる。

【００５６】

【表５】

```
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;
};
```

変数ｓｕｐｅｒ＿ｂｌｏｃｋ－＞ｓ＿ｏｐは、以下のＣ関数を有し、スーパー・ブロックへのアクセスを提供できる。

【００５７】

【表６】

```
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ノード・インターフェースの例は、以下の通りである。

【００５８】

【表７】

```
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;
        Usngined char I_sock;
        Atomic_t I_writecount;
        Unsigned int I_attr_flags;
        __u32 I_generation;
 }.
```

変数ｉｎｏｄｅ－＞ｉ＿ｏｐは、以下のＣ関数を有し、以下の通りにスーパー・ブロックへの方法アクセスを提供できる。

【００５９】

【表８】

```
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) (sturct 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 *);
};
```

【００６０】

また、システム６００は機能ｃｒｅａｔｅ（ｓｔｒｕｃｔ　ｉｎｏｄｅ＊，ｃｏｎｓｔ　ｃｈａｒ＊，ｉｎｔ，ｉｎｔ，ｓｔｒｕｃｔ　ｉｎｏｄｅ＊＊）も提供する。この機能は、表２に記載された入力を受け入れることができる。

【００６１】

【表９】

表２

変数タイプ	変数識別子	記述
struct inode*	ParentDirectory	この入力は、createオペレーションの入力ディレクトリを指定できる。
const char*	NewName	この入力は、新しいファイル・システム・オブジェクトの名称を指定できる。
int	NewSize	この入力は、オブジェクトの新しいサイズを指定できる。
Int	NewMode	この入力は、オブジェクトの新しいモードを指定できる。
struct inode**	NewInode	この入力は、作成されたオブジェクトの新しいiノードを指定できる。

この機能は、ｇｅｔ＿ｅｍｐｔｙ＿ｉｎｏｄｅ（）を使用して新しいｉノードを作成できる。機能は、ｄｅｎｔｒｙ関係を介してｃｏｎｓｔ　ｃｈａｒ＊変数をそれにアタッチすることにより新しいｉノードを初期化できる。機能は、指定されたサイズを有する新しいｉノード構造を初期化できる。機能は、指定されたモードを有する新しいｉノード構造を初期化できる。機能は、ＳｔｒｅａｍＣｏｎｅＣｒｅａｔｅメッセージをストリーム・ツリーＩＳＭに送ることができる。メッセージは、以下の通りに構成できる。

１．　ＨＡＮＤＬＥを、ＯＳＭハンドル・マネージャから検索できる。

２．　ＴＹＰＥを、ファイルを示す１に設定できる。

３．　ＳＧＬを、ＮｅｗＮａｍｅに設定できる。

ＮｅｗＩｎｏｄｅのデリファレンスは、新しいｉノード構造とともに設定できる。

【００６２】
また、システム６００は機能ｓｔｒｕｃｔ　ｄｅｎｔｒｙ＊ｌｏｏｋｕｐ（ｓｔｒｕｃｔ　ｉｎｏｄｅ＊，ｓｔｒｕｃｔ　ｄｅｎｔｒｙ＊）を提供できる。この機能は、表３に記載された入力を受け入れることができる。
【００６３】
【表１０】

表３

変数タイプ	変数識別子	記述
struct inode*	ParentDirectory	この入力は、ルップアップ・オペレーションの入力ディレクトリを指定できる。

この機能は、ＳｔｒｅａｍＣｏｎｅＩｄｅｎｔｉｆｙメッセージをストリーム・ツリーＩＳＭ６１８に送ることができる。メッセージは、以下の通りに構成できる。
１．　ＰＡＲＥＮＴＨＡＮＤＬＥを、ＰａｒｅｎｔＤｉｒｅｃｔｏｒｙｉノード・ローカル・データ内で識別されるハンドルに設定できる。
２．　ＣＨＩＬＤＨＡＮＤＬＥを、ＯＳＭハンドル・マネージャから検索されるハンドルに設定できる。
３．　ＡＴＴＲＩＢＵＴＥＳを、ＳＴＲＥＡＭＣＯＮＥＣＲＥＡＴＥ＿ＱＵＥＲＹに設定できる。
４．　ＳＧＬを、Ｎａｍｅに設定できる。
機能は、ｇｅｔ＿ｅｍｐｔｙ＿ｉｎｏｄｅ（）を使用して新しいｉノードを作成できる。機能は、ｄｅｎｔｒｙ関係を介してｃｏｎｓｔ　ｃｈａｒ＊変数をそれにアタッチすることにより新しいｉノードを初期化できる。機能は、ＳｔｒｅａｍＣｏｎｅＧｅｔＩｎｆｏｒｍａｔｉｏｎメッセージをストリーム・ツリーＩＳＭに送ることができる。メッセージは、以下の通りに構成できる。
１．　ＨＡＮＤＬＥを、上記のＣＨＩＬＤＨＡＮＤＬＥに設定できる。
２．　ＳＧＬを、タイプＩｎｆｏｒｍａｔｉｏｎＲｅｓｕｌｔＢｌｏｃｋのローカル・データ構造に設定できる。
機能は、ＩｎｆｏｒｍａｔｉｏｎＲｅｓｕｌｔＢｌｏｃｋからのｉノード値を設定できる。機能は、作成されたｉノードにＮｅｗＩｎｏｄｅ変数のデリファレンスを設定できる。
【００６４】
システム６００は、機能ｍｋｄｉｒ（ｓｔｒｕｃｔ　ｉｎｏｄｅ＊，ｓｔｒｕｃｔ　ｄｅｎｔｒｙ＊，ｉｎｔ）を提供できる。この機能は、表４に記載された入力を受け入れることができる。
【００６５】
【表１１】

表４

変数タイプ	変数識別子	記述
struct inode*	ParentDirectory	この入力は、mkdirオペレーションの入力ディレクトリを指定できる。
Struct dentry*	Name	この入力は、作成するためのオブジェクトの名称を指定できる。
int	Mode	この入力は、新しいディレクトリのモードを指定できる。

この機能は、ディレクトリを作成するためのＳｔｒｅａｍＣｏｎｅＣｒｅａｔｅメッセージを送ることができる。メッセージは、以下の通りに構成される。
１．　ＨＡＮＤＬＥを、ｉノード構造入力ＰａｒｅｎｔＤｉｒｅｃｔｏｒｙ内で識別され

るハンドルに設定できる。
２．　ＴＹＰＥを、２に設定できる。
３．　ＳＧＬを、実際の名称を有する入力Ｎａｍｅに設定できる。
【００６６】
システム６００は、機能ｒｍｄｉｒ（ｓｔｒｕｃｔ　ｉｎｏｄｅ＊，ｓｔｒｕｃｔ　ｄｅｎｔｒｙ＊）を提供できる。この機能は、表５に記載された入力を受け入れることができる。
【００６７】
【表１２】

表５

変数タイプ	変数識別子	記述
struct inode*	ParentDirectory	この入力は、ルックアップ・オペレーションの入力ディレクトリを指定できる。
struct dentry*	Name	この入力は、除去するためのオブジェクトの名称を指定できる。
int	Length	この入力は、名称の長さを文字で指定できる。

この機能は、ディレクトリを除去するためのＳｔｒｅａｍＣｏｎｅＥｒａｓｅメッセージを送ることができる。メッセージは、以下の通りに構成される。
１．　ＨＡＮＤＬＥを、ｉノード構造入力ＰａｒｅｎｔＤｉｒｅｃｔｏｒｙ内で識別されるハンドルに設定できる。
２．　ＳＧＬを、入力Ｎａｍｅに設定できる。
【００６８】
また、システム６００は機能ｒｅｎａｍｅ（ｓｔｒｕｃｔ　ｉｎｏｄｅ＊，ｓｔｒｕｃｔ　ｄｅｎｔｒｙ＊，ｓｔｕｒｃｔ　ｉｎｏｄｅ＊，ｓｔｒｕｃｔ　ｄｅｎｔｒｙ＊）を提供できる。この機能は、表６に記載された入力を受け入れることができる。
【００６９】
【表１３】

表６

変数タイプ	変数識別子	記述
struct inode*	OldDir	この入力は、名称変更するためのファイルのディレクトリを指定できる。
struct dentry*	OldName	この入力は、名称変更するためのオブジェクトの名称を指定できる。
struct inode*	NewDir	この入力は、オブジェクトの新しいディレクトリを指定できる。
struct dentry*	NewName	この入力は、オブジェクトの新しい名称を指定できる。

この機能は、ＳｔｒｅａｍＣｏｎｅＩｄｅｎｔｉｆｙメッセージをＩＳＭに送ることができる。メッセージは、以下の通りに構成される。
１．　ＰＡＲＥＮＴＨＡＮＤＬＥを、ＯＳＭハンドルのためのＯｌｄＤｉｒｉノード内部記憶装置から設定できる。
２．　ＣＨＩＬＤＨＡＮＤＬＥを、ＯＳＭハンドル・マネージャから検索される新しいハンドルに設定できる。
３．　ＡＴＴＲＩＢＵＴＥＳを、ＳＴＲＥＡＭＣＯＮＥＣＲＥＡＴＥ＿ＱＵＥＲＹに設定できる。

４．　ＳＧＬを、ＯｌｄＮａｍｅに設定できる。
機能は、ＳｔｒｅａｍＣｏｎｅＲｅｎａｍｅメッセージをＩＳＭに送ることができる。メッセージは、以下の通りに構成される。
１．　ＨＡＮＤＬＥを、ＳｔｒｅａｍＣｏｎｅＩｄｅｎｔｉｆｙのＣＨＩＬＤＨＡＮＤＬＥに設定できる。
２．　ＮＥＷＰＡＲＥＮＴを、ＯＳＭハンドルのためのＮｅｗＤｉｒｉノード内部記憶装置に設定できる。
３．　ＳＧＬを、ＮｅｗＮａｍｅに設定できる。
機能は、ＳｔｒｅａｍＣｏｎｅＣｌｏｓｅメッセージをＩＳＭに送ることができる。メッセージは、以下の通りに構成される。ＨＡＮＤＬＥを、ＳｔｒｅａｍＣｏｎｅＲｅｎａｍｅメッセージから以前のＨＡＮＤＬＥに設定できる。

【００７０】
また、システム６００は機能ｔｒｕｎｃａｔｅ（ｓｔｒｕｃｔ　ｉｎｏｄｅ＊）を提供できる。この機能は、表７に記載された入力を受け入れることができる。

【００７１】
【表１４】

表７

変数タイプ	変数識別子	記述
struct inode*	File	この入力は、トランケートするためのファイルを指定できる。

機能は、ＳｔｒｅａｍＣｏｎｅＲｅｓｉｚｅメッセージをＩＳＭに送ることができる。メッセージは、以下の通りに構成される。
１．　ＨＡＮＤＬＥを、ｉノード内部ＯＳＭハンドル記憶装置から検索できる。
２．　ＳＩＺＥを、ｉノード変数ｉ＿ｓｉｚｅから検索できる。
ＶＦＳ６０６内のそれぞれのファイルが、少なくとも１つのスーパー・ブロックを有することができ、これにはファイル・システムのアクティビティを開始するためのファイル・システムについての十分な情報が含まれる。このスーパー・ブロックは、以前に記述したものなどのＣ構造ｓｔｒｕｃｔ　ｓｕｐｅｒ＿ｂｌｏｃｋ内で詳述される。方法変数、ｆ＿ｏｐは、ファイル・オペレーションへのアクセスを提供できる。ｓｔｒｕｃｔ　ｓｕｐｅｒ＿ｏｐｅｒａｔｉｏｎｓ＊ｓ＿ｏｐ機能は、ＯＳＭのために所望されることのあるルートｉノードへのアクセスを提供できる。

【００７２】
【表１５】

```
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;
};
```

システム６００は、以下に記述する１つまたは複数のファイル・オペレーションを提供できる。変数ｆ－＞ｓ＿ｏｐは、以下のＣ関数を有することができ、スーパー・ブロックへのアクセスを提供できる。

【００７３】

【表１６】

```
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 *);
};
```

【００７４】

システム６００は、機能ｌｌｓｅｅｋ（ｓｔｒｕｃｔ　ｆｉｌｅ＊，ｏｆｆ＿ｔ，ｉｎｔ）を提供できる。この機能は、表８に記載された入力を受け入れることができる。

【００７５】

【表１７】

表８

変数タイプ	変数識別子	記述
struct file*	File	この入力は、シークするためのファイル・ポインタを指定できる。
off_t	Offset	この入力は、シークするためにOriginからオフセットを指定できる。
Int	Origin	この入力は、Originを指定できる。

この機能は、ＳｔｒｅａｍＣｏｎｅＳｅｅｋメッセージをＩＳＭに送ることができる。メッセージは、以下の通りに構成される。
１．　ＨＡＮＤＬＥを、ＯＳＭハンドルのためのノード内部ハンドル記憶装置から設定できる。
２．　ＮＥＷＰＯＳＩＴＩＯＮを、ＯｆｆｓｅｔおよびＯｒｉｇｉｎから計算によって得られた位置に設定できる。
【００７６】
システム６００は、機能ｒｅａｄ（ｓｔｒｕｃｔ　ｆｉｌｅ＊，ｃｈａｒ＊，ｓｉｚｅ＿ｔ，ｌｏｆｆ＿ｔ）を提供できる。この機能は、表９に記載された入力を受け入れることができる。
【００７７】
【表１８】

表９

変数タイプ	変数識別子	記述
struct file*	File	この入力は、読み取るためのファイル・ポインタを指定できる。
char*	Buffer	この入力は、読み取るためバッファを指定できる。
Size_t*	Size	この入力は、読み取るためのバイトのサイズを指定できる。
loff_t*	SeekChange	この入力は、どれだけのデータが読み取られたかを指定できる。

この機能は、ＳｔｒｅａｍＣｏｎｅＲｅａｄメッセージをＩＳＭに送ることができる。メッセージは、以下の通りに構成される。
１．　ＨＡＮＤＬＥを、ＯＳＭハンドルのためのノード内部ハンドル記憶装置から設定できる。
２．　ＳＧＬを、ＢｕｆｆｅｒおよびＳｉｚｅに設定できる。
【００７８】
システム６００は、機能ｗｒｉｔｅ（ｓｔｒｕｃｔ　ｆｉｌｅ＊，ｃｏｎｓｔ　ｃｈａｒ＊，ｓｉｚｅ＿ｔ，ｌｏｆｆ＿ｔ）を提供できる。この機能は、表１０に記載された入力を受け入れることができる。
【００７９】
【表１９】

表１０

変数タイプ	変数識別子	記述
struct file*	File	この入力は、読み取るためのファイル・ポインタを指定できる。
Const char*	Buffer	この入力は、書き込むためのバッファを指定できる。
Size_t*	Size	この入力は、読み取るためのバイトのサイズを指定できる。
Loff_t*	SeekChange	この入力は、どれだけのデータが書き込まれたかを指定できる。

この機能は、ＳｔｒｅａｍＣｏｎｅＷｒｉｔｅメッセージをＩＳＭに送ることができる。メッセージは、以下の通りに構成される。
１．　ＨＡＮＤＬＥを、ＯＳＭハンドルのためのノード内部記憶装置から設定できる。
２．　ＳＧＬを、ＢｕｆｆｅｒおよびＳｉｚｅに設定できる。
【００８０】
システム６００は、機能ｒｅａｄｄｉｒ（ｓｔｒｕｃｔ　ｆｉｌｅ＊，ｖｏｉｄ＊，ｆｉｌｌｄｉｒ＿ｔ）を提供できる。この機能は、表１１に記載された入力を受け入れることができる。
【００８１】
【表２０】

表１１

変数タイプ	変数識別子	記述
struct file*	File	この入力は、読み取るためのファイル・ポインタを指定できる。

この機能は、ＳｔｒｅａｍＣｏｎｅＥｎｕｍｅｒａｔｅメッセージをＩＳＭに送ることができる。メッセージは、以下の通りに構成される。
１．　ＨＡＮＤＬＥを、ＯＳＭハンドルのためのノード内部記憶装置から設定できる。
２．　ＥＮＵＭＥＲＡＴＯＲを、Ｃｏｕｎｔに設定する。
３．　ＳＧＬを、Ｅｎｔｒｙのファイル名バッファに設定できる。
４．　ＳＧＬのサイズを２５５に設定できる。
【００８２】
また、システム６００は以下の通りにｄｅｎｔｒｙインターフェースを提供できる。
【００８３】
【表２１】

```
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];
}.
```

【００８４】
ストリーム・ツリーＩＳＭ６１８が、本明細書に定義するように、１つまたは複数のストリーム・ツリー・クラス・メッセージをサポートできる。ストリーム・ツリー６１８が機能し、ストリーム・ツリーＯＳＭ６０８などのストリーム・ツリーＯＳＭからのメッセージをサポートできる。ストリーム・フォレストＩＳＭ６１６が、本明細書に定義するように、ストリーム・フォレスト・クラス・メッセージをサポートできる。ストリーム・フォレストＩＳＭ６１６が機能し、ストリーム・フォレストＯＳＭ６０２などのストリーム・フォレストＯＳＭからのメッセージをサポートできる。
【００８５】
ストリーム・フォレストＩＳＭ６１６が、ユーザ・インターフェースを有することができる。ユーザ・インターフェースは、初期設定で１つまたは複数のストリーム・ツリーを表示できるユーザ・スクリーンを有することができる。ユーザが、新しいストリーム・ツリーを作成できる。新しいストリーム・ツリーを作成する場合に、ユーザには、サイズと識別を尋ねるスクリーンが提示される。ストリーム・ツリーが作成される場合は、ストリーム・ツリー・クラス・オブジェクトが作成される。ユーザはまた、ストリーム・ツリーを消去することもできる。ユーザ・インターフェースは、ストリーム・ツリーの消去の確認を提供できる。適宜、ストリーム・フォレスト・メッセージを取り扱い、ツリー・オブジェクトを作成することができる。
【００８６】
システム６００は、メッセージＳｔｒｅａｍＣｏｎｅＣｒｅａｔｅを提供できる。このメッセージを使用して、情報を格納する新しいストリーム・コーンを作成でき、表１２に記載された入力を受け入れることができる。
【００８７】
【表２２】

表１２

変数タイプ	変数識別子	記述
U32	HANDLE	この入力は、閉じるべきハンドルを指定できる。
U32	TYPE	この入力は、作成すべきタイプを指定できる。1はファイルを示し、2はディレクトリを示す。
SGL	SGL	識別子を指定する。

【００８８】
また、システム６００はメッセージＳｔｒｅａｍＣｏｎｅＥｎｕｍｅｒａｔｅを提供できる。このメッセージを使用して、ストリーム・コーン・コンテナ内に１つまたは複数のストリーム・コーンをリストでき、表１３に記載された入力を受け入れることができる。
【００８９】
【表２３】

表１３

変数タイプ	変数識別子	記述
U32	HANDLE	この入力は、閉じるべきハンドルを指定できる。
U32	ENUMERATOR	この入力は、どのエントリをSGL内にエニュメレートすべきかを示すゼロベースのインデックスを指定できる。
SGL	SGL	エニュメレートされたストリーム・コーン識別子の場所を指定する。

【００９０】
システム６００は、ハンドル・マネージャからＨＡＮＤＬＥに関係するｉノードを検索できる。エニュメレータが０に設定されているとき、親としてハンドルに関係するｉノードとともに「ｅｘｔ２ｆｓ」ライブラリ呼出しｄｉｒ＿ｉｔｅｒａｔｅ（）を使用して、「ｅｘｔ２」ファイルシステム上にストリーム・コーン識別子のリストを生成することができる。ＥＮＵＭＥＲＡＴＯＲインデックスに対応するリスト・エントリをＳＧＬ内にコピーできる。
【００９１】
システム６００は、メッセージＳｔｒｅａｍＣｏｎｅＥｒａｓｅを提供できる。このメッセージを使用して、ストリーム・コーン識別子を消去でき、表１４に記載された入力を受け入れることができる。
【００９２】
【表２４】

表１４

変数タイプ	変数識別子	記述
U32	HANDLE	この入力は、閉じるべきハンドルを指定できる。
SGL	SGL	この入力は、消去すべき識別子の名称を指定できる。

【００９３】
システム６００は、メッセージＳｔｒｅａｍＣｏｎｅＧｅｔＩｎｆｏｒｍａｔｉｏｎを提

供できる。このメッセージを使用して、ストリーム・コーンについての情報を検索でき、表１５に記載された入力を受け入れることができる。

【００９４】

【表２５】

表１５

変数タイプ	変数識別子	記述
U32	HANDLE	この入力は、情報をそこから検索すべきハンドルを指定できる。
SGL	SGL	この入力は、消去すべき識別子の名称を指定できる。

【００９５】

システム６００は、メッセージＳｔｒｅａｍＣｏｎｅＩｄｅｎｔｉｆｙを提供できる。このメッセージを使用して、ストリング識別子に対してハンドル識別子をマッピングでき、表１６に記載された入力を受け入れることができる。

【００９６】

【表２６】

表１６

変数タイプ	変数識別子	記述
U32	PARENT HANDLE	この入力は、親ハンドル・ストリーム・コーン・コンテナを指定できる。
U32	CHILD HANDLE	この入力は、ストリーム・コーン識別子にマッピングすべき子ハンドルを指定できる。
U32	ATTRIBUTES	この入力は、識別されたストリーム・コーンの属性を指定できる。
SGL	SGL	この入力は、識別子の名称を指定できる。

【００９７】

システム６００は、メッセージＳｔｒｅａｍＣｏｎｅＬｏｃｋを提供できる。このメッセージを使用して、１バイトの範囲のファイルをロックでき、表１７に記載された入力を受け入れることができる。

【００９８】

【表２７】

表１７

変数タイプ	変数識別子	記述
U32	HANDLE	この入力は、親ハンドル・ストリーム・コーン・コンテナを指定できる。
U64	INDEX	この入力は、ロックすべきバイト・インデックスを指定できる。
U64	SIZE	この入力は、ロックすべきバイトのサイズを指定できる。

【００９９】

システム６００は、メッセージＳｔｒｅａｍＣｏｎｅＲｅａｄを提供できる。このメッセージを使用して、ストリーム・コーンからブロックを読み取ることができ、表１８に記載

された入力を受け入れることができる。
【０１００】
【表２８】

表１８

変数タイプ	変数識別子	記述
U32	HANDLE	この入力は、親ハンドル・ストリーム・コーン・コンテナを指定できる。
U64	SIZE	この入力は、読み取るべきブロックのサイズを指定できる。
SGL	SGL	この入力は、メモリ内のどこにファイルを格納すべきかを指定できる。

【０１０１】
また、システム６００はメッセージＳｔｒｅａｍＣｏｎｅＲｅｌｅａｓｅを提供できる。このメッセージを使用して、指定したハンドルの識別を閉じることができ、表１９に記載された入力を受け入れることができる。
【０１０２】
【表２９】

表１９

変数タイプ	変数識別子	記述
U32	HANDLE	この入力は、閉じるべきハンドルを指定できる。

この機能は、ｉノード識別子からＨＡＮＤＬＥをアンリンクできる。
【０１０３】
システム６００は、メッセージＳｔｒｅａｍＣｏｎｅＲｅｎａｍｅを提供できる。このメッセージを使用して、ストリーム・コーンを名称変更することができ、表２０に記載された入力を受け入れることができる。
【０１０４】
【表３０】

表２０

変数タイプ	変数識別子	記述
U32	HANDLE	この入力は、名称変更すべきハンドルを指定できる。
U32	NEWPARENT	この入力は、現在のまたは新しい親ハンドルを指定できる。
SGL	SGL	この入力は、新しいSGLの名称を指定できる。

【０１０５】
また、システム６００はメッセージＳｔｒｅａｍＣｏｎｅＲｅｓｉｚｅを提供できる。このメッセージを使用して、ストリーム・コーンをサイズ変更することができ、表２１に記載された入力を受け入れることができる。
【０１０６】
【表３１】

表２１

変数タイプ	変数識別子	記述
U32	HANDLE	この入力は、サイズ変更すべきハンドルを指定できる。
U64	SIZE	この入力は、ストリーム・コーンの新しいサイズを指定できる。

【０１０７】
システム６００は、メッセージＳｔｒｅａｍＣｏｎｅＳｅｅｋを提供できる。このメッセージを使用して、ストリーム・コーンの位置を変更することができ、表２２に記載された入力を受け入れることができる。
【０１０８】
【表３２】

表２２

変数タイプ	変数識別子	記述
U32	HANDLE	この入力は、名称変更すべきハンドルを指定できる。
U64	NEW POSITION	この入力は、ストリーム・コーンの新しい位置を指定できる。

【０１０９】
システム６００は、メッセージＳｔｒｅａｍＣｏｎｅＳｅｔＩｎｆｏｒｍａｔｉｏｎを提供できる。このメッセージを使用して、ストリーム・コーンに関する情報を設定することができ、表２３に記載された入力を受け入れることができる。
【０１１０】
【表３３】

表２３

変数タイプ	変数識別子	記述
U32	HANDLE	この入力は、情報を設定すべきハンドルを指定できる。
SGL	SGL	この入力は、情報設定ブロックを指定できる。

【０１１１】
システム６００は、メッセージＳｔｒｅａｍＣｏｎｅＵｎｌｏｃｋを提供できる。このメッセージを使用して、以前に設定されたバイトのロック範囲をアンロックすることができ、表２４に記載された入力を受け入れることができる。
【０１１２】
【表３４】

表２４

変数タイプ	変数識別子	記述
U32	HANDLE	この入力は、アンロックすべきハンドルを指定できる。
U64	INDEX	この入力は、アンロックすべきバイトの範囲のスタート・バイトを指定できる。
U64	BYTECOUNT	この入力は、アンロックすべきバイト・カウントを指定できる。

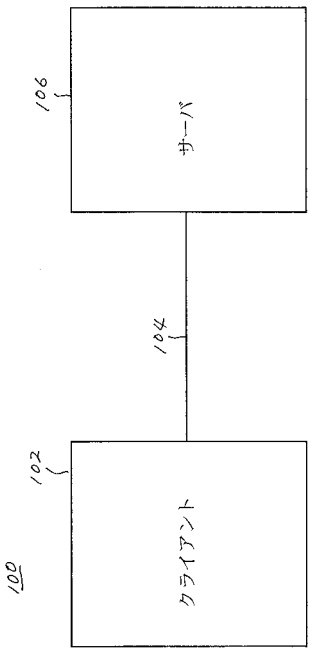
【０１１３】
システム６００は、メッセージＳｔｒｅａｍＣｏｎｅＷｒｉｔｅを提供できる。このメッセージを使用して、ブロックをストリーム・コーンに書き込むことができ、表２５に記載された入力を受け入れることができる。
【０１１４】
【表３５】

表２５

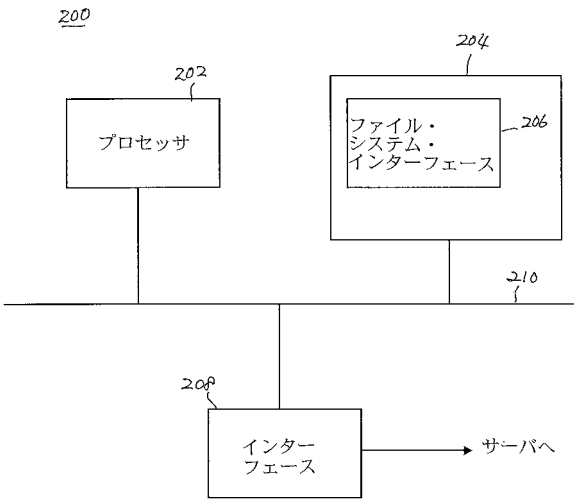
変数タイプ	変数識別子	記述
U32	HANDLE	この入力は、書き込まれるべきハンドルを指定できる。
U64	BYTECOUNT	この入力は、書き込まれるべきバイトのカウントを指定できる。
SGL	SGL	この入力は、書き込まれるべきブロックを指定できる。

【０１１５】
本発明の実施形態の一定の特徴を本明細書に記述し例示してきたが、当業者には、多くの修正形態、代替物、変更形態、均等物が思い浮かべられることであろう。したがって、頭記の特許請求の範囲は、本発明の実施形態の真の趣旨内に含まれる、このようなすべての修正形態および変更形態を包含するためのものであることを理解されたい。
【図面の簡単な説明】
【０１１６】
【図１】本発明の一実施形態を実践するのに適切なシステムを示す図である。
【図２】本発明の一実施形態によるクライアント・システムを示すブロック図である。
【図３】本発明の一実施形態によるサーバ・システムを示すブロック図である。
【図４】本発明の一実施形態によるクライアント・システムによって実施されるオペレーションを示すブロック流れ図である。
【図５】本発明の一実施形態によるサーバ・システムによって実施されるオペレーションを示すブロック流れ図である。
【図６】本発明の一実施形態によるソフトウェア・アーキテクチャを示す図である。

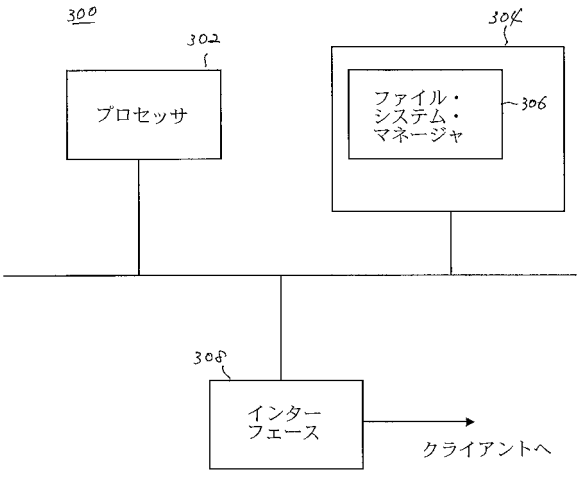
【図１】


106
サーバ
104
102
クライアント
100

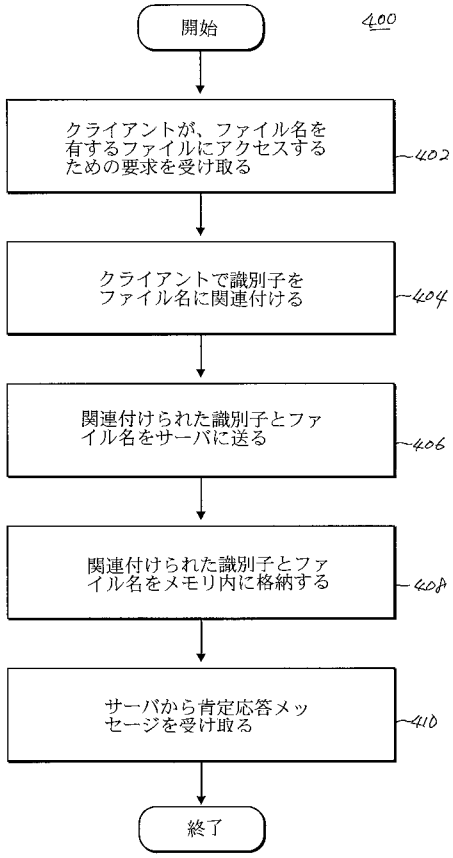
【図２】


200
202
プロセッサ
204
ファイル・
システム・
インターフェース
206
210
208
インター
フェース
サーバへ

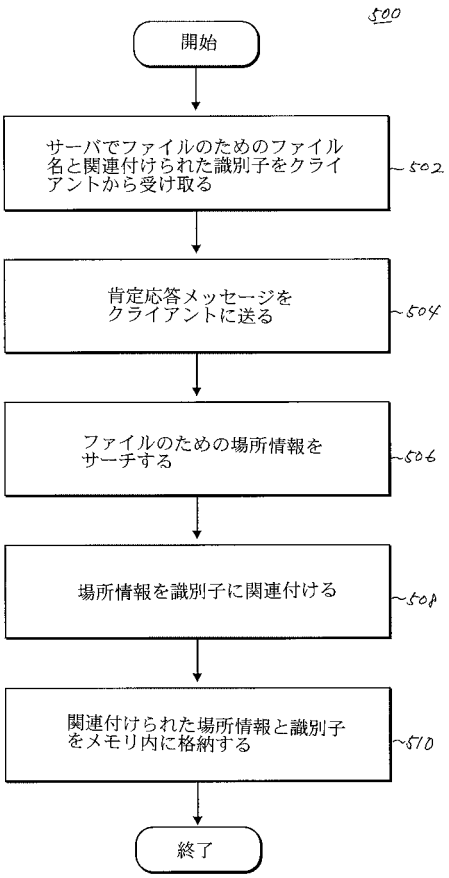
【図３】


300
302
プロセッサ
304
ファイル・
システム・
マネージャ
306
308
インター
フェース
クライアントへ

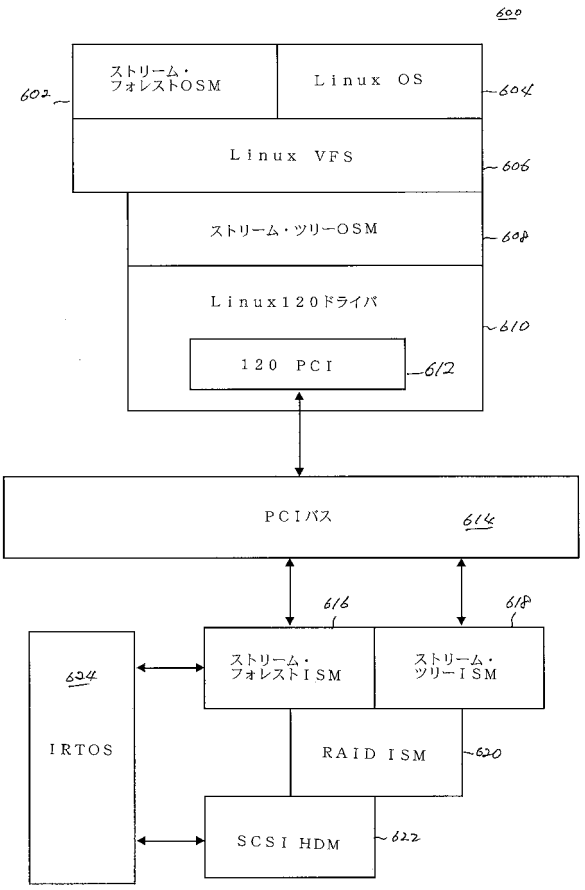
【図４】


開始
400
クライアントが、ファイル名を
有するファイルにアクセスする
ための要求を受け取る
402
クライアントで識別子を
ファイル名に関連付ける
404
関連付けられた識別子とファ
イル名をサーバに送る
406
関連付けられた識別子とファ
イル名をメモリ内に格納する
408
サーバから肯定応答メッ
セージを受け取る
410
終了

【図５】


500
開始
サーバでファイルのためのファイル名と関連付けられた識別子をクライアントから受け取る
502
肯定応答メッセージをクライアントに送る
504
ファイルのための場所情報をサーチする
506
場所情報を識別子に関連付ける
508
関連付けられた場所情報と識別子をメモリ内に格納する
510
終了

【図６】


600
ストリーム・フォレストOSM
602
Linux OS
604
Linux VFS
606
ストリーム・ツリーOSM
608
Linux120ドライバ
610
120 PCI
612
PCIバス
614
616
618
624
IRTOS
ストリーム・フォレストISM
ストリーム・ツリーISM
RAID ISM
620
SCSI HDM
622

【国際公開パンフレット】

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
11 July 2002 (11.07.2002)

PCT

(10) International Publication Number
WO 02/054286 A2

(51) International Patent Classification⁷: G06F 17/30

(21) International Application Number: PCT/US01/44883

(22) International Filing Date:
20 November 2001 (20.11.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/751,862 30 December 2000 (30.12.2000) US

(71) Applicant: INTEL CORPORATION [US/US]; 2200 Mission College Boulevard, Santa Clara, CA 95052 (US).

(72) Inventor: DAKE, Steven; 1972 West Bartlett Court, Chandler, AZ 85248 (US).

(74) Agents: MALLIE, Michael, J. et al.; Blakely Sokoloff Taylor & Zafman, 12400 Wilshire Boulevard, Los Angeles, CA 90025 (US).

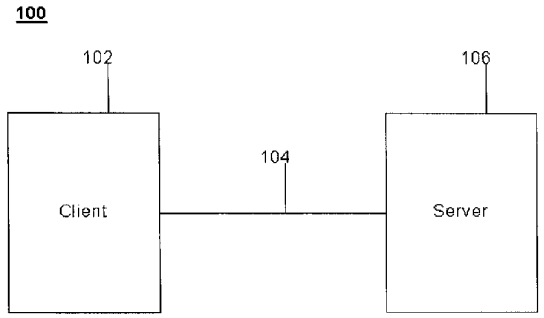
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:
without international search report and to be republished upon receipt of that report

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: METHOD AND APPARATUS TO IMPROVE FILE MANAGEMENT



(57) Abstract: A method and apparatus to manage a file in a computer and communications network is described.

WO 02/054286 A2

METHOD AND APPARATUS TO IMPROVE FILE MANAGEMENT

FIELD

This disclosure relates to computer and communication systems. More particularly, it relates to methods and apparatus to improve file management within a computer and communication system.

BACKGROUND

Computer and communications systems are frequently similar in architecture. Each system may comprise a number of separate components each designed to perform certain functions. The components may communicate information to other components over an interconnect system. An interconnect system operates to manage the transfer of information over a communications medium, such as metal leads, twisted-pair wire, co-axial cable, fiber optics, radio frequencies and so forth. Communications between components typically help coordinate operations of the individual components so that they may act as a cohesive system. This type of distributed system architecture may provide certain advantages in terms of performance, redundancy, scalability and efficiency.

There are disadvantages, however, associated with this type of system design. One disadvantage is that a component may have to remain idle as it waits for information from another component. Idle time may represent an inefficient use of system resources. Another disadvantage is that functionality may be allocated to components in a manner that increases the amount of information communicated between components. This

increase in communications increases demands on the finite resources of the interconnect system.

BRIEF DESCRIPTION OF THE DRAWINGS

The subject matter regarded as embodiments of the invention is particularly pointed out and distinctly claimed in the concluding portion of the specification. Embodiments of the invention, however, both as to organization and method of operation, together with objects, features, and advantages thereof, may best be understood by reference to the following detailed description when read with the accompanying drawings in which:

FIG. 1 is a system suitable for practicing one embodiment of the invention.

FIG. 2 is a block diagram of a client system in accordance with one embodiment of the invention.

FIG. 3 is a block diagram of a server system in accordance with one embodiment of the invention.

FIG. 4 is a block flow diagram of operations performed by a client system in accordance with one embodiment of the invention.

FIG. 5 is a block flow diagram of operations performed by a server system in accordance with one embodiment of the invention.

FIG. 6 illustrates a software architecture in accordance with one embodiment of the invention.

DETAILED DESCRIPTION

In the following detailed description, numerous specific details are set forth in order to provide a thorough understanding of the embodiments of the invention. It will be

understood by those skilled in the art, however, that the embodiments of the invention may be practiced without these specific details. In other instances, well-known methods, procedures, components and circuits have not been described in detail so as not to obscure the embodiments of the invention.

Embodiments of the invention may improve performance of a distributed system by reducing idle time for distributed components as well as bandwidth demands for the interconnect system. More particularly, one embodiment of the invention may improve performance of a distributed system used for file management. Consequently, this may reduce delays associated with file operations. Accordingly, a user may benefit in terms of more responsive applications and services.

A file management system typically comprises a client and a server. A client in this context may refer to a requestor of services. A server in this context may refer to a provider of services. The client typically sends a file request to a server over an interconnect system. The file request may be in the form of a message, and may include a function request and a file name. A message in this context may comprise one or more alphanumeric characters, symbols or logical expressions that when combined represent, for example, control words, commands, instructions, information or data. The message could vary in composition from a single bit to entire phrases, for example. The server associates a unique identifier with the file name, identifies location information for the file, and stores it with the unique identifier. The server then sends the unique identifier back to the client. The client receives the unique identifier and uses it in lieu of the file name for subsequent file requests.

A unique identifier is assigned to a file name to reduce bandwidth demands on the interconnect systems. A file name may comprise a number of alphanumeric characters or symbols ("character string"), for example. Each character or symbol is converted to one

or more bits, typically with 8 bits (e.g., 1 byte) per character. Since a file name may comprise many characters, the interconnect system may have to transport a relatively large number of bits. To reduce this problem, the server may assign a unique identifier to each file name and sends the unique identifier to the client. The unique identifier is typically smaller in length than the file name. For example, a unique identifier may have a length of 32 or 64 bits. The client may then use the unique identifier for subsequent file requests.

There are disadvantages, however, to having the server assign a unique identifier to a file name. One disadvantage is that the client may have to remain idle as it waits for the unique identifier prior to processing subsequent file request. Another disadvantage is that the client and server may need to communicate more information than necessary, thereby increasing demands on the interconnect system.

With respect to the first disadvantage, the client may have to remain idle as it waits for the unique identifier from the server. To initiate the assignment process, the client sends a message to the server requesting assignment of a unique identifier to a file name. The server associates a unique identifier with the file name, identifies location information for the file, and stores it with the unique identifier. The server then sends a message with the unique identifier back to the client. While waiting to receive the unique identifier from the server, the client may receive subsequent file requests with the same file name. The client may not be able to begin processing these file requests until the entire assignment process is completed. As a result, the client may be forced to remain idle during this time period.

With respect to the second disadvantage, the client and server may need to communicate unnecessary information to perform the assignment function, thereby increasing demands on the interconnect system. A file management system as described above requires at least two messages. The client sends a first message to the server

requesting assignment of a unique identifier to a file name. The server sends a second message to the client with the unique identifier. Each message requires use of a certain amount of bandwidth from the interconnect system. Bandwidth in this context may refer to the speed at which information can be transferred over an interconnect system, and is typically measured in kilobits per second (kbps). By way of contrast, one embodiment of the invention may perform the assignment process using only one message, thereby potentially reducing bandwidth demands on the interconnect system by as much as 50%.

Embodiments of the invention may improve performance of a distributed system by reducing idle time for a client as well as bandwidth demands for the interconnect system. One embodiment of the invention assigns a unique identifier to a file name at the client, and sends the unique identifier to the server. This may reduce client idle time since the client may begin processing subsequent file requests without having to wait for a message from the server. This may also reduce bandwidth demands since the server does not need to send a message back to the server to complete the assignment process, or alternatively, may send a message that is shorter than those needed by previous file management systems.

It is worthy to note that any reference in the specification to "one embodiment" or "an embodiment" means in this context that a particular feature, structure, or characteristic described in connection with the embodiment may be included in at least one embodiment of the invention. The appearances of the phrase "in one embodiment" in various places in the specification do not necessarily all refer to the same embodiment.

Referring now in detail to the drawings wherein like parts are designated by like reference numerals throughout, there is illustrated in FIG. 1 a system 100 suitable for practicing one embodiment of the invention. As shown in FIG. 1, system 100 comprises a client 102 and a server 106 connected by an interconnect system 104. The term "client" as

used herein may refer to any requestor of information. The term "server" as used herein may refer to any provider of information.

FIG. 2 is a block diagram of a client system in accordance with one embodiment of the invention. FIG. 2 illustrates a client 200 that may be representative of client 102. As shown in FIG. 2, client 200 comprises a processor 202, a memory 204 and an interface 208, all connected by connection 210. Memory 204 may store program instructions and data. The term "program instructions" include computer code segments comprising words, values and symbols from a predefined computer language that, when placed in combination according to a predefined manner or syntax, cause a processor to perform a certain function. Examples of a computer language include C, C++ and assembly. Processor 202 executes the program instructions, and processes the data, stored in memory 204. Interface 208 coordinates the transport of data from client 200 to another device. Connection 210 transports data between processor 202, memory 204, and interface 208.

Processor 202 can be any type of processor capable of providing the speed and functionality desirable for various embodiments of the invention. For example, processor 202 could be a processor from a family of processors made by Intel Corporation, Motorola, Compaq or Sun Microsystems. In one embodiment of the invention, processor 202 may be a dedicated processor to manage input/output (I/O) devices, such as hard drives, keyboards, printers, network interface cards and so forth. This processor is typically referred to as an I/O processor (IOP).

In one embodiment of the invention, memory 204 comprises a machine-readable medium and may include any medium capable of storing instructions adapted to be executed by a processor. Some examples of such media include, but are not limited to, read-only memory (ROM), random-access memory (RAM), programmable ROM, erasable programmable ROM, electronically erasable programmable ROM, dynamic

RAM, magnetic disk (e.g., floppy disk and hard drive), optical disk (e.g., CD-ROM) and any other media that may store digital information. In one embodiment of the invention, the instructions are stored on the medium in a compressed and/or encrypted format. As used herein, the phrase "adapted to be executed by a processor" is meant to encompass instructions stored in a compressed and/or encrypted format, as well as instructions that have to be compiled or installed by an installer before being executed by the processor. Further, client 200 may contain various combinations of machine-readable storage devices through various I/O controllers, which are accessible by processor 202 and which are capable of storing a combination of computer program instructions and data.

Memory 204 may store and allow execution by processor 202 of program instructions and data to implement the functions of a client, such as client 106 and client 200. In one embodiment of the invention, memory 204 includes a set of program instructions that will be collectively referred to herein as a file system interface 206.

File system interface 206 may be an interface that operates to provide access to one or more files for system 100. An interface in this context may refer to a defined protocol by which one software module may access functionality from another software module. A file in this context refers to a discrete set of data stored in memory, such as in memory 204 or a hard drive. File system interface 206 may receive a request to perform certain operations for a file, such as create, open, seek, read, write, rename, delete, copy, move, and so forth. The request may originate from a host OS or an application program, for example. A host OS may comprise an OS for a system. For example, if an embodiment of the invention was implemented as part of a personal computer, the host OS might comprise an OS sold by Microsoft Corporation, such as Microsoft Windows® 95, 98, 2000 and NT, for example.

In one embodiment of the invention file system interface 206 operates as an Operating System Service Module (OSM) as defined by the Intelligent I/O Specification (I_2O) developed by the I_2O Special Interest Group (SIG) (I_2O SIG), version 1.5, adopted in April of 1997, and available from "www.i20sig.org" ("I_2O Specification"), although the invention is not limited in scope in this respect.

By way of background, the I_2O Specification defines a standard architecture for intelligent I/O that is independent of both the specific device being controlled and the host operating system (OS). Intelligent I/O in this context refers to moving the function of processing low-level interrupts from a central processing unit (CPU) or other processor to I/O processors (IOPs) designed specifically to provide processing for I/O functions. This may improve I/O performance as well as permit the CPU or other processor to provide processing functionality for other tasks. An interrupt in this context refers to a request to access an I/O device, such as a hard drive, floppy disk drive, printer, monitor, keyboard, network interface card (NIC) and so forth.

The I_2O Specification describes an OSM, an Intermediate Services Module (ISM) and a Hardware Device Module (HDM). The OSM may be a driver that operates as an interface between a host OS and an ISM. A driver in this context refers to a set of program instructions that manage operations of a particular component, device or software module. The ISM may operate as an interface between the OSM and a Hardware Device Module (HDM). The ISM may perform specific functionality for I/O management functions, network protocols or peer-to-peer functionality such as background archiving, for example. The HDM may be a driver that operates to control a particular I/O device.

The I_2O Specification defines a communications model that comprises a message-passing system. The OSM, ISM and HDM communicate and coordinate operations by passing information in the form of messages through a message layer. A message layer in

this context may manage and dispatch requests, provide a set of Application Programming Interfaces (APIs) for delivering messages, and provide a set of support routines to process messages.

In one embodiment of the invention, file system interface 206 operates as an OSM in accordance with the I_2O Specification. In one embodiment of the invention, file system interface 206 receives file requests from an application program via the host OS, translates the request into a message in accordance with the I_2O Specification, and sends it to a file system manager (described below) for processing. An application program in this context refers to a program that provides a predetermined set of functions for specialized tasks, typically having a user interface to facilitate the processing of commands and instructions between a user and the computer system. Examples of application programs might include a word processor, spread sheet, database or Internet browser.

Interface 208 may comprise any suitable technique for controlling communication signals between computer or network devices using a desired set of communications protocols, services and operating procedures, for example. In one embodiment of the invention, interface 208 may operate, for example, in accordance with the PCI Specification and the I_2O Specification. In another embodiment of the invention, interface 208 may operate in accordance with the Transmission Control Protocol (TCP) as defined by the Internet Engineering Task Force (IETF) standard 7, Request For Comment (RFC) 793, adopted in September, 1981, and the Internet Protocol (IP) as defined by the IETF standard 5, RFC 791, adopted in September, 1981, both available from "www.ietf.org." Although interface 208 may operate with in accordance with the above described protocols, it can be appreciated that interface 208 may operate with any suitable technique for controlling communication signals between computer or network devices using a

desired set of communications protocols, services and operating procedures, for example, and still fall within the scope of the invention.

Interface 208 also includes connectors for connecting interface 208 with a suitable communications medium. Interface 208 may receive communication signals over any suitable medium such as copper leads, twisted-pair wire, co-axial cable, fiber optics, radio frequencies, and so forth. In one embodiment of the invention, the connectors are suitable for use with a bus to carry signals that comply with the PCI Specification.

FIG. 3 is a block diagram of a server system in accordance with one embodiment of the invention. FIG. 3 illustrates a server 300 that is representative of server 106, in accordance with one embodiment of the invention. As shown in FIG. 3, server 300 comprises a processor 302, a memory 304 and an interface 308, all connected by connection 310. Elements 302, 304, 308 and 310 of FIG. 3 are similar in structure and operation as corresponding elements 202, 204, 208 and 210 described with reference to FIG. 2. Although server 300 is shown with a processor 302, it can be appreciated that server 300 may operate without processor 302 by using another processor available to server 300 (e.g., processor 202), and still fall within the scope of the invention. For example, such a configuration may occur if the embodiments of the invention were incorporated into a personal computer where the client and server were connected by a PCI bus and both shared a single processor.

In one embodiment of the invention, memory 304 contains program instructions for a file system manager 306. File system manager 306 performs file management and provides access to a storage medium (not shown) containing a plurality of files. File system 306 performs file operations such as create, open, seek, read, write, rename, delete, copy, move, and so forth, in response to file requests received from file system interface 206. One example of file system interface 206 includes an ISM operating in accordance

with the I_2O Specification, although the scope of the invention is not limited in this respect.

The operation of systems 100, 200 and 300 will be described in more detail with reference to FIGS. 4 and 5. Although FIGS. 4 and 5 presented herein include a particular sequence of operations, it can be appreciated that the sequence of operations merely provides an example of how the general functionality described herein may be implemented. Further, the sequence of operations does not necessarily have to be executed in the order presented unless otherwise indicated.

FIG. 4 is a block flow diagram of the operations performed by a client in accordance with one embodiment of the invention. In this embodiment of the invention, file system interface 206 operates as part of client 106. It can be appreciated that file system interface 206, however, can be implemented by any device, or combination of devices, located anywhere in a computer or network system and still fall within the scope of the invention.

As shown in FIG. 4, a client receives a request to access a file having a file name at block 402. The client associates the file name with an identifier at block 404. The client sends the associated identifier and file name to a server at block 406. The client stores the associated identifier and file name in memory at block 408. The client receives an acknowledgement message from the server at block 410.

Once the client assigns an identifier to a file, the identifier is used for future requests for the file. The client receives a second request at the client to access the file. The client retrieves the identifier associated with the file name from memory. The client sends the second request to the server using the associated identifier.

FIG. 5 is a block flow diagram of the operations performed by a server in accordance with one embodiment of the invention. In this embodiment of the invention,

file system manager 306 operates as part of client 106. It can be appreciated that this functionality, however, can be implemented by any device, or combination of devices, located anywhere in a computer or network system and still fall within the scope of the invention.

As shown in FIG. 5, a server receives a file name and associated identifier for a file at block 502. The server sends an acknowledgement message to the client at block 504. The server searches for location information for the file at block 506. The server associates the location information with the identifier at block 508. The server stores the associated location information and identifier in memory.

Once the server indexes the location information for a file using the identifier, the server can use the identifier to access the location information for subsequent file requests. The server receives a second request to access the file having the identifier. The server retrieves the location information from memory using the identifier.

The operation of systems 100, 200 and 300, and the flow diagrams shown in FIGS. 4 and 5, may be better understood by way of example. An application program sends a request to read information from a file with a file name "test file one" to the host OS of system 100. A unique identifier in this context refers to a series of alphanumeric characters that when combined represent a unique word, value or binary string to the client, server and/or system with respect to other words, values or binary strings used by the client, server and/or system. The host OS passes the file request to file system interface 206. File system interface 206 generates a unique identifier "A123" and assigns it to file name "test file one." In this embodiment of the invention, the unique identifier "A123" may be a hexidecimal 32 bit number, for example. File system interface 206 creates a message "identify (test file one, A123)," and places it in an outbound message queue for transport over connection 104 to file system manager 306. The outbound

message queue in this context refers to a queue such as first-in-first-out (FIFO) that is used to hold messages until the interconnect system can transport the message. File system interface 206 stores "test file one" with "A123" in a lookup table in memory 204.

File system manager 306 receives the message over connection 104. File system manager 306 parses the message and invokes the function "identify (test file one, A123)." The term parses in this context refers to separating individual characters or sub-sets of characters from the message that represent, for example, commands, control words, file names, data, function calls, sub-routine names, flags and so forth. The term "invokes" in this context refers to a command sent to the processor to begin executing program instructions associated with a given function or sub-routine. This function takes as inputs "test file one" and "A123," and informs file system manager 306 that the file name "test file one" will be referenced in subsequent file requests as "A123." This may be accomplished by updating a lookup table in memory by storing the file name and unique identifier together as corresponding or linked terms, that is, one may be found by searching for the other. File system manager 306 searches for location information for file "A123," which is typically located on a storage device such as a hard drive. Examples of location information may include addressing information, device, cylinder number and track number, although the invention is not limited in scope in this respect. File system manager 306 associates the location information with identifier "A123," and stores the location information with identifier "A123" in a lookup table in memory 304. File system manager 306 sends an acknowledgement message to file system interface 206 that the file name identifier and location information have been received. An acknowledgement message in this context refers to a short message indicating that a previous message was received. The acknowledgement message may comprise a single bit, character, word or phrase, as desired by a particular system.

Subsequent file requests received by file system manager 306 will then use identifier "A123" when requesting operations for file name "test file one." For example, file system interface 206 receives a second request to perform a "delete" operation for "test file one." File system interface 206 retrieves the previously associated identifier "A123" for "test file one" from memory. File system interface 206 sends a message "delete('A123')" to file system manager 306. File system manager 306 receives the message "delete('A123')" and retrieves the location information for file name "test file one" using the identifier "A123." File system manager 306 then performs the requested file operation using the retrieved location information.

In one embodiment of the invention, file system interface 206 may be implemented as an OSM in accordance with the I_20 Specification and a particular host OS, such as the Linux OS version 2.3.99 pre-3 kernel available from "www.kernel.org" ("Linux Kernel"). In this embodiment of the invention, the OSM may further comprise a stream forest OSM, a stream tree OSM, and a class specification. A class in this context may refer to a specific interface definition. A tree in this context may refer to a collection of storage objects called cones. An object in this context may refer to an instance of a class. A cone in this context may refer to a storage object that supports read, write and lock capabilities, for example. A forest in this context may refer to a collection of trees.

In this embodiment of the invention, the stream forest OSM may be used to model a collection of file systems. The stream forest OSM may provide file naming operations, such as creating a name, erasing a name, or renaming a particular forest, for example. Further, open and close operations may also be supported. The tree OSM may be used to model a particular file system. A file may be modeled as a cone. The stream forest OSM may also function to support file operations, such as to name a file, rename it, erase it, lock it from change, read it or write it, for example.

The OSM may be designed to communicate with an ISM. The ISM may further comprise a stream forest ISM and a stream tree ISM. The stream forest ISM may support the file system naming capability of the OSM. In one embodiment of the invention, the stream tree ISM may support stream cone identifiers with lengths of 2^8 characters. The stream tree ISM may also support 2^{16} open stream cones, as well as 2^{32} possible stream cones. The tree ISM may support 2^{64} bytes for all contained stream cones. It can be appreciated that these values do not limit the scope of the invention in this respect.

In operation, the stream tree OSM, stream forest OSM, stream tree ISM and stream forest ISM may all communicate using a messaging scheme in accordance with a class specification as discussed below. This messaging scheme will be referred to as a streaming messaging scheme, and will be discussed in further detail below. A host OS may use the functionality provided by the stream forest OSM and stream tree OSM. The host OS may use the stream forest OSM to model a grouping of file systems, and the stream tree OSM to model a particular file system, for example. The stream forest OSM may use the stream forest ISM to manage groupings of file systems within an IRTOS environment. The stream tree OSM may use the stream tree ISM to manage a file system within an I_20 Real-Time Operating System (IRTOS) environment. The stream forest OSM and stream tree OSM may communicate with the stream forest OSM and stream tree ISM, respectively, using the stream messaging scheme. This embodiment of the invention will be further described with reference to FIG. 7.

FIG. 6 illustrates a software architecture in accordance with one embodiment of the invention. As shown in FIG. 6, a system 600 may comprise a file system interface having a stream forest OSM 602, a Linux OS 604, a Linux Virtual File System (VFS) 606, a stream tree OSM 608, and one or more Linux I_20 drivers 610. Linux I_20 drivers 610 may operate in accordance with the Linux Kernel and the I_20 Specification. Linux I_20 drivers

610 may include PCI I_20 drivers 612. PCI I20 drivers 612 may operate in accordance with the PCI Specification and the I_20 Specification.

The file system interface communicates with a file system manager over a bus 614 that communicates signals in accordance with the PCI Specification. The file system manager may comprise a stream forest ISM 616, a stream tree ISM 618, a Redundant Array of Inexpensive Disks (RAID) ISM 620, a Small Computer System Interface (SCSI) HDM 622 and IRTOS 624.

In this embodiment of the invention, the modules shown in system 600 may be implemented using the C programming language, although the scope of the invention is not limited in this respect. Further, in this embodiment of the invention modules shown in system 600 support identifiers having a length of 255 characters.

Stream tree OSM 608 may be configured to provide access to a typical file system. Stream tree OSM 608 may be configured to support one or more Linux VFS required functions, as defined by the Linux Specification. Stream tree OSM 608 may be configured to support stream tree class messages, as discussed in more detail below. Stream tree OSM 608 may be configured to operate with Linux OS 604 using, for example, the Linux Kernel. Stream tree OSM 608 may support a stream tree ISM, such as stream tree ISM 618.

Stream forest OSM 602 may provide the function **loctl() kernel interface**. Stream forest OSM 602 may function to create a stream tree within a stream forest, using a function name such as "IOCTL_SF_TREECREATE." This function may accept as inputs, for example, an input buffer defined in the C language as follows:

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

Stream forest OSM 602 may also function to rename an existing stream tree, using a function name such as "IOCTL_SF_TREERENAME." This function may accept as inputs, for example, an input buffer defined in the C language as follows:

```
struct ik_sf_treerename {
        char    IdentifierBeforeRename [256];
        char    IdentifierAfterRename [256];
}.
```

Stream forest OSM 602 may also function to erase an existing stream tree, using a function name such as "IOCTL_SF_TREEERASE." This function may accept as inputs, for example, an input buffer defined in the C language as follows:

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

Stream forest OSM 602 may use one or more of the following data structures. A data structure referred to herein as a "super block" may be used to represent the initial "inode" of a file system. An "inode" in this context may refer to a file node of the file system. A super block operation list may be a grouping of functions used to manipulate the super block. An inode record data structure may be used to represent each file node of the file system. An inode operation list may be a grouping of functions used to manipulate an inode. A file record data structure may be used to represent an abstract file within a file system. A file operation list may be used to support file operations. All of these data structures combined with operation list operations may be used to represent a file system.

Stream tree class messages may be issued for several of the data structures. A stream tree class message may be issued, for example, for the inode operations create, lookup, make directory ("mkdir"), remove directory ("rmdir"), and rename directory. A stream tree class message may be issued for various file operations, such as open, read, write, seek, and close.

Stream forest OSM 602 may register the contained file system with the identifier "i2ofs," for example, within the Linux kernel. Stream forest OSM 602 may use the NULL identifier for the root tree cone container, for example. Stream forest OSM 602 may use the identifier '.' for the current tree cone container, for example. Stream forest OSM 602 may use the identifier '..' for the parent tree cone container, for example.

System 600 may contain system descriptions for one or more of the following modules:

1. A MODULE_AUTHOR description for the kernel;
2. A MODULE_DESCRIPTION of "I2O Filesystem Offload Driver";
3. A module_init () which may register the file system;
4. A module_exit () which may un-register the file system;
5. A DECLARE_FSTYPE module which may have parameters of the i2ofs_type i2ofs, read super operation, and 0;
6. An OSM handle manager module that may have a first handle of zero that increases by 1 per used handle;
    a. An internal software interface AcquireHandle() which may retrieve an unused handle; and
    b. An internal software interface ReleaseHandle (int) which may release a currently used handle into the free pool.

System 600 may also provide the function **VFS i2ofs_read_super (struct super_block *sb, void *options, int silent)**. This function may accept the inputs as defined in Table 1.

TABLE 1

Variable Type	Variable Identifier	Description
Struct super_block *	Sb	This input may specify the location where the super block should be set.
Void *	Options	This input may specify the options passed by mount.
int	Silent	This input may specify if the mount operation is silent.

This function sets the values in the super block structure to 0. The following variables may be set as follows:

sb->s_blocksize = 512.
sb->s_blocksize_bits = 10.
sb->S_s_magic = I2OFS_SUPER_MAGIC
sb->s_op = the super_operations structure defined in the OSM.

The function may create a new inode using a module referred to as **get_empty_inode()**.

The inode may be set to zero, with various associated variables having the following settings:

inode i_uid = 20.
inode i_gid = 20.
inode operations may be set to the pointer describing the inode operation list.
inode i_fop may be set to the point describing the file operation list.
inode I_mode may be set to S_IFDIR|S_IRUGO|S_IXUGO.
inode may be inserted into the inode hash table.
Sb->s_root may be set to d_alloc_root() of the root inode.

Each file system within VFS 606 may have at least one super block, which contains enough information about the file system to initiate activity of the file system. This super block may be implemented in a C structure struct super_block, as follows:

```
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;
};
```

The variable super_block->s_op may contain the following C functions and provide access to the super block.

```
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 *);
};
```

An example of an inode interface may be as follows:

```
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;
        Usngined char I_sock;
        Atomic_t I_writecount;
        Unsigned int I_attr_flags;
        __u32 I_generation;
};.
```

The variable inode->i_op may contain the following C functions and provide method access to the super block, as follows:

```
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 *);
}.
```

System 600 also provides a function **create (struct inode *, const char *, int, int, struct inode **)**. This function may accept the inputs set forth in Table 2.

TABLE 2

Variable Type	Variable Identifier	Description
struct inode *	ParentDirectory	This input may specify the input directory of the create operation.
const char *	NewName	This input may specify the name of the new file system object.
int	NewSize	This input may specify the new size of the object
Int	NewMode	This input may specify the new mode of the object.
struct inode **	NewInode	This input may specify the new inode of the created object.

This function may create a new inode using **get_empty_inode ()**. The function may initialize the new inode by attaching the const char * variable to it via dentry relationship. The function may initialize the new inode structure with the size specified. The function may initialize the new inode structure with the mode specified. The function may send a

StreamConeCreate message to the stream tree ISM. The message may be configured as follows:

1. HANDLE may be retrieved from the OSM handle manager;
2. TYPE may be set to 1, indicating a file; and
3. SGL may be set to NewName.

The dereference of NewInode may be set with the new inode structure.

System 600 may also provide a function **struct dentry *lookup (struct inode *, struct dentry *)**. This function may accept the inputs set forth in Table 3.

TABLE 3

Variable Type	Variable Identifier	Description
struct inode *	ParentDirectory	This input may specify the input directory of the lookup operation.

This function may send a **StreamConeIdentify** message to stream tree ISM 618. The message may be configured as follows:

1. PARENTHANDLE may be set to the handle identified within the ParentDirectory inode local data;
2. CHILDHANDLE may be set to a handle retrieved from the OSM handle manager;
3. ATTRIBUTES may be set to STREAMCONECREATE_QUERY; and
4. SGL may be set to Name.

The function may create a new inode using **get_empty_inode ()**. The function may initialize the new inode by attaching the const char * variable to it via dentry relationship. The function may send a **StreamConeGetInformation** message to the stream tree ISM. The message may be configured as follows:

1. HANDLE may be set to CHILDHANDLE above; and
2. SGL may be set to a local data structure of type Information Result Block.

The function may set the inode values from the Information Result Block. The function may set the dereference of the NewInode variable to the inode that has been created.

System 600 may provide the function **mkdir (struct inode *, struct dentry *, int)**. This function may accept the inputs as set forth in Table 4.

TABLE 4

Variable Type	Variable Identifier	Description
struct inode *	ParentDirectory	This input may specify the input directory of the mkdir operation.
Struct dentry *	Name	This input may specify the name of the object to create.
int	Mode	This input may specify the mode of the new directory.

The function may send a **StreamConeCreate** message to create a directory. The message may be configured as follows:

1. HANDLE may be set to the handle identified in the inode structure input ParentDirectory;
2. TYPE may be set to 2; and
3. SGL may be set to the input Name that contains the actual name.

System 600 may provide the function **rmdir (struct inode *, struct dentry *)**. This function may accept the inputs set forth in Table 5.

TABLE 5

Variable Type	Variable Identifier	Description

struct inode *	ParentDirectory	This input may specify the input directory of the lookup operation.
struct dentry *	Name	This input may specify the name of the object to remove.
int	Length	This input may specify the length of Name in characters.

This function may send a **StreamConeErase** message to remove a directory. The message may be configured as follows:

1. HANDLE may be set to the handle identified in the inode structure input ParentDirectory; and
2. SGL may be set to the input Name.

System 600 may also provide the function **rename (struct inode *, struct dentry *, struct inode *, struct dentry *)**. This function may accept the inputs set forth in Table 6.

TABLE 6

Variable Type	Variable Identifier	Description
struct inode *	OldDir	This input may specify the directory of the file to rename.
struct dentry *	OldName	This input may specify the name of the object to rename.
struct inode *	NewDir	This input may specify the new directory of the object.
struct dentry *	NewName	This input may specify the new name of the object.

This function may send a **StreamConeIdentify** message to the ISM. The message may be configured as follows:

1. PARENTHANDLE may be set from OldDir inode internal storage for OSM handles;

2. CHILDHANDLE may be set to new handle retrieved from the OSM handle manager;
3. ATTRIBUTES may be set to STREAMCONECREATE_QUERY; and
4. SGL may be set to OldName.

The function may send a StreamConeRename message to the ISM. The message may be configured as follows:

1. HANDLE may be set to the CHILDHANDLE of StreamConeIdentify;
2. NEWPARENT may be set to NewDir inode internal storage for OSM handles; and
3. SGL may be set to NewName.

The function may send a **StreamConeClose** message to the ISM. The message may be configured as follows: HANDLE may be set to the previous HANDLE from **StreamConeRename** message.

System 600 may also provide the function **truncate (struct inode *)**. This function may accept the inputs set forth in Table 7.

TABLE 7

Variable Type	Variable Identifier	Description
struct inode *	File	This input may specify the file to truncate.

The function may send a **StreamConeResize** message to the ISM. The message may be configured as follows:

1. HANDLE may be retrieved from the inode internal OSM handle storage; and

2. SIZE may be retrieved from the inode variable i_size.

Each file within VFS 606 may have at least one super block, which may contain enough information about the file system to initiate activity of the file system. This super block is detailed in the C structure struct super_block, such as the one described previously. The methods variable, f_op, may provide access to file operations. The struct super_operations *s_op function may provide access to the root inode, which may be desired for the OSM.

```
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;
};
```

System 600 may provide one or more file operations described below. The variable f->s_op may contain the following C functions and may provide access to the super block.

```
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 *);
};
```

System 600 may provide the function **llseek (struct file *, off_t, int)**. This function may accept the inputs as set forth in Table 8.

TABLE 8

Variable Type	Variable Identifier	Description
struct file *	File	This input may specify the file pointer to seek.
off_t	Offset	This input may specify the offset from Origin to seek to.
Int	Origin	This input may specify the Origin.

This function may send a **StreamConeSeek** message to the ISM. The message may be configured as follows:

1. HANDLE may be set from node internal storage for OSM handles; and
2. NEWPOSITION may be set to the position obtained by calculation from Offset and Origin.

System 600 may provide the function **read (struct file *, char *, size_t, loff_t)**. This function may accept the inputs set forth in Table 9.

TABLE 9

Variable Type	Variable Identifier	Description
struct file *	File	This input may specify the file pointer to read.
char *	Buffer	This input may specify the buffer to read into.
Size_t	Size	This input may specify size in bytes to read.
loff_t *	SeekChange	This input specifies how much data was read.

This function may send a **StreamConeRead** message to the ISM. The message may be configured as follows:

1. HANDLE may be set from node internal storage for OSM handles; and
2. SGL may be set to Buffer and Size.

System 600 may provide the function **write (struct file *, const char *, size_t, loff_t *)**. This function may accept the inputs set forth in Table 10.

TABLE 10

Variable Type	Variable Identifier	Description
struct file *	File	This input may specify the file pointer to read.
Const char *	Buffer	This input may specify the buffer to write.
Size_t	Size	This input may specify size in bytes to read.
Loff_t	SeekChange	This input specifies how much data was written.

This function may send a **StreamConeWrite** message to the ISM. The message may be configured as follows:

1. HANDLE may be set from node internal storage for OSM handles; and
2. SGL may be set to Buffer and Size.

System 600 may provide the function **readdir (struct file *, void *, filldir_t)**. This function may accept the inputs set forth in Table 11.

TABLE 11

Variable Type	Variable Identifier	Description
struct file *	File	This input may specify the file pointer to read.

This function may send a **StreamConeEnumerate** message to the ISM. The message may be configured as follows:

1. HANDLE may be set from node internal storage for OSM handles;

2. ENUMERATOR is set to Count;

3. SGL may be set to Entry's file name buffer;

4. SGL's size may be set to 255.

System 600 may also provide a dentry interface as follows:

```
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];
};
```

Stream tree ISM 618 may support one or more stream tree class messages, as defined herein. Stream tree 618 may function and support messages from a stream tree OSM, such as stream tree OSM 608. Stream forest ISM 616 may support stream forest class messages, as defined herein. Stream forest ISM 616 may function and support messages from a stream forest OSM, such as stream forest OSM 602.

Stream forest ISM 616 may include a user interface. The user interface may include a user screen that may display one or more stream trees on initialization. A user may be able to create new stream trees. When creating new stream trees, the user may be presented with a screen asking for size and identification. If a stream tree is created, a

stream tree class object may be created. The user may also be able to erase stream trees. The user interface may provide for confirmation of erasure of stream trees. Stream forest messages may be handled and tree objects created as appropriate.

System 600 may provide the message **StreamConeCreate**. This message may be used to create a new stream cone that will store information, and may accept the inputs as set forth in Table 12.

TABLE 12

Variable Type	Variable Identifier	Description
U32	HANDLE	This input may specify the handle that should be closed.
U32	TYPE	This input may specify the type to create. 1 indicates a file and 2 indicates a directory.
SGL	SGL	Specifies the identifier.

System 600 may also provide the message **StreamConeEnumerate**. This message may be used to list one or more stream cones within a stream cone container, and may accept the inputs as set forth in Table 13.

TABLE 13

Variable Type	Variable Identifier	Description
U32	HANDLE	This input may specify the handle that should be closed.
U32	ENUMERATOR	This input may specify a zero-based index indicating which entry should be enumerated into the SGL.
SGL	SGL	Specifies the location of the enumerated stream cone identifier.

System 600 may retrieve the inode relating to HANDLE from the handle manager. When an enumerator is set to 0, the "ext2fs" library call dir_iterate() may be used with the inode relating to handle as the parent to generate a list of stream cone identifiers on the "ext2" filesystem. A list entry corresponding to the ENUMERATOR index may be copied into SGL.

System 600 may provide the message **StreamConeErase**. This message is used to erase a stream cone identifier, and may accept the inputs as set forth in Table 14.

TABLE 14

Variable Type	Variable Identifier	Description
U32	HANDLE	This input may specify the handle that should be closed.
SGL	SGL	This input may specify the name of the identifier that should be erased.

System 600 may provide the message **StreamConeGetInformation**. This message may be used to retrieve information about the stream cone, and may accept the inputs as set forth in Table 15.

TABLE 15

Variable Type	Variable Identifier	Description
U32	HANDLE	This input may specify the handle where information should be retrieved from.
SGL	SGL	This input may specify the name of the identifier that should be erased.

System 600 may provide the message **StreamConeIdentify**. This message may be used to map a handle identifier to a string identifier, and may accept the inputs as set forth in Table 16.

TABLE 16

Variable Type	Variable Identifier	Description
U32	PARENT HANDLE	This input may specify the parent handle stream cone container.
U32	CHILD HANDLE	This input may specify the child handle that should be mapped to the stream cone identifier.
U32	ATTRIBUTES	This input may specify the attributes of the identified stream cone.
SGL	SGL	This input may specify the name of the identifier.

System 600 may provide the message **StreamConeLock**. This message may be used to lock a byte range of a file, and may accept the inputs as set forth in Table 17.

TABLE 17

Variable Type	Variable Identifier	Description
U32	HANDLE	This input may specify the parent handle stream cone container.
U64	INDEX	This input may specify the byte index that should be locked.
U64	SIZE	This input may specify the size in bytes that should be locked.

System 600 may provide the message **StreamConeRead**. This message may be used to read a block from a stream cone, and may accept the inputs as set forth in Table 18.

TABLE 18

Variable Type	Variable Identifier	Description
U32	HANDLE	This input may specify the parent handle stream cone container.
U64	SIZE	This input may specify the size of the block to read.
SGL	SGL	This input may specify where in memory the file should be stored.

System 600 may also provide a message **StreamConeRelease**. This message may be used to close an identification of the specified handle, and may accept the inputs as set forth in Table 19.

TABLE 19

Variable Type	Variable Identifier	Description
U32	HANDLE	This input may specify the handle that should be closed.

This function may unlink HANDLE from the inode identifier.

System 600 may provide a message **StreamConeRename**. This message may be used to rename a stream cone, and may accept the inputs as set forth in Table 20.

TABLE 20

Variable Type	Variable Identifier	Description
U32	HANDLE	This input may specify the handle that should be renamed.

U32	NEWPARENT	This input may specify the current or new parent handle.
SGL	SGL	This input may specify the name of the new SGL.

System 600 may also provide a message **StreamConeResize**. This message is used to resize a stream cone, and may accept the inputs as set forth in Table 21.

TABLE 21

Variable Type	Variable Identifier	Description
U32	HANDLE	This input may specify the handle that should be renamed.
U64	SIZE	This input may specify the new size of the stream cone.

System 600 may provide a message **StreamConeSeek**. This message is used to change the position of a stream cone, and may accept the inputs as set forth in Table 22.

TABLE 22

Variable Type	Variable Identifier	Description
U32	HANDLE	This input may specify the handle that should be renamed.
U64	NEW POSITION	This input may specify the new position of the stream cone.

System 600 may provide a message **StreamConeSetInformation**. This message may be used to set information regarding the stream cone, and may accept the inputs as set forth in Table 23.

TABLE 23

Variable Type	Variable Identifier	Description
U32	HANDLE	This input may specify the handle that should have information set.
SGL	SGL	This input may specify the information set block.

System 600 may provide a message **StreamConeUnlock**. This message may be used to unlock a previously set byte lock range, and may accept the inputs as set forth in Table 24.

TABLE 24

Variable Type	Variable Identifier	Description
U32	HANDLE	This input may specify the handle that should be unlocked.
U64	INDEX	This input may specify the start byte of the range of bytes to unlock.
U64	BYTECOUNT	This input may specify the byte count to unlock.

System 600 may provide a message **StreamConeWrite**. This message may be used to write a block to a stream cone, and may accept the inputs as set forth in Table 25.

TABLE 25

Variable Type	Variable Identifier	Description
U32	HANDLE	This input may specify the handle that should be written.
U64	BYTECOUNT	This input may specify the count in bytes that should be written.
SGL	SGL	This input may specify the block to be written.

While certain features of the embodiments of the invention have been illustrated as described herein, many modifications, substitutions, changes and equivalents will now occur to those skilled in the art. It is, therefore, to be understood that the appended claims are intended to cover all such modifications and changes as fall within the true spirit of the embodiments of the invention.

CLAIMS:

1. A method to manage a file, comprising:

receiving a first request at a client to access a file having a file name;

associating an identifier with said file name at said client; and

sending said first request to a server with said identifier and said file name.

2. The method of claim 1, further comprising storing said identifier and file name in memory.

3. The method of claim 1, further comprising receiving an acknowledgement message from said server.

4. The method of claim 1, further comprising:

receiving a second request at said client to access said file;

retrieving said identifier associated with said file name from said memory; and

sending said second request to said server using said associated identifier.

5. The method of claim 4, wherein said first and second requests specify a file operation.

6. A method to manage a file, comprising:

receiving a first request at a server to access a file having a file name and identifier; and

sending an acknowledgement message to said client.

7. The method of claim 6, further comprising:

searching for location information for said file;

associating said location information with said identifier; and

storing said location information and said identifier in memory.

8. The method of claim 7, further comprising:

receiving a second request at said server to access said file using said identifier; and

retrieving said location information from said memory using said identifier.

9. A method to manage a file, comprising:

receiving a file request having a file name at a client;

associating an identifier with said file name;

sending said identifier and said file name to a server;

searching for location information using said file name; and

storing said location information with said identifier.

10. The method of claim 9, further comprising sending an acknowledgement message to said client.

11. A method to manage file operations, comprising:

receiving a file request with a file name by a file system interface;

assigning a unique identifier to said file name by said file system interface ; and

sending said unique identifier and file name to a file system manager.

12. The method of claim 11, further comprising:

receiving said unique identifier and said file name at said file system manager;

searching for file information using said file name; and

storing said file information using said unique identifier.

13. An article comprising:

a storage medium;

said storage medium including stored instructions that, when executed by a processor, result in receiving a first request at a client to access a file having a file name, associating an identifier with said file name at said client, and sending said first request to a server with said identifier and said file name.

14. The article of claim 13, wherein the stored instructions, when executed by a processor, further results in storing said identifier and file name in memory.

15. The article of claim 13, wherein the stored instructions, when executed by a processor, further results in receiving an acknowledgement message from said server.

16. The article of claim 13, wherein the stored instructions, when executed by a processor, further results in receiving a second request at said client to access said file,

retrieving said identifier associated with said file name from said memory, and sending said second request to said server using said associated identifier.

17. An article comprising:

a storage medium;

said storage medium including stored instructions that, when executed by a processor, result in receiving a file request having a file name at a client, associating an identifier with said file name, sending said identifier and said file name to a server, searching for location information using said file name, and storing said location information with said identifier.

18. The article of claim 17, wherein the stored instructions, when executed by a processor, further result in sending an acknowledgement message to said client.

19. An apparatus to perform file management, comprising:

a client to receive a request for a file having a file name, said client to assign a unique identifier to said file name and send said unique identifier and said file name to a server; and

an interconnect system connected to said client to communicate said unique identifier and said file name to said server.

20. The apparatus of claim 19, further comprising a server to receive said unique identifier and file name, said server to locate information for said file and store said information using said unique identifier.

21. An apparatus to perform file management, comprising:

a client to assign an identifier to a file name;

a server to locate file information using said file name and store said file information using said identifier; and

an interconnect system to transport said file name and said identifier between said client and said server.

22. The apparatus of claim 21, wherein said client comprises an operating system service module.

23. The apparatus of claim 21, wherein said server comprises an intermediate service module.

24. The apparatus of claim 21, wherein said interconnect system operates in accordance with a peripheral component interconnect system and an I_2O system.

25. An apparatus to perform file management, comprising:

a file system interface to receive a request for a file having a file name and assign a unique identifier to said file name;

a file system manager to locate file information using said file name and store said file information using said unique identifier; and

a communications system to communicate said unique identifier and said file name between said file system interface and said file system manager.

26. The apparatus of claim 25, wherein said communications system comprises:

a communications medium comprising at least one of a group comprising twisted pair wire, co-axial cable, fiber optic and radio-frequencies; and

a communications interface to operate in accordance with a set of communication protocols.

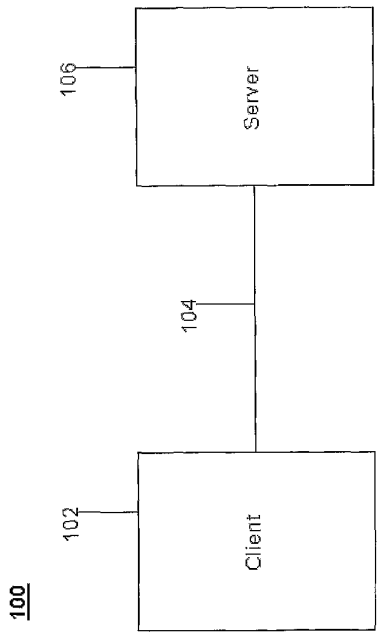


FIG. 1

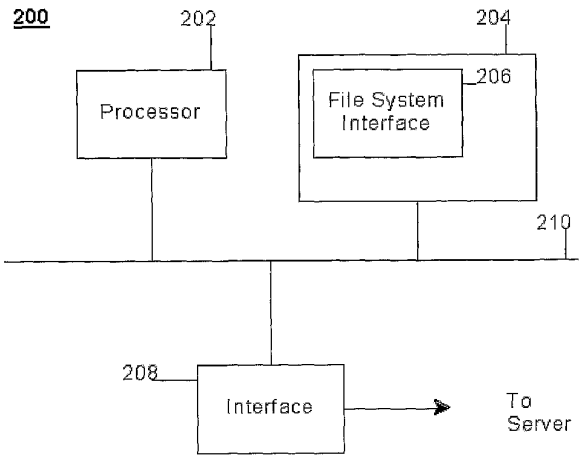

200
202
204
206
Processor
File System Interface
210
208
Interface
To Server

FIG. 2

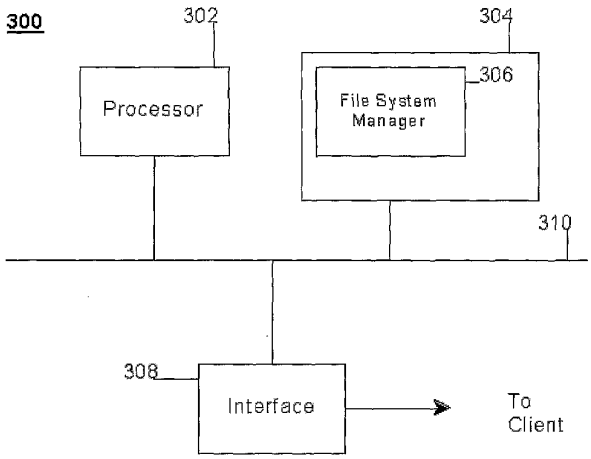
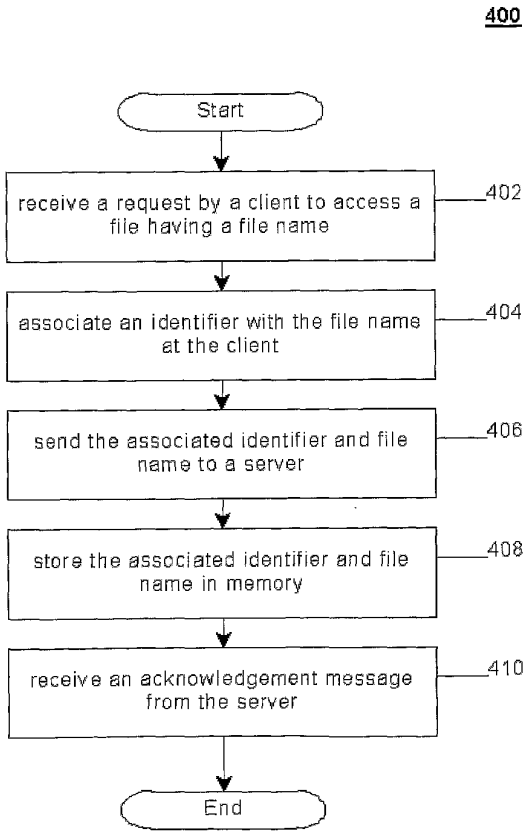
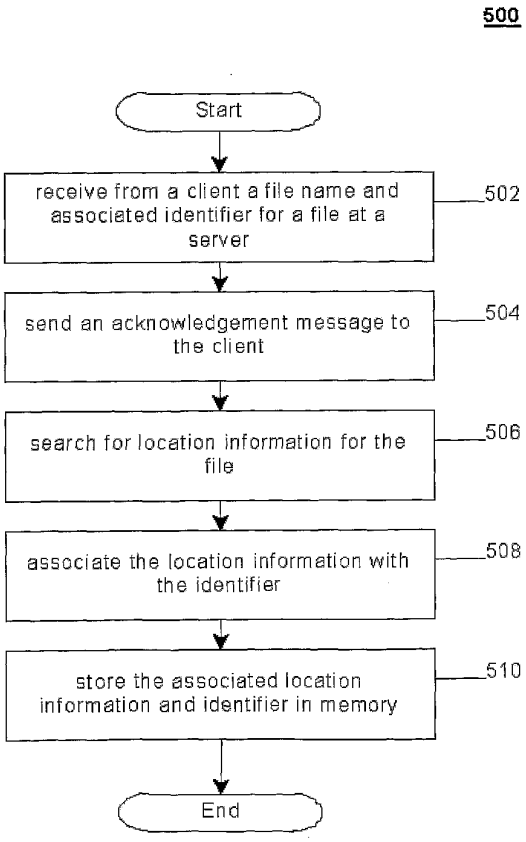

300
302
304
306
Processor
File System Manager
310
308
Interface
To Client

FIG. 3


400
Start
receive a request by a client to access a file having a file name
402
associate an identifier with the file name at the client
404
send the associated identifier and file name to a server
406
store the associated identifier and file name in memory
408
receive an acknowledgement message from the server
410
End


500
Start
receive from a client a file name and associated identifier for a file at a server
502
send an acknowledgement message to the client
504
search for location information for the file
506
associate the location information with the identifier
508
store the associated location information and identifier in memory
510
End

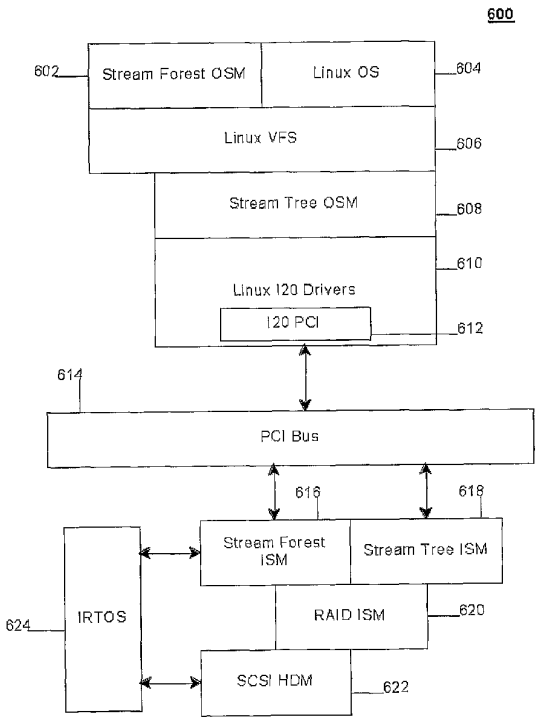

600
602
Stream Forest OSM
Linux OS
604
Linux VFS
606
Stream Tree OSM
608
610
Linux I20 Drivers
I20 PCI
612
614
PCI Bus
616
618
Stream Forest ISM
Stream Tree ISM
624
IRTOS
RAID ISM
620
SCSI HDM
622

FIG. 6

【国際公開パンフレット（コレクトバージョン）】

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
11 July 2002 (11.07.2002)

PCT

(10) International Publication Number
WO 2002/054286 A3

(51) International Patent Classification?: G06F 17/30

(21) International Application Number:
PCT/US2001/044883

(22) International Filing Date:
20 November 2001 (20.11.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/751,862 30 December 2000 (30.12.2000) US

(71) Applicant: INTEL CORPORATION [US/US]; 2200 Mission College Boulevard, Santa Clara, CA 95052 (US).

(72) Inventor: DAKE, Steven; 1972 West Bartlett Court, Chandler, AZ 85248 (US).

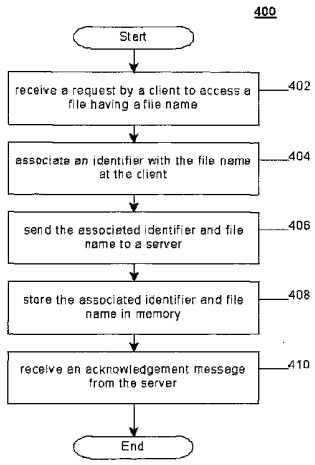
(74) Agents: MALLIE, Michael, J. et al.; Blakely Sokoloff Taylor & Zafman, 12400 Wilshire Boulevard, Los Angeles, CA 90025 (US).

(81) Designated States (national): AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PH, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States (regional): ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: METHOD AND APPARATUS TO IMPROVE FILE MANAGEMENT



(57) Abstract: A method and apparatus to manage a file in a computer and communications network is described comprising, receiving a first request at a client to access a file having a file name, associating an identifier with said file name at said client and sending said first request to a server with said identifier and said file name.

WO 2002/054286 A3

WO 2002/054286 A3

Published:
— *with international search report*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments*

(88) Date of publication of the international search report:
12 February 2004

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

【国際調査報告】

INTERNATIONAL SEARCH REPORT

Interne plication No
PCT/US 01/44883

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F17/30

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, WPI Data, PAJ, INSPEC, IBM-TDB

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	PAWLOWSKI B ET AL: "The NFS Version 4 Protocol" PROCEEDINGS OF 2ND INTERNATIONAL SANE CONFERENCE, 'Online! 22 - 25 May 2000, pages 1-20, XP002262978 Maastricht, The Netherlands Retrieved from the Internet: <URL:http://www.nluug.nl/events/sane2000/papers/pawlowski.pdf> 'retrieved on 2003-11-25!	1-3,6,7, 11-15,19
A	* sections 2, 4, 6 *	4,5, 8-10, 16-18, 20-26
	--- -/--	

[X] Further documents are listed in the continuation of box C.

[X] Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
28 November 2003	15/12/2003

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Tx. 31 651 epo nl, Fax: (+31-70) 340-3016	Wienold, N

Form PCT/ISA/210 (second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Interna ...lication No
PCT/US 01/44883

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication,where appropriate, of the relevant passages	Relevant to claim No.
X	CALLAGHAN B: "RFC 2054 - WebNFS Client Specification" IETF, October 1996 (1996-10), XP002242530 Retrieved from the Internet: <URL:http://www.ietf.org/rfc/rfc2054.txt> 'retrieved on 2003-05-26!	1-3,6,7, 11-15,19
A	* sections 3 - 5 *	4,5, 8-10, 16-18, 20-26
X	FRIESENHAHN B: "A FILE SYSTEM FOR THE WEB" BYTE, MCGRAW-HILL INC. ST PETERBOROUGH, US, vol. 21, no. 11, 1 November 1996 (1996-11-01), pages 63-64, XP000641067 ISSN: 0360-5280	1-3,6,7, 11-15,19
A	* whole document *	4,5, 8-10, 16-18, 20-26
X	US 5 742 817 A (PINKOSKI AIKO M) 21 April 1998 (1998-04-21)	1-3,6,7, 11-15,19
A	* abstract * column 1, line 10 -column 2, last line	4,5, 8-10, 16-18, 20-26

Form PCT/ISA/210 (continuation of second sheet) (July 1992)

INTERNATIONAL SEARCH REPORT

Information on patent family members

Interna application No
PCT/US 01/44883

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5742817 A	21-04-1998	NONE	

Form PCT/ISA/210 (patent family annex) (July 1992)

フロントページの続き

(81)指定国 AP(GH,GM,KE,LS,MW,MZ,SD,SL,SZ,TZ,UG,ZM,ZW),EA(AM,AZ,BY,KG,KZ,MD,RU,TJ,TM),EP(AT,BE,CH,CY,DE,DK,ES,FI,FR,GB,GR,IE,IT,LU,MC,NL,PT,SE,TR),OA(BF,BJ,CF,CG,CI,CM,GA,GN,GQ,GW,ML,MR,NE,SN,TD,TG),AE,AG,AL,AM,AT,AU,AZ,BA,BB,BG,BR,BY,BZ,CA,CH,CN,CO,CR,CU,CZ,DE,DK,DM,DZ,EC,EE,ES,FI,GB,GD,GE,GH,GM,HR,HU,ID,IL,IN,IS,JP,KE,KG,KP,KR,KZ,LC,LK,LR,LS,LT,LU,LV,MA,MD,MG,MK,MN,MW,MX,MZ,NO,NZ,PH,PL,PT,RO,RU,SD,SE,SG,SI,SK,SL,TJ,TM,TR,TT,TZ,UA,UG,UZ,VN,YU,ZA,ZW

（特許庁注：以下のものは登録商標）
Ｌｉｎｕｘ