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

特許第5128484号
(P5128484)

(45)発行日　平成25年1月23日(2013.1.23)　(24)登録日　平成24年11月9日(2012.11.9)

(51)Int.Cl.		F I		
G06F 9/48	(2006.01)	G06F	9/46	457
G06F 9/54	(2006.01)	G06F	9/06	640C
G06F 11/36	(2006.01)	G06F	9/06	620M

請求項の数 9　(全 19 頁)

(21)出願番号	特願2008-537768 (P2008-537768)
(86)(22)出願日	平成18年10月16日 (2006.10.16)
(65)公表番号	特表2009-514098 (P2009-514098A)
(43)公表日	平成21年4月2日 (2009.4.2)
(86)国際出願番号	PCT/US2006/040527
(87)国際公開番号	WO2007/050363
(87)国際公開日	平成19年5月3日 (2007.5.3)
審査請求日	平成21年10月16日 (2009.10.16)
(31)優先権主張番号	60/730,546
(32)優先日	平成17年10月26日 (2005.10.26)
(33)優先権主張国	米国 (US)
(31)優先権主張番号	11/428,162
(32)優先日	平成18年6月30日 (2006.6.30)
(33)優先権主張国	米国 (US)

前置審査

(73)特許権者　500046438
マイクロソフト　コーポレーション
アメリカ合衆国　ワシントン州　９８０５２－６３９９　レッドモンド　ワン　マイクロソフト　ウェイ
(74)代理人　110001243
特許業務法人　谷・阿部特許事務所
(74)復代理人　100115624
弁理士　濱中　淳宏
(74)復代理人　100156971
弁理士　稲　綾子

最終頁に続く

(54)【発明の名称】静的に検証可能なプロセス間通信の分離プロセス

(57)【特許請求の範囲】

【請求項１】

コンピュータにおいて実行される方法であって、

前記コンピュータのオペレーションシステムが、特定のデータセットの所有権を第１のプロセスに関連付けることと、

前記オペレーティングシステムが、前記第１のプロセスによって所有される第１のエンドポイントと、第２のプロセスによって所有される第２のエンドポイントとを含む２つのエンドポイントのみを有するプロセス間通信チャネルを介して、該プロセス間通信チャネルに関連付けられた静的に検証可能なチャネル規約に従って、前記特定のデータセットを前記第１のプロセスから前記第２のプロセスに送信することと

を含み、前記チャネル規約は、前記第１のプロセスが通信すべきプロセスと、前記プロセス間通信チャネルにおける通信方法とを指定し、および前記プロセス間通信チャネルの前記２つのエンドポイントの各々が一度に１つのプロセスのみによって所有されることを規定しており、前記チャネル規約が順守されているかどうかを、前記特定のデータセットの送信の前に検証し、

前記特定のデータセットを送信することによって、前記特定のデータセットの前記所有権が前記第１のプロセスから前記第２のプロセスに転送され、前記所有権を転送した後、前記第１のプロセスによる前記特定のデータセットに対するアクセスは制限されることを特徴とする方法。

【請求項２】

前記特定のデータセットは、メッセージを含み、
前記チャネル規約は、さらに前記プロセス間通信チャネルにおけるメッセージのシーケンスを定義することを特徴とする請求項１に記載の方法。
【請求項３】
前記特定のデータセットは、割り当てられたメモリ内のアドレス可能な位置に格納され、前記割り当てられたメモリの前記アドレス可能な位置に対するアクセスは、一度に１つのプロセスのみに制限されることを特徴とする請求項１に記載の方法。
【請求項４】
請求項１に記載の方法をコンピュータに実行させるためのプログラム。
【請求項５】
コンピュータにおいて実行される方法であって、
前記コンピュータのオペレーティングシステムが、２つのエンドポイントのみを有するプロセス間通信チャネルの特定のエンドポイントの所有権を、第１のプロセスに関連付けることと、
前記オペレーティングシステムが、前記プロセス間通信チャネルを介して、前記プロセス間通信チャネルに関連付けられた静的に検証可能なチャネル規約に従って、前記特定のエンドポイントを前記第１のプロセスから第２のプロセスに送信することと
を含み、前記チャネル規約は、前記第１のプロセスが通信すべきプロセスと、前記プロセス間通信チャネルにおける通信方法とを指定し、および前記プロセス間通信チャネルの前記２つのエンドポイントの各々が一度に１つのプロセスのみによって所有されることを規定しており、前記チャネル規約が順守されているかどうかを、前記特定のエンドポイントの送信の前に検証し、
前記特定のエンドポイントを送信することによって、前記特定のエンドポイントの所有権が、前記第１のプロセスから前記第２のプロセスに転送され、前記所有権を転送した後、前記第１のプロセスによる前記特定のエンドポイントへのアクセスは制限されることを特徴とする方法。
【請求項６】
前記特定のエンドポイントを送信することは、前記特定のエンドポイントが格納されているアドレス可能な位置へのポインタを送信することを含むことを特徴とする請求項５に記載の方法。
【請求項７】
請求項５に記載の方法をコンピュータに実行させるためのプログラム。
【請求項８】
コンピュータにおいて実行される方法であって、
前記コンピュータのオペレーティングシステムが、独立の実行環境を有する複数の分離ソフトウェアプロセスを構築することと、
前記コンピュータの共有交換ヒープ内の複数のメモリブロックが、それぞれ、前記複数の分離ソフトウェアプロセスのうち２つ以上の分離ソフトウェアプロセスによって同時に所有されていないことを確認することと、
前記オペレーティングシステムが、第１のエンドポイントと第２のエンドポイントとからなる２つのエンドポイントのみを有するプロセス間通信チャネルを介して、該プロセス間通信チャネルに関連付けられたチャネル規約に従って、前記複数の分離ソフトウェアプロセスのうち前記第１のエンドポイントを有する送信側の分離ソフトウェアプロセスから、前記第２のエンドポイントを有する受信側の分離ソフトウェアプロセスにメッセージを送信することと
を含み、前記チャネル規約は、前記送信側の分離ソフトウェアプロセスが通信すべき受信側の分離ソフトウェアプロセスと、前記プロセス間通信チャネルにおける通信方法とを指定し、および前記プロセス間通信チャネルの前記２つのエンドポイントの各々が一度に１つのプロセスのみによって所有されることを規定しており、前記チャネル規約が順守されているかどうかを、前記メッセージの送信の前に検証し、

前記メッセージを送信することによって、前記メッセージに対する所有権が、前記送信側の分離ソフトウェアプロセスから前記受信側の分離ソフトウェアプロセスに転送され、前記所有権を転送した後、前記メッセージによって示される前記共有交換ヒープ内の特定のメモリブロックに対する前記送信側の分離ソフトウェアプロセスのアクセスは制限されることを特徴とする方法。

【請求項９】

前記メッセージを送信する前に、前記特定のメモリブロックが前記送信側の分離ソフトウェアプロセス以外の別の分離ソフトウェアプロセスによってアクセスされていないかどうかを判定することと、

前記判定の結果を提示することと

をさらに含むことを特徴とする請求項８に記載の方法。

【発明の詳細な説明】

【技術分野】

【０００１】

本発明は、静的に検証可能なプロセス間通信の分離プロセスに関する。

【背景技術】

【０００２】

一部のオペレーティングシステム（ＯＳ）は、プロセス分離（process isolation）およびプロセス間通信（inter-process communication）を提供する。ＯＳは、あるプロセスが、データまたは別のプロセスの実行命令にアクセスまたは破壊できないように、そのプロセスを分離しようとする。加えて、分離により、他のプロセスからの協力なしに、プロセスを終了させてそのリソースを再利用するための明確な境界が提供される。プロセス間通信により、プロセスは、データを交換し、シグナル通知をすることができる。

【０００３】

しかし、プロセス間の分離と通信の間には自然の引張（natural tension）がある。典型的に、プロセスが互いからより分離されるほど、プロセスが互いに通信することは、より複雑で高価になる可能性がある。逆に、プロセスが互いからあまり分離されないほど、プロセスが互いに通信することはより簡単になる。

【０００４】

例えば、メモリを共有するプロセスは、低い程度の分離であるとみなすことができる。共有メモリプロセスは典型的に、共有メモリを直接読み書きすることによる、明らかに単純な方法で通信することができる。他方、プロセスがメモリを共有することをＯＳが許可しない場合、ＯＳは典型的に、プロセスが情報を交換するための何らかのメカニズムを提供する。

【発明の開示】

【発明が解決しようとする課題】

【０００５】

パフォーマンスの考慮を尊重すると、分離と通信との間のトレードオフは、従来、分離の利点を犠牲にする手法で解決される。特に、従来のＯＳは、プロセス間でメモリの共有を可能にすることが多い。したがって、ＯＳはさらに、同じプロセスの内にコンポーネントを同一場所に配置して、通信を最大化する。このように同一場所に配置されるものの例には、デバイスドライバ、ブラウザ拡張、およびウェブサービスプラグインがある。このようなコンポーネントに対するアクセスの容易さのためにプロセス分離を避けることは、障害分離および明確なリソース管理などの分離の利点の多くを、複雑にし、または損なう可能性がある。あるコンポーネントに障害が起こると、その障害は、共有メモリを、残りのコンポーネントを動作不能な状態にする可能性のある、矛盾または破損した状態のままにすることが多い。

【０００６】

反対に、真に分離したプロセスは当然、分離の利点を享受する。しかし、そのような分離プロセスは従来、プロセス間通信が困難であった。

【課題を解決するための手段】
【０００７】
本明細書では、静的に検証可能な、分離プロセス間のプロセス間通信を提供するオペレーティングシステムの１つまたは複数の実装を説明する。また、本明細書では、静的に検証可能な、プロセス間通信を有する分離プロセスの開発を促進するプログラミングツールの１つまたは複数の実装を説明する。
【０００８】
本要約は、以下の「発明を実施するための最良の形態」でさらに説明される概念の選択を、簡略化した形式で紹介するために提供される。本要約は、特許請求される主題の主要な特徴または本質的な特徴を特定することは意図されておらず、特許請求される主題の範囲を定めるための助けとして使用されることも意図されていない。
【０００９】
同じ参照番号は、図面通じて、同様の要素または特徴を指すのに使用される。
【発明を実施するための最良の形態】
【００１０】
以下の記述は、プロセス間通信のための機能を有する分離プロセスを提供するＯＳについて説明する。説明されるＯＳの分離プロセスの分離は、静的に検証可能である。分離プロセスの実行可能命令を、コンパイル時か実行時、またはその両方で検証することができる。また、静的に検証可能な分離プロセス間のプロセス間通信の開発を促進する１つまたは複数のプログラミング言語ツールを、本明細書で説明する。
【００１１】
静的に検証可能なプロセスとは、実際にそのプロセスの命令を実行することなく自己の実行可能命令を分析することができるソフトウェアプロセスである。その分析により、プロセスが、許可されない振舞いをしないこと、および／または他のプロセスもしくはオペレーティングシステム自体の操作を妨害しないことが保証される。
【００１２】
本明細書で説明される１つまたは複数の実装は、ソフトウェアがより良く構築される可能性が高く、プログラムの振舞いの検証が容易であり、実行時の障害を抑制し軽減することができる環境を作るために、プログラミング言語ツールを使用する。本明細書で説明される１つまたは複数の実装の特徴の一部は、以下を含む（しかし、これらに限定されるものではない）。
・データは、各チャネルが厳密に２つのエンドポイントから成る双方向チャネル上で、交換される。任意の時点で、各チャネルエンドポイントは、単一のスレッドによって所有される（すなわち、単一のプロセスが所有する）。
・バッファおよび他のメモリデータ構造は、そのバッファ内およびメモリデータ構造内に含まれるデータをコピーすることによってではなく、ポインタによって転送される。これらの転送は、メモリブロックの所有権（ownership）を渡す。
・チャネル通信は、メッセージ、メッセージの引数の型、および有効なメッセージインタラクションのシーケンスを、セッション型と類似の有限状態マシンとして記述する、静的に検証可能なチャネル規約（channel contracts）によって規定される。
・チャネルエンドポイントを、メッセージでチャネルを介して送信することができる。したがって、通信ネットワークは、動的に発展することがある。
・チャネル上の送受信は、メモリ割り当てを必要としない。
・送信は、ノンブロッキングかつノンフェーリング（non-failing）である。ノンブロッキングは、送信側は通信の成功を待たないことを意味する。ノンフェーリングは、通信は、常に最終的には成功することを意味する。本実装では、定義によってこれを達成する。すなわち、送信操作は、結果を待つことなく完了する。（しかし、「チャネル」は失敗する可能性があり、チャネル上で受信するときにこれを監視することができる）。
【００１３】
（例示的なオペレーティングシステムおよびプログラミングツール）

図１は、静的に検証可能なプロセス間通信ソフトウェア分離プロセス（ＳＩＰ：Software-Isolated Process）と、そのような静的に検証可能なプロセス間通信ＳＩＰのプログラミングを促進するプログラミングツールの使用とをサポートする例示的な動作シナリオを示す。

【００１４】

図１は、コンピュータ１２０のメモリ１１０内に格納されるおよび／またはコンピュータ１２０のメモリ１１０内で実行中の、オペレーティングシステム１００およびプログラミングツール１６０を示す。コンピュータ１２０は典型的に、（メモリ１１０を含め）様々なプロセッサ読み取り可能な媒体を含む。このような媒体は、コンピュータ１２０によってアクセス可能な任意の利用可能な媒体とすることができ、揮発性および不揮発性媒体、取り外し可能および取り外し不能媒体の両方を含む。

【００１５】

コンピュータ１２０は、ロードモジュールのセット１２４と、（メモリ１１０の一部、またはメモリ１１０とは別個にすることができる）作業メモリ１３０とを格納するコンピュータストレージデバイス１２２（例えば、ハードドライブ、ＲＡＩＤシステムなど）を含む。

【００１６】

作業メモリ１３０は、（作業メモリ１３０内の位置へのポインタなどの）情報を保持するのに使用されるバッファである、交換ヒープ１３２も含む。本明細書では、交換ヒープは、「バッファ」、「共有交換バッファ」、またはこれらと同等の何かと呼ばれることがある。（ブロック１３４によって示されるように）ヒープは、複数のアドレス可能なメモリブロックを含む。交換ヒープ１３２は全体として、複数のプロセスによってアクセス可能であるが、個々のブロックそれぞれは、（そのブロックが使用中のとき）同時に１つのプロセスによって所有される。しかし、メモリブロックの所有権は、別のアクティブなプロセスと交換されることがある。したがって、このように、交換ヒープ１３２は、ＳＩＰに、データを交換するためのメカニズムを提供する。

【００１７】

示されるように、オペレーティングシステム１００は、プロセスコンストラクタ（process constructor）１５０モジュールを備える。プロセスコンストラクタは、オペレーティングシステム１００のカーネルの一部とすることができる。プロセスコンストラクタ１５０は、典型的にコンピュータストレージに格納されたロードモジュールセットとして明示される構成コンポーネント（constituent components）の動的なセットから、コンピュータの作業メモリ内のプロセスを構築する。

【００１８】

図１における例では、プロセスコンストラクタ１５０は、作業メモリ１３０に格納されるプロセス１４０を構築する。ここで示されるように、プロセス１４０は、プロセスの拡張コンポーネントによって編集されるプロセスの構成コンポーネントの明示（manifestation）であるロードモジュール１２４から、構築される。

【００１９】

プロセス１４０は、プロセス１４０のコンテンツ、プロセスの許可された振る舞い、およびプロセスの他の可能性のある特性（properties）を定義する、プロセスマニフェスト（process manifest）１４２を有する。ここで示されるように、プロセスマニフェスト１４２は、プロセスマニフェスト１４２がその構成を記述する（プロセス１４０などの）プロセスと直接関連付けられる。

【００２０】

プログラミングツール１６０は、モジュールおよびデータ構造を備える。これらとともに、プログラミングツール１６０は、静的変数と、定義および制限されたプロセスのプロセス間通信を有する分離プロセスとの作成において、プロセスを開発する者を助ける。プログラミングツール１６０は、コンパイル時、実行時、または両方の時点で施行される強い不変条件（strong invariants）を使用することによって本開発を促進する。強い不変

条件について、「検証」の節で後述する。

【００２１】

プログラミングツール１６０は、プログラマが、時間のかかるテストおよびデバッグをせずに、プロセス間通信のエラーを発見、修正、および／または回避することを助ける、静的分析ツールを提供する。決定論的な静的事前計算分析ツール（deterministic static pre-computation analysis tool）の効率性および適用性を向上させることによって、プログラミングツール１６０は、プログラマまたはプログラマ集団が、プロセス間通信関連のエラーがないプログラムまたはプログラムのセットを作成する可能性をさらに向上させ、そのようなプログラムまたはプログラムのセットを作成するのに必要とされる、テストおよびデバッグする労力をさらに減少させる。

【００２２】

説明されたプログラミングツール（例えば、図１のプログラミングツール１６０）は、（本明細書で説明されるように）開発者のＳＩＰの使用および生成を促進するプログラミング構成およびアプローチを使用する。説明されたプログラミングツールを使用して、ＳＩＰ通信を静的に検証することができる。

【００２３】

（ソフトウェア分離プロセス）

コンピュータ科学の分野、より詳細にはオペレーティングシステムの分野では、「ソフトウェアプロセス」（または、より単純には「プロセス」）という用語は、よく知られている。アプリケーションは、１つまたは複数のプロセスから成ることが多い。ＯＳは、コンピュータ上で実行中の１つまたは複数の別個のプロセスを認識し、実際には、管理および監視することがある。

【００２４】

本明細書では、ＳＩＰ抽象化モデル（SIP abstraction model）を提供および／またはサポートするＯＳモデル内で動作する、１つまたは複数の実装を説明する。ＳＩＰは、プログラムまたはシステムの部分をカプセル化し、情報の隠蔽、障害の分離、および強いインタフェースを提供する。ＳＩＰは、説明される実装にしたがって、ＯＳおよびアプリケーションソフトウェアを通じて使用される。

【００２５】

ＳＩＰでは、カーネルの外部の実行可能コードは、あるＳＩＰ内で実行して、強く型付けされた通信チャネルを通して通信する。ＳＩＰは、データ共有または動的なコードローディングを許容しない、閉じた環境である。ＳＩＰは、従来のＯＳのプロセスとは多くの点で異なる。以下は、ＳＩＰが従来のＯＳのプロセスと異なる点の例である。

・ＳＩＰは、閉じたオブジェクト空間であり、アドレス空間ではない。２つのＳＩＰは、同時にオブジェクトにアクセスすることができない。プロセス間の通信は、排他的なデータ所有権を転送する。

・ＳＩＰは、閉じたコード空間でもある。プロセスは、動的にコードをロードまたは生成しない。

・ＳＩＰは、分離に関してメモリ管理ハードウェアに依拠せず、したがって、複数のＳＩＰは、物理または仮想アドレス空間内に存在することができる。

・ＳＩＰ間の通信は、双方向であり強く型付けされた高次のチャネルを通して行われる。チャネルの型は、チャネルの通信プロトコル、ならびにチャネルが転送する値を記述し、両方の態様が検証される。

・ＳＩＰは、作成が安価であり、ＳＩＰ間の通信が受けるオーバヘッドは低い。この低コストにより、ＳＩＰを、細かい分離および拡張メカニズムとして使用することが現実的になる。

・ＳＩＰは、オペレーティングシステムによって作成および管理され、その結果、終了時にＳＩＰのリソースを、効率的に再利用することができる。

・ＳＩＰは、異なるデータレイアウト、ランタイムシステム、およびガーベジコレクタを有する範囲にいたるまで、独立した実行環境を有する。他の安全な言語システムは、１

つの実行環境をサポートする。

【００２６】

「ソフトウェア分離プロセス」または「ＳＩＰ」という用語は、本明細書では便宜上使用される。この用語は、本概念の範囲を限定することは意図されていない。実際、本概念を、ソフトウェア、ハードウェア、ファームウェア、またはそれらの組合せで実装することができる。

【００２７】

（プロセス間通信）

図２は、ＳＩＰ間の予期せぬインタラクション（unanticipated interaction）なしにプロセス間通信を促進する例示的なプロセス間通信（ＩＰＣ：inter-process communication）アーキテクチャ２００を示す。プロセスの間の通信を提供することに加えて、例示的なＩＰＣアークテクチャ２００は、プロセスとオペレーティングシステムのカーネルとの間の通信を提供することができる。

【００２８】

例示的なＩＰＣアーキテクチャ２００により、ＳＩＰは、双方向であり２つのプロセスの間で振舞いから型付けされた接続（behaviorally typed connection）であるチャネル上で、メッセージを送信することによって、排他的に通信する。メッセージは、送信側から受信プロセスに転送される、（上記の図１の交換ヒープ１３２などの）「交換ヒープ」内の値またはメッセージブロックのタグ付けされたコレクションである。チャネルは、チャネルに沿ったメッセージのフォーマットと有効なメッセージシーケンスとを指定する規約によって型付けされる。

【００２９】

図２に示されるように、例示的なＩＰＣアーキテクチャ２００は、メモリ２１０（例えば、揮発性、不揮発性、取り外し可能、取り外し不能など）で構成されるコンピュータ２０２上に実装される。ＯＳ２１２は、メモリ２１０内に格納され、コンピュータ２０２上で実行されるように示される。

【００３０】

ＯＳ２１２は、カーネル２２０を有する。ＯＳのカーネル２２０は、ＩＰＣファシリテータ（facilitator）２２２を組み込む。ＯＳのカーネル２２０は、１つまたは複数のプロセスを構築することができる。図２は、例えばメモリ２１０内で実行中の３つのアクティブなプロセス（２３０、２４０、および２５０）を示す。

【００３１】

ＩＰＣファシリテータ２２２は、（プロセス２３０、２４０、および２５０などの）アクティブなプロセス間の通信を促進する。図２は、ＩＰＣファシリテータ２２２を実装するＯＳのカーネル２２０を図示するが、他の実装は、ＯＳのカーネルの外部にあるＩＰＣファシリテータを有することができる。その場合、それぞれが協調して動作し、および／またはＯＳと連携して動作するであろう。

【００３２】

メモリ２１０は、複数のメモリブロック２９２を有する交換ヒープ２９０も含む。交換ヒープ２９０は、（プロセス２３０、２４０、および２５０などの）複数のアクティブなプロセスによってアクセス可能である。交換ヒープ２９０は、ＳＩＰに、データを交換するためのメカニズムを提供する。

【００３３】

“Inter-Process Communications Employing Bi-directional Message Conduits”は、本明細書で説明した１つまたは複数の実装に適する、例示的なＩＰＣアーキテクチャ２００に関するさらなる詳細を開示している。

【００３４】

（交換ヒープ）

各ＳＩＰは、自己の独立した専用のヒープを維持する。ＳＩＰは、互いにメモリを共有しない。したがって、データが、あるＳＩＰから別のＳＩＰに渡されるとき、その渡され

たデータは、プロセスの専用ヒープ（private heap）からは来ない。代わりに、そのデータは、プロセス間で移動可能なデータの保持に使用される別個のヒープから来る。その別個のヒープは、図１に示される交換ヒープ１３２または図２に示される交換ヒープ２９０などの、交換ヒープである。

【００３５】

ＳＩＰは、自己の専用ヒープへのポインタを含むことができる。加えて、ＳＩＰは、共用交換ヒープ（public exchange heap）へのポインタを有することができる。少なくとも１つの説明された実施形態では、交換ヒープは、交換ヒープ自身へのポインタを含むのみである。各ＳＩＰは、交換ヒープへの複数のポインタを保持することができる。しかし、交換ヒープ内の各メモリブロックは、システムの実行の間の任意の時点において、多くても１つのＳＩＰによって所有される（すなわち、アクセス可能である）。

【００３６】

静的検証を実施するとき、プログラミングツール１６０は、交換ヒープ内のメモリブロックの所有権を追跡することができるが、各ブロックが、任意の時点で多くても１つのプロセスによって所有されるためである。交換ヒープ内の各ブロックが任意の時点で単一のプロセスによってアクセス可能であるという事実は、有用な相互排他の保証（mutual exclusion guarantee）も提供する。

【００３７】

（チャネル）

ＩＰＣアーキテクチャ２００では、チャネルは、厳密に２つのエンドポイントから成る双方向のメッセージコンジット（message conduit）である。エンドポイントは、チャネルピアと呼ばれることもある。チャネルは、損失を少なく（loss-lessly）かつ順番にメッセージを送達する。また、メッセージは典型的に、送信された順序で取り出される。意味的に、各エンドポイントは、受信キューを有し、エンドポイント上の送信側は、ピアのキュー上のメッセージを待ち行列に入れる。

【００３８】

チャネルは、チャネル規約によって記述される。換言すると、各チャネルの規約は、そのチャネル上でプロセス間通信の制約を指定する。例えば、規約は、プロセスが通信できる他のプロセスと、そのような通信が発生する方法とを指定することができる。チャネルの２つのエンドは、典型的に対称ではない。説明の目的で本明細書では、一方のエンドポイントをインポーティングエンド（Ｉｍｐ）と呼び、他方をエクスポーティングエンド（Ｅｘｐ）と呼ぶ。これらを、それぞれ、型Ｃ．ＩｍｐおよびＣ．Ｅｘｐを使用して型レベルで区別するが、Ｃは、インタラクションを規定するチャネル規約である。

【００３９】

図２は、チャネルを、電気プラグ、コード、およびコンセントとして比喩的に図示する。少なくとも１つの説明された実装では、チャネルは、厳密に２つのみのエンドポイントを有し、各エンドポイントは、多くても１つのプロセスによって所有される。示されるように、チャネル２６０は、プロセス２３０とＯＳのカーネル２２０を連結（link）し、２つのエンドポイント２６２および２６４のみを有する。チャネル２７０は、プロセス２４０とプロセス２５０とを連結し、２つのエンドポイント２７２および２７４のみを有する。チャネル２８０は、最初にプロセス２５０を自身に連結するが、それでも２つのエンドポイント２８２および２８４を有するのみである、新規に形成されたチャネルである。

【００４０】

これらのチャネルは、厳密に２つの（エンドポイントを表す）「プラグ」を有する「電気コード」のグラフィックメタファによって表される。電気を通すのではなく、これらの「コード」は、「コード」が差し込まれる場合に、各参加者（participant）によって（「双方的に」）送受信されるメッセージを通す。この双方向のメッセージパッシングは、チャネル２７０の隣にある方向を示す封筒（directional envelope）によって図示されている。

【００４１】

ＩＰＣアーキテクチャ２００は、メッセージパッシングＩＰＣ通信メカニズムを提供する。（一部の従来のアプローチにあるように）何らかの共有メモリの適時な読み書きを使用する代わりに、ＩＰＣアーキテクチャ２００は、プロセス間通信をメッセージの送受信に制限する。

【００４２】

従来的なＯＳのメッセージパッシングのアプローチは、１つの送信側と複数の受信側、または複数の送信側と１つの受信側のいずれかを有することが多い一方向のメカニズムである。これらの従来的なアプローチと異なり、ＩＰＣアーキテクチャ２００のチャネルは、厳密に２つのエンドポイントと多くても２つの参加者とを有する二方向のメカニズムである。

【００４３】

このことは、図２のチャネル２６０およびチャネル２７０によって図示されている。チャネル２６０は、プロセス２３０とＯＳのカーネル２２０を連結し、これら２つを連結するのみである。チャネル２７０は、プロセス２４０とプロセス２５０を連結し、これら２つを連結するのみである。

【００４４】

図２に図示されるように、双方向ＩＰＣチャネルのそれぞれは、厳密に２つのチャネルエンドポイントを有する。各チャネルエンドポイントは、同時に多くても１つのプロセスによって所有される。例えば、一方のチャネルエンドポイントは、あるプロセスによって所有され、他方のチャネルエンドポイントは、別のプロセスによって所有されるか、またはオペレーティングシステムのカーネルによって所有される。エンドポイントは、チャネル上で転送されることがある。それを行う際、これらのエンドポイントの所有権が転送される。

【００４５】

ＩＰＣファシリテータ２２２は、各メッセージおよび各メッセージのカプセル化が、任意の瞬間に、多くても１つのプロセスによって所有されることを保証する。チャネルレベルの抽象化を各チャネルに使用することによって、このことを実現することができる。さらに、チャネルの抽象化レベルでは、メッセージは、任意の瞬間に多くても１つのプロセスのアクセス可能なメモリ内に存在する。通信プロセス（communicating process）の観点からすると、メッセージ内部に含まれる状態またはメッセージからアクセス可能な状態が、共有されることはない。少なくとも１つの説明した実装では、メッセージは、送信されるまではメッセージ作成者のみによってアクセス可能である。少なくとも１つの説明した実装では、メッセージは、受信された後にメッセージ受信者のみによってアクセス可能である。

【００４６】

（所有権）

チャネル上で転送されるエンドポイントと他のデータとのメモリ分離は、コンパイル時に交換ヒープ内の全てのブロックを追跡することによって保証される。特に静的チェックは、これらのリソースがリソースを所有するプログラムポイントで発生し、方法がこれらのリソースの所有権をリークしないようにアクセスを実施する。追跡されたリソースは、正確な所有権モデルを有する。

【００４７】

各リソースは、任意の時点で、多くても１つのプロセスによって所有される。例えば、エンドポイントが、スレッドＴ１からスレッドＴ２にメッセージで送信されると、エンドポイントの所有権は、メッセージの受信時に、Ｔ１からそのメッセージに変わり、次いでＴ２に変わる。

【００４８】

従来のアプローチでは、プロセスは、データのコピーを作成してそのデータを転送する。結果として、そのデータは複数のプロセスによって所有される。データを送信したプロセスは、なおそのデータのコピーに作用することができる。

【００４９】
少なくとも１つの説明した実装では、データの所有権は、特定のＳＩＰに関連付けられる。データの所有権は、渡されているデータとともに転送される。したがって、送信ＳＩＰは、データを一旦渡すと、このデータに対するアクセスをもはや有さず、このデータのコピーを作成しなかったので、このデータに作用することができない。本明細書で説明される１つまたは複数の実装では、データは１つのＳＩＰによって所有され、チャネル上でデータが一旦送信されるとその所有権はデータとともに転送される。
【００５０】
同様に、チャネルの各エンドポイントは、ちょうど１つのＳＩＰによって所有される。エンドポイントの所有権は、エンドポイントの転送とともに別のＳＩＰに渡る。一旦送信されると、送信ＳＩＰは、ちょうど送信したチャネルのエンドポイントに対するアクセスはもはや有さない。
【００５１】
この（エンドポイントおよびデータの）所有権の転送は、図１に示される交換ヒープ１３２または図２に示される交換ヒープ２９０などの交換ヒープを介して達成される。より詳細には、交換ヒープ内のメモリブロックは、（対象データまたは対象エンドポイントのいずれかのメモリ位置への）ポインタを含む。チャネルに渡って別のプロセスと交換するとき、送信プロセスは、交換ヒープ内のメモリブロックへのポインタを、受信プロセスに転送する。
【００５２】
この手法では、送信プロセスは、効果的に対象データを受信プロセスに転送するが、自身のためのコピーを作成または維持せずにそれを行う。さらに、送信プロセスは、効果的に対象エンドポイントの所有権を、所有権を維持せずに受信プロセスに転送する。所有権の転送は、メッセージの送信側が、メッセージへのポインタを受信側のエンドポイントに格納することによって、メッセージ交換プロトコルの現在の状態によって決定される位置で所有権を渡すとして、説明されることもある。
【００５３】
コピーされるデータがないこれらの交換を、「ゼロコピー」アプローチと呼ぶことができる。このようなアプローチを使用すると、ディスクバッファおよびネットワークパケットを、送信されるデータをコピーまたはいずれも保存することなく、複数のチャネルにわたって、プロトコルスタックを通してアプリケーションプロセス内に転送することができる。
【００５４】
（チャネル規約）
チャネル規約は、プロセス分離アーキテクチャを促進するために、本明細書で説明される実装によって使用される。チャネル規約（およびプロセス間通信の他の態様）は、“Inter-Process Communications Employing Bi-directional Message Conduits”でも説明されている。
【００５５】
チャネル上の単純なインタラクションを記述する規約の例を以下に示す。

```
contract C1{
  in message Request (int x) requires x>0;
  out message Reply (int y);
  out message Error ();

  state start: Request?
               -> (Reply! Or Error!)
               -> Start;
}
```

【００５６】

この例では、Ｃｏｎｔｒａｃｔ　Ｃ１は、３つのメッセージ、すなわち、Ｒｅｑｕｅｓｔ、Ｒｅｐｌｙ、およびＥｒｒｏｒを宣言する。各メッセージ宣言は、そのメッセージに含まれる引数の型を指定する。例えば、ＲｅｑｕｅｓｔおよびＲｅｐｌｙの両方は、単一の整数値を含み、一方、Ｅｒｒｏｒは、いかなる値も持たない。さらに、各メッセージは、Ｓｐｅｃ＃が引数をさらに制限する節（clauses）を要求することを指定することができる。

【００５７】

メッセージに、方向（direction）をタグ付けすることもできる。規約は、エクスポータ（exporter）の観点から書かれている。したがって、本例では、Ｒｅｑｕｅｓｔは、インポータ（importer）によってエクスポータに送信することができるメッセージであり、一方、ＲｅｐｌｙおよびＥｒｒｏｒは、エクスポータからインポータに送信される。修飾詞なしで、メッセージは両方向に移動することができる。

【００５８】

メッセージ宣言の後、規約は、送受信動作によって駆動される状態マシンを介して、許容メッセージインタラクション（allowable message interaction）を指定する。宣言された第１の状態は、インタラクションの初期状態とみなされる。例示の規約　Ｃ１は、Ｓｔａｒｔと呼ばれる単一の状態を宣言する。状態名の後の、アクションのＲｅｑｕｅｓｔは、Ｓｔａｒｔ状態においてチャネルのエクスポート側がＲｅｑｕｅｓｔメッセージを受信する意思があることを示す。これに続いて、構文（Ｒｅｐｌｙ！　ｏｒ　Ｅｒｒｏｒ！）は、エクスポータがＲｅｐｌｙまたはＥｒｒｏｒメッセージのいずれかを送信（！）することを指定する。最後の部分（－＞Ｓｔａｒｔ）は、インタラクションがその後Ｓｔａｒｔ状態に続き、それにより無限にループすることを指定する。

【００５９】

もう少し関連する例は、ネットワークスタックに対する規約の一部分である。

```
public contract TcpConnectionContract{
  // Request（要求）
  in message connect (uint dstIP, ushort dstPort);

  out message Ready ();

  // Initial state（初期状態）
  state Start : Ready! -> ReadyState;

  state ReadyState : one {
    Connect? -> ConnectResult;
    BindLocalEndpoint? -> BindResult;
    Close? -> Closed;
  }

  // Binding to a local endpoint（ローカルエンドポイントにバインド）
  state BindResult : one {
    OK! -> Bound;
    InvalidEndPoint! -> ReadyState;
  }

  in message Listen ();

  state Bound : one {
    Listen? -> ListenResult;
    Connect? -> ConnectResult;
```

```
    Close? -> Closed;
   }
・・・
```

【００６０】

規約内のプロトコル仕様は、いくつかの目的を果たす。そのプロトコル仕様は、実行時または静的分析ツールを通してのいずれかで、プログラミングエラーの検出を助けることができる。ランタイムモニタリングは、チャネル上で交換されたメッセージに応じて、規約の状態マシンを駆動し、誤った遷移（transition）を監視する。それ自体により、ランタイムモニタリング技術は、１つのプログラム実行におけるエラーを検出するが、非終了（non-termination）などの「活性（liveness）」エラーを検出することができない。活性特性（liveness properties）は、「何か良いことが最終的に発生する」形態、例えば「最終的にプログラムがメッセージを送信する」形態の特性である。静的プログラム分析は、プロセスが、全てのプログラム実行において正しく、スタックフリーであるという強い保証を提供することができる。一般に、静的分析は、発生時に１つの実行をモニタリングすることに限定されない。静的分析は、例えば、プロセスが最終的に何かを行うか否かを判定するために、プロセスの命令を調べることに依拠することがある。これは常に機能するわけではないことを示す基本的結果が論理内に存在するが、多数の場合では、これは十分に良く機能することができる。

【００６１】

一実装では、ランタイムモニタリングと静的検証の組合せを使用する。チャネル上の全てのメッセージは、正確性を検出するが活性の問題は検出しないチャネルの規約とチェックされる。本明細書で説明される実装は、安全性の特性を検証する静的チェッカを有する。

【００６２】

加えて、コンパイラは、チャネル上で未完了（outstanding）の可能性があるメッセージの最大数を判定する規約を使用し、その規約により、コンパイラがチャネルエンドポイント内のバッファを静的に割り当てることが可能となる。静的に割り当てられたバッファは、通信のパフォーマンスを改善する。

【００６３】

（エンドポイント）

チャネルは、そのチャネルのインポート側およびエクスポート側を表すエンドポイントの対として明示される。各エンドポイントは、チャネルが順守する規約を指定する型を有する。エンドポイント型は、各規約内で暗黙的に宣言される。規約Ｃ１は、クラスとして表され、エンドポイント型は、以下のようにそのクラス内部のネストされた型である。

・　Ｃ１．Ｉｍｐ　－規約Ｃ１でのチャネルのインポートエンドポイントの型
・　Ｃ１．Ｅｘｐ　－規約Ｃ１でのチャネルのエクスポートエンドポイントの型

【００６４】

（送受信メソッド）

各規約クラスは、その規約内で宣言したメッセージを送受信するためのメソッドを含む。例として、以下のメソッドを与える。

```
C1.Imp {
  void SendRequest (int x);
  void RecvReply (out int y);
  void RecvError ();
}

C1.Exp {
  void RecvRequest (out int x);
  void SendReply (int y);
  void SendError ();
```

}
【００６５】
Ｓｅｎｄメソッドの意味は、それらがメッセージを非同期的に送信することである。受信メソッドは、所与のメッセージが到着するまでブロックされる。異なるメッセージが最初に到着するとエラーが発生する。そのようなエラーは、プログラムが規約検証チェックを通る場合は、発生することはない。受信者が次に要求するメッセージを正確に知らなければ、これらのメソッドは適切ではない。
【００６６】
（方法の実装）
図３は、静的に検証可能なＳＩＰのための効率的なプロセス間通信を促進する方法３００および４００を示す。これらの方法３００および４００は、図１および図２に示される様々なコンポーネントの１つまたは複数によって実施される。さらに、これらの方法３００および４００を、ソフトウェア、ハードウェア、ファームウェア、またはそれらの組合せで実施することができる。
【００６７】
図３のブロック３０２で、ＯＳは、コンピュータオペレーティングシステム環境における１つまたは複数のＳＩＰの実行を提供する。
【００６８】
ブロック３０４で、ＯＳは、特定のデータセットの所有権を第１のＳＩＰに関連付ける。このデータセットは、図１に示される交換ヒープ１３２または図２に示される交換ヒープ２９０などの、交換ヒープ内のメモリブロックとすることができる。このデータセットは、メッセージとすることができる。このデータセットは、データ、あるいはデータを含むメモリ位置への１つまたは複数のポインタを含むことができる。また、このデータセットは、チャネルエンドポイントへの１つまたは複数のポインタを含むことができる。
【００６９】
ブロック３０６で、ＯＳは、特定のデータセットを第１のＳＩＰから第２のＳＩＰに送信する。ここでの送信は、（交換ヒープ内の）データセットへのポインタを、第２のＳＩＰに提供することから成ることがある。代替として、送信は、第２のＳＩＰに接続されたチャネルのエンドポイントにメッセージを書き込むことから成ることがある。
【００７０】
ブロック３０８で、ＯＳは、特定のデータセットの所有権を第１のＳＩＰから第２のＳＩＰに転送する。メッセージがチャネル上で送信されると、所有権は、送信ＳＩＰから受信ＳＩＰに渡る。送信ＳＩＰは、もはやメッセージに対する参照を保持しない。実質的に、送信ＳＩＰは、もはや送信メッセージに対するアクセスを有さない。
【００７１】
送信３０６および転送３０８の間、送信された情報のコピーは、保持されない。実際、送信された情報のコピーは、作成されない。データセットへのポインタ（またはより正確には、データまたはポインタを格納しているメモリブロックへのポインタ）だけが転送されるので、コピーは、作成および送信されない。
【００７２】
この所有権の不変条件は、（プログラミングツール１６０およびＯＳ１００などの）プログラミングツールおよびオペレーティングシステムによって施行される。この所有権の不変条件は、少なくとも３つの目的を果たす。１つめは、プロセス間の共有を回避することである。２つめは、メッセージのポインタエイリアス（pointer aliasing）を排除することによって、静的プログラム分析を促進することである。３つめは、コピーまたはポインタを渡すこと（pointer passing）により実装することができるメッセージパッシングの意味（semantics）を提供することによって、実装の柔軟性を許容することである。
【００７３】
図４に示されるように、４０２で、オペレーティングシステムは、コンピュータオペレーティングシステム環境における１つまたは複数のＳＩＰの実行を提供する。

【００７４】
ブロック４０４で、ＯＳは、特定のプロセス間通信チャネルの特定のエンドポイントの所有権を、第１のＳＩＰに関連付ける。このデータセットは、図１に示される交換ヒープ１３２または図２に示される交換ヒープ２９０などの、交換ヒープ内のメモリブロックとすることができる。このデータセットは、メッセージとすることができる。このデータセットは、１つまたは複数のポインタを含むことができる。このデータセットは、１つまたは複数のポインタを含むメモリ位置への１つまたは複数のポインタを含むことができる。また、このデータセットは、チャネルエンドポイントへの１つまたは複数のポインタを含むことができる。
【００７５】
ブロック４０６で、ＯＳは、特定のプロセス間通信チャネルの特定のエンドポイントを、第１のＳＩＰから第２のＳＩＰに送信する。ここでの送信は、（交換ヒープ内の）特定のエンドポイントへのポインタを、第２のＳＩＰに提供することから成ることがある。代替として、送信は、第２のＳＩＰに接続されたチャネルのエンドポイントにメッセージを書き込むことから成ることがある。
【００７６】
ブロック４０８で、ＯＳは、特定のプロセス間通信チャネルの特定のエンドポイントの所有権を、第１のＳＩＰから第２のＳＩＰに転送する。エンドポイントの所有権が、送信ＳＩＰから受信ＳＩＰに渡ると、送信ＳＩＰは、もはやメッセージに対する参照を保持しない。実質的に、送信ＳＩＰは、もはや送信されたデータに対するアクセスを有さない。
【００７７】
さらに、エンドポイント所有権のこの転送は、「コピー」を生成および転送することなく発生する。エンドポイントへのポインタ（または、エンドポイントへのポインタを格納しているメモリブロックへのポインタ）だけが転送されるので、コピーは、作成および送信されない。
【００７８】
（検証）
プログラミングツール１６０は、１つまたは複数のＳＩＰのプログラミングを検証することができる。プログラミングツール１６０は、実行されるコードがタイプセーフであることを検証し、コンパイラによる強い不変条件の使用、および実行時の強い不変条件の使用の実施を検証する。このような強い不変条件は、（限定ではなく例として）以下を含む。
・　交換ヒープ内の各ブロックは、任意の時点で、多くても１つの所有スレッド（すなわち、プロセス）を有する。
・　交換ヒープ内のブロックは、そのブロックの所有者によってアクセスされるのみである。したがって、ブロックが解放された後、または所有権の転送後、アクセスはない。
・　チャネル規約の実施形態は、プロセス間の通信（例えば、そのチャネル規約に対応するチャネルにおいて観察されたメッセージのシーケンス）を定義し、制限する。
【００７９】
（検証の方法の実装）
図５は、分離プロセスの検証方法５００を示す。この方法５００は、図１および図２に示される様々なコンポーネントの１つまたは複数によって実施される。さらに、この方法５００を、ソフトウェア、ハードウェア、ファームウェア、またはそれらの組合せで実施することができる。
【００８０】
図５のブロック５０２で、１つまたは複数のＳＩＰのための実行可能コードを、ＳＩＰをサポートするコンピュータオペレーティングシステム環境でコンパイルする。
【００８１】
ブロック５０４で、コンパイル時間の間に、プログラミングツール１６０は、交換ヒープ内の各メモリブロックが、任意の時点において多くても１つの所有プロセスを有するこ

とを、確認する。これは、任意の一時点において１つのＳＩＰのみが、任意の特定のメモリブロックを所有することを意味する。

【００８２】

ブロック５０６で、コンパイル時間の間に、プログラミングツール１６０は、交換ヒープ内の各メモリブロックが、その正当な所有者（例えば、ＳＩＰ）によってアクセスされるのみであることを確認する。

【００８３】

ブロック５０８で、コンパイル時間の間に、プログラミングツール１６０は、チャネルの規約条件（contract term）が守られていることを確認する。例えば、ツールは、コントロール内で定義されたメッセージシーケンスが順守されていることを確認する。

【００８４】

プログラミングツール１６０は、このような確認の結果を、ユーザ、プログラムモジュール、および／またはオペレーティングシステムに報告することができる。プログラミングツール１６０は、その検証をコンパイル中に実施することができる。加えて、プログラミングツール１６０は、生成された中間言語コード上でこれらの同じ特性を検証することもできる。さらに、プログラミングツール１６０は、型付けされたアセンブリ言語についての結果として生じる形態を再度検証することができる。

【００８５】

（結論）

本明細書で説明される技術を、１つまたは複数のコンピュータネットワークの一部、あるいはそれらの組合せとして、プログラムモジュール、汎用および専用のコンピューティングシステム、ネットワークサーバおよび装置、専用電子回路およびハードウェア、ファームウェアを含む（がこれらに限らない）多くの方法で実装することができる。

【００８６】

本明細書で説明される１つまたは複数の実装を、限定ではないがＰＣ、サーバコンピュータ、ハンドヘルドまたはラップトップデバイス、マルチプロセッサシステム、マイクロプロセッサベースのシステム、プログラム可能な家庭用電化製品、無線電話および装置、汎用および専用の装置、ＡＳＩＣ、ネットワークＰＣ、シンクライアント、シッククライアント、セットトップボックス、ミニコンピュータ、メインフレームコンピュータ、上記システムまたは装置の任意のものを含む分散コンピューティング環境などを含む、使用に適した多くのよく知られたコンピューティングシステム、環境、および／または構成を通して、実装することができる。

【００８７】

１つまたは複数の上述の実装を、構造的特徴および／または方法のステップに特有の言葉で説明してきたが、他の実装を、本明細書で説明される特定の例示的な特徴またはステップなしに実施することができることを理解されたい。むしろ、その特定の例示的な特徴およびステップは、１つまたは複数の実装の好ましい形態として開示されている。一部の例では、例示的な実装の説明を明確にするため、公知の特徴は、省略または簡略化されていることもある。さらに、理解を容易にするために、ある方法のステップを別々のステップとして区別してある。しかし、これらの別々に区別されたステップを、それらのパフォーマンスに依存する順序として必ずしも解釈すべきではない。

【図面の簡単な説明】

【００８８】

【図１】本明細書で説明される１つまたは複数の実装をサポートする、オペレーティングシステムアーキテクチャに関する動作シナリオの図である。

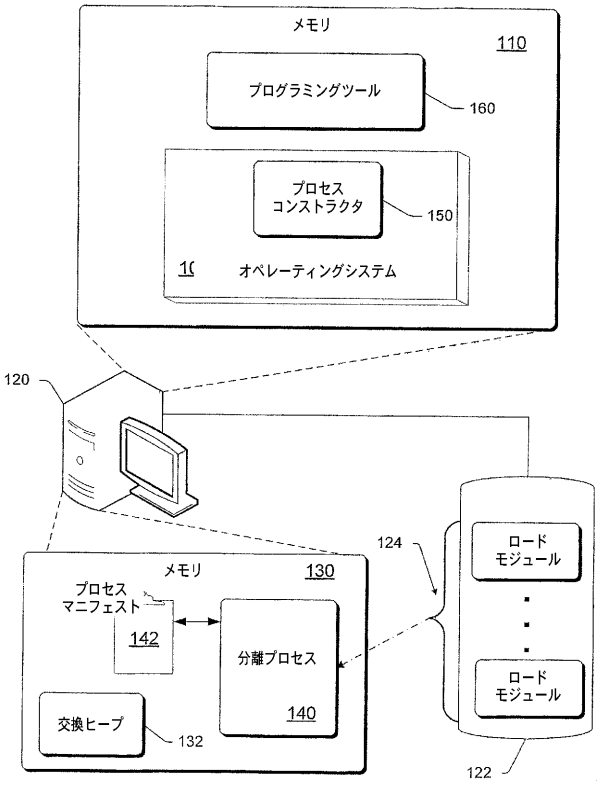
【図２】本明細書で説明される１つまたは複数の実装をサポートする、オペレーティングシステムアーキテクチャに関する別の動作シナリオの図である。

【図３】本明細書で説明される１つまたは複数の実装をサポートする、オペレーティングシステムアーキテクチャのブロック図である。

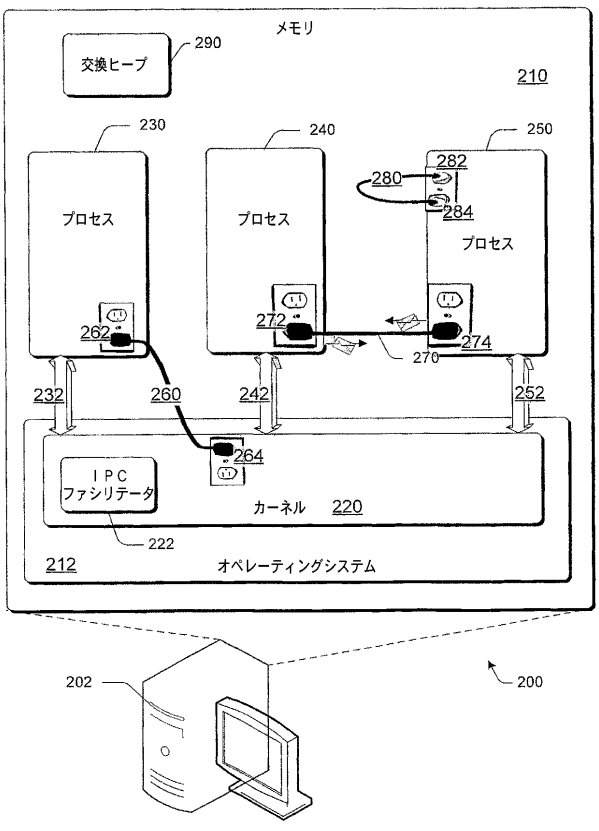
【図４】本明細書で説明される、別の方法の実装についてのフローチャートである。

【図５】本明細書で説明される、別の方法の実装についてのフローチャートである。

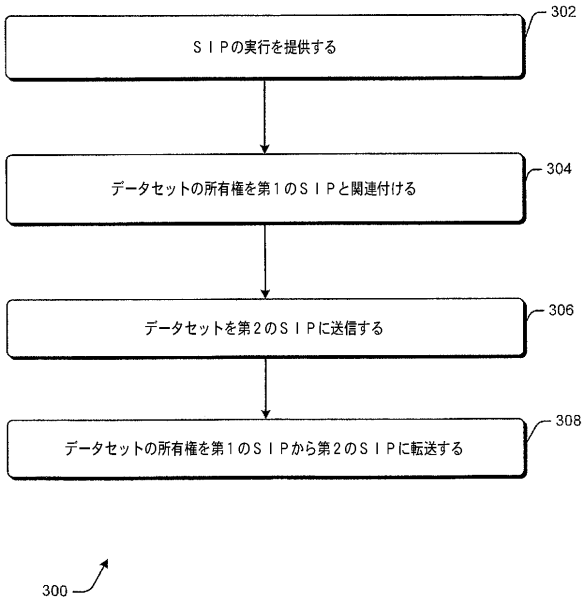
【図１】



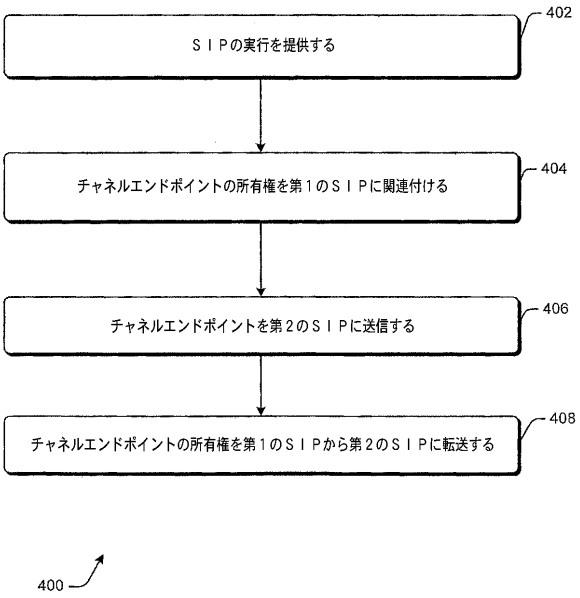
【図２】



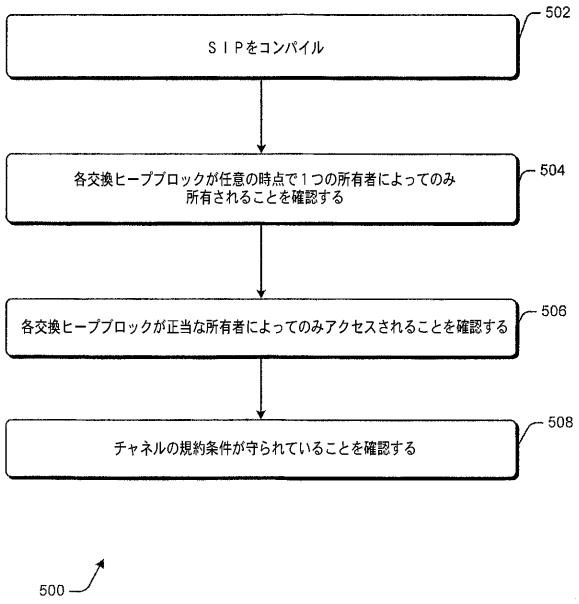
【図３】


ＳＩＰの実行を提供する
302
データセットの所有権を第１のＳＩＰと関連付ける
304
データセットを第２のＳＩＰに送信する
306
データセットの所有権を第１のＳＩＰから第２のＳＩＰに転送する
308
300

【図４】


ＳＩＰの実行を提供する
402
チャネルエンドポイントの所有権を第１のＳＩＰに関連付ける
404
チャネルエンドポイントを第２のＳＩＰに送信する
406
チャネルエンドポイントの所有権を第１のＳＩＰから第２のＳＩＰに転送する
408
400

【図５】


ＳＩＰをコンパイル
502
各交換ヒープブロックが任意の時点で１つの所有者によってのみ
所有されることを確認する
504
各交換ヒープブロックが正当な所有者によってのみアクセスされることを確認する
506
チャネルの規約条件が守られていることを確認する
508
500

フロントページの続き

(72)発明者 ガレン シー. ハント
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション インターナショナル パテンツ内
(72)発明者 ジェームズ アール. ラルス
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション インターナショナル パテンツ内
(72)発明者 マーティン アバディ
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション インターナショナル パテンツ内
(72)発明者 マーク エイケン
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション インターナショナル パテンツ内
(72)発明者 ポール バーラム
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション インターナショナル パテンツ内
(72)発明者 マヌエル エー. ファンドリッチ
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション インターナショナル パテンツ内
(72)発明者 クリス ハウブリッツェル
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション インターナショナル パテンツ内
(72)発明者 オリオン ハドソン
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション インターナショナル パテンツ内
(72)発明者 スティーブン レビ
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション インターナショナル パテンツ内
(72)発明者 ニック マーフィー
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション インターナショナル パテンツ内
(72)発明者 ビャルネ スティーンズガード
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション インターナショナル パテンツ内
(72)発明者 デビッド タルディティ
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション インターナショナル パテンツ内
(72)発明者 テッド ウォッバー
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション インターナショナル パテンツ内
(72)発明者 ブライアン ジル
アメリカ合衆国 ９８０５２ ワシントン州 レッドモンド ワン マイクロソフト ウェイ マイクロソフト コーポレーション インターナショナル パテンツ内

審査官 吉田 美彦

(56)参考文献 特開平０５－２２４９５６（ＪＰ，Ａ）

(58)調査した分野(Int.Cl., ＤＢ名)

G06F 9/48
G06F 9/54
G06F 11/36