(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第4921018号
(P4921018)

(45)発行日　平成24年4月18日(2012.4.18)　(24)登録日　平成24年2月10日(2012.2.10)

(51)Int.Cl.		F I		
G06F 12/10	(2006.01)	G06F	12/10	559
G06F 12/00	(2006.01)	G06F	12/00	514Z
G06F 9/46	(2006.01)	G06F	9/46	350
G06F 13/10	(2006.01)	G06F	12/00	599
		G06F	12/00	597U

請求項の数 2　(全 21 頁)　最終頁に続く

(21)出願番号　特願2006-102266 (P2006-102266)
(22)出願日　平成18年4月3日 (2006.4.3)
(65)公開番号　特開2006-294028 (P2006-294028A)
(43)公開日　平成18年10月26日 (2006.10.26)
審査請求日　平成21年1月22日 (2009.1.22)
(31)優先権主張番号　05102679.7
(32)優先日　平成17年4月5日 (2005.4.5)
(33)優先権主張国　欧州特許庁 (EP)

(73)特許権者　390009531
インターナショナル・ビジネス・マシーンズ・コーポレーション
ＩＮＴＥＲＮＡＴＩＯＮＡＬ　ＢＵＳＩＮＥＳＳ　ＭＡＳＣＨＩＮＥＳ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国１０５０４　ニューヨーク州　アーモンク　ニュー　オーチャード　ロード
(74)代理人　100108501
弁理士　上野　剛史
(74)代理人　100112690
弁理士　太佐　種一
(74)代理人　100091568
弁理士　市位　嘉宏

最終頁に続く

(54)【発明の名称】直接実行機能を提供するためのシステム、コンピュータシステム、方法およびプログラム

(57)【特許請求の範囲】
【請求項１】
メモリアドレス指定デバイスに対する直接実行機能を自動的に提供する方法であって、
アプリケーションプログラムによりファイルにアクセスするための要求を受理するステップと、
前記ファイルを保持するファイルシステムを識別するステップと、
前記ファイルシステムを管理するファイルシステムドライバが、前記メモリアドレス指定デバイスにあるファイルの内容のシステムメモリアドレスを検索するためのファイルシステムダイレクトアクセスインターフェースを提供するかどうかを決定するステップと、
前記ファイルシステムドライバが、前記ファイルシステムダイレクトアクセスインターフェースを提供しない場合には前記直接実行機能を使用せず、前記ファイルシステムドライバが、前記ファイルシステムダイレクトアクセスインターフェースを提供する場合には前記ファイルシステムが存在するデバイスを識別するステップと、
前記デバイスを管理するデバイスドライバが、前記メモリアドレス指定デバイスの指定されたブロックのシステムメモリアドレスを検索するためのデバイスダイレクトアクセスインターフェースを提供するかどうかを決定するステップと、
前記デバイスドライバが、前記デバイスダイレクトアクセスインターフェースを提供しない場合には前記直接実行機能を使用せず、前記デバイスドライバが、前記デバイスダイレクトアクセスインターフェースを提供する場合には前記ファイルシステムが、前記直接実行機能を可能にするように構成されるかどうかを決定するステップと、

前記ファイルシステムが、前記直接実行機能を可能にするように構成されると決定することに応答して、前記ファイルシステムダイレクトアクセスインターフェースを使用して、前記直接実行機能を提供するステップと、
を含む方法。
【請求項２】
ディジタルコンピュータの内部メモリ内に格納されたコンピュータプログラムであって、請求項１に記載の各ステップをコンピュータに実行させるためのコンピュータプログラム。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、全体としてオペレーティングシステム、特に、直接実行機能をインプリメントするためのシステムおよび方法を提供するオペレーティングシステムに関する。
【背景技術】
【０００２】
先行技術のコンピュータシステムは、プログラムおよびデータファイルを保持するための不揮発性大容量記憶装置（たとえば、ハードディスクドライブ）を含む。これらのファイルの内容は、ＲＡＭ（「ランダムアクセスメモリ」）タイプのシステムメモリ内にロードされ、ＣＰＵ（「中央処理装置」）によりアクセスまたは実行される。この動作は、一般に、アプリケーションプログラムに代わってオペレーティングシステムにより実行される。先行技術のコンピュータシステムおよびオペレーティングシステムは、仮想メモリおよびデマンドページングをサポートしている。アプリケーションは、システムメモリアドレスを直接使用して、アプリケーションが使用するコードおよびデータを指示するのではなく、代わりに「仮想アドレス」を使用して記憶場所を指示し、記憶場所は、ＣＰＵ回路によりインプリメントされ、オペレーティングシステムにより制御されるページング機構により、システムメモリアドレスに変換される。その結果、オペレーティングシステムは、プログラムおよびデータファイル全体をＲＡＭ内にロードする必要性をなくすことができる。むしろ、システムメモリは特定サイズの塊（「ページ」と呼ばれる）に分割され、オペレーティングシステムは、この特定のページにアクセスした時に、ファイル内容の対応する塊を各々のメモリページ内にロードする。このプロセスは、通常、「デマンドページング」と呼ばれる。
【０００３】
この方法の不利な点の１つに、ＲＡＭがプログラムおよびデータファイルの内容を保持する必要があり、他の目的に使用可能なＲＡＭの量が減少することがある。さらに、一般に、場合に応じて内容をＲＡＭ内にダウンロードする必要がある。したがって、先行技術のコンピュータシステムによっては、ＲＡＭと同様に、ＣＰＵが直接アクセスできる別のタイプの不揮発性記憶装置（「メモリアドレス指定デバイス」）が提供されている。メモリアドレス指定デバイスの先行技術の一実施態様は、フラッシュメモリカードである。メモリアドレス指定デバイスを用いれば、ＣＰＵは、最初にＲＡＭ内に内容をダウンロードすることなく、メモリアドレス指定デバイス上に格納されたコードおよびアクセスデータを直接実行できるようになる。メモリアドレス指定デバイス上に存在するコードを直接実行するこの方法は、「直接実行」と呼ばれる。仮想メモリをサポートするオペレーティングシステム上で実行されるアプリケーションに直接実行機能を提供するには、オペレーティングシステムは、アプリケーションのアドレス空間の特定の仮想アドレスが、メモリアドレス指定デバイスによりサポートされるアドレスの範囲内のシステムメモリアドレスにマップされるように、ページング機構を制御する必要がある。
【０００４】
先行技術のその他のコンピュータシステムは、仮想化機能を提供する。仮想化は、単一のコンピュータシステムを実行する「ハイパーバイザ」と呼ばれることの多いソフトウェアプログラムによってインプリメントされるが、複数の「ゲスト」オペレーティングシス

テムが各々別個の「仮想マシン」で同時に実行をすることができるようにする。オペレーティングシステムから各々の仮想マシンを見ると、仮想マシン自体が、ＣＰＵ、ＲＡＭおよびＩ／Ｏデバイスを備えた実際のコンピュータシステムとして内部で動作しているように見える。これらの仮想コンポーネントに対するアクセスは、ハイパーバイザによって傍受され、実際のコンポーネントに対するアクセスに変換される。これによって、コンピュータシステムのリソースを複数のゲストオペレーティングシステム間で共有し、システムリソース全体の利用率を増加させることができる。

【０００５】

いくつかの先行技術の仮想化コンピュータシステムの不利な点の１つとして、同じプログラムまたはデータが同じハイパーバイザ下で動作する複数のゲストにより同時にアクセスされることと、各々のゲストオペレーティングシステムは、それらの内容を保持するための仮想ＲＡＭを別個に割り当て、ひいては、ハイパーバイザが、前記内容の複数の同じコピーを物理ＲＡＭに割り当てる必要性が出てくる点が挙げられる。これは、他の目的に使用することのできるメモリが減少することを意味し、これによって効率的に同時に動作可能なゲストの数が制限されてしまう。したがって、先行技術のハイパーバイザによっては、複数のゲストから同時にアクセスできる物理メモリのセグメント（「共有メモリセグメント」）が提供されている。プログラムまたはデータファイルを共有メモリセグメント内に格納することにより、複数のゲストが、前記ファイルに同時にアクセスすることができ、最初に内容を仮想ＲＡＭ内にダウンロードする必要はなくなる。ゲストオペレーティングシステムから前記共有セグメントを見ると、物理メモリアドレス指定デバイスであるかのように見える。

【０００６】

データおよびプログラムファイルは、一般に、標準のファイルシステムレイアウトを使用するデバイス上に格納され、オペレーティングシステムによっては、異なる利用シナリオのために最適化された複数の異なるファイルシステムレイアウトを使用することができる。これをできるようにするため、先行技術のオペレーティングシステムは、一般に、複数のコンポーネント状に構造化される。オペレーティングシステムによっては、中央ファイルおよびメモリ管理コンポーネント、複数のファイルシステムドライバ、並びに複数のＩ／Ｏデバイスドライバが存在する。したがって、オペレーティングシステムは、ファイルおよびメモリ管理コンポーネントと組み合わせて、ファイルシステムドライバおよびＩ／Ｏデバイスドライバの適切な対を使用することにより、サポートされる何れかのＩ／Ｏデバイス上で、サポートされる何れかのファイルシステムレイアウトを使用できるようにする。しかし、先行技術のオペレーティングシステムは、既存のファイルシステムドライバを使用して、直接実行をできるようにする形態でメモリアドレス指定デバイスにアクセスすることはできない。メモリアドレス指定デバイスに対するアクセスをサポートする直接実行は、モノリシック形態でインプリメントされる。

【０００７】

実際、先行技術のオペレーティングシステムのインプリメンテーションによっては、標準ファイルシステムレイアウトを使用して、メモリアドレス指定デバイス上にデータを格納することができず、むしろ、このようなデバイスに特有の方法で、これらのデバイス上のデータを配列する必要がある。この配列には、特にＩ／Ｏデバイスおよびメモリアドレス指定デバイスの両方を使用するコンピュータシステムの場合、不利な点が複数ある。異なるファイルシステムレイアウトをサポートすることにより、システム管理はさらに難しくなる可能性がある。異なるレイアウトをフォーマット、管理、バックアップおよび復元するには、異なるツールが必要になる。既存のファイルの集合をＩ／Ｏデバイスからメモリアドレス指定デバイス、またはこの逆に移行することはさらに難しくなる。メモリアドレス指定デバイスにより要求される特定のレイアウトは、存在するすべての特徴（たとえば、アクセス制御および特権チェックをインプリメントする）に、標準のファイルシステムレイアウトを提供するわけではない。

【０００８】

もう１つの先行技術のインプリメンテーション（ｚＳｅｒｉｅｓ（ＩＢＭ　Ｃｏｒｐｏｒａｔｉｏｎの商標）上のＬｉｎｕｘ（Ｌｉｎｕｓ　Ｔｏｒｖａｌｄｓの米国およびその他の国における商標、以下、同様なので略）用ＸＩＰ２ＦＳファイルシステム）は、Ｌｉｎｕｘオペレーティングシステムが提供する標準ファイルシステムフォーマットの１つである第２拡張ファイルシステム（「ｅｘｔ２」）を使用して、プログラムおよびデータを仮想メモリアドレス指定デバイス上（ｚ／ＶＭ（ＩＢＭ　Ｃｏｒｐｏｒａｔｉｏｎの商標）ハイパーバイザにより提供される共有メモリセグメント）に格納するためのサポートを提供する。しかし、この方法には、上記の段落に記載した不利な点の殆どがあり、他の標準Ｌｉｎｕｘファイルシステムフォーマットを使用することができないほか、ＸＩＰ２ＦＳは、ｅｘｔ２のすべての特徴を提供するわけではない（たとえば、ＸＩＰ２ＦＳは、書き込みアクセスをサポートしない）。

【０００９】

ＸＩＰ２ＦＳのもう１つの不利な点は、ＸＩＰ２ＦＳがオペレーティングシステムの上記のコンポーネント構造に組み込まれないことであり、ＸＩＰ２ＦＳが、ｅｘｔ２ファイルシステムレイアウトを使用してファイルにアクセスした場合も、ＸＩＰ２ＦＳは、Ｌｉｎｕｘ　ｅｘｔ２ファイルシステムドライバを使用してアクセスするのではなく、代わりにｅｘｔ２ファイルシステムレイアウトにアクセスするのに必要なアクセスロジックを再インプリメントする。この場合もやはり、完全なｅｘｔ２ロジックの一部のみが再インプリメントされるため、ＸＩＰ２ＦＳはｅｘｔ２のすべての特徴をサポートするわけではない。さらに他の不利な点として、Ｌｉｎｕｘオペレーティングシステムの標準ｅｘｔ２ファイルシステムコンポーネントは、長期にわたって開発され、新しい特徴が追加され、たとえば、Ｌｉｎｕｘカーネルバージョン２．６が提供されたｅｘｔ２ファイルシステムドライバのバージョンは、非常に大きいディレクトリ構造、およびより精巧なアクセス制御機構に対してより迅速にアクセスするためのサポートを追加した。ＸＩＰ２ＦＳは、ｅｘｔ２ドライバのこのような強化から自動的に効果を得るのではなく、必要なすべての特徴をＸＩＰ２ＦＳコード内に再インプリメントする必要がある。

【発明の開示】

【発明が解決しようとする課題】

【００１０】

本発明の目的は、オペレーティングシステムによる直接実行機能を提供して、上記の先行技術の不利な点を解消する方法を提供することである。

【課題を解決するための手段】

【００１１】

本発明は、直接実行機能をインプリメントするための新しいシステムおよび方法を提供するオペレーティングシステムを開示する。

【００１２】

本発明の基礎となる先行技術のオペレーティングシステムは、アプリケーションプログラムに対するインターフェースを有するメモリ／ファイルマネジャと、メモリ／ファイルマネジャに対するファイルシステムＩ／Ｏインターフェースを有する少なくとも１個のファイルシステムドライバと、ファイルシステムドライバに対するデバイスＩ／Ｏインターフェースを有する少なくとも１個のデバイスドライバとを含み、少なくとも１個のデバイスドライバは、少なくとも１個のＩ／Ｏベースデバイス、およびファイルシステムドライバに対するデバイスＩ／Ｏインターフェースを有する少なくとも１個のデバイスドライバに対するアクセスを提供し、少なくとも１個のデバイスドライバは、少なくとも１個のメモリアドレス指定デバイスに対するアクセスを提供し、オペレーティングシステムは、少なくとも１個のメモリアドレス指定デバイスにアクセスするための直接実行機能を提供する。

【００１３】

先行技術のオペレーティングシステムは、直接実行機能をインプリメントするための以下の新しい独創的な機能コンポーネント：

メモリ／ファイルマネジャと少なくとも１個のファイルシステムドライバとの間のファイルシステムのダイレクトアクセスインターフェースであって、ファイルシステムのダイレクトアクセスインターフェースが、指定のオフセットにおけるファイルの内容のシステムメモリアドレスを検索する機能を検索する機能を提供し、ファイルがメモリアドレス指定デバイス上に存在するインターフェースと、

少なくとも１個のファイルシステムドライバと、少なくとも１個のメモリアドレス指定デバイスに対するアクセスを提供する少なくとも１個のデバイスドライバとの間のデバイスダイレクトアクセスインターフェースであって、デバイスダイレクトアクセスインターフェースが、少なくとも１個のメモリアドレス指定デバイスの指定ブロックのシステムメモリアドレスを検索する機能を提供するインターフェースとにより拡張され、

直接実行機能は、メモリ／ファイルマネジャ、少なくとも１個のファイルシステムドライバ、ファイルシステムのダイレクトアクセスインターフェースおよびデバイスダイレクトアクセスインターフェースを使用することにより、少なくとも１個のメモリアドレス指定デバイスに対するアクセスを提供する少なくとも１個のデバイスドライバにより提供される。

【００１４】

本発明の上記および追加の目的、特徴および効果は、以下の詳細な説明で明らかになる。

【００１５】

本発明の新規な特徴は、添付の請求の範囲に記載する。しかし、本発明自体、および好ましい使用形態、他の目的、および効果は、具体的な実施態様に関する以下の詳細な説明を参照し、添付の図面に関連して読むと最も良く理解されるであろう。

【発明を実施するための最良の形態】

【００１６】

図１は、コンピュータシステム１０のブロック図を示す。コンピュータシステム１０は、パーソナルコンピュータ、メインフレームコンピュータ、または他の何らかのタイプのコンピュータもしくはデータ処理システムで良い。コンピュータシステム１０は、以下に述べるように、別のコンピュータシステム上で動作するハイパーバイザにより提供される仮想マシンでも良い。コンピュータシステム１０は、中央演算処理装置（「ＣＰＵ」）１１と、ランダムーアクセスメモリ（「ＲＡＭ」）１２と、メモリアドレス指定デバイス１３と、Ｉ／Ｏベースデバイス１４とを備える。一実施態様では、メモリアドレス指定デバイス１３はフラッシュメモリカードで良い。他の実施態様では、メモリアドレス指定デバイス１３は、ＣＰＵ１１がメモリ動作のために直接アクセスできる何らかのデバイスで良い。一実施態様では、Ｉ／Ｏベースデバイス１４はハードディスクドライブで良い。他の実施態様では、Ｉ／Ｏベースデバイス１４は、Ｉ／Ｏ操作を使用して、データをＲＡＭ１２からコピーするか、またはＲＡＭ１２にコピーできる何らかのデバイスで良い。コンピュータシステム１０は、メモリアドレス指定デバイス１３またはＩ／Ｏベースデバイス１４あるいはその両方の複数のインスタンスも含む場合がある。ＣＰＵ１１、ＲＡＭ１２、並びにデバイス１３および１４は、システムバス１５に結合される。ＣＰＵ１１は、メモリ動作のために、ＲＡＭ１２およびメモリアドレス指定デバイス１３に直接アクセスすることができる。ＣＰＵ１１は、メモリ動作のためにＩ／Ｏベースデバイス１４に直接アクセスすることはできないが、データは、Ｉ／Ｏ操作を使用して、デバイス１４からＲＡＭ１２に、あるいは逆にコピーすることができる。コンピュータシステム１０は、１つまたは複数のアプリケーションプログラムを実行させることが可能なオペレーティングシステムを実行させる（以下で詳細に説明する）。オペレーティングシステムは、コンピュータシステム１０の様々なリソース（ＣＰＵ１１、ＲＡＭ１２、デバイス１３および１４）に対するアプリケーションプログラムによるアクセスを管理および調整する。

【００１７】

いくつかの実施態様では、コンピュータシステム１０は、別のコンピュータシステム上で動作するハイパーバイザによりエミュレートされる仮想マシンで良い。図２は、コンピ

ュータシステム１０が、仮想ＣＰＵ１１、仮想ＲＡＭ１２、並びに仮想デバイス１３および１４から成るような一実施態様を示す。すべての仮想コンポーネントはハイパーバイザ２１により提供され、ハイパーバイザ２１は、別のコンピュータシステム２０上で動作するソフトウェアプログラムであり、それ自体はＣＰＵ、ＲＡＭおよびデバイスから成る。ハイパーバイザ２１は、完全にソフトウェアによって、またはコンピュータシステム２０が提供する視覚化ハードウェア支援機能を使用することによって、仮想コンポーネント１０～１４を提供する。仮想マシン１０のほかに、他の仮想マシン２２が、コンピュータシステム２０上のハイパーバイザ２１下で同時に動作する。一実施態様では、ハイパーバイザ２１は、ＩＢＭ Ｃｏｒｐ．が製造販売するｚ／ＶＭ（ＩＢＭ Ｃｏｒｐｏｒａｔｉｏｎの商標）ソフトウェアプログラムであって、ＩＢＭｅＳｅｒｖｅｒ（ＩＢＭ Ｃｏｒｐｏｒａｔｉｏｎの商標）ｚＳｅｒｉｅｓ（ＩＢＭ Ｃｏｒｐｏｒａｔｉｏｎの商標）メインフレームコンピュータ上で動作するプログラムで良い。この実施態様では、コンピュータシステム１０は、ｚ／ＶＭにより提供される仮想マシンで良く、メモリアドレス指定デバイス１３は、ｚ／ＶＭに基づいて画定される非連続保管セグメント（「ＤＣＳＳ）」）で良い。ＤＣＳＳは、複数の仮想マシンが同時に使用可能なｚ／ＶＭにより管理されるメモリのセグメントである。ｚ／ＶＭは、複数の仮想マシンがＤＣＳＳを同時に使用可能である場合でも、コンピュータシステム２０の実ＲＡＭ内におけるＤＣＳＳの内容の１つのコピーのみを保持する。

【００１８】

ＣＰＵ１１がメモリ動作のために直接アクセス可能なすべてのコンポーネントは、図３に示すように、コンピュータシステム１０のシステムメモリアドレス空間３０を構成する。ＲＡＭ１２およびメモリアドレス指定デバイス１３は、システムメモリアドレス空間３０の一部である。さらに、その他のコンポーネント（図示しない）は、システムメモリアドレス空間３０、たとえば読出し専用メモリ（「ＲＯＭ」）またはビデオカードフレームバッファである。ＣＰＵ１１により実行されるすべてのプログラムインストラクション、およびＣＰＵ１１により実行されるインストラクションによりアクセスされるすべてのメモリデータは、実行が生じた時点で、システムメモリアドレス空間３０内に存在しなければならない。使用可能なメモリの見掛けのサイズを増加し、同じコンピュータシステム上で同時に動作する異なるアプリケーションプログラムが、偶発的に互いに他のメモリを修正するのを防止するため、オペレーティングシステムは、各々のアプリケーションプログラムに「仮想アドレス空間」を提供し、仮想アドレス空間内のシステムメモリアドレス空間の選択部分に対するアクセスを提供する。ＣＰＵ１１は、アプリケーションコードを実行するが、アプリケーションの仮想アドレス空間内に存在するメモリにのみアクセスすることができる。仮想アドレス空間とシステムメモリアドレス空間との間で変換することができるように、仮想アドレス空間とシステムメモリアドレス空間の両方が、一般に「ページ」と呼ばれる等サイズの塊状に分割される。

【００１９】

図３は、コンピュータシステム１０のシステムメモリアドレス空間３０を示す。さらに、アプリケーションプログラムの仮想アドレス空間３１が示されている。仮想アドレス空間３１でアドレス指定できる各々のページについては、当該ページの状態を実際に記述するページ記述子が存在しなければならない。ページは、ある時点でシステムメモリアドレス空間３０内に存在しても、あるいは存在しなくても良い。ページがシステムメモリアドレス空間３０内に存在する場合、ページ記述子はその事実を指示し、システムメモリアドレス空間３０内のページの位置を指示することになる。ページがシステムメモリアドレス空間３０内に存在しない場合、記述子はその事実を指示し、アプリケーションプログラムがこのページを介してアクセスしようとしている内容の場所をオペレーティングシステムが確認できるようにする追加の情報を含むことになる。仮想アドレス空間に対するすべてのページ記述子の集合は、ページテーブルと呼ばれる。仮想アドレス空間のページテーブルは、オペレーティングシステムにより維持される。

【００２０】

ＣＰＵ１１が仮想アドレス空間３１でアプリケーションコードを実行している間、ＣＰＵ１１によりアクセスされるすべてのメモリアドレスは、仮想アドレス空間３１用のページテーブルを使用して、仮想アドレス空間３１内の仮想アドレスからシステムメモリアドレス空間３０内のシステムメモリアドレスに変換される。この変換はページング機構により実行され、ページング機構は、ハードウェアもしくはソフトウェアインプリメンテーション、またはこれらの組合せで良い。実施態様によっては、ページング機構は、ＣＰＵ１１内に存在するページングユニットによりインプリメントされる。ＣＰＵ１１が、システムメモリアドレス空間３０に対応するページを現在持たない仮想アドレス空間３１内のページにアクセスする場合、ページング機構は、ＣＰＵ１１にページ障害割込みを生成させる。この割込みにより、「ページ障害ハンドラ」と呼ばれるオペレーティングシステムコードが、必要な内容をシステムメモリアドレス空間３０内のいくつかのページ内にもたらし、ページテーブルを更新して、仮想ページをシステムメモリアドレス空間３０内のこのページにマップする。次に、アプリケーションは、ページの実行およびアクセスを継続する。このプロセスは、一般に、「デマンドページング」と呼ばれる。

【００２１】

図３に示す実施例の状況では、仮想アドレス空間３１は、現在４つのページを保持している。３つのページ（３２ａ～３２ｃ）は、現在実行されているアプリケーションコードを含み、１つのページ（３２ｄ）は、アプリケーションによりアクセスされるデータを含む。ページ３２ａ～３２ｃに含まれるアプリケーションコードは、ブロック３４ａ～３４ｃ内のＩ／Ｏベースデバイス１４上に存在するアプリケーションプログラムファイルからロードされる。アプリケーションコードページ３２ａおよび３２ｂは、システムメモリアドレス空間３０内に現在実際に存在しており、ＲＡＭ１２のページ３３ａおよび３３ｂ内に存在する。同様に、データページ３２ｄはシステムメモリアドレス空間内に存在しており、ＲＡＭ１２のページ３３ｄ内に存在する。アプリケーションコードページ３２ｃは、現在、システムメモリアドレス空間３０内に存在しない。アプリケーションがページ３２ｃにアクセスすると、オペレーティングシステムのページ障害ハンドラは、ＲＡＭ１２に新しいページ３３ｃ（図示しない）を割り当て、Ｉ／Ｏ操作を実行してブロック３４ｃの内容をページ３３ｃにコピーし、ページテーブルを更新して、仮想アドレス空間３１のページ３２ｃをシステムメモリアドレス空間３０のページ３３ｃにマップする。ページ３２ａおよび３２ｂの場合、このプロセスは、図３に示す時点で既に完了している。データページ３２ｄ／３３ｄは、アプリケーションにより実行時に生成された内容を保持する。この内容は、Ｉ／Ｏベースデバイス１４からロードされたものではない。

【００２２】

図３に示すように、ＣＰＵ１１がＩ／Ｏベースデバイス１４上に存在するアプリケーションプログラムを実行している時、ＲＡＭ１２のページは、実行しているアプリケーションプログラムのコードを保持する必要がある。しかし、アプリケーションプログラムがメモリアドレス指定デバイス上に存在する場合、その必要はなく、ＲＡＭ１２のより多くのページを他の目的に使用できる（あるいは別法によると、より少ないＲＡＭの全体量で、意図されたタスクを履行することができる）。このプロセスは、一般に「直接実行」と呼ばれる。図４は、仮想アドレス空間３１で実行されている図３と同じアプリケーションを示すが、この場合、アプリケーションは、Ｉ／Ｏベースデバイス１４ではなくメモリアドレス指定デバイス上に存在し、直接実行されている。図４に示すように、アプリケーションプログラムファイルからのアプリケーションコードは、メモリアドレス指定デバイス１３のブロック３５ａ～ｃに存在する。メモリアドレス指定デバイス１３は、システムメモリアドレス空間３０内に存在するため、ブロック３５ａおよび３５ｂは、それぞれ仮想アドレス空間３１のページ３２ａおよび３２ｂに直接マップされる。同様に、アプリケーションがページ３２ｃにアクセスすると、オペレーティングシステムのページ障害ハンドラは、単に、仮想アドレス空間３１のページ３２ｃとシステムメモリアドレス空間３０のページ３５ｃとの間の別のマッピングを確立し、オペレーティングシステムは、どのページもＲＡＭ１２に割り当てる必要はない。一方、データページ３２ｄは、図３に示されたシ

ナリオと同様に、ＲＡＭ１２のページ３３ｄにマップされる点に注目する。

【００２３】

図５は、本発明を具現しない先行技術のオペレーティングシステム４０のコンポーネント構造を視覚化する。図３および図４に示すデマンドページングおよび直接実行機能をインプリメントするために必要なコンポーネントのみを図示する。オペレーティングシステム４０は、インターフェース４８を介したアプリケーションプログラム４９から発せられるＩ／Ｏベースデバイス１４またはメモリアドレス指定デバイス１３上に存在するファイルへのアクセス要求を処理するメモリ／ファイルマネジャ４１を含む。Ｉ／Ｏベースデバイス１４上のファイルにアクセスするために、メモリ／ファイルマネジャ４１は、ファイルシステムＩ／Ｏインターフェース４５を介して、ファイルシステムドライバ４３と対話する。オペレーティングシステム４０は、複数のバージョンのファイルシステムドライバを含み、各々のファイルシステムドライバは、特定のファイルシステムタイプの処理を担い、すべてのファイルシステムドライバは、同じファイルシステムＩ／Ｏインターフェース４５をインプリメントする。同様に、オペレーティングシステム４０は複数のバージョンのデバイスドライバを含み、各々のデバイスドライバは特定タイプのデバイスの処理を担い、すべてのデバイスドライバは同じデバイスＩ／Ｏインターフェース４６をインプリメントする。

【００２４】

図５に示すように、オペレーティングシステム４０のコンポーネント構造内では、デバイスドライバ４４（およびすべてのデバイスドライバ）が担っているのは、データをＩ／Ｏベースデバイス上の指定の場所からＲＡＭ内に、あるいはこの逆にコピーすることである。デバイスドライバは、デバイスの内容またはこれらの内容が組織化される方法に関する知識は持たない。デバイス上に存在するデータは、一般に、各々が「ブロック数」により識別される「ブロック」と呼ばれる塊状に分割される。デバイスＩ／Ｏインターフェース４６は、デバイスドライバの要求により、ブロック数Ｂで識別されるブロックから、アドレスＡで開始するＲＡＭのブロックに、あるいはこの逆にデータをコピーできるようにする。

【００２５】

デバイス上にデータを構造化ストレージで格納できるように、オペレーティングシステム４０は、ファイルシステムドライバを提供する。ファイルシステムドライバは、「ファイル」の集合としてデバイス上に格納されたデータへのアクセスを可能にし、各々のファイルは、順に並んだバイトのシーケンスの抽象化を提供する。すべてのファイルは、一般に「ファイル名」と呼ばれるファイルシステムにより提供されるいくつかの手段により識別される。ファイルシステムは、各々のファイルのどのバイトが、下のデバイスのどのブロック内に格納されるかというトラックを保持する。この記録保持を実行するのに必要な情報は、通常、「ファイルシステムメタデータ」と呼ばれ、下にあるデバイス上に格納される。ファイルシステムメタデータの特定のレイアウトは、ファイルシステムごとに異なる。図５に示すように、オペレーティングシステム４０のコンポーネント構造内では、ファイルシステムメタデータのレイアウトを処理するために必要なすべてのロジックをインプリメントするのは、ファイルシステムドライバ４３（およびすべてのファイルシステムドライバ）が担っている。ファイルシステムＩ／Ｏインターフェース４５は、あるファイル名で識別されるファイルＦからデータを、ファイルＦ内の指定のオフセット０で開始して、アドレスＡで開始するＲＡＭのブロック内に、あるいはこの逆にコピーするというファイルシステムドライバの要求を認める。このような要求を処理するため、ファイルシステムドライバは、ファイルシステムメタデータを調査して、下にあるデバイスのどのブロックＢが、ファイルＦ内のオフセット０に対応するデータを保持するかを決定し、下にあるデバイスを処理するデバイスドライバのデバイスＩ／Ｏインターフェース４６を使用して、前記デバイスとメモリの指定ブロックとの間でコピーを行うかを決定する。

【００２６】

したがって、図５に示すように、オペレーティングシステム４０のコンポーネント構造

内では、図３に示すデマンドページング機能は以下のようにインプリメントされる：アプリケーションが、システムメモリアドレス空間内で現在使用されていないページにアクセスすると、オペレーティングシステムのページ障害ハンドラ（通常、メモリ／ファイルマネジャ４１の一部）は、ページ記述子から、アプリケーションがページ中のどの内容を求めているのか決定する。ページ記述子は、ファイルＦ、および前記内容が発見されるファイルＦ内のオフセット０を指定することにより、前記内容を識別する。次に、メモリ／ファイルマネジャ４１は、新しいページをＲＡＭ１２に割り当て、ファイルシステムドライバ４３からファイルシステムＩ／Ｏインターフェース４５を介して、データをファイルＦ内のオフセット０から前記の新しいページにコピーするように要求する。上記のとおり、ファイルシステムドライバ４３は、ファイルシステムメタデータを調査して、下にあるデバイス（この場合、Ｉ／Ｏベースデバイス１４）のどのブロックＢがデータを保持しているかを決定し、デバイスドライバ４４のデバイスＩ／Ｏインターフェース４６を使用して、そのデータを前記の新しいページにコピーする。Ｉ／Ｏ操作が完了すると、メモリ／ファイルマネジャ４１は、それに応じてページテーブルを更新する。

【００２７】

しかし、先行技術のファイルシステムＩ／Ｏインターフェース４５は、この目的に適していないため、ファイルシステムドライバ４３を使用して、メモリアドレス指定デバイス１３上に存在するファイルに対する直接実行アクセスを実行するのは不可能である。その代わりに、図５に示すように、オペレーティングシステム４０のコンポーネント構造内では、直接実行アクセスは、メモリ／ファイルマネジャ４１内にしっかりと組み込まれているＸＩＰマネジャ４２により処理され、メモリアドレス指定デバイス１３に直接アクセスする。したがって、ＸＩＰマネジャ４２は、メモリアドレス指定デバイス１３にアクセスし、前記デバイス上に格納されたデータのファイルシステムレイアウトを処理することの両方を担っている。ＸＩＰマネジャ４２によりサポートされるファイルシステムレイアウト以外のファイルシステムレイアウトを使用して、メモリアドレス指定デバイス１３上のデータにアクセスすることは、オペレーティングシステム４０が、さもなければ、このようなファイルシステムにファイルシステムドライバを提供すると思われる場合でも不可能である。

【００２８】

本発明は、このような制約を除去する。図６は、本発明を具現するメモリアドレス指定デバイス１３に対する直接実行アクセスをインプリメントするオペレーティングシステム５０のコンポーネント構造を示す。図５に示す先行技術のオペレーティングシステム４０と比べて、オペレーティングシステム５０は、コンピュータシステム１０のすべてのハードウェアコンポーネント（ＲＡＭ１２、Ｉ／Ｏベースデバイス１４、メモリアドレス指定デバイス１３）に対して同じインターフェースを保持する。また、オペレーティングシステム５０は、アプリケーションプログラム４９に対して同じインターフェース４８を使用する。メモリ／ファイルマネジャ５１は、先行技術（図５参照）により提供される修正バージョンのメモリ／ファイルマネジャ４１であり、ファイルシステムドライバ５２は、先行技術（図５参照）により提供される修正バージョンのファイルシステムドライバ４３である。オペレーティングシステム５０は、メモリアドレス指定デバイス１３にアクセスするデバイスドライバ５３も提供する。オペレーティングシステム４０の先行技術のいくつかのバージョンは、同様にメモリアドレス指定デバイス１３にアクセスするデバイスドライバも提供していたが、このようなドライバは、デバイスＩ／Ｏインターフェース４６（図５参照）のみをインプリメントし、したがって、直接実行機能を提供することはできなかった点に注目する。この機能は、ＸＩＰマネジャ４２によりインプリメントされたが、ＸＩＰマネジャ４２は、デバイスドライバを使用せずに、メモリアドレス指定デバイス１３に直接アクセスする。図６に示すように、本発明は、ＸＩＰマネジャのコンポーネントを必要としない。その代わりに、直接実行機能は、この場合、メモリ／ファイルマネジャ５１、ファイルシステムドライバ５２およびデバイスドライバ５３内に組み込まれる。これは、２つの新しいインターフェース：ファイルシステムドライバ５２により提供される

ファイルシステムのダイレクトアクセスインターフェース５４、およびデバイスドライバ５３により提供されるデバイスダイレクトアクセスインターフェース５５を使用することにより可能である。デバイスダイレクトアクセスインターフェース５５の基本的な特徴は、システムメモリアドレス空間内に存在するメモリアドレス指定デバイスのブロックＢのシステムメモリアドレスＡを検索する手段を提供することである。同様に、ファイルシステムのダイレクトアクセスインターフェース５４の基本的な特徴は、ファイルＦが、システムメモリアドレス空間内に存在するメモリアドレス指定デバイス上に存在する場合、オフセット０におけるファイルＦの内容のシステムメモリアドレスＡを検索する手段を提供することである。これらのダイレクトアクセスインターフェースの使用については、以下に詳細に説明する。

【００２９】

図６に示すように、オペレーティングシステム５０は、インターフェース４８を通してアプリケーションプログラム４９と対話する。インターフェース４８の一部は、アプリケーションプログラム４９によりトリガされるデマンドページング要求から成り、アプリケーションプログラム４９は、システムメモリアドレス空間３０のページに現在マップされない仮想アドレス空間３１のページにアクセスする。（仮想アドレス空間３１およびシステムメモリアドレス空間３０は、図３および図４に示され、仮想アドレス空間３１は、アプリケーション４９の仮想アドレス空間に対応する点に注目する。）インターフェース４８のその他の部分は、アプリケーションプログラム４９によるオペレーティングシステム５０に対する要求（「システムコール」）であって、Ｉ／Ｏベースデバイス１４またはメモリアドレス指定デバイス１３上に存在するファイルの内容を読み取るか、書き込むか、さもなければアクセスするための要求から成る。メモリ／ファイルマネジャ５１は、インターフェース４８を介したアプリケーションプログラム４９による要求を処理するオペレーティングシステム５０のコンポーネントである。Ｉ／Ｏベースデバイス１４またはメモリアドレス指定デバイス１３上に存在するファイルの内容にアクセスするには、メモリ／ファイルマネジャ５１は、ファイルシステムＩ／Ｏインターフェース４５またはファイルシステムのダイレクトアクセスインターフェース５４あるいはその両方を介して、ファイルシステムドライバ５２と対話する。オペレーティングシステム５０は、各々が特定のファイルシステムレイアウトの処理を担う複数のバージョンのファイルシステムドライバを含む。すべてのファイルシステムドライバは同じファイルシステムＩ／Ｏインターフェース４５をインプリメントし、いくつかのファイルシステムドライバは、さらに、メモリアドレス指定デバイス上に存在するファイルに対する直接実行アクセスを実行するのに適するファイルシステムのダイレクトアクセスインターフェース５４をインプリメントする。ファイルシステムドライバ５２は、デバイスＩ／Ｏインターフェース４６またはデバイスダイレクトアクセスインターフェース５５あるいはその両方を介して、デバイスドライバ４４および５３と対話する。オペレーティングシステム５０は、各々が特定タイプのデバイスの処理を担う複数バージョンのデバイスドライバも含む。すべてのデバイスドライバは同じデバイスＩ／Ｏインターフェース４６をインプリメントし、メモリアドレス指定デバイス１３のデバイスドライバは、さらに、メモリアドレス指定デバイス１３上に存在するファイルに対する直接実行アクセスを実行するのに適するデバイスダイレクトアクセスインターフェース５５をインプリメントする。Ｉ／Ｏインターフェース４５およびダイレクトアクセスインターフェース５４の両方を提供するファイルシステムドライバの場合、メモリ／ファイルマネジャ５１は、このようなファイルシステムドライバにより処理されるファイルに対する定期的なアクセスおよび直接実行アクセスの両方を実行すること可能である。図７および図８は、それぞれ定期的なアクセスおよび直接実行アクセスを実行するのに必要な制御フローを詳細に示す。図９は、２つの方法のどちらを特定ファイルのアクセスに使用するかを選択するために必要な決定ロジックを詳細に示す。

【００３０】

図７は、オペレーティングシステム５０が、Ｉ／Ｏベースデバイス１４上に存在するファイルにアクセスするための要求を処理する時の制御フローを示す。この制御フローは、

先行技術のオペレーティングシステム４０が対応するアクセスに使用する制御フローと同じである点に注目する。制御フローのステップは、順にアクション６０ａ～ｆにより表現される。ここに示す特定のアクションは、アプリケーション４９が、仮想アドレス空間３１のページ３２ｃにアクセスしようと試みた後、図３に示す状況に対応する。このページは、システムメモリアドレス空間３０のどのページにも現在マップされていないため、ページ障害割込みが生成され、オペレーティングシステム５０のメモリ／ファイルマネジャ５１コンポーネントにより処理される。これは、図７にアクション６０ａで表現される。メモリ／ファイルマネジャ５１は、ページ３２ｃのページ記述子から、アプリケーションが、ページ３２ｃの内容が、オフセット０におけるファイルＦの内容に対応するはずであると予想していることを読み取る。メモリ／ファイルマネジャ５１は、ファイルＦが、ファイルシステムドライバ５２により処理されるファイルシステム上に存在すると決定する。メモリ／ファイルマネジャ５１は、さらに、ファイルＦが直接実行をサポートしないと決定する（この点について、以下で詳細に説明する）。メモリ／ファイルマネジャ５１は、次に、ＲＡＭ１３内の自由ページ３３ｃを割り当て、そのシステムメモリアドレスＡを決定し、ファイルシステムドライバ５２のファイルシステムＩ／Ｏインターフェース４５を介して、オフセット０におけるファイルＦの内容をシステムメモリアドレスＡのページにコピーするように要求する（アクション６０ｂ）。ファイルシステムドライバ５２は、オフセット０におけるファイルＦのデータが、Ｉ／Ｏベースデバイス１４のブロックＢ上に存在し、デバイスドライバ４４が、Ｉ／Ｏベースデバイス１４へのアクセスを担うことを決定する。ファイルシステムドライバ５２は、次に、デバイスドライバ４４のデバイスＩ／Ｏインターフェース４６を介して、Ｉ／Ｏベースデバイス１４のブロックＢをシステムメモリアドレスＡのページにコピーするように要求する（アクション６０ｃ）。デバイスドライバ４４は、Ｉ／Ｏベースデバイス４４上のＩ／Ｏ操作６１に、そのコピーを実行させる。Ｉ／Ｏ操作６１が完了すると、デバイスドライバ４４は、デバイスＩ／Ｏインターフェース４６を介して、ファイルシステムドライバ５２に完了を報告し（アクション６０ｄ）、ファイルシステムドライバ５２は、同様に、ファイルシステムＩ／Ｏインターフェース４５を介してメモリ／ファイルマネジャ５１に完了を報告する（アクション６０ｅ）。メモリ／ファイルマネジャ５１は、最後に、システムメモリアドレス空間３０におけるページ３３ｃに対する仮想アドレス空間３１のページ３２ｃのマッピングを確立する。

【００３１】

図８は、同様に、オペレーティングシステム５０が、メモリアドレス指定デバイス１３上に存在するファイルに対する直接実行アクセスを実行するという要求を処理する時の制御フローを示す。このフローは、先行技術のオペレーティングシステム４０が対応するアクセスに使用するフローとは異なり、本発明により記述される新しいダイレクトアクセスインターフェースを使用する点に注目する。さらに、メモリアドレス指定デバイス１３上のファイルに対するいくつかのアクセスは、直接実行アクセスに適さず、この場合、その代わりに、図７に示し、上記で説明した制御フローと等価な制御フローを使用して、ファイルに対する定期的なＩ／Ｏアクセスが実行される。これは、デバイスドライバ５３が、デバイスドライバ４４と同様にデバイスＩ／Ｏインターフェース４６もインプリメントするために可能である。

【００３２】

図８に示す制御フローのステップは、順にアクション７０ａ～ｆで表現される。図８に示す特定のアクションは、アプリケーション４９が仮想アドレス空間３１のページにアクセスすることを試みた後の図４に示す状況に対応する。図７のように、ページ障害割込みが生成されて、メモリ／ファイルマネジャ５１により処理され（アクション７０ａ）、メモリ／ファイルマネジャ５１は、やはり、ページ３２ｃのページ記述子から、アプリケーションが、ページ３２ｃの内容がオフセット０におけるファイルＦの内容に対応するはずであると予想していることを読み取る。メモリ／ファイルマネジャ５１は、ファイルＦが、ファイルシステムドライバ５２により処理されるファイルシステム上に存在すると決定する。メモリ／ファイルマネジャ５１は、さらに、ファイルＦが直接実行をサポートしな

いと決定する（この点について、以下で詳細に説明する）。メモリ／ファイルマネジャ５１は、次に、ファイルシステムドライバ５２のファイルシステムのダイレクトアクセスインターフェースを介して、オフセット０におけるファイルＦの内容のシステムメモリアドレスを検索するように要求する（アクション７０ｂ）。ファイルシステムドライバ５２は、オフセット０におけるファイルＦの内容が、メモリアドレス指定デバイス１３のブロックＢ上に存在し、デバイスドライバ５３がメモリアドレス指定デバイス１３へのアクセスを担うと決定する。ファイルシステムドライバ５２は、次に、デバイスドライバ５３のデバイスダイレクトアクセスインターフェース５５を介して、メモリアドレス指定デバイス１３のブロックＢのシステムメモリアドレスを検索するように要求する（アクション７０ｃ）。デバイスドライバ５３は、メモリアドレス指定デバイス１３のブロックＢが、システムメモリアドレス空間３０のページ３５ｃに対応し、デバイスダイレクトアクセスインターフェース５５を介して、そのアドレスＡをファイルシステムドライバ５２に返し（アクション７０ｄ）、同様に、ファイルシステムのダイレクトアクセスインターフェース５４を介して、アドレスＡをメモリ／ファイルマネジャ５１に返す（アクション７０ｅ）と決定する。メモリ／ファイルマネジャ５１は、最後に、システムメモリアドレス空間３０のアドレスＡにおけるページ３５ｃに対する仮想アドレス空間３１のページ３２ｃのマッピングを確立する。

【００３３】

図９は、ファイルシステムＩ／Ｏインターフェースまたはファイルシステムのダイレクトアクセスインターフェースを使用してファイルＦにアクセスするかどうかを決定する時のオペレーティングシステム５０（図６）内の制御フローを示す。オペレーティングシステムは、最初に、どのファイルシステムドライバが、ファイルＦを保持するファイルシステムＦＳに責任を負うかを決定する。ファイルシステムドライバがファイルシステムのダイレクトアクセスインターフェースを全然サポートしていない場合、ファイルシステムＩ／Ｏインターフェースが使用される。さもなければ、オペレーティングシステムは、どのデバイスドライバが、ファイルシステムＦＳの下にあるデバイスＤに責任を負うかを決定する。デバイスドライバが、デバイスダイレクトアクセスインターフェースを全然サポートしていない場合、同様にファイルシステムＩ／Ｏインターフェースが使用される。さもなければ、オペレーティングシステムは、ファイルシステムＦＳが、ファイルＦに直接実行アクセスするためにユーザにより構成されたかどうかを決定する。構成された場合は、ファイルシステムのダイレクトアクセスインターフェースが使用され、さもなければ、ファイルシステムＩ／Ｏインターフェースが使用される。実施態様によっては、ユーザは、ＦＳ上のすべてのファイルに対する直接実行アクセスをできるようにするか、またはＦＳ上のすべてのファイルに対して直接実行アクセスできないようにするかのみ選択する。他の実施態様では、ユーザは、各々の単一ファイルに関して別個に、このような選択を行う。さらに、実施態様によっては、その他の構成可能なファイルシステムパラメータが、ユーザにより、直接実行と互換性のある値に設定された場合にのみ、直接実行アクセスを可能にすることができ、たとえば、ファイルシステムのブロックサイズは、複数のシステムメモリページのサイズに等しい値、または複数のシステムメモリページのサイズに構成する必要がある。

【００３４】

オペレーティングシステムは、必ずしも、ファイルＦに１回アクセスするごとに、図９に示す完全な決定ロジックを実行する必要はない点に注目する。その代わりに、選択は、ファイルＦが最初にアクセスされ、ファイルＦに関連するオペレーティングシステムのデータ構造内に記憶されている時点で選択され、その後のアクセスは、前記データ構造内に格納されている決定結果を再利用する。

【００３５】

図１０～図１８は、Ｌｉｎｕｘオペレーティングシステム内における本発明の一実施態様をさらに詳細に示す。本発明は、インターフェースを拡張することにより、直接実行機能を標準オペレーティングシステムのＩ／Ｏコンポーネント構造内に組み込むのであって

、コンポーネント構造全体を交換するのではない。Ｌｉｎｕｘは、Ｉ／Ｏ操作の実行に関連して、以下のコンポーネント層を提供する：

関連する特定ハードウェアを認識せずに、外部記憶装置にアクセスできるようにするデバイスドライバ層と、

外部記憶装置を認識せずに、ロジックファイルのビューを使用して外部記憶装置にアクセスすることが可能なファイルシステム層と、

アプリケーションが仮想メモリ、およびオペレーティングシステムが使用する動的アドレス変換技術を認識せずに、アプリケーションの実行をできるようにするメモリ管理層。

【００３６】

図１０は、多くの現代的なオペレーティングシステムにインプリメントされるデバイスの抽象化を示す。この実施例は、Ｌｉｎｕｘのデバイスドライバを示す。ｍａｋｅ＿ｒｅｑｕｅｓｔ関数は、データをデバイスに読み書きするための要求を提出するために呼び出される。ｒｅｑｕｅｓｔ＿ｑｕｅｕｅおよびｄｏ＿ｒｅｑｕｅｓｔ関数は、Ｌｉｎｕｘ内部の最適化の一部である。デバイスドライバは、「要求をディスパッチする　－＞　要求を処理する　－＞　割り込みハンドラ　－＞　要求をディスパッチする」のループで動作し、デバイスドライバに提出されたすべての作業が実行される。

【００３７】

デバイスドライバ層は、読み取り要求または書き込み要求を提出できるようにする。この層の頻繁なユーザは、データをファイルから転送するか、またはファイルに転送するファイルシステムのｒｅａｄｐａｇｅ（ｓ）／ｗｒｉｔｅｐａｇｅ（ｓ）動作である。データをアドレス指定するには、物理ブロック数が使用される。

【００３８】

本発明は、図１１に示されているデバイスドライバ層のインターフェースに対する拡張を提供する。既存のインターフェースを元の状態に維持しつつ、この拡張は、デバイス上のデータを直接参照するために使用できる機能を提供する。この参照は、要求を提出せずにデバイス上のデータにアクセスし、その完了を待つために使用することができる。この拡張は、任意に、メモリアドレス指定デバイスにアクセスするデバイスドライバによりインプリメントすることが可能である。新しい動作ｄｉｒｅｃｔ＿ａｃｃｅｓｓは、ｍａｋｅ＿ｒｅｑｕｅｓｔに類似する物理ブロック番号を取得するが、どのデータも転送しない。その代わりに、データの参照が返される。この参照は、デバイスが、さらにデバイスドライバ層と対話せずに、再び閉鎖／アンマウントされるまで、何らかのデータをその後いつでも物理ブロックに読み書きするために使用することができる。新しいインターフェースの使用は任意であり、従来のｍａｋｅ＿ｒｅｑｕｅｓｔインターフェースは、生デバイスドライバなどの新しいインターフェースをサポートしていないユーザのために、元の状態を維持する。従来のインターフェースは汎用ファイルシステムをサポートすることが重要であり、汎用ファイルシステムを使用して、ファイルシステムのメタデータを転送する（ｉｎｏｄｅ、ｄｉｒｅｃｔｏｒｙエントリなど）。

【００３９】

図１２は、Ｌｉｎｕｘ内の汎用ファイルシステムが、デバイスドライバのｍａｋｅ＿ｒｅｑｕｅｓｔ関数を使用して、どのようにアドレス空間操作をインプリメントするかを示す。ｒｅａｄ　ｐａｇｅ（ｓ）／ｗｒｉｔｅ　ｐａｇｅ（ｓ）関数は、ｇｅｔ＿ｂｌｏｃｋ関数を使用して［複数のページを処理する場合、繰り返し］、複数の対象ページに関連するデバイス上の物理ブロック数を識別する。図１０に示すｍａｋｅ＿ｒｅｑｕｅｓｔ関数は、データにアクセスするために使用される。Ｌｉｎｕｘでは、ファイルシステムは、一般に、ファイルシステムのライブラリ関数と共にアドレス空間操作を使用して、ｓｙｓ＿ｒｅａｄ（）およびｓｙｓ＿ｗｒｉｔｅ（）などのファイル操作を実行する。これらのアドレス空間操作およびライブラリ関数の使用は任意だが、殆どの汎用ファイルシステムはこれらを使用する。アドレス空間操作、ｒｅａｄ　ｐａｇｅ（）、ｒｅａｄ　ｐａｇｅｓ（）、ｗｒｉｔｅ　ｐａｇｅ（）、ｗｒｉｔｅ　ｐａｇｅｓ（）は、データの１つまたは複数のメモリページをデバイスドライバから読み取るか、またはデバイスドライバに書

き込むために使用される。メモリページをアドレス指定するには、ロジックファイルハンドルおよびオフセットが使用される。このアドレス指定は、ファイルシステムにより物理ブロック番号に変換される。

【００４０】

本発明は、図１３に示すように、あるメモリページの背後で記憶装置の参照を検索できるようにするｇｅｔ＿ｘｉｐ＿ｐａｇｅという名称の関数により、アドレス空間操作インターフェースに対する拡張を提供する。この機能は、ファイルハンドル、およびアドレス指定用オフセットを使用し、ｇｅｔ＿ｂｌｏｃｋ関数を呼び出すことにより、ファイルハンドルおよびアドレス指定用オフセットを物理ブロック番号に変換し、その物理ブロックの背後で、デバイスドライバ層のｄｉｒｅｃｔ＿ａｃｃｅｓｓ関数から記憶装置の参照を検索する（図１１に示す）。この参照は、ファイルシステムまたはデバイスドライバ層とさらに対話せずに、ファイルが切り捨てられるか、またはファイルシステムがアンマウントされるまで、何らかのデータをその後いつでも物理ブロックに読み書きするために使用することができる。新しいインターフェースの使用は、サポートされている場合は必須であり、データの完全性の点で、従来のｒｅａｄ　ｐａｇｅ（ｓ）／ｗｒｉｔｅ　ｐａｇｅ（ｓ）インターフェースまたは新しいｇｅｔ＿ｘｉｐ＿ｐａｇｅインターフェースがファイルに対してサポートされる。

【００４１】

ｒｅａｄ　ｐａｇｅ（ｓ）／ｗｒｉｔｅ　ｐａｇｅ（ｓ）関数の主なユーザは、ファイルシステムのライブラリ関数である。図１４は、ファイルシステムのライブラリ関数が、Ｌｉｎｕｘの汎用ファイルシステムのためのｒｅａｄ＿ｔｙｐｅファイル操作をどのように実行するかを示す。

【００４２】

ｇｅｎｅｒｉｃ＿ｆｉｌｅ＿ｒｅａｄ関数は、ｓｙｓ＿ｒｅａｄ（）システムコールに関連するファイル操作を実行する。

【００４３】

ｇｅｎｅｒｉｃ＿ｆｉｌｅ＿ｒｅａｄｖ関数は、ｓｙｓ＿ｒｅａｄｖ（）システムコールに関連するファイル操作を実行する。

【００４４】

ｇｅｎｅｒｉｃ＿ｆｉｌｅ＿ａｉｏ＿ｒｅａｄ関数は、非同期ＩＯシステムコールに関連する読み取り操作を実行する。

【００４５】

ｇｅｎｅｒｉｃ＿ｆｉｌｅ＿ｓｅｎｄｆｉｌｅファイル操作は、ｓｙｓ＿ｓｅｎｄｆｉｌｅ（）システムコールに関連するファイル操作を実行する。

【００４６】

これらのすべての関数はｇｅｎｅｒｉｃ＿ｍａｐｐｉｎｇ＿ｒｅａｄを呼び出し、ｇｅｎｅｒｉｃ＿ｍａｐｐｉｎｇ＿ｒｅａｄは、ｒｅａｄ　ｐａｇｅ（ｓ）関数を使用して（図１２に示すように）、デバイスから１つまたは複数のページを読み取る。これらの関数の使用は、ファイルシステムの場合は任意だが、汎用ファイルシステムは、独自のファイル操作のインプリメンテーションを行う代わりに、すべての関数を使用する。

【００４７】

図１６は、ファイルシステムのライブラリ関数が、どのようにＬｉｎｕｘの汎用ファイルシステムのためのｗｒｉｔｅ　ｔｙｐｅファイル操作を実行するかを示す。

【００４８】

ｇｅｎｅｒｉｃ＿ｆｉｌｅ＿ｗｒｉｔｅ関数は、ｓｙｓ＿ｗｒｉｔｅ（）システムコールに関連するファイル操作を実行する。

【００４９】

ｇｅｎｅｒｉｃ＿ｆｉｌｅ＿ｗｒｉｔｅｖ関数は、ｓｙｓ＿ｗｒｉｔｅｖ（）システムコールに関連するファイル操作を実行する。

【００５０】

ｇｅｎｅｒｉｃ＿ｆｉｌｅ＿ａｉｏ＿ｗｒｉｔｅ関数は、非同期ＩＯシステムコールに関連する書き込み操作を実行する。

【００５１】

これらのすべての関数は、ｇｅｎｅｒｉｃ＿ｆｉｌｅ＿ｄｉｒｅｃｔ＿ｗｒｉｔｅ（オプションのＯ＿ＤＩＲＥＣＴを使用して、対象ファイルを開く場合）、またはｇｅｎｅｒｉｃ＿ｆｉｌｅ＿ｂｕｆｆｅｒｅｄ＿ｗｒｉｔｅを呼び出す。これらの関数はｗｒｉｔｅｐａｇｅ（ｓ）関数を使用して、１つまたは複数のページをデバイスに書き込む。

【００５２】

本発明は、ライブラリ関数が、ｇｅｔ＿ｘｉｐ＿ｐａｇｅインターフェースがサポートされている場合、それを使用できるようにするために、ライブラリ関数の拡張を提供する。図１５は、ｒｅａｄ　ｔｙｐｅ操作のために、ファイルシステムのライブラリ関数に対して行われる拡張を示す。ｇｅｔ＿ｘｉｐ＿ｐａｇｅアドレス空間操作が存在するかどうかに応じて、操作を実行するために、ｇｅｎｅｒｉｃ＿ｍａｐｐｉｎｇ＿ｒｅａｄ関数、または新しいｄｏ＿ｘｉｐ＿ｍａｐｐｉｎｇ＿ｒｅａｄ関数を使用して、操作が実行される。ｄｏ＿ｘｉｐ＿ｍａｐｐｉｎｇ＿ｒｅａｄ関数は、ｇｅｔ＿ｘｉｐ＿ｐａｇｅアドレス空間操作を使用して、デバイス上の対象データの参照を検索する。データ転送の場合、この参照が直接使用され、Ｉ／Ｏ操作を実行する必要はない。図１７は、ｗｒｉｔｅ　ｔｙｐｅ操作のために、ファイルシステムライブラリ関数に対して行われる拡張を示す。ｇｅｔ＿ｘｉｐ＿ｐａｇｅアドレス空間操作が存在するかどうかに応じて、ｇｅｎｅｒｉｃ＿ｆｉｌｅ＿ｂｕｆｆｅｒｅｄ＿ｗｒｉｔｅ／ｇｅｎｅｒｉｃ＿ｆｉｌｅ＿ｂｕｆｆｅｒｅｄ＿ｗｒｉｔｅ関数を使用するか、または新しいｇｅｎｅｒｉｃ＿ｆｉｌｅ＿ｘｉｐ＿ｗｒｉｔｅ関数を使用して操作を行う。ｇｅｎｅｒｉｃ＿ｆｉｌｅ＿ｘｉｐ＿ｗｒｉｔｅ関数は、ｇｅｔ＿ｘｉｐ＿ｐａｇｅアドレス空間操作を使用して、デバイス上の対象データに対する参照を検索する。データ転送の場合、この参照を直接使用して、Ｉ／Ｏ操作を実行する必要はない。

【００５３】

ｇｅｔ＿ｘｉｐ＿ｐａｇｅアドレス空間操作をインプリメントするすべてのファイルシステムは、ライブラリ関数によりインプリメントされるすべてのファイル操作を使って、直接実行アクセスを実行するために、さらにコードを変更する必要はない。図１５、図１７および図１８は、拡張されたライブラリ関数が、ｇｅｔ＿ｘｉｐ＿ｐａｇｅアドレス空間操作がインプリメントされるかどうかに応じて、それぞれの機能をどのようにインプリメントするかを示す。ｇｅｔ＿ｘｉｐ＿ｐａｇｅがインプリメントされる場合、すべてのライブラリ関数は、ｇｅｔ＿ｘｉｐ＿ｐａｇｅから検索された参照を使用して、記憶装置に対するすべてのデータ転送を直接実行する。図１８は、ファイルメモリマッピングのために、ファイルシステムのライブラリ関数に対して行われる拡張を示す。このアプリケーションは、現在存在しない仮想メモリアドレス空間の一部にアクセスした。Ｌｉｎｕｘの標準のアーキテクチャ依存およびコアメモリ管理機能は、結果として得られるページ障害を処理するために使用される。定期的な処理と違って、ファイルシステムは、対象ページに関連するファイルのために、ｆｉｌｅｍａｐ＿ｘｉｐ＿ｎｏｐａｇｅハンドラをインストールした。このハンドラは、ｇｅｔ＿ｘｉｐ＿ｐａｇｅを使用して、デバイス上の対象データの参照を検索する。この参照はｄｏ＿ｎｏ＿ｐａｇｅ関数に返され、アプリケーションの仮想アドレス空間ページテーブル内にページテーブルエントリを生成し、アプリケーションが、オペレーティングシステムがさらに関与せずに、デバイス上のデータを直接使用できるようにする。ページテーブルのエントリは、ファイルマッピングがプライベートであることが選択された場合（標準機構が適用される）、後に、書き込み機構上のコピーの対象になる。

【００５４】

直接実行の効果は、アプリケーションのバイナリファイルおよび共有ライブラリが、Ｌｉｎｕｘのファイルマッピングの対象になり、その結果、上記のページ障害機構の対象になるため達成される。

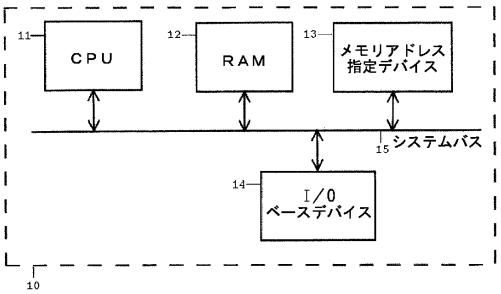
【００５５】
上記の拡張は、Ｌｉｎｕｘオペレーティングシステムの Ｉ／Ｏ関連コンポーネント内で、全体の構造および層隔離を元の状態に維持する。デバイスドライバ層は、物理ブロック番号をデバイスにマップするが、ファイルまたはその他のロジックオブジェクトと連動しない。ファイルシステムは、ロジックファイルと物理ブロック番号のアドレス指定との間でマッピングを実行するが、デバイスの構造と連動しない。一方、データ転送自体は、上下逆転する。デバイスドライバは、新たな拡張を使用する場合、データを転送しない。データは、ファイル操作ライブラリ関数で直接背転送される。この解決方法の効果は、デバイスドライバ層および汎用ファイルシステムに対して、非常に小さくかつ非侵入的な変化のみを含むことである。汎用ファイルシステムは、メモリ消費量の減少および実行の増加など、直接機構の利点を容易に取得することができる。

【図面の簡単な説明】

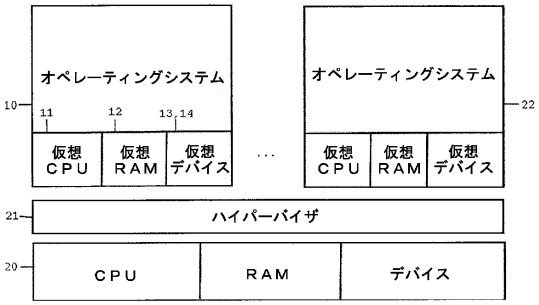
【００５６】
【図１】本発明をインプリメントするために必要なコンピュータシステムのブロック図を示す。
【図２】本発明のいくつかの実施態様のゲストとして、図１に示すコンピュータシステムをホストする仮想マシン環境を示す。
【図３】先行技術のオペレーティングシステムにより提供される仮想メモリおよびデマンドページング機能を示す。
【図４】先行技術のオペレーティングシステムにより提供される直接実行機能を示す。
【図５】直接実行をインプリメントする先行技術のオペレーティングシステムのコンポーネント構造を示す。
【図６】本発明を使用する直接実行をインプリメントするオペレーティングシステムのコンポーネント構造を示す。
【図７】本発明によるＩ／Ｏベースデバイスにアクセスするために使用される図６のオペレーティングシステムのコンポーネントを通る制御フローを示す。
【図８】本発明によるメモリアドレス指定デバイスに対する直接実行アクセスを実行するために使用される図６のオペレーティングシステムのコンポーネントを通る制御フローを示す。
【図９】図７および図８に示される２つの制御フローのどちらを使用するかを選択するために使用される図６のオペレーティングシステムによる決定ロジックを示す。
【図１０】先行技術のＬｉｎｕｘオペレーティングシステムにインプリメントされるデバイスの抽象化を示す。
【図１１】本発明をインプリメントするＬｉｎｕｘオペレーティングシステム内のデバイスドライバ層に行われる拡張を示す。
【図１２】汎用ファイルシステムが、１つまたは複数のページを先行技術のＬｉｎｕｘオペレーティングシステム内のデバイスに読み書きするためのアドレス空間操作をどのように提供するかを示す。
【図１３】本発明をインプリメントするＬｉｎｕｘオペレーティングシステム内のファイルシステムのアドレス空間操作に行われる拡張を示す。
【図１４】ファイルシステムのライブラリ関数が、先行技術のＬｉｎｕｘオペレーティングシステム内において、汎用ファイルシステムに対してｒｅａｄ　ｔｙｐｅファイル操作をどのように実行するかを示す。
【図１５】本発明をインプリメントするＬｉｎｕｘオペレーティングシステム内において、ｒｅａｄ　ｔｙｐｅ操作のためにファイルシステムのライブラリ関数に対して行われる拡張を示す。
【図１６】ファイルシステムのライブラリ関数が、先行技術のＬｉｎｕｘオペレーティングシステム内において、汎用ファイルシステムのためのｗｒｉｔｅ　ｔｙｐｅファイル操作をどのように実行するかを示す。
【図１７】本発明をインプリメントするＬｉｎｕｘオペレーティングシステム内において

、ｗｒｉｔｅ　ｔｙｐｅ操作のためにファイルシステムのライブラリ関数に行われる拡張を示す。
【図１８】本発明をインプリメントするＬｉｎｕｘオペレーティングシステムにおいて、ファイルメモリマッピングのためにファイルシステムのライブラリ関数に行われる拡張を示す。
【符号の説明】
【００５７】
１２　ＲＡＭ
１３　メモリアドレス指定デバイス
１４　Ｉ／Ｏベースデバイス
４４、５３　デバイスドライバ
４９　アプリケーションプログラム
５０　オペレーティングシステム
５１　メモリ／ファイルマネジャ
５２　ファイルシステムドライバ

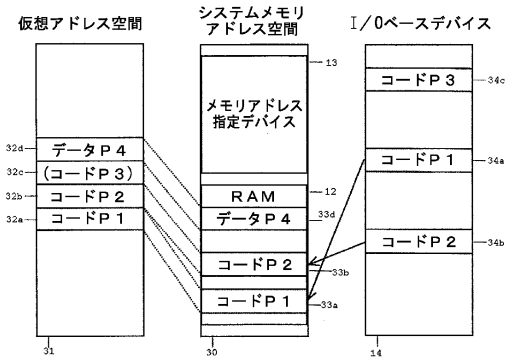
【図１】



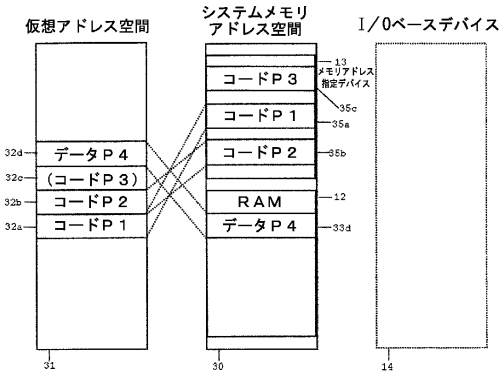
【図２】



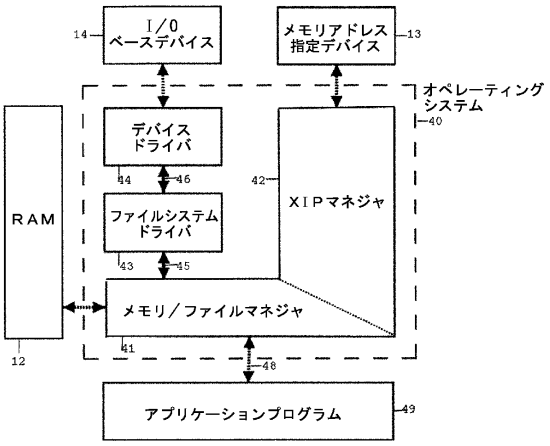
【図３】



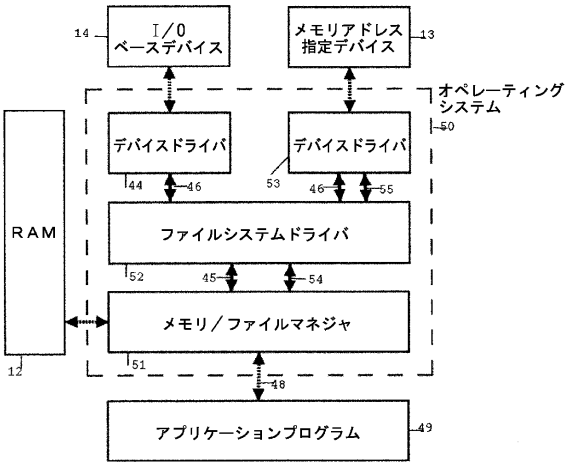
【図４】


仮想アドレス空間
システムメモリ
アドレス空間
I／0ベースデバイス
データＰ４
（コードＰ３）
コードＰ２
コードＰ１
コードＰ３
コードＰ１
コードＰ２
RAM
データＰ４
メモリアドレス
指定デバイス
31
30
14

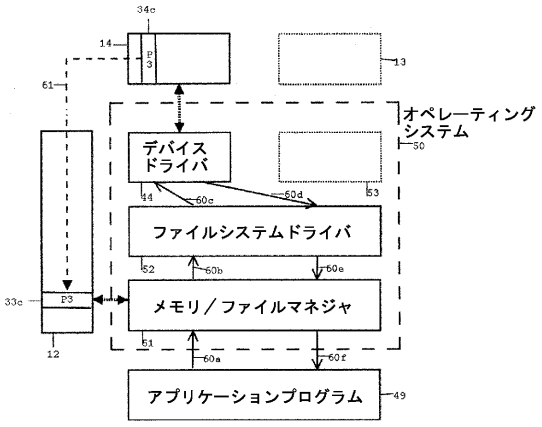
【図５】


I／0
ベースデバイス
メモリアドレス
指定デバイス
オペレーティング
システム
デバイス
ドライバ
ファイルシステム
ドライバ
XIPマネジャ
RAM
メモリ／ファイルマネジャ
アプリケーションプログラム

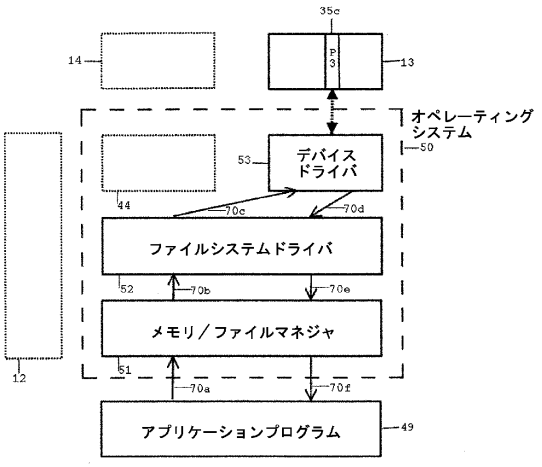
【図６】


I／0
ベースデバイス
メモリアドレス
指定デバイス
オペレーティング
システム
デバイスドライバ
デバイスドライバ
RAM
ファイルシステムドライバ
メモリ／ファイルマネジャ
アプリケーションプログラム

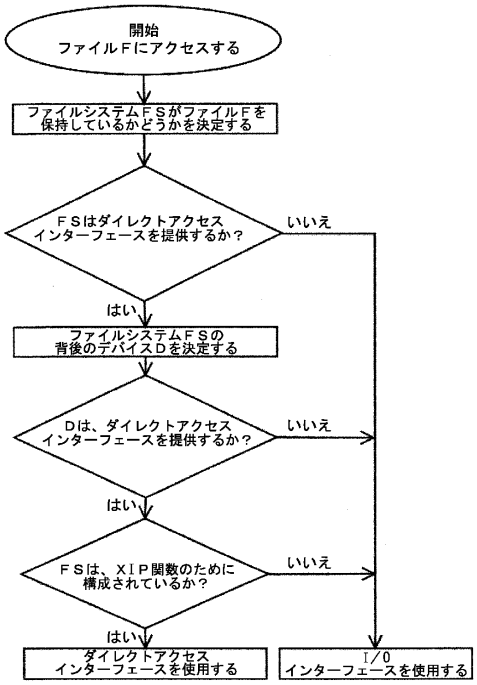
【図７】


オペレーティング
システム
デバイス
ドライバ
ファイルシステムドライバ
メモリ／ファイルマネジャ
アプリケーションプログラム

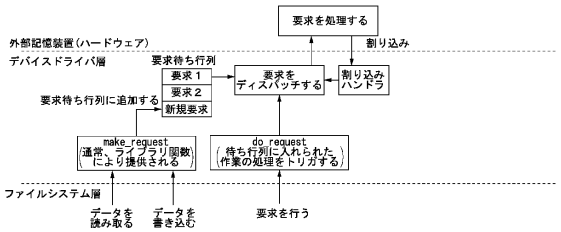
【図８】



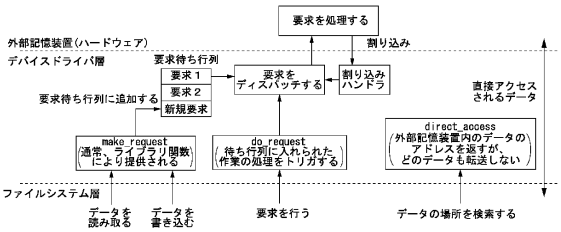
【図９】



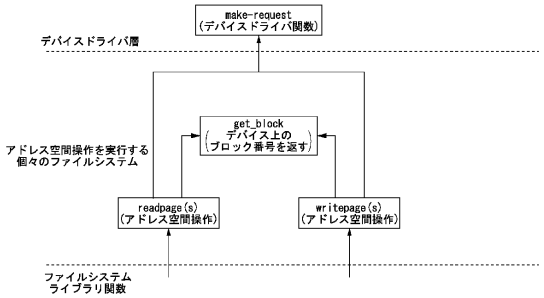
【図１０】



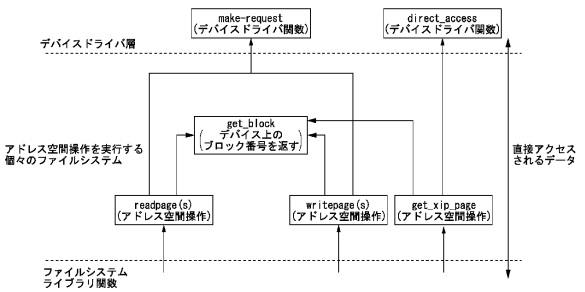
【図１１】



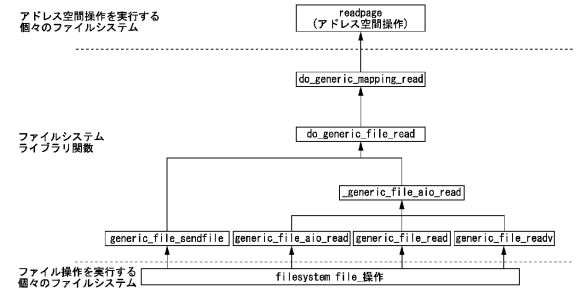
【図１２】



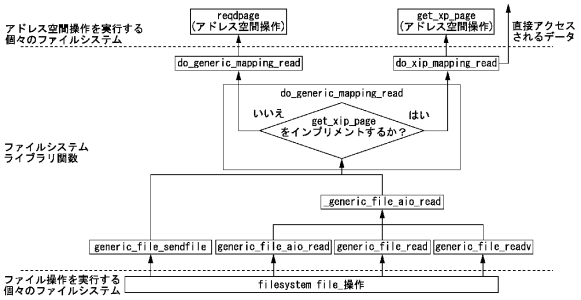
【図１３】



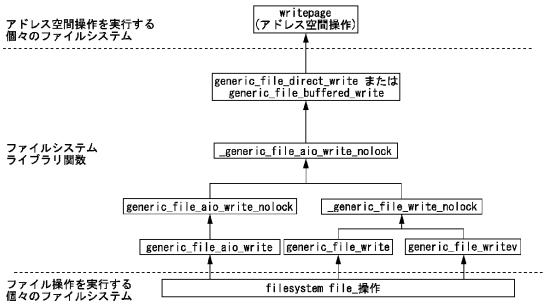
【図１４】



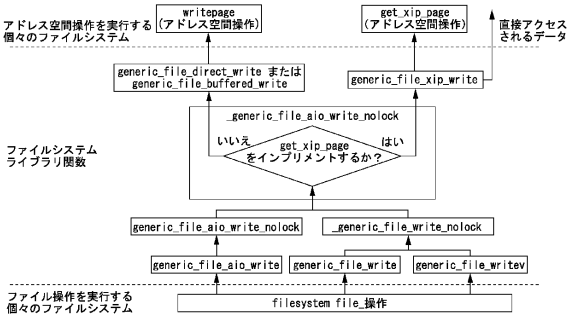
【図１５】



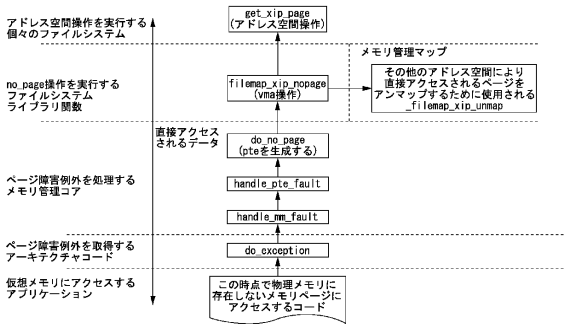
【図１６】



【図１７】



【図１８】



フロントページの続き

(51)Int.Cl. FI
G06F 13/10 340Z

(74)代理人 100086243
弁理士 坂口 博
(72)発明者 カーステン・オット
ドイツ連邦共和国 D－７１０３４、ボエブリンゲン、モーレネル・ウェグ １１７
(72)発明者 ウルリヒ・ウェイガンド
ドイツ連邦共和国 D－７１０８８、ホルツガーリンゲン、リヒテンステイン通り ８

審査官 原 秀人

(56)参考文献 特開平６－１３１２６６（ＪＰ，Ａ）
米国特許出願公開第２００２／００６９３４２（ＵＳ，Ａ１）
前川 守 外２名，分散オペレーティングシステム，日本，共立出版株式会社，１９９３年１０月 5日，第1版，ｐ．１２６～１２９
後藤 大地，ファイルシステムって、なに？ より広範な可用性・堅牢性を実現するための方法，ＴｒａｎｓＴＥＣＨ，日本，株式会社翔泳社，１９９９年１２月 1日，第８巻 第１４号，ｐ．３０～３９
Ｆｒａｎｋ Ｇ． Ｓｏｌｔｉｓ，Ｉｎｓｉｄｅ ｔｈｅ ＡＳ／４００，日本，（株）インフォ・クリエイツ，２０００年 2月１０日，日本語版 第２版，ｐ．２２７～２４１

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ １２／１０
Ｇ０６Ｆ １２／００
Ｇ０６Ｆ ９／４６
Ｇ０６Ｆ １３／１０