(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第6708860号
(P6708860)
(45)発行日　令和2年6月10日(2020.6.10)　(24)登録日　令和2年5月26日(2020.5.26)

(51)Int.Cl.　　F I
G06F 12/14 (2006.01)　　G06F 12/14 510E
G06F 21/62 (2013.01)　　G06F 21/62 318

請求項の数 14　(全 22 頁)

(21)出願番号　特願2019-79548(P2019-79548)
(22)出願日　平成31年4月18日(2019.4.18)
(62)分割の表示　特願2018-532157(P2018-532157)の分割
原出願日　平成27年12月15日(2015.12.15)
(65)公開番号　特開2019-117664(P2019-117664A)
(43)公開日　令和1年7月18日(2019.7.18)
審査請求日　平成31年4月18日(2019.4.18)

(73)特許権者　518210362
エルゼットラブズ　ゲーエムベーハー
スイス国　ツェーハー－８３０４　ヴァリゼレン,　リヒティアーカーデ　１６
(74)代理人　100078282
弁理士　山本　秀策
(74)代理人　100113413
弁理士　森下　夏樹
(74)代理人　100181674
弁理士　飯田　貴敏
(74)代理人　100181641
弁理士　石川　大輔
(74)代理人　230113332
弁護士　山本　健策

最終頁に続く

(54)【発明の名称】仮想アドレス空間レガシーエミュレーションシステムにおける保護キー管理およびプレフィックス変換

(57)【特許請求の範囲】
【請求項１】
保護キーメモリアクセス制御をサポートするように適合されていないプロセッサを有するコンピューティングシステムにおいて保護キーメモリアクセス制御を実装する方法であって、前記プロセッサは、メモリ管理ユニット（ＭＭＵ）を含み、仮想メモリを管理するオペレーティングシステムを実行し、前記プロセッサは、ユーザプロセスおよびタスクを実行するように適合され、前記方法は、
各プロセスに、仮想アドレスメモリの連続範囲と連続アドレス空間内の各ブロックへの記憶域キーとを割り当てることと、
前記プロセスにおける各タスクに記憶域アクセスキーを割り当てることと、
その割り当てられたアクセスキーを伴うタスクに仮想アドレスマッピングを割り当てることによって、特定のアクセスキーを使用してそのタスクの実行を開始することと、
前記タスクによる後続のメモリアクセスに応答して、前記後続のメモリアクセスが前記タスクによって以前に使用されたアクセスキーと異なるアクセスキーを使用する場合、前記タスクが前記異なるアクセスキーを使用する権限を与えられているかどうかを決定し、第２のタスクが権限を与えられている場合、後続の仮想アドレスマッピングを前記タスクおよび前記異なるアクセスキーに割り当てることと、
前記タスクによる記憶域を配分するコマンドまたは解放するコマンドに応答して、制御バイトを設定することであって、前記制御バイトを設定することは、前記記憶域キーデータと、各制御バイトに関連付けられたページが有効であるというインジケータとを設定す

ることを含む、ことと、
任意のアクセスキーを使用する前記タスクのコンピュータ命令の実行時、
前記タスクに関連付けられた前記アドレス空間内に仮想アドレスを生成することと、
第１のアドレスが第１のプロセスに関連付けられた前記アドレスの範囲内にあるかどうかを決定することと、
前記アドレスが前記第１のプロセスに関連付けられた前記アドレスの範囲内にある場合、
セグメンテーション違反を前記ＭＭＵによって生成することと、
前記アクセスキー値が０であること、または前記タスクが前記アクセスキーを使用して前記仮想アドレスにアクセスする権限を与えられていることを例外ハンドラによって検証することと、
前記例外ハンドラを検証すると、前記ＭＭＵにおいて、前記仮想アドレスを備えているデータのページに関連付けられるネイティブ保護設定を変更し、前記仮想アドレスへのアクセスを可能にすることと、
続いて、前記命令を再試行し、前記ＭＭＵによって、前記タスクの前記命令が前記データのページ内の仮想アドレスにアクセスすることを可能にすることと
を含む、方法。
【請求項２】
前記タスクは、前記プロセスの第２のタスクを備えている、請求項１に記載の方法。
【請求項３】
前記第１および第２のタスクに関連付けられたプレフィックスデータは、異なる物理アドレスの中であるが、同じ論理アドレスにおいて記憶される、請求項２に記載の方法。
【請求項４】
前記変更することは、対応するページテーブル項目のビット０の値を設定し、前記物理ページがメモリの中に存在していることを示すことを含む、請求項１に記載の方法。
【請求項５】
ＭＰＲＯＴＥＣＴ（）関数を呼び出し、前記ネイティブ保護設定を変更することをさらに含む、請求項１に記載の方法。
【請求項６】
前記セグメンテーション違反は、前記仮想アドレスに対応する前記ページが物理メモリの中に存在しないこと、または、前記タスクが読み取り専用アクセス許可設定を伴ってメモリの中に存在するページに書き込もうとしていること、または、前記タスクがマッチングアクセス許可設定を伴わずにページを読み込もうとしていることを示すことを含む、請求項１～５のうちのいずれかに記載の方法。
【請求項７】
前記ブロックは、４ｋブロックを備えている、請求項１～６のうちのいずれかに記載の方法。
【請求項８】
コンピューティングシステムであって、前記コンピューティングシステムは、
保護キーメモリアクセス制御をサポートするように適合されていないプロセッサであって、前記プロセッサは、メモリ管理ユニット（ＭＭＵ）を含み、仮想メモリを管理するオペレーティングシステムを実行し、前記プロセッサは、ユーザプロセスおよびタスクを実行するように適合されている、プロセッサと、
命令を記憶している非一過性のメモリと
を備え、
前記命令は、前記プロセッサ上で実行されると、
プロセスに、仮想アドレスメモリの連続範囲と連続アドレス空間内の各ブロックへの記憶域キーとを割り当てることと、
前記プロセスにおける各タスクに記憶域アクセスキーを割り当てることと、
その割り当てられたアクセスキーを伴うタスクに仮想アドレスマッピングを割り当てる

ことによって、特定のアクセスキーを使用してそのタスクの実行を開始することと、
前記タスクによる後続のメモリアクセスに応答して、前記後続のメモリアクセスが前記タスクによって以前に使用されたアクセスキーと異なるアクセスキーを使用する場合、前記タスクが前記異なるアクセスキーを使用する権限を与えられているかどうかを決定し、第２のタスクが権限を与えられている場合、後続の仮想アドレスマッピングを前記タスクおよび前記異なるアクセスキーに割り当てることと、
前記タスクによる記憶域を配分するコマンドまたは解放するコマンドに応答して、制御バイトを設定することであって、前記制御バイトを設定することは、前記記憶域キーデータと、各制御バイトに関連付けられるページが有効であるというインジケータとを設定することを含む、ことと、
任意のアクセスキーを使用する前記タスクのコンピュータ命令の実行時、
前記タスクに関連付けられた前記アドレス空間内に仮想アドレスを生成することと、
第１のアドレスが第１のプロセスに関連付けられた前記アドレスの範囲内にあるかどうかを決定することと、
前記アドレスが前記第１のプロセスに関連付けられた前記アドレスの範囲内にある場合、
セグメンテーション違反を前記ＭＭＵによって生成することと、
前記アクセスキー値が０であること、または前記タスクが前記アクセスキーを使用して前記仮想アドレスにアクセスする権限を与えられていることを例外ハンドラによって検証することと、
前記例外ハンドラを検証すると、前記ＭＭＵにおいて、前記ゲスト仮想アドレスを備えているデータのページに関連付けられたネイティブ保護設定を変更し、前記仮想アドレスへのアクセスを可能にすることと、
続いて、前記命令を再試行し、前記ＭＭＵによって、前記タスクの前記命令が前記データのページ内の仮想アドレスにアクセスする可能にすることと
を前記プロセッサに行わせる、コンピューティングシステム。
【請求項９】
前記タスクは、前記プロセスの第２のタスクを備えている、請求項８に記載のシステム。
【請求項１０】
前記第１および第２のタスクに関連付けられたプレフィックスデータは、異なる物理アドレスの中であるが、同じ論理アドレスにおいて記憶される、請求項９に記載のシステム。
【請求項１１】
前記ネイティブ保護設定を変更することは、対応するページテーブル項目のビット０の値を設定し、前記物理ページがメモリの中に存在していることを示すことを含む、請求項８に記載のシステム。
【請求項１２】
前記ネイティブ保護設定を変更するように動作可能であるＭＰＲＯＴＥＣＴ（）関数を前記システムの前記オペレーティングシステムの中にさらに備えている、請求項８に記載のシステム。
【請求項１３】
前記セグメンテーション違反は、前記仮想アドレスに対応する前記ページが物理メモリの中に存在しないこと、または、前記タスクが読み取り専用アクセス許可設定を伴ってメモリの中に存在するページに書き込もうとしていること、または、前記タスクがマッチングアクセス許可設定を伴わずにページを読み込もうとしていることを示すことを含む、請求項８～１２のうちのいずれかに記載のシステム。
【請求項１４】
前記ブロックは、４ｋブロックを備えている、請求項８～１３のうちのいずれかに記載のシステム。

【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、保護されたメモリアクセスを実装するための技法に関し、具体的には、そのプロセッサおよびオペレーティングシステムが記憶域保護キーをサポートしないシステムにおける記憶域保護キーの実装に関する。
【背景技術】
【０００２】
コンピュータシステムの重要な制限は、所与のコンパイルされたプログラムが、それがコンパイルされたオペレーティングシステムおよび機械命令組の下でしか起動できないことである。これは、既知のレジスタの組と、既知のオペレーティングシステムへの呼び出しを行うことによって入出力動作を実施する能力とを用いて、コンパイルされたプログラムが特定の命令組（すなわち、システムが認識して実行することができる命令）に書き込まれるので、確かである。例えば、図１に図示されるように、コンパイルされたアプリケーション（１０）は、特定のオペレーティングシステム（２０）と、ハードウェアプラットフォーム（３０）とを含む特定のプラットフォーム上で実行するように構成される。そのようなオペレーティングシステム（２０）およびハードウェアプラットフォーム（３０）は、様々な程度の複雑性のものであり得る。しかし、異なるハードウェア命令の組を実装する環境内で、または異なる関数呼び出しを伴うオペレーティングシステムの下で、アプリケーションを起動することを望む場合、典型的には、アプリケーションプログラムは、再コンパイルされなければならない。この制約は、異種環境内で動作するコンピュータプログラムの能力を限定する。
【０００３】
１つのプラットフォームから別のプラットフォームにコンピュータプログラムを拡張するために、それが異なるハードウェアプラットフォーム上で自然に起動するように、クロスコンパイラがプログラムを再コンパイルするために使用され得る。しかしながら、多くの状況で、ソースコードを再コンパイルすることは望ましくない。再コンパイルすることは、エラー、システム性能の変化、またはシステム挙動の変化をもたらし得る。これらの問題を解決することは、元のソースコードの変更を要求し得、それは、コードベースを断片化し、管理複雑性を増加させる。加えて、特定のアプリケーションのためのソースコードは、常に利用可能であるわけではないこともあり、異なるプラットフォーム上で所与のプログラムを動作させる能力にさらなる制限を加える。
【０００４】
この問題に対処するための１つのアプローチは、標的プラットフォーム上で起動するが、異なる（例えば、レガシー）プラットフォームの挙動をエミュレートするエミュレート型システムを使用することである。図２は、そのようなエミュレート型システムを描写する。エミュレート型システム（９０）は、典型的には、標的ハードウェアプラットフォーム（８０）と、好適なデバイスドライバ（７０）と、ネイティブオペレーティングシステム（６０）とを含む。レガシーシステム環境をシミュレーションするために、エミュレータ（５０）が提供され、エミュレータは、１つのアーキテクチャのための命令を標的アーキテクチャのための対応する命令の組に変換する命令ハンドリングルーチンを含む。実行中、エミュレータは、ネイティブオペレーティングシステム（６０）関数を呼び出し、標的ハードウェア（８０）上で起動し、レガシーハードウェアシステムの挙動をシミュレーションする。レガシープラットフォームのゲストオペレーティングシステム（４０）が、エミュレート型システムの中にインストールされる場合、コンパイルされたアプリケーションプログラム（１０）が、実際には異なるプラットフォーム上で起動していることを意識せず、エミュレートされた環境内で実行することができる。レガシーメインフレームコンピュータの例は、ＯＳ／３６０™、Ｓｙｓｔｅｍ／３７０™、Ｓｙｓｔｅｍ／３９０™またはＥＳＡ／３９０™、およびｓｙｓｔｅｍ／Ｚを起動するＩＢＭメインフレーム（Ｉｎｔｅｒｎａｔｉｏｎａｌ Ｂｕｓｉｎｅｓｓ Ｍａｃｈｉｎｅｓ Ｃｏｒｐ．

ＮＹ，ＵＳ）を含む。

【０００５】

種々のハードウェアプラットフォームのためのエミュレータが公知である。例えば、Ｈｅｒｃｕｌｅｓは、ＬＩＮＵＸ（登録商標）（Ｌｉｎｕｘ Ｆｏｕｎｄａｔｉｏｎ，ＣＡ，ＵＳ）、ＷＩＮＤＯＷＳ（登録商標）（Ｍｉｃｒｏｓｏｆｔ Ｃｏｒｐ．ＷＡ，ＵＳ）、ＳＯＬＡＲＩＳ（登録商標）（Ｏｒａｃｌｅ Ａｍｅｒｉｃａ，Ｉｎｃ．，ＣＡ，ＵＳ）、またはＯＳ Ｘ（登録商標）（Ａｐｐｌｅ Ｉｎｃ．，ＣＡ，ＵＳ）オペレーティングシステムを起動するＸ８６マシンが、メインフレームＳｙｓｔｅｍ／３７０、ＥＳＡ／３９０、およびｚ／Ａｒｃｈｉｔｅｃｔｕｒｅハードウェアを模倣することを可能にする、エミュレータである。Ｈｅｒｃｕｌｅｓ等のハードウェアエミュレータを使用して、ＭＶＳ（登録商標）（Ｉｎｔｅｒｎａｔｉｏｎａｌ Ｂｕｓｉｎｅｓｓ Ｍａｃｈｉｎｅｓ Ｃｏｒｐ．ＮＹ，ＵＳ）、ＯＳ／３６０™等のメインフレームオペレーティングシステムがインストールされ得、したがって、異なるプラットフォーム上でメインフレーム環境を提供する。したがって、レガシーオペレーティングシステム下のレガシープラットフォーム上で起動するようにコンパイルされた実行可能ロードモジュールを含むアプリケーションが、ハードウェアエミュレータ上にインストールされたそのオペレーティングシステムのインスタンスにおいて起動し得る。

【０００６】

この従来のエミュレーションアプローチは、ソフトウェアを実行するために要求される変換の複数の層に起因して、低下した性能に悩まされ得る。具体的には、そのようなエミュレーションシステムは、典型的には、エミュレーションにおいて起動するゲストプログラムによってアクセスされる仮想ゲストアドレスを決定するだけではなく、実アドレスおよび絶対システムアドレスをエミュレートするように、それぞれ、動的アドレス変換およびプレフィックス変換もエミュレートしなければならない。加えて、アプリケーションを起動するために、オペレーティングシステムのコピーが、インストールされ、エミュレート型マシン上での使用のために確かめられなければならない。

【０００７】

アドレス空間は、コンピュータ記憶域の中のバイト場所に対応する整数の連続範囲である。実アドレスまたは物理アドレスは、物理メモリの中の場所のアドレスを指す。絶対アドレスは、システムメモリの中の場所のアドレスを指す物理アドレスである。システムは、実アドレスを絶対アドレスに変換するプレフィックス変換を採用する。仮想アドレスは、一方で、アドレス変換機構を用いて、物理アドレスに変換される。動的アドレス変換（ＤＡＴ）は、メモリアドレス指定の分野で公知であるような１つの機構である。

【０００８】

現在の６４ビットプロセッサは、（１６ＥｉＢの理論的最大値を伴う）２５６ＴｉＢ仮想アドレス空間をサポートする。ページングは、物理ＲＡＭの全量が物理的にインストールされることを実際に要求することなく、各プロセスが全仮想アドレス空間を調べることを可能にする技法である。実際に、多くの現在の実装は、１ＴｉＢの物理ＲＡＭ限界と、物理ＲＡＭの４ＰｉＢの理論的限界とを有する。物理ＲＡＭの低減された量に適応することに加えて、ページングは、ページレベル保護の利益を導入する。そのようなシステムは、ユーザレベルプロセスが、それら自身のアドレス空間の中にページングされるデータのみを調べて修正することができるので、ハードウェア分離を提供することができる。システムページは、ユーザプロセスから保護されることもできる。６４ビットｘ８６アーキテクチャの場合、ページレベル保護が、ここでは、メモリ保護機構としてセグメンテーションに優先する。そのようなシステムでは、メモリ管理ユニットまたはＭＭＵは、仮想アドレスを物理アドレスに変換するユニットである。ＭＭＵは、典型的には、２つのテーブル、すなわち、ページングディレクトリおよびページングテーブルの使用を通して、このメモリマッピング変換を行う。

【０００９】

Ｉｎｔｅｌ実装の一例では、両方のテーブルは、１，０２４個の８バイト項目を備えて

いる。ページディレクトリでは、各項目は、ページテーブルを指し示す。ページテーブルでは、各項目は、物理アドレスを指し示し、物理アドレスは、次いで、ディレクトリ内のオフセットおよびテーブル内のオフセットを計算することによって見出される仮想アドレスにマップされる。これは、テーブルシステム全体が線形４ＧＢ仮想メモリマップを表すので、行われることができる。

【００１０】

図３Ａは、ページディレクトリ項目の例を描写する。ビット１２－６３の中で見出されるページテーブル４ＫＢアラインされたアドレスは、その点で４メガバイトを管理するページテーブルの物理アドレスを表す。低次ビットがアクセスビットの値を含み、アドレスの一部ではないので、このアドレスが４Ｋアラインされていることは、重要である。ビット９－１１は、システムプログラマによる使用のために利用可能である。「Ｇｌｏｂａｌ」のＧで標識されたビット８は、無視される。「Ｐａｇｅ　Ｓｉｚｅ」の「Ｓ」で標識されたビット７は、その特定の項目のためのページサイズを記憶する。このビットが設定される場合、ページは、サイズが４ＭＢである。そうでなければ、それらは、サイズが４ＫＢである。「０」で表されるビット６は、将来の使用のために予備にされ、値「０」に設定される。「Ａｃｃｅｓｓｅｄ」のＡで標識されたビット５は、ページが読み取られたか、または書き込まれたかを示すために、使用される。このビットは、ページがアクセスされる度にＭＭＵによって設定される。「Ｄｉｓａｂｌｅｄ」のＤで標識されたビット４は、キャッシュ無効化ビットである。このビットが設定される場合、ページは、キャッシュされないであろう。「Ｗｒｉｔｅ－Ｔｈｒｏｕｇｈ」の「Ｗ」で標識されたビット３は、ライトスルーキャッシングが有効にされているかどうかを示す。「Ｕｓｅｒ／Ｓｕｐｅｒｖｉｓｏｒ」のＵで標識されたビット２は、特権レベルに基づいてページへのアクセスを制御する。このビットが設定されている場合、ページは、全てのプロセスによってアクセスされ得る。このビットが設定されていない場合、ページは、監視プロセスによってのみアクセスされ得る。ページディレクトリ項目の場合、ユーザビットは、ページディレクトリ項目によって参照される全てのページへのアクセスを制御する。したがって、ページをユーザプロセスにアクセス可能にすることが所望される場合、ユーザビットが、関連ページディレクトリ項目の中ならびにページテーブル項目の中で設定されなければならない。「Ｒｅａｄ／Ｗｒｉｔｅ」のＲで標識されたビット１は、読み込み／書き込み許可フラグである。このビットが設定される場合、ページは、読み込み／書き込みページである。そうではなく、ビットが設定されていないとき、ページは、読み取り専用ページである。ＣＲ０の中のＷＰビットが、これらがユーザプロセスのみに適用されることを決定する場合、デフォルト設定でカーネル書き込みアクセス、またはＲビット設定がユーザおよびカーネルプロセスの両方によるアクセスを制御するかどうかを可能にする。「Ｐｒｅｓｅｎｔ」のＰで標識されたビット０は、設定された場合、ページが物理メモリの中に常駐すること、または、設定されていない場合、物理メモリの中に存在しないことを示す。ビットが空である場合、ページフォールトが、参照試行時、生じるであろう。

【００１１】

図３Ｂは、ページテーブル項目の例を描写する。ページテーブル項目は、以下の例外を伴って、ページディレクトリ項目に非常に類似する。すなわち、「Ｇｌｏｂａｌ」のＧで標識されたビット８は、アドレスがキャッシュされ、ＣＲ３がリセットされた場合、バッファを別にして参照がアドレスを更新することを防止する。アドレスは、ＣＲ３設定にかかわらず有効なままであろう。ページディレクトリ項目の場合に確保されたビット６ではなく、ページテーブル項目のビット７が予備にされる。「Ｄｉｒｔｙ」のＤで標識されたビット６は、ページが書き込まれていることを示す。ページテーブル項目の中の「Ｃａｃｈｅ　Ｄｉｓａｂｌｅｄ」のＣで標識されたビット５は、ページディレクトリの中のＤで標識されたビット４と同じ機能を果たす。

【００１２】

レガシーメインフレーム環境では、各プロセスは、仮想アドレス空間を割り当てられる。所与のプロセスは、複数のタスクを開始し得、共通プロセス下で動作するタスクは、同

じ仮想アドレス空間内で動作する。

【００１３】

メインフレームＣＰＵは、典型的には、ブロック０の中、または０－４０９５バイトに対応する記憶場所の中に、それらの状態情報の一部を記憶する。複数のプロセッサが同じ物理メモリをより容易に共有することを可能にするために、そのようなシステムは、多くの場合、０－４０９５の範囲内の実アドレスが各ＣＰＵのために実メモリの中の異なる場所に対応することを可能にするプレフィックス変換として公知の技法を採用するが、残りの実アドレスは、同一であろう。したがって、プレフィックス変換は、プロセッサの実記憶域の中の場所を表す実アドレスを、メインシステム記憶域の中で割り当てられる物理アドレスである絶対アドレスに変換する。これは、各プロセッサが、現在のプログラムステータスワード、古いプログラムステータスワード、および他の状態情報を記憶するためのそれ自身のプレフィックス記憶エリアを有することを可能にする。プレフィックスエリアのサイズは、変動し得る。例えば、いくつかの６４ビットシステムは、プレフィックスエリアを場所０－８１９１に対応するアドレスに割り当てる。

【００１４】

ＭＭＵの重要な機能は、プロセスまたはタスクが、そのプロセスまたはタスクに配分されていないメモリにアクセスすることを防止することである。配分されていないメモリにアクセスする試行は、ハードウェア故障をもたらし、それは、オペレーティングシステムによって阻止され、多くの場合、セグメンテーション違反と呼ばれ、概して、プロセスの終了を引き起こす。メモリの中へのデータの不正記憶に対するさらなる保護として、メインフレームシステムは、メモリへのアクセスを制御する記憶域キーの概念を実装する。メモリまたはページフレームの各連続４Ｋブロックは、関連付けられる記憶域キーを有する。記憶域キーは、システムメモリ内の確保された空間の中のテーブルの中に記憶される。要求された記憶域アクセスキーを有するタスク、または０の記憶域アクセスキーを有するタスクのみ、ブロックへの完全なタスクを与えられる。

【００１５】

記憶域キーは、典型的には、メモリの各４ＫＢブロックに関連付けられる制御バイトを有するテーブルの中に記憶される。Ｓｙｓｔｅｍ／３６０™、Ｓｙｓｔｅｍ／３９０™、またはＳｙｓｔｅｍ／Ｚアーキテクチャ等のメインフレームシステムでは、記憶域キーは、物理メモリアドレスに関連付けられる。より具体的には、メモリの各物理ページに対して、記憶域キーを記憶する制御バイトがあり、メモリの中にある４Ｋバイトブロックと同じくらいの数の記憶域キーがある。メインフレームシステムでは、制御バイトは、典型的には、４ビット記憶域キーと、保護ビットと、それぞれ、変更および参照を記録するために使用される２ビットとを含む、１バイトフィールドの７ビットを含む。図５Ａは、４ビットキー５１０がビット０－３の中に記憶され、保護ビット５２０がビット４の中に記憶され、変更ビット５３０がビット５の中に記憶され、参照ビット５４０がビット６の中に記憶された制御バイト（５００）の７ビットの例を描写する。所与の制御バイトのフェッチビットが０に設定される場合、書き込みアクセスのみが保護され、任意の保護キーとともに動作するタスクが、ブロックを読み取ることを可能にされる。フェッチビットが１に設定される場合、保護は、ブロックへの読み取り（フェッチ）および書き込み（記憶）アクセスの両方に適用される。

【００１６】

４ビットの中の保護キーを符号化するシステムでは、０から１５まで付番された１６個の保護キーがある。所与のタスクに関連付けられる保護キーは、記憶域アクセスキーとも称されるプログラムステータスワード（ＰＳＷ）の中に記憶される。動作時、システムは、アクセスが許可されているかどうかを決定するように、記憶域キーに対して記憶域アクセスキーをチェックし、メモリのブロックに対して制御バイトの中に記憶されたアクセス制御ビットをチェックする。記憶域キーがアクセス制御ビットに合致しない場合、記憶域保護論理が戻り、タスクをインタラプトし、保護例外を開始するであろう。記憶域キー値０は、特別な場合である。タスクが０のアクセスキー値とともに動作するとき、そのアド

レスのためのシステムメモリ内の記憶域キーの値がどのようであっても、アクセスが許可される。典型的には、オペレーティングシステムによる使用のために確保されるメモリエリアのみが、０の記憶域キー値を割り当てられる。

【００１７】

そのようなシステムの制御バイトの中の記憶域キーは、オペレーティングシステムの制御下にあり、オペレーティングシステムは、データのページが物理メモリの中へコピーされるとき、またはプロセスもしくはタスクによってアクセスまたは修正されるとき、各項目の中のビットを記憶して修正する。多くのユーザタスクは、キー番号８のみにアクセスするが、所与のタスクに関連付けられる複数の記憶域キーの使用がサポートされ、例えば、典型的には、キー番号９を使用する、ＣＩＣＳの下で行われる。殆どのシステムプロセスは、キー０の下で動作する。

【００１８】

記憶域キーは、階層的ではないリングシステムと異なり、０の記憶域キーは、常にアクセスを許可する「マスタキー」であり、０ではない記憶域キーは、固有であり、それらの値は、固有であること以外の具体的な意味を有していない。好ましくは、記憶域キーを使用するシステムでは、各メモリアドレスは、単一のキーを割り当てられる。

【００１９】

メインフレーム動作をエミュレートするシステムは、典型的には、メインフレームシステムのものと異なる命令組を有する標的プロセッサ上でそのようにする。そのような標的プロセッサは、記憶域へのキー制御型アクセスのためのハードウェアサポートを提供しない。したがって、ｘ８６アーキテクチャ上のメインフレーム動作をエミュレートするシステムでは、記憶域キーの動作をエミュレートすることが望ましいであろう。同時に、Ｈｅｒｃｕｌｅｓエミュレータ等の従来技術のエミュレーションシステムは、実アドレスへの仮想アドレスのＤＡＴのエミュレーションを実装し、その後、物理アドレスの管理のエミュレーションを実装した。エミュレートされた動的アドレス変換の実装は、記憶域へのキー制御型アクセスのエミュレーションに複雑性を導入し、複数のエミュレートされたテーブルルックアップを行う必要性に起因して、システム性能を限定する。

【００２０】

第１のアーキテクチャ上で起動するようにコンパイルされたプログラムが、異なる標的アーキテクチャ上で起動することを可能にされるために、別の代替は、オブジェクトコードを逆コンパイルし、次いで、標的アーキテクチャ上で起動するようにそれを再コンパイルする、プログラムを変換することである。種々の逆コンパイラが公知であるが、概して、再コンパイルされたプログラムが元のプログラムの挙動を正確に再現するために要求される確実性で、逆コンパイラがコンピュータ命令を識別し、データから分離することが可能ではないので、１つのプラットフォームからのオブジェクトコードの逆コンパイルおよび再コンパイルは困難である。しかしながら、そのコードおよびデータが正しく識別されることができるプログラム（既知のコンパイラによって出力される、または既知のフラグもしくは設定の組を伴って最初にコンパイルされるプログラム等）の組に、逆コンパイルおよび変換が適用される場合、第１のアーキテクチャ上で起動するようにコンパイルされるコードの逆コンパイル、および標的プラットフォームのための実行可能コードを作成するための逆コンパイルされたコードの再コンパイルは、エミュレーションの代替を提示する。一例では、入力として、ＩＢＭメインフレーム上で起動するようにコンパイルされる再配置可能コボルロードモジュールを受信するロードモジュールコンパイラは、入力として受信され、ｘ８６マシン上で起動するように適合される実行可能オブジェクトプログラムは、出力として生成される。

【発明の概要】

【発明が解決しようとする課題】

【００２１】

エミュレートされる動的アドレス変換の追加オーバーヘッドを伴わずに、記憶域へのキー制御型アクセスのためのサポートを提供するエミュレート型システムが説明される。加

えて、インタラプトまたはタスク間のコンテキスト切り替えに応答して、プレフィックスエリアを保存および復元することに関連付けられるオーバーヘッドを伴わずに、複数のプロセッサに関連付けられるであろう複数のタスクをエミュレート型システムが実行することが有益であろう。メインフレーム上で実行可能であるロードモジュールを代替的プラットフォームのための実行可能コードに変換するロードモジュールコンパイラを採用するシステムも説明され、システムは、ロードモジュールコンパイラによって生成される実行可能コードを、記憶域へのキー制御型アクセスを採用するメインフレームシステムまたはエミュレート型システムと相互運用可能にするために、記憶域へのキー制御型アクセスのためのサポートを提供する。

【課題を解決するための手段】

【００２２】

一実施形態では、本発明は、保護キーメモリアクセス制御をサポートするように適合されていないプロセッサを有するコンピューティングシステムを提供し、プロセッサは、メモリ管理ユニット（ＭＭＵ）を含み、仮想メモリを管理するオペレーティングシステムを実行し、プロセッサは、ユーザプロセスおよびタスクを実行するように適合される。システムは、各プロセスに、仮想アドレスメモリの連続範囲および連続アドレス空間内の各４Ｋブロックへの記憶域キーを割り当てることと、プロセスにおいて記憶域アクセスキーを各タスクに割り当てることと、その割り当てられたアクセスキーを伴うタスクに仮想アドレスマッピングを割り当てることによって、特定のアクセスキーを使用してそのタスクの実行を開始することと、仮想アドレスマッピングを割り当てられた記憶域キーのためのタスクに割り当てることとによって、保護キーメモリアクセス制御を実装する方法のために使用され得る。該タスクによる後続のメモリアクセスに応答して、後続のメモリアクセスが該タスクによって以前に使用されたアクセスキーと異なるアクセスキーを使用する場合、該タスクが異なるアクセスキーを使用する権限を与えられているかどうかを決定し、第２のタスクが権限を与えられている場合、後続の仮想アドレスマッピングをタスクおよびキーに割り当てる。タスクによる記憶域を配分するコマンドまたは解放するコマンドに応答して、記憶域キーデータと、各制御バイトに関連付けられるページが有効であるというインジケータとを設定することを含む制御バイトを設定する。アクセスキーを使用する該タスクのコンピュータ命令の実行時、方法は、タスクに関連付けられるアドレス空間内で仮想アドレスを生成することと、第１のアドレスが第１のプロセスに関連付けられるアドレスの範囲内にあるかどうかを決定することと、アドレスが第１のプロセスに関連付けられるアドレスの範囲内にある場合、仮想アドレスに対応するページが物理メモリの中に存在しないこと、またはタスクが読み取り専用アクセス許可設定を伴ってメモリの中に存在するページに書き込もうとしていることを示すセグメンテーション違反をＭＭＵによって生成することと、該アクセスキー値が０であること、または該タスクがアクセスキーを使用して仮想アドレスにアクセスする権限を与えられていることを例外ハンドラによって検証することと、例外ハンドラを検証すると、ＭＭＵの中に該ゲスト仮想アドレスを備えているデータのページに関連付けられるネイティブ保護設定を変更し、仮想アドレスへのアクセスを可能にすることと、続いて、命令を再試行し、ＭＭＵによって、該データのページ内の仮想アドレスにアクセスするタスクの命令を可能にすることとをさらに含む。

【００２３】

互いにおよび上記の実施形態と組み合わせられ得る方法の追加の実施形態では、タスクは、該プロセスの第２のタスクを含み、第１および第２のタスクに関連付けられるプレフィックスデータは、異なる物理アドレスの中であるが、同じ論理アドレスにおいて記憶され、変更することは、対応するページテーブル項目のビット０の値を設定し、物理ページがメモリの中に存在していることを示すことを含み、方法は、ＭＰＲＯＴＥＣＴ（）関数を呼び出し、ネイティブ保護設定を変更することをさらに含み得る。

【００２４】

本発明の実施形態は、上記の方法のうちのいずれかを実装することが可能であり得るコンピューティングシステムをさらに提供する。コンピューティングシステムは、保護キー

メモリアクセス制御をサポートするように適合されていないプロセッサであって、プロセッサは、メモリ管理ユニット（ＭＭＵ）を含み、仮想メモリを管理するオペレーティングシステムを実行し、プロセッサは、ユーザプロセスおよびタスクを実行するように適合される、プロセッサと、命令を記憶する非一過性のメモリとを含み、命令は、プロセッサ上で実行されると、プロセスに、仮想アドレスメモリの連続範囲および連続アドレス空間内の各４Ｋブロックへの記憶域キーを割り当てることと、プロセスにおいて記憶域アクセスキーを各タスクに割り当てることと、その割り当てられたアクセスキーを伴うタスクに仮想アドレスマッピングを割り当てることによって、特定のアクセスキーを使用してそのタスクの実行を開始することと、仮想アドレスマッピングを割り当てられた記憶域キーのためのタスクに割り当てることと、該タスクによる後続のメモリアクセスに応答して、後続のメモリアクセスが該タスクによって以前に使用されたアクセスキーと異なるアクセスキーを使用する場合、該タスクが該異なるアクセスキーを使用する権限を与えられているかどうかを決定することと、第２のタスクが権限を与えられている場合、後続の仮想アドレスマッピングをタスクおよびキーに割り当てることと、タスクによる記憶域を配分するコマンドまたは解放するコマンドに応答して、記憶域キーデータと、各制御バイトに関連付けられるページが有効であるというインジケータとを設定することを含む制御バイトを設定することと、アクセスキーを使用する該タスクのコンピュータ命令の実行時、タスクに関連付けられるアドレス空間内で仮想アドレスを生成することと、第１のアドレスが第１のプロセスに関連付けられるアドレスの範囲内にあるかどうかを決定することと、アドレスが第１のプロセスに関連付けられるアドレスの範囲内にある場合、仮想アドレスに対応するページが物理メモリの中に存在しないこと、またはタスクが読み取り専用アクセス許可設定を伴ってメモリの中に存在するページに書き込もうとしていることを示すセグメンテーション違反をＭＭＵによって生成することと、該アクセスキー値が０であること、または該タスクがアクセスキーを使用して仮想アドレスにアクセスする権限を与えられていることを例外ハンドラによって検証することと、例外ハンドラを検証すると、ＭＭＵの中に該ゲスト仮想アドレスを備えているデータのページに関連付けられるネイティブ保護設定を変更し、仮想アドレスへのアクセスを可能にすることと、続いて、命令を再試行し、ＭＭＵによって、該データのページ内の仮想アドレスにアクセスする該タスクの命令を可能にすることとをプロセッサに行わせる。

【００２５】

相互および上記の実施形態と組み合わせられ得る、システムの追加の実施形態では、タスクは、該プロセスの第２のタスクを含み、第１および第２のタスクに関連付けられるプレフィックスデータは、異なる物理アドレスの中であるが、同じ論理アドレスにおいて記憶され、ネイティブ保護設定を変更することは、対応するページテーブル項目のビット０の値を設定し、物理ページがメモリの中に存在していることを示すことを含み、システムはさらに、ネイティブ保護設定を変更するように動作可能であるＭＰＲＯＴＥＣＴ（）関数を該システムのオペレーティングシステムの中に含む。

本願明細書は、例えば、以下の項目も提供する。

（項目１）

保護キーメモリアクセス制御をサポートするように適合されていないプロセッサを有するコンピューティングシステムにおいて保護キーメモリアクセス制御を実装する方法であって、前記プロセッサは、メモリ管理ユニット（ＭＭＵ）を含み、仮想メモリを管理するオペレーティングシステムを実行し、前記プロセッサは、ユーザプロセスおよびタスクを実行するように適合され、前記方法は、

各プロセスに、仮想アドレスメモリの連続範囲と連続アドレス空間内の各４Ｋブロックへの記憶域キーとを割り当てることと、

前記プロセスにおける各タスクに記憶域アクセスキーを割り当てることと、

その割り当てられたアクセスキーを伴うタスクに仮想アドレスマッピングを割り当てることによって、特定のアクセスキーを使用してそのタスクの実行を開始することと、

仮想アドレスマッピングを前記割り当てられた記憶域キーのための前記タスクに割り当

てることと、
前記タスクによる後続のメモリアクセスに応答して、前記後続のメモリアクセスが前記タスクによって以前に使用されたアクセスキーと異なるアクセスキーを使用する場合、前記タスクが前記異なるアクセスキーを使用する権限を与えられているかどうかを決定し、前記第２のタスクが権限を与えられている場合、後続の仮想アドレスマッピングを前記タスクおよびキーに割り当てることと、
前記タスクによる記憶域を配分するコマンドまたは解放するコマンドに応答して、制御バイトを設定することであって、前記制御バイトを設定することは、前記記憶域キーデータと、各制御バイトに関連付けられたページが有効であるというインジケータとを設定することを含む、ことと、
前記アクセスキーを使用する前記タスクのコンピュータ命令の実行時、
前記タスクに関連付けられた前記アドレス空間内に仮想アドレスを生成することと、
前記第１のアドレスが前記第１のプロセスに関連付けられた前記アドレスの範囲内にあるかどうかを決定することと、
前記アドレスが前記第１のプロセスに関連付けられた前記アドレスの範囲内にある場合、
前記仮想アドレスに対応する前記ページが物理メモリの中に存在しないこと、または、前記タスクが読み取り専用アクセス許可設定を伴ってメモリの中に存在するページに書き込もうとしていることを示すセグメンテーション違反を前記ＭＭＵによって生成することと、
前記アクセスキー値が０であること、または前記タスクが前記アクセスキーを使用して前記仮想アドレスにアクセスする権限を与えられていることを例外ハンドラによって検証することと、
前記例外ハンドラを検証すると、前記ＭＭＵにおいて、前記仮想アドレスを備えているデータのページに関連付けられるネイティブ保護設定を変更し、前記仮想アドレスへのアクセスを可能にすることと、
続いて、前記命令を再試行し、前記ＭＭＵによって、前記タスクの前記命令が前記データのページ内の仮想アドレスにアクセスすることを可能にすることと
を含む、方法。
（項目２）
前記タスクは、前記プロセスの第２のタスクを備えている、項目１に記載の方法。
（項目３）
前記第１および第２のタスクに関連付けられたプレフィックスデータは、異なる物理アドレスの中であるが、同じ論理アドレスにおいて記憶される、項目２に記載の方法。
（項目４）
前記変更することは、対応するページテーブル項目のビット０の値を設定し、前記物理ページがメモリの中に存在していることを示すことを含む、項目１に記載の方法。
（項目５）
ＭＰＲＯＴＥＣＴ（）関数を呼び出し、前記ネイティブ保護設定を変更することをさらに含む、項目１に記載の方法。
（項目６）
コンピューティングシステムであって、前記コンピューティングシステムは、
保護キーメモリアクセス制御をサポートするように適合されていないプロセッサであって、前記プロセッサは、メモリ管理ユニット（ＭＭＵ）を含み、仮想メモリを管理するオペレーティングシステムを実行し、前記プロセッサは、ユーザプロセスおよびタスクを実行するように適合されている、プロセッサと、
命令を記憶している非一過性のメモリと
を備え、
前記命令は、前記プロセッサ上で実行されると、
プロセスに、仮想アドレスメモリの連続範囲と連続アドレス空間内の各４Ｋブロックへ

の記憶域キーとを割り当てることと、
前記プロセスにおける各タスクに記憶域アクセスキーを割り当てることと、
その割り当てられたアクセスキーを伴うタスクに仮想アドレスマッピングを割り当てることによって、特定のアクセスキーを使用してそのタスクの実行を開始することと、
仮想アドレスマッピングを前記割り当てられた記憶域キーのための前記タスクに割り当てることと、
前記タスクによる後続のメモリアクセスに応答して、前記後続のメモリアクセスが前記タスクによって以前に使用されたアクセスキーと異なるアクセスキーを使用する場合、前記タスクが前記異なるアクセスキーを使用する権限を与えられているかどうかを決定することと、前記第２のタスクが権限を与えられている場合、後続の仮想アドレスマッピングを前記タスクおよびキーに割り当てることと、
前記タスクによる記憶域を配分するコマンドまたは解放するコマンドに応答して、制御バイトを設定することであって、前記制御バイトを設定することは、前記記憶域キーデータと、各制御バイトに関連付けられるページが有効であるというインジケータとを設定することを含む、ことと、
前記アクセスキーを使用する前記タスクのコンピュータ命令の実行時、
前記タスクに関連付けられた前記アドレス空間内に仮想アドレスを生成することと、
前記第１のアドレスが前記第１のプロセスに関連付けられた前記アドレスの範囲内にあるかどうかを決定することと、
前記アドレスが前記第１のプロセスに関連付けられた前記アドレスの範囲内にある場合、
前記仮想アドレスに対応する前記ページが物理メモリの中に存在しないこと、または、前記タスクが読み取り専用アクセス許可設定を伴ってメモリの中に存在するページに書き込もうとしていることを示すセグメンテーション違反を前記ＭＭＵによって生成することと、
前記アクセスキー値が０であること、または前記タスクが前記アクセスキーを使用して前記仮想アドレスにアクセスする権限を与えられていることを例外ハンドラによって検証することと、
前記例外ハンドラを検証すると、前記ＭＭＵにおいて、ゲスト仮想アドレスを備えているデータのページに関連付けられたネイティブ保護設定を変更し、前記仮想アドレスへのアクセスを可能にすることと、
続いて、前記命令を再試行し、前記ＭＭＵによって、前記タスクの前記命令が前記データのページ内の仮想アドレスにアクセスする可能にすることと
を前記プロセッサに行わせる、コンピューティングシステム。
（項目７）
前記タスクは、前記プロセスの第２のタスクを備えている、項目６に記載のシステム。
（項目８）
前記第１および第２のタスクに関連付けられたプレフィックスデータは、異なる物理アドレスの中であるが、同じ論理アドレスにおいて記憶される、項目６に記載のシステム。
（項目９）
前記ネイティブ保護設定を変更することは、対応するページテーブル項目のビット０の値を設定し、前記物理ページがメモリの中に存在していることを示すことを含む、項目６に記載のシステム。
（項目１０）
前記ネイティブ保護設定を変更するように動作可能であるＭＰＲＯＴＥＣＴ（）関数を前記システムの前記オペレーティングシステムの中にさらに備えている、項目６に記載のシステム。
【００２６】
本開示およびその特徴ならびに利点のさらに完全な理解のために、ここで、付随する図面と併せて解釈される、以下の説明が参照される。

【図面の簡単な説明】
【００２７】
【図１】図１は、従来技術のアプリケーション、オペレーティングシステム、およびハードウェアの概略図である。
【図２】図２は、従来技術のエミュレート型システムの概略図である。
【図３Ａ】図３Ａは、従来技術とともに、または本発明の実施形態とともに使用され得る、ページディレクトリ項目の概略図である。
【図３Ｂ】図３Ｂは、従来技術とともに、または本発明の実施形態とともに使用され得る、ページテーブル項目の概略図である。
【図４Ａ】図４Ａは、本発明のある実施形態による、アプライアンスの概略図である。
【図４Ｂ】図４Ｂは、図４Ａのアプライアンスで使用するためのロードモジュールコンパイルドアプリケーションに再コンパイルされている、モジュール間の関係の概略図である。
【図５Ａ】図５Ａは、従来技術の制御バイトの概略図である。
【図５Ｂ】図５Ｂは、本発明の実施形態とともに使用され得る制御バイトの概略図である。
【図６】図６は、本発明のある実施形態によるシステムの動作のフローチャートである。
【図７】図７は、本発明の実施形態とともに使用され得る４プロセッサシステムの概略図である。
【発明を実施するための形態】
【００２８】
図２に示されるように、エミュレーションシステムは、ゲストアプリケーションの実行をサポートするために、実行すべきゲストオペレーティングシステム（４０）のインストールのための設備を提供し得る。本発明のシステムの一側面では、レガシーアプリケーションエンジンを含むアプライアンスが提供され、レガシーアプリケーションエンジンは、ゲストオペレーティングシステムを採用することなく、ゲストアプリケーションの実行を可能にするように構築される。図４Ａは、ｘ８６コンピュータまたはコンピュートブレード（４１０）が構成され、標的オペレーティングシステム（４２０）を起動する一実施形態を図示する。一例では、標的オペレーティングシステムは、ＬＩＮＵＸ（登録商標）である。ネイティブＡＰＩの組（４４０）が、エミュレートされた命令の迅速な実行を可能にするためにさらに提供される。これらのネイティブＡＰＩは、レガシーアプリケーション環境（４３０）によって呼び出され、レガシーアプリケーション環境は、ゲストシステム（図示せず）の挙動をエミュレートするように適合され、レガシーまたはゲストアプリケーションがアプライアンス上で起動することを可能にする。一実施形態では、レガシーアプリケーション環境は、プロセスのためのコンテナとして動作するレガシー動作エンジンモジュール（４３２）と、メモリ管理および他のハードウェアエミュレーション機能を実装するレガシーハードウェア環境モジュール（４３５）とで構成される。
【００２９】
図４Ｂは、ロードモジュールコンパイラを使用してロードモジュールコンパイルドアプリケーション（４５１）に再コンパイルされているレガシーアプリケーションの場合のモジュール間の関係を図示する。エミュレートされたタスクまたはプロセスと同様に、ロードモジュールコンパイル型アプリケーション（４５１）は、レガシー動作環境モジュール（４３２）によって提供されるコンテナの中で動作する。アプリケーションは、ネイティブＡＰＩ（４４０）と、メモリ管理機能を実装するレガシーハードウェア環境モジュール（４３５）とを呼び出すことができる。図４Ａのレガシーアプリケーションと異なり、ロードモジュールコンパイル型アプリケーション（４５１）は、標的アーキテクチャ上で実行するようにコンパイルされているので、レガシープラットフォームに対応するレガシーハードウェア呼び出しを行わない。
【００３０】
レガシーメインフレームシステム上で起動するプロセスの仮想アドレス空間は、典型的

には、２ＧＢであり、それは、空間をアドレス指定するための３１ビットに対応する。物理メモリに値する全アドレス空間を各プロセスに配分することが高価すぎるので、仮想メモリは、典型的には、物理メモリをより小さい量の物理メモリ、典型的には、４Ｋに分割するが、他の量も使用される。実際に、そのようなメインフレームシステムは、典型的には、仮想アドレス空間のための２ＧＢ限界よりもはるかに小さい全物理メモリで動作する。動作時、そのようなレガシーシステムは、仮想メモリアドレスを、物理記憶デバイスにアクセスするために使用される物理アドレスに変換するために、ＤＡＴを行う。仮想メモリはまた、保護されたメモリ空間の共有、コンピュータメモリ階層の自動管理も可能にし、プログラムのロードおよび実行を促進する。

【００３１】

エミュレータは、概して、１つのシステムが別のシステムの仕様に従って挙動することを可能にするハードウェアまたはソフトウェアを指す。例えば、エミュレータは、いわゆるゲストシステム上で起動するように設計されたソフトウェアが、異なる設計またはアーキテクチャのハードウェアを組み込み得るホストシステム上で起動することを可能にするであろう。メインフレームゲストシステム上で起動するように設計されたソフトウェアが、代わりに異なるコンピュータシステム上で動作することを可能にするエミュレーションシステムが公知である。そのようなエミュレータは、典型的には、エミュレーションシステムが標的アーキテクチャ上でゲストシステムのハードウェア特徴を複製するために、下層メインフレームシステムのハードウェアをエミュレートする。メモリ管理の場合、そのようなハードウェアエミュレータは、典型的には、物理メモリをエミュレートし、したがって、動的アドレス変換挙動をエミュレートするであろう。

【００３２】

一実施形態によると、エミュレーションシステムは、下層物理記憶域をエミュレートすることなく、メインフレームの仮想アドレス空間に関連付けられる２ＧＢ仮想アドレス空間をエミュレートする。エミュレーションシステムは、そのハードウェアおよびオペレーティングシステムがより広いアドレス空間をサポートするネイティブマシン上で実装され得る。好ましい実施形態では、６４ビットのワードサイズおよびメモリアドレス幅を伴う６４ビットプロセッサ、ならびに６４ビット仮想メモリアドレス指定を使用する６４ビットオペレーティングシステムが採用される。当業者は、エミュレーションシステムが、他のワードサイズ、アドレスバス幅を有するプロセッサを伴って、または仮想アドレスのための他の数のビットを採用するオペレーティングシステムを伴って、実装され得ることを認識するであろう。

【００３３】

一実施形態によると、エミュレートされるシステムにおける各タスクまたはプロセスは、２ＧＢアドレス空間への異なるマッピングの組を割り当てられる。好ましい実施形態では、各タスクは、タスクまたはプロセスに関連付けられる各記憶域キーのために異なるマッピングを割り当てられる。好ましくは、記憶域がタスクに配分されるとき、記憶域配分ルーチンは、特定のページが所与のキーによってアクセスされ得ることを示す、制御ブロックの中のビットを変更する。一例では、制御バイトの第８のビットが、将来の使用のために予備にされる。別の実装では、ビット８は、それがｍａｌｌｏｃまたはｇｅｔｍａｉｎ等のメモリ配分ルーチンを通して取得されているので、ページが現在有効であるかどうかを示すために使用され得る。本実施形態では、記憶域キーを保持する制御バイトの第８のビットは、有効／無効ビットを含み、有効／無効ビットは、仮想アドレス空間の特定のページが配分されていることを示すために記憶域配分ルーチンによって設定されることができる。ページが配分解除される場合、ルーチンは、同様に、ページがもはや配分されていないことを示すために有効／無効ビットを設定するであろう。

【００３４】

そのようなシステムでは、各タスクまたはプロセスは、１～１６個の異なるアドレスマッピングを割り当てられるであろう。一実施形態では、エミュレーションシステムは、ｘ８６プロセッサ上のＬＩＮＵＸ（登録商標）オペレーティングシステムの下で起動し、Ｓ

３９０メインフレームシステムにおける仮想記憶域へのアクセスを制御するための記憶域キーの動作をエミュレートする。ＬＩＮＵＸ（登録商標）関数ＭＰＲＯＴＥＣＴ（）は、呼び出しプロセスのメモリページのためのＭＭＵステータスにおける保護を変更する。呼び出しプロセスが保護に違反する様式でメモリにアクセスしようとする場合、カーネルがプロセスのためのＳＩＧＳＥＧＶ信号を生成する。

【００３５】

一例では、記憶域保護ビットは、特定のメモリアドレスが存在していることを示す有効／無効ビットと一緒に制御バイトの中に記憶されている。図５Ｂは、４ビットキー５５５がビット０－３の中に記憶され、保護ビット５６０がビット４の中に記憶され、変更ビット５６５がビット５の中に記憶され、参照ビット５７０がビット６の中に記憶され、有効／無効ビット５７５がビット７の中に記憶された例示的制御バイト（５５０）の８つのビットの例を描写する。単一バイトの使用は、制御情報が異なるサイズの制御ワードの中または他の記憶場所で非連続ビットの中に記憶され得るので、例示的であると理解されるべきである。

【００３６】

システムの動作の説明が、図６を参照して行われる。一実施形態によると、データの全てのページのためのＭＭＵの中の物理ページ設定（図３Ｂ）が、最初に、アクセスを禁止するように設定される。例えば、対応するページテーブル項目のための存在ビット（ビット０）は、ページがメモリの中に存在しないことを示す「０」に設定されるであろう。動作時、エミュレーションにおいて動作するタスクまたはプロセスが仮想レガシーメモリアドレスへの命令を実行するとき、システムは、最初に、６１０で示されるように、所与の記憶域キーの下で動作するタスクに関連付けられる特定の仮想アドレスを決定する。システムは、次いで、アドレスがタスクに割り当てられたアドレスの範囲内にあるかどうかをチェックする。アドレスがプロセスに割り当てられたアドレスの範囲内にない場合（６１５）、アクセスが拒否される（６８０）。アドレスがプロセスに割り当てられた範囲内にある場合、システムは、ページがすでにアクセスされているかどうかを決定する６２０。第１のアクセスの場合、ページは、以前にアクセスされておらず、物理ページのアクセス制御ビットは、０であり、ＭＭＵは、アクセスを禁止し、カーネルは、ＳＩＧＳＥＧＶ信号を生成する。以下に説明される例示的実装は、ページが存在しないことを検出することに応答して、物理ページのアクセス制御ビットを設定し、アクセスビットが変化したことをＭＭＵにシグナリングするために、ＬＩＮＵＸ（登録商標）ＭＰＲＯＴＥＣＴ（）動作を使用するが、異なるシステムでは、セグメンテーション違反を検出するための異なる信号、および非存在から存在にページの状態を変更するための異なるルーチンが、本発明に従って使用され得る。ページが以前にアクセスされた場合において、読み取り／書き込みアクセス許可が検証される。読み取りアクセスのみが設定されているページに対して書き込みアクセスが求められている場合、制御は、示されるようにインタラプトハンドラに戻る（６２５）。読み取り／書き込みアクセス許可が合致する場合（６２５）、アクセスが許可される（６２７）。

【００３７】

本発明の一実施形態では、その上でエミュレータが起動するＬＩＮＵＸ（登録商標）システムのインタラプトハンドラは、キー検証ルーチンを含むように修正され、キー検証ルーチンが呼び出される６２０。キー検証ルーチンは、現在のタスクに関連付けられる記憶域アクセスキーを、記憶域キーテーブルの制御バイト５５０の中の記憶域キー５５５と比較し、キーが等しいかどうかを調べる６４０。一実施形態では、現在のプロセスに関連付けられる保護キーは、レジスタ、キー、制御ブロック、プレフィックス情報、および他のコンテキスト情報を含むデータ構造の中でレガシー動作環境（４３２）によって維持される。キー検証ルーチンが、アクセスキーが記憶域キーと同一であることを見出す場合、アクセスが許可されるべきであり、ルーチンが記憶域論理アドレスに関連付けられるネイティブ保護ステータスを変更する６５０。ＬＩＮＵＸ（登録商標）オペレーティングシステムの下の一例では、ＭＰＲＯＴＥＣＴ（）動作が、ＰＲＯＴ＿ＮＯＮＥからＰＲＯＴ＿Ｒ

ＥＡＤまたはＰＲＯＴ＿ＷＲＩＴＥに保護ステータスを設定するために使用される。この例におけるＭＰＲＯＴＥＣＴ（）動作は、対応するページテーブル項目の存在ビット（ビット０）を「１」に設定し、読み取り／書き込みビット（ビット１）を０またはなしに設定するであろう。一実施形態では、エミュレータは、３状態テーブルを実装し、設定をＰＲＯＴ＿ＷＲＩＴＥに変更するが、追加の保護状態がサポートされ得る。対応するページテーブル項目のための存在ビットも、ページが現在存在していることを示すように、「０」から「１」に変更される。現時点で、エミュレーションシステムは、最初にセグメンテーション違反を引き起こしたプロセッサ命令６５０の実行を再試行する。

【００３８】

キー検証ルーチンが、キーが合致しないことを決定する場合、アクセスキーがキー０であるかどうかを調べるためにチェックする６７０。キー０が、典型的には、システム動作に使用されるので、アクセスキーが０である場合、エミュレータは、ステップ６５０に進み、システムアクセスを可能にするようにネイティブ保護設定を変更する。キーが合致せず、キーがキー０以外である場合、システムは、アクセスを拒否し６８０、エミュレーションシステムは、命令を実行せず、エミュレーションシステムは、記憶域保護例外をエミュレートする。

【００３９】

エミュレータ上で起動する同じタスクまたはプロセスによる、同じ仮想アドレスへの後続のアクセスは、加速された様式で動作する。タスクおよびキーに関連付けられた仮想アドレスが決定されると、システムは、セグメンテーション違反およびＳＩＧＳＥＧＶ信号がないので、ページが以前にアクセスされていること６２０を調べる。好ましくは、ＭＭＵは、ページがすでにメモリの中に存在するかどうかを調べるように、要求されたページに対応するページテーブル項目の保護ビット（ビット０）の状態をチェックする。ページが存在する場合、これは、キー検証が、この特定の記憶域キーを使用してこのアドレスにアクセスするこのタスクのために以前に行われたことを意味する。したがって、保護された記憶域キーの下のアクセスは、許可設定が以前に検証されているので、許可される。ページがメモリの中に存在したままである間、キー検証が繰り返し行われる必要がないので、存在するページへの反復アクセスは、低減したオーバーヘッドを経験する。このようにエミュレーションにおいて動作するタスクまたはプロセスは、そうでなければ各アクセスのために行われるであろう例外処理、コンテキスト切り替え、およびテーブルルックアップの排除に起因して、著しい性能改良を経験するであろう。

【００４０】

同じタスクまたはプロセスが、続いて、仮想メインフレーム記憶域の中の同じ場所にアクセスする命令を実行するが、異なる記憶域アクセスキーを使用してそうする場合、システムは、各タスクおよび記憶域アクセスキーに関連付けられた異なるアドレスマッピングがあるので、異なる仮想アドレス６１０を決定するであろう。この条件下で、異なるアクセスキーを使用する仮想記憶域へのアクセスが、キーを使用する第１のそのようなアクセスである場合、エミュレータは、再びセグメンテーション違反を経験し、キー検証ルーチンを呼び出し、アクセスキーが記憶域キーに合致する場合６４０、ＭＰＲＯＴＥＣＴ（）動作６５０を呼び出し、ページの保護ステータスを変更する。アクセスが異なる記憶域キーの下で許可されない場合、命令は、エミュレーションにおいて実行されず、エミュレータは、適切な例外をエミュレートして記録するであろう。

【００４１】

物理アドレスではなく仮想アドレスをエミュレートするエミュレータにおける別個の仮想アドレスの本発明の使用の別の目的は、複数のタスクまたはプロセスの加速されたエミュレーションである。上で議論されるように、レガシーメインフレームシステムは、プレフィックス変換として公知の技法を実装し、プレフィックス変換は、エミュレートされるシステムにおける各プロセッサが、０－４０９５バイトの範囲内の同じ物理アドレスを使用してメモリの異なる物理ブロックにアクセスすることを可能にする。４プロセッサシステムの例が、図７に図示されている。この例では、０のプレフィックス設定を有して示さ

れるＣＰＵ０のための実際の物理アドレス０－４０９５が、絶対物理アドレス０－４０９５にマップされる。ＣＰＵ１の場合、プレフィックス設定は、１として示され、実際の物理アドレス０－４０９５は、４Ｋのオフセットにおいて絶対システム物理アドレスの中へマップされる。同様に、２および３のプレフィックス設定をそれぞれ有するＣＰＵ２およびＣＰＵ３は、０－４０９５からのそれらの実際の物理アドレスを有し、それらの実際の物理アドレスは、それぞれ、８Ｋおよび１２Ｋのオフセットにおいて絶対システム物理アドレス空間の中へマップされる。この説明図では、各ＣＰＵのためのプレフィックスエリアは、４Ｋバイトであり、プレフィックス設定は、４Ｋのインクリメントを示す整数によって表される。異なるサイズのプレフィックスエリアのための他のインクリメントも使用され得る。代替として、プレフィックス設定は、この例では、４Ｋ、８Ｋ、および１２Ｋ等のオフセットアドレス、またはプレフィックスレジスタの中に記憶されたオフセットアドレスビットの組に記憶され得る。連続アドレスが、例証を容易にするために図７に示されている。しかしながら、プレフィックスアドレスは、アドレス空間内の他の場所にあり得、メモリ内の連続場所を指し示す必要はない。プレフィックス変換は、複数のプロセッサが、互いのデータを上書きすることなく、それらのそれぞれのブロック０場所で状態情報を管理することを可能にするが、インタラプトおよびコンテキスト切り替えの場合、プレフィックスエリアデータを交換して管理する必要性が、そのようなインタラプトまたはコンテキスト切り替えをハンドリングすることに関連付けられるオーバーヘッドを増加させる。

【００４２】

レガシーシステム内のプレフィックスエリアは、ハードウェアにおいて設定され、物理ＣＰＵに関連付けられるので、仮想アドレス指定によって管理されることができない。エミュレートされた環境では、プレフィックスエリアは、実アドレスに固定される必要はない。しかしながら、そのようなレガシーシステムのハードウェア挙動をエミュレートするエミュレーションシステムは、エミュレートされた物理記憶域内のプレフィックスエリアをエミュレートするであろう。そのようなシステムは、プレフィックスエリアのコピー、修正、および以降の復元をエミュレートする必要性に起因して、インタラプトまたはコンテキスト切り替えをエミュレートするときに有意なオーバーヘッドを経験する。例えば、ＣＰＵレジスタおよびプログラムステータスワードの状態だけではなく、エミュレーションにおけるプレフィックス記憶エリアの状態も保存することが、計算リソースを消費し、システム性能に悪影響を及ぼすので、プレフィックス変換の使用は、状態変化に関連付けられるオーバーヘッドを増加させる。そのようなエミュレーションシステムでは、より迅速なインタラプトハンドリングおよびコンテキスト切り替えを可能にするために、プレフィックス記憶エリアをコピーすることに関連付けられるオーバーヘッドを低減または排除することが有益であろう。

【００４３】

上で説明されるように、新規のシステムの実施形態は、タスクまたはプロセスによって使用される各記憶域キーのために別個のマッピングを用いて、固有の仮想アドレスマッピングの組を各タスクまたはプロセスに割り当てる。新規のシステムの一側面では、プレフィックスエリアも、エミュレートされる実ＣＰＵアドレス空間およびエミュレートされる絶対システムアドレス空間内ではなく、エミュレートされた仮想アドレス空間内で管理される。

【００４４】

一例では、プレフィックスエリアのコピーが、所与のタスクの記憶域キーの各々に関連付けられる仮想アドレスマッピングの各々の中に記憶される。このアプローチは、タスクが開始されるときにプレフィックスエリアの複数のコピーを書き込むことを要求するが、実記憶域のブロック０または絶対物理記憶域内の設定された場所ではなくて、仮想記憶域の中にプレフィックスエリアを記憶することが、システム性能を改良する。Ｓｙｓｔｅｍ３９０ＴＭシステムにおいて、またはＳｙｓｔｅｍ３９０ＴＭシステムの物理記憶域をエミュレートするエミュレーションシステムにおいて、インタラプトもしくはコンテキスト

切り替えは、プレフィックスエリアをメモリ内の異なる場所にコピーする必要性、およびタスクが再開するときにプレフィックスエリアを復元する必要性に起因して、増加したオーバーヘッドをもたらす。多数のコンテキスト切り替えを伴うプロダクション環境で、かつ大量の入出力を行うアプリケーションの場合、この余分なデータをコピーすることに関連付けられる性能ペナルティが著しい。

【００４５】

上で解説されるように、レガシーメインフレーム環境内の一般的なプロセスに関連付けられるタスクの組は、共有仮想アドレス空間内で動作する。マルチプロセッサシステムでは、タスクは、同じＣＰＵまたは別個のＣＰＵに割り当てられ得る。本発明の実施形態によると、各タスクは、各記憶域キーのために１つ、それ自身の仮想アドレス空間の組を割り当てられ、それ自身の仮想ＣＰＵを割り当てられる。各仮想ＣＰＵは、別個のＬＩＮＵＸ（登録商標）スレッドとして実装され得る。エミュレートされるシステムが物理アドレスではなく仮想アドレスをエミュレートするので、エミュレーションシステムは、タスクを、レガシーシステムにおいて生じるであろうＣＰＵへのタスクの割当と一致している仮想ＣＰＵに関連付ける必要はない。実際に、エミュレートされるＣＰＵが仮想であるので、エミュレータは、レガシーハードウェアと同程度にリソース配分において制約される必要はない。

【００４６】

一実施形態によると、タスクが開始し、仮想ＣＰＵが開始されるとき、記憶域初期化ルーチンは、記憶域が特定のタスクのために有効であることを示すように、記憶域キー値およびビット設定を含む制御バイトのテーブルの組を初期化する。タスクに関連付けられる各記憶域キーのために１つのマッピングである仮想アドレスマッピングの組が必要とされるので、初期化ルーチンは、各利用可能記憶域キーに対応する項目を作成する。本実施形態では、記憶域がマップされた後、次いで、システムは、仮想アドレス０をプレフィックスページに再マップするために、かつプレフィックスページをアドレス０に戻して再マップするために、ＬＩＮＵＸ（登録商標） ｓｙｓｔｅｍ ｃａｌｌ ｒｅｍａｐ＿ｆｉｌｅ＿ｐａｇｅｓ（）を使用する、ルーチン呼び出し再マッププレフィックスを呼び出す。

【００４７】

記憶域がタスクから配分解除されるとき、ルーチンは、ｍｐｒｏｔｅｃｔ（）動作を使用するアクセスを禁止するように、そのネイティブ保護ステータスをＰＲＯＴ＿ＮＯＮＥに設定し、対応するページフレームがそのタスクのために無効であることを示すために、対応する有効／無効ビットも制御バイトの中で無効に設定する。

【００４８】

上で説明されるエミュレーションシステムは、レガシーメインフレームの仮想アドレスをエミュレートするが、実または絶対アドレスへの動的アドレス変換をエミュレートしないエミュレーションシステムの例である。システムは、各タスクへの異なる仮想アドレス空間の組のマッピング、およびエミュレートされた環境内で記憶域キー保護およびプレフィックス変換のエミュレーションの性能を改良する技法を通して、性能を改良する。ｘ８６標的プラットフォームおよびｘ８６標的オペレーティングシステムを使用して、実施形態が説明される。他の標的プロセッサアーキテクチャまたはオペレーティングシステムも使用され得る。ＬＩＮＵＸ（登録商標）オペレーティングシステムにＳＩＧＳＥＧＶ信号を生成させるＭＭＵの特徴を使用し、ＬＩＮＵＸ（登録商標）ｍｐｒｏｔｅｃｔ（）動作を呼び出す記憶域キー保護を実装する技法が説明される。仮想アドレスに対応するページが、利用可能であること、利用可能でないこと、または、保護されていること、保護されていないことを検出する他の方法が、他のオペレーティングシステムおよびコンピュータプラットフォームを使用して実装され得る。

【００４９】

各タスクへの異なる仮想アドレス空間の組のマッピングの上で説明された技法は、上で説明されるロードモジュールコンパイラを使用してコンパイルされたプログラムの実行に適用されることもできる。そのようなシステムでは、ロードモジュールコンパイラによっ

て出力される実行可能ｘ８６プログラムは、エミュレーションにおいて起動するプログラムが仮想アドレスの範囲を割り当てられるであろうことと同程度に、仮想アドレスの範囲を割り当てられる。一実施形態によると、レガシー動作環境は、エミュレーションプログラムの実行のためのみならず、ロードモジュールコンパイラを使用してｘ８６プラットフォーム上で自然に実行するようにコンパイルされているプログラムの実行のためにもコンテナとして動作する。エミュレーションにおいて起動するタスクまたはプロセスに対して、レガシー動作環境は、ＬＭＣコンパイル型プロセスに関連付けられるレジスタ、キー、制御ブロック、およびプレフィックス情報を含むコンテキスト情報を維持する。

【００５０】

本発明の１つ以上の側面は、例えば、物理コンピュータ読み取り可能な媒体を備えている製造品（例えば、１つ以上のコンピュータプログラム製品）の中に含まれることができる。そのような媒体は、例えば、ソースまたはオブジェクトコード形式であり得るコンピュータプログラム命令、または、本発明の能力を提供するように構成される他のコマンドまたは論理を含む。製造品は、ディスクドライブ、光学ドライブ、半導体メモリ、テープドライブの中に含まれるか、または、別個であり得るかまたはコンピュータもしくはコンピュータシステムにインストールされ得る記憶デバイスの中に含まれることができる。

【００５１】

上で説明される例示的システムは、Ｉｎｔｅｌプロセッサ上で見出されるメモリ管理ユニットを採用し、特定のタスクによって使用するための物理メモリの中へ設置され得るページを制御するためにＬＩＮＵＸ（登録商標）オペレーティングシステムによって使用されるページテーブルの管理において保護キー検証を行うための特殊例外ハンドラを採用した。本明細書に開示される実施形態は、Ｉｎｔｅｌプロセッサを伴うコンピュータを使用したが、ＡＭＤ（ＣＡ，ＵＳ）、ＩＢＭ、Ｍｏｔｏｒｏｌａ（ＩＬ，ＵＳ）、ＡＲＭ（ＵＫ）、または他の供給源によって提供されるプロセッサ等の他のプロセッサが、本明細書に開示される本発明に影響を及ぼすことなく使用され得る。

【００５２】

プログラム命令を記憶および／または実行するためのシステムは、典型的には、システムバスもしくは他のデータチャネル、またはスイッチ、バッファ、ネットワーク、ならびにチャネルの配列を通してメモリに結合される少なくとも１つのプロセッサを含む。メモリは、プログラムの実行中に採用されるローカルメモリであるキャッシュメモリを含み得る。メインメモリは、ランダムアクセスメモリ（ＲＡＭ）または当技術分野で公知である他の動的記憶デバイスであることができる。好ましくは、そのようなシステムは、メモリの持続性を確実にするためにバッテリバックアップを採用する。システムによって使用される読み取り専用メモリは、ＲＯＭ、ＰＲＯＭ、ＥＰＲＯＭ、Ｆｌａｓｈ／ＥＥＰＲＯＭ、または他の公知のメモリ技術であることができる。大容量記憶装置が、データまたはプログラム命令を記憶するために使用されることができる。大容量記憶装置の例は、ディスク、ディスクのアレイ、テープ、ソリッドステートデバイスを含み、直接アタッチ型、ネットワーク化アタッチ型、記憶エリアネットワーク、または当技術分野で公知である他の記憶域構成で構成され得る。リムーバブル記憶媒体は、テープ、ハードドライブ、フロッピー（登録商標）ディスク、ジップドライブ、フラッシュメモリおよびフラッシュメモリドライブ、光学ディスク等を含む。

【００５３】

多くの例が、本明細書で提供される。これらの例は、本発明の精神から逸脱することなく修正され得る。本明細書に説明される例および実施形態は、例として提供され、他の構成要素、ルーチン、またはモジュールも、使用され得る。

【図１】

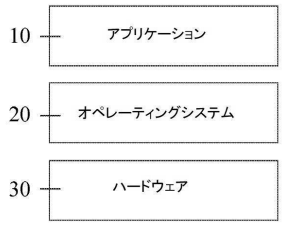

FIG. 1 - 従来技術

【図２】

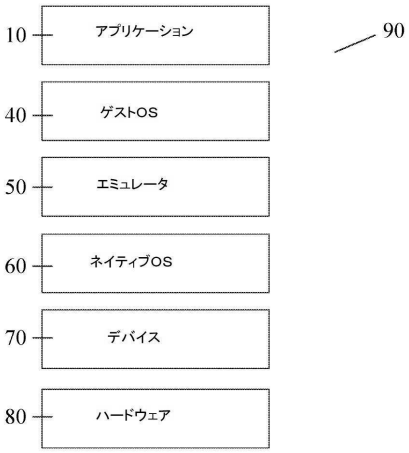

FIG. 2 - 従来技術

【図３Ａ】

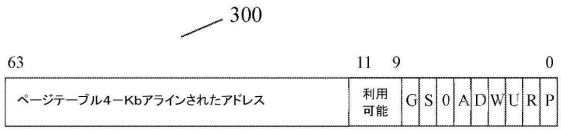

FIG. 3A

【図３Ｂ】

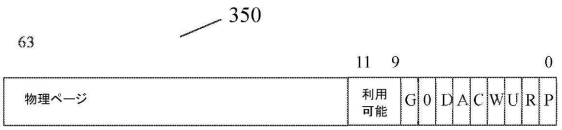

FIG. 3B

【図４Ａ】

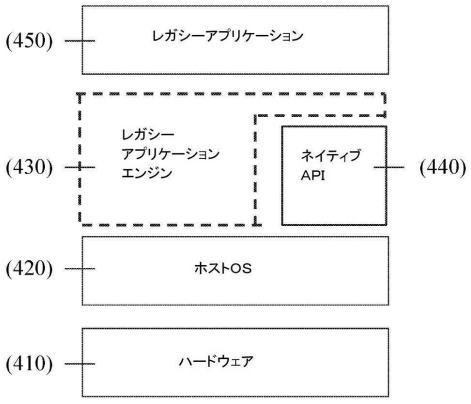

FIG. 4A

【図４Ｂ】

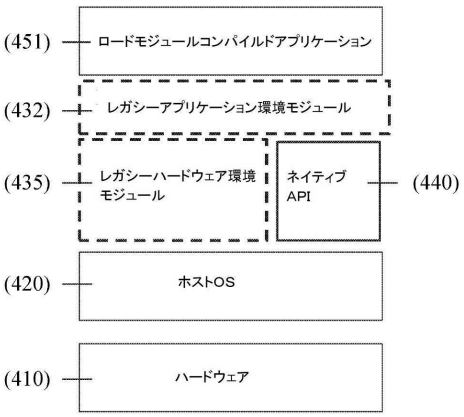

FIG. 4B

【図５Ａ】

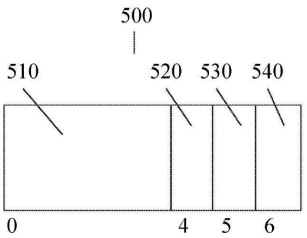

FIG. 5A - 従来技術

【図５Ｂ】

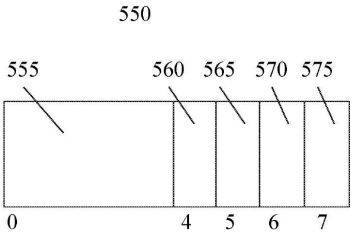

FIG. 5B

【図６】

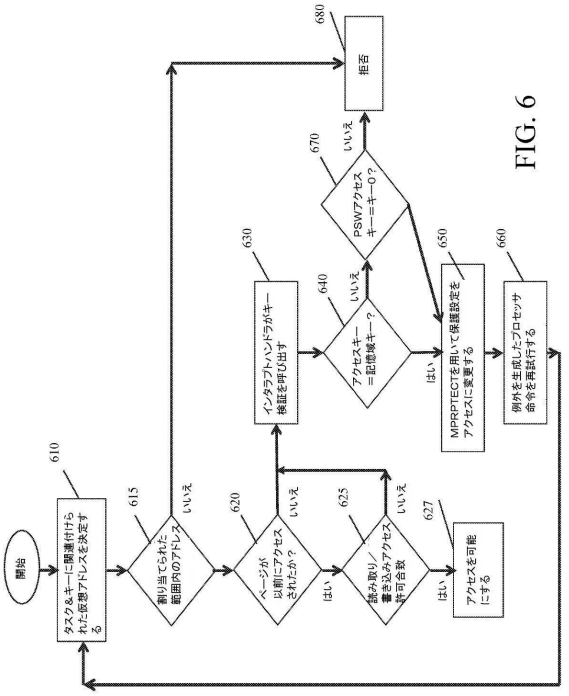

FIG. 6

【図７】

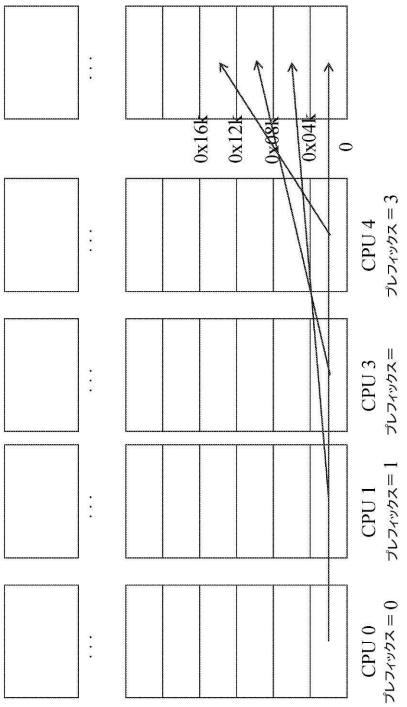

FIG. 7

フロントページの続き

(72)発明者　ジャン　イエーガー
スイス国　８８０３　リュシュリコン,　アルペンシュトラーセ　２１

審査官　平井　誠

(56)参考文献　特開２０１３－２０５９７２（ＪＰ，Ａ）
特表２０１１－５１７７９７（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０６Ｆ　１２／１４
Ｇ０６Ｆ　２１／６２