(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4410795号
(P4410795)
(45)発行日　平成22年2月3日(2010.2.3)　(24)登録日　平成21年11月20日(2009.11.20)

(51)Int.Cl.　F I
G06F 9/52 (2006.01)　G06F 9/46 472Z

請求項の数 13 (全 18 頁)

(21)出願番号　特願2006-506127(P2006-506127)
(86)(22)出願日　平成16年4月15日(2004.4.15)
(65)公表番号　特表2006-524381(P2006-524381A)
(43)公表日　平成18年10月26日(2006.10.26)
(86)国際出願番号　PCT/GB2004/001624
(87)国際公開番号　WO2004/094863
(87)国際公開日　平成16年11月4日(2004.11.4)
審査請求日　平成19年4月13日(2007.4.13)
(31)優先権主張番号　10/422,193
(32)優先日　平成15年4月24日(2003.4.24)
(33)優先権主張国　米国(US)

早期審査対象出願

(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】共有リソースの同時アクセス

(57)【特許請求の範囲】
【請求項１】
コンピュータの共有リソースにアクセスする方法であって、前記共有リソースを使用する複数の要求を非同期的に発することが可能な複数のプロセスのうちの第１のプロセスにおいて、
（ａ）前記共有リソースを使用する現在の要求に固有の識別子をアトミックに割当てるステップであって、前記固有の識別子を、前記複数のプロセスによって発せられた他の要求に対し前記現在の要求が受け取られる順番に基づいて前記現在の要求に割当てるステップと、
（ｂ）前記固有の識別子を用いる前記現在の要求を準備することにより当該現在の要求の処理を完了させるステップと、
（ｃ）前記現在の要求の処理が完了した後に、完了した要求のカウントをアトミックに増分させるステップと、
（ｄ）前記現在の要求の処理が完了した後に、最近完了した要求より前に発せられた他の要求の処理が完了したかどうかをアトミックに判別するステップと、
（ｅ）前記最近完了した要求より前に発せられた他の要求の処理が完了したという判別に応答して、処理が完了したディスパッチされていない要求の各々をディスパッチするステップと、
を含む方法。
【請求項２】

前記ステップ（ｂ）が、非アトミックに行われる、請求項１に記載の方法。
【請求項３】
前記第１のプロセスにおいて、前記現在の要求の処理が完了した後に、前記現在の要求が前記最近完了した要求であるかどうかをアトミックに判別するステップをさらに含む、請求項１に記載の方法。
【請求項４】
前記第１のプロセスにおいて、前記現在の要求の処理が完了した後に、前記現在の要求が最後のディスパッチ動作の後に発せられた最初の要求であるかどうかをアトミックに判別するステップと、
前記現在の要求が前記最後のディスパッチ動作の後に発せられた最初の要求であるという判別に応答して前記現在の要求をディスパッチするステップをさらに含む、請求項１に記載の方法。
【請求項５】
前記ステップ（ｅ）が、複数の要求をバッチでディスパッチするステップを含む、請求項１に記載の方法。
【請求項６】
前記ステップ（ａ）が、要求カウンタを増分させるステップを含み、
前記ステップ（ｃ）が、完了カウンタを増分させるステップを含み、
前記方法が、
前記現在の要求が最近完了した要求であるという判別に応答して、前記現在の要求の処理が完了した後に、前記現在の要求の前記固有の識別子と同値になるように最後の完了カウンタをアトミックに更新するステップをさらに含み、
前記ステップ（ｄ）が、前記最後の完了カウンタを前記完了カウンタと比較するステップを含み、
前記ステップ（ｅ）が、前記最後の完了カウンタと同値になるように最後のディスパッチ・カウンタを更新するステップを含む、請求項１に記載の方法。
【請求項７】
前記要求カウンタ、前記完了カウンタ、前記最後の完了カウンタ、及び前記最後のディスパッチ・カウンタの各々が、単調に増加する一連の番号を用いて更新される、請求項６に記載の方法。
【請求項８】
前記最近完了した要求より前に発せられた他の要求の処理がまだ完了されていないという前記ステップ（ｄ）の否定的な判別に応答して、処理が完了したディスパッチされていない要求の各々のディスパッチを据え置くステップをさらに含み、
前記複数のプロセスのうちの第２のプロセスにおいて、
前記現在の要求より前に割当てられた固有の識別子を有する、前記現在の要求に対し先行する要求の処理を完了させるステップと、
前記先行する要求の処理が完了した後に、前記最近完了した要求より前に発せられた他の要求の処理が完了されたことをアトミックに判別するステップと、
前記最近完了した要求より前に発せられた他の要求の処理が完了されたという判別に応答して、処理が完了したディスパッチされていない要求の各々をディスパッチするステップと、
を含む、請求項１に記載の方法。
【請求項９】
前記共有リソースが、入力／出力アダプタのフレーム・バッファの少なくとも１つのスロットを含み、
前記ステップ（ｂ）が、前記スロットを用いてデータを通信するステップを含み、
前記ステップ（ｅ）が、前記通信されるデータの受信をクライアントに通知するステップを含む、請求項１に記載の方法。
【請求項１０】

前記コンピュータが、論理的に区画化されたコンピュータを含み、
前記入力／出力アダプタが、前記論理的に区画化されたコンピュータの区画間通信のために用いられる仮想ローカル・エリア・ネットワークに結合された仮想入力／出力アダプタである、請求項９に記載の方法。
【請求項１１】
前記共有リソースが、プロセッサ、ハードウェア・デバイス、入力／出力デバイス、ストレージ・デバイス、アダプタ、コントローラ、データベース、ファイルシステム、サービス、及びこれらのコンポーネントからなる群から選択される、請求項１に記載の方法。
【請求項１２】
共有リソースと、
前記共有リソースを使用する複数の要求を非同期的に発することが可能な複数のプロセスを実行するように構成された少なくとも１つのプロセッサと、
第１のプロセスにおいて、前記共有リソースを使用する現在の要求に固有の識別子をアトミックに割当てるステップと、前記固有の識別子を用いる前記現在の要求を準備することにより当該現在の要求の処理を完了させるステップと、前記現在の要求の処理が完了した後に、完了した要求のカウントをアトミックに増分させるステップと、前記現在の要求の処理が完了した後に、最近完了した要求より前に発せられた他の要求の処理が完了したかどうかをアトミックに判別するステップと、前記最近完了した要求より前に発せられた他の要求の処理が完了されたという判別に応答して、処理が完了したディスパッチされていない要求の各々をディスパッチするステップとを行うために、前記少なくとも１つのプロセッサ上で実行されるように構成されたプログラム・コードと、
を含み、前記固有の識別子は、前記複数のプロセスによって発せられた他の要求に対し前記現在の要求が受け取られる順番に基づいて当該現在の要求に割当てられる、装置。
【請求項１３】
請求項１ないし請求項１１の何れか１項に記載の方法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、一般に、コンピュータ及びコンピュータ・ソフトウェア、特に、コンピュータの共有リソースへの同時アクセスを管理することに関する。
【背景技術】
【０００２】
現代社会においてはコンピュータへの依存度が高まり続けているので、コンピュータ・技術は、増加する需要についていくべく最前線で進歩していかなければならない。重要な研究及び開発努力の１つの特定の対象は、並列性、すなわち多数のタスクを並列に処理する性能である。
【０００３】
増加した並列処理を容易にするために、多数のコンピュータ・ソフトウェア及びハードウェア技術が開発されている。ハードウェアの観点から、コンピュータは、作業負荷の受容能力の増加を与えるために、ますます多数のマイクロプロセッサに依存しつつある。さらに、多数のスレッドを並列に実行する能力をサポートして、多数のマイクロプロセッサの使用を通して達成できるのと同じ多くの性能ゲインを効果的に与える幾つかのマイクロプロセッサが開発されている。ソフトウェアの観点から、コンピュータ・プログラムが多数のスレッドを同時に実行できるようにし、それにより多数のタスクを本質的に同時に実行することができるマルチスレッド化オペレーティング・システム及びカーネルが開発されている。
【０００４】
さらに、幾つかのコンピュータは、論理的区画化の概念を実装し、そこでは、単一の物理的コンピュータが、本質的に多数の独立した「仮想」コンピュータ（論理区画と呼ばれ

る）のように動作可能にされ、物理的コンピュータ（例えば、プロセッサ、メモリ、入力／出力デバイス）における種々のリソースは、種々の論理区画の間で割当てられる。各論理区画は、個別のオペレーティング・システムを実行し、ユーザ及び論理区画上で実行するソフトウェア・アプリケーションの見地から、完全に独立したコンピュータのように動作する。

【０００５】

並列性は、多数のタスクを一度に行う能力によりシステム性能を効果的に高めるが、並列性の１つの副作用は、多数の同時処理動作を同期させる必要があるために、特に、多数のスレッドによって共有されることが可能なシステム・リソースに関するシステムの複雑さが増すことである。特定のリソースにアクセスすることができる個別のスレッド又はプロセスは、典型的には、他のスレッド又はプロセスのアクティビティを認識しない。したがって、１つのプロセスが別のプロセスに関する特定のリソースに予期しない形でアクセスして、不確定の結果及び潜在的なシステムエラーをもたらす恐れがある。

【０００６】

しかし、ほんの一例として、イーサネット（登録商標）・ネットワーク・アダプタのような入力／出力（Ｉ／Ｏ）アダプタは、コンピュータにおいて実行される多数のスレッド又はプロセスによって共有されることが可能な有限の数のリソースを有することができる。こうしたタイプの１つのリソースはフレーム・バッファであり、それは、イーサネット（登録商標）と互換性のあるネットワークに及び該ネットワークから通信されるデータ・フレームを格納するのに用いられる。フレーム・バッファは、異なるプロセスによって順番に用いられるスロットに配置されることが多く、Ｉ／Ｏアダプタは、各スロットの消費を通知される。しかしながら、スロットの１つの消費が多大な時間量を占領し、その間、他のプロセスが他のスロットを利用できないことがある。さらに、２つのプロセスが同じスロットを同時に使用しようとする懸念がある。

【０００７】

これらの懸念に対処するために、ロック又はセマフォーといった逐次化機構を用いて、共有リソースへのアクセスを一度に一つのプロセスに制限することができる。ロック又はセマフォーは、本質的には、特定の共有エンティティ又はリソースにアクセスするためにマルチスレッド化環境におけるプロセス又はスレッドによって排他的に得られる「トークン」である。プロセス又はスレッドがリソースにアクセスする前に、最初にシステムからトークンを取得しなければならない。別のプロセス又はスレッドがトークンを同時に所有する場合には、他のプロセス又はスレッドによってトークンが解放されるまで、前者のプロセス又はスレッドは、リソースにアクセスすることを許可されない。このようにして、リソースへのアクセスは、不確定の動作が起こるのを防止するために効果的に「逐次化」される。

【０００８】

ロックされたリソースにアクセスする必要があるあらゆるスレッド又はプロセスは、リソースへのアクセスを進める前に、そのリソースのロックが解除されるまで待たなければならないことが多いので、共有リソースが逐次化機構によって管理されるときにはいつも、並列化の機会が減少されることが多い。したがって、共有リソースにロックがかけられる頻度及び持続時間を最小にすることへの強い要望がある。

【０００９】

前述のＩ／Ｏアダプタの例においては、フレーム・バッファへの対立のないアクセスを保証するための最も直接的な解決策は、フレーム・バッファのあらゆる使用に対して単純にロックをかけることである。しかしながら、そのようにすると、フレーム・バッファに格納されたデータを準備し、解釈し、その他の処理をするプロセスによって大量の作業が実行される必要があるので、並列化の機会が大幅に制限される。したがって、これらの全てのアクティビティが発生する間の他のプロセスによるフレーム・バッファへのアクセスのロックアウトは、並列化の観点からは逆効果である。

【００１０】

或る状況においては、共有リソースにロックをかける前に、共有リソースの使用に関連する幾つかの事前作業を行って、共有リソースが他のプロセスに対してロックされる必要がある時間の量を減らすことが可能である。しかしながら、そのようにすることを意図された従来の試みは、リソースの利用可能性の重複チェックと、より複雑な事前作業が実行されることを要求することが見出されている。

【発明の開示】

【発明が解決しようとする課題】

【００１１】

したがって、マルチスレッド化されたコンピュータにおいて実行される多数のスレッド又はプロセスによる共有リソースへのアクセスを調整する改善された方法への大きな必要性がある。

【課題を解決するための手段】

【００１２】

本発明は、従来技術に関連するこれらの問題及び他の問題に対処するために、マルチスレッド化されたコンピュータにおける複数のプロセスによる共有リソースへのアクセスが、共有リソースを使用する要求を受け取る順番と、こうした要求を受け取った後にそれらの要求の処理が完了する順番との両方を追跡するアトミック・オペレーションの集合を介して管理されるというような装置、プログラム及び方法を提供する。要求のディスパッチは、最近完了された要求より前に受け取った、ディスパッチされていない要求の処理が完了するまで、効果的に据え置かれる。

【００１３】

特に、本発明の１つの態様によれば、共有リソースは、共有リソースを使用する要求を非同期的に発することが可能な複数のプロセスによってアクセスされることができる。第１のプロセスにおいて、共有リソースを使用する現在の要求に固有の識別子がアトミックに割当てられる。この場合、固有の識別子は、複数のプロセスによって発せられた他の要求に対し現在の要求を受け取る順番に基づいて現在の要求に割当てられる。その後、固有の識別子を用いる現在の要求の処理が完了し、そうした処理が完了した後に、完了した要求のカウントがアトミックに増分され、最近完了した要求より前に発せられた他の要求の処理が完了したかどうかについてのアトミックな判別がなされる。完了していた場合、処理が完了した、ディスパッチされていない要求の各々がディスパッチされる。

【００１４】

本発明に係る幾つかの実施形態においては、要求の処理の完了は、非アトミックに行うことができ、したがって、要求の処理と関連して行われるアトミック・オペレーションの持続時間を制限し、それにより共有リソースへの競合を減少させる。さらに、幾つかの実施形態においては、ディスパッチされていない要求のディスパッチの結果として、多数の要求がバッチで対処され、それにより個々のディスパッチのオーバーヘッドが減少する。

【００１５】

ここで、本発明を、添付の図面を参照しながら、一例として説明する。

【発明を実施するための最良の形態】

【００１６】

以下に示される実施形態は、複数のクライアントによって発せられた、共有リソースを使用する要求を受け取る順番と、共有リソースを使用する要求のディスパッチと関連してこうした要求の処理が完了する順番との両方を追跡するものである。要求は、非同期的に発せられることができ、さらに、順番どおりではなく完了させることができる。結果として、要求を受け取る際に、要求を受け取る順番に基づいて、要求に固有の識別子が割当てられる。さらに、完了した要求の番号が追跡され、最近完了した要求より前に発せられた他の要求の処理が完了したかどうかを判別するために、各要求の取り扱いと関連して判別がなされる。この条件が満たされたと判断されると、処理が完了した、ディスパッチされていない要求の各々がディスパッチされる。

【００１７】

以下で一層明らかとなるように、要求を受け取る順番、及びこうした要求が完了する順番の追跡は、共有リソースに関連する一連のカウンタの使用を通じて行われ、これらのカウンタは、セマフォーのような排他的ロック機構を介してアトミックにアクセスされることが可能である。要求カウンタは、要求を受け取る際に、要求に固有の識別子を割当てるために用いることができる。完了した要求（すなわち、準備及びサブミッションといった処理が完了した要求）の番号を追跡するために、完了カウンタを用いることができる。最後の完了カウンタは、最近完了した要求を追跡するのに用いることができ、一方、最後のディスパッチ・カウンタは、最近ディスパッチされた要求を追跡するのに用いることができる。

【００１８】

本発明に係る多くの実施形態においては、上述のカウンタへのアクセスのみがアトミックである必要がある。要求の取り扱いに関係する他の処理は、非アトミックに取り扱われることが多く、したがって、要求の取り扱いに関連して行われるアトミック・オペレーションの持続時間が制限され、それにより、そうでなければ共有リソースへの同時アクセスを妨げる競合が減少される。

【００１９】

さらに、本発明に係る実施形態は、要求のバッチでのディスパッチを与える能力をサポートする。要求のディスパッチは、本発明に係る異なる形で行われる。例えば、共有リソースが、ネットワーク・アダプタのような入力／出力アダプタのフレーム・バッファにおけるスロットである場合には、ディスパッチオペレーションは、入力／出力アダプタを介してそのクライアントに関するパケットが受け取られるときにはいつでも、クライアントへの割り込みの生成として実装することができる。共有リソースが、ＳＣＳＩネットワークのようなネットワークにおける大容量ストレージ・デバイスである場合には、ディスパッチ・オペレーションは、オペレーティング・システムへのＩ／Ｏ要求の完了通知とすることができる。

【００２０】

要求の性質は、典型的には、当該共有リソースの特定のタイプに応じて変わることになる。例えば、データベース・リソースにおいては、要求はデータベース・トランザクションを表すことができ、一方、ファイル管理システム・リソースにおいては、要求はファイルアクセス要求を表すことができる。ストレージ・デバイスにおいては、要求は入力／出力（Ｉ／Ｏ）要求を表すことができ、一方、プロセッサ・リソースにおいては、要求はプロセッサに論理区画を割当てるための要求を表すことができる。本発明の開示内容の利点を有する他の共有リソース及び要求の実装が、当業者には明らかであろう。

【００２１】

ここで、幾つかの視点を通じて同じ番号が同じ部分を表す図面に移ると、図１は、本発明に係る共有リソースへの同時及び非同期アクセスをサポートするために用いることができる汎用コンピュータ・アーキテクチャ５０の例示的な構成を示す。特に、アーキテクチャ５０は、汎用共有リソース５６への複数のクライアント５４によるアクセスを管理するリソース・マネージャ５２を組み入れていることが示されている。

【００２２】

共有リソース５６は、コンピュータ、例えば、種々のタイプの入力／出力、イーサネット（登録商標）・アダプタ、ＳＣＳＩアダプタその他のネットワーク・アダプタ、ストレージ・アダプタ及びコントローラ、ワークステーション・アダプタ及びコントローラ、データベース・コントローラ、プロセッサといったその他のハードウェア・リソース、その他の入力／出力アダプタ及びコントローラなどによってアクセスされることができる事実上あらゆるタイプのコンピュータ・リソースを表すと考えられる。さらに、共有リソース５６は、例えば、カーネル又はオペレーティング・システム・サービス、ファイル管理システム、データベースなどといった種々のタイプのソフトウェア・リソースを含むと考えられる。共有リソースはまた、例えば、バッファ又は入力／出力アダプタに常駐する他のメモリ・リソースといった上記のタイプのリソースのサブコンポーネントを含むことがで

きる。

【００２３】

クライアント５４は、それぞれ、事実上あらゆるプログラム・コード、又はその他のクライアントに対して非同期的な形でリソース５６にアクセスすることができる外部プログラマブル・デバイスも表すと考えられる。図示された実施形態においては、典型的には、各クライアント５４は、例えば、シングルスレッド化及び／又はマルチスレッド化プロセッサを含む１つ又はそれ以上のプロセッサ、ソフトウェアベースの又はオペレーティング・システム支援マルチスレッド化、論理的区画化、及び／又は多数の実行インスタンスが共有リソースに並列に非同期的にアクセスできるようにする他の技術を使用して、コンピュータに常駐する複数の並列処理又はスレッドの１つにおいて実行される、コンピュータ上のプログラム・コードを用いて実装される。

【００２４】

共有リソース５６へのアクセスを管理するために、リソース・マネージャ５２は、種々のクライアント５４によって生成されたリソース・アクセス要求のための個別のサーバとして働く複数のサーバ・プロセス５８を含む。各サーバ・プロセス５８は、所与のクライアント５４に静的に又は動的に割当てられ、サーバ・プロセスとクライアントとの間のマッピングは、必ずしも１：１である必要はなく、時間の経過と共に変化してもよいことがわかるであろう。

【００２５】

リソース・マネージャ５２はまた、共有リソース５６へのアクセスを調整するためにサーバ・プロセス５８と対話するディスパッチ・プロセス６０を含む。以下でより明らかとなるように、ディスパッチ・プロセス６０は、一度に多数のアクセス要求をディスパッチするためにバッチでのディスパッチを実行することができる。さらに、ディスパッチ・プロセス６０はまた、共有リソース５６からのデータのリターンを管理することができ、入来データをしかるべきクライアントに送付するために、サーバ・プロセス５８と、あるいは直接各クライアントと対話することができる。しかしながら、他の実施形態においては、リターン・データは、ディスパッチ・プロセス６０とは別に取り扱われてもよい。さらに、リソース・マネージャ５２は、多数の共有リソースを管理することができ、又は多数の共有リソースを管理するために多数のリソース・マネージャを用いることができる。

【００２６】

本発明に係る共有リソース５６への同時アクセスをサポートするために、リソース・マネージャ５２は、４つのアトミック・カウンタ６２、６４、６６及び６８を含む。カウンタ６２は、ここでは要求カウンタと呼ばれ、クライアント５４の１つによって生成された次のアクセス要求に割当てられるべき識別子を与えるために用いられる。生成された各要求のために、要求カウンタ６２が次の識別子に増分され、識別子は、要求を固有に識別するために、サーバ・プロセスにおけるＭＹ＿ＲＥＱＵＥＳＴ変数７０に格納される。種々の実施形態においては、要求カウンタ６２は、カウンタの現在値が、最後に受け取られた要求を表すか、又は要求が受け取られたときに次の要求に割当てられることになる値を表すように、事前に又は事後に増分される。

【００２７】

カウンタ６４は、ここでは完了カウンタと呼ばれ、完了したアクセス要求の数を追うために用いられる。カウンタ６６は、ここでは最後の完了（「ＬＡＳＴ＿ＤＯＮＥ」）カウンタと呼ばれ、完了した最後の（すなわち最新の）アクセス要求の識別子を維持するために用いられる。典型的には、要求が順番どおりではなく完了することが許される、すなわち、後続する要求が先行する要求より先に完了することが許されるとすれば、カウンタ６４及び６６は両方とも必要とされる。

【００２８】

カウンタ６８は、最後のディスパッチ・カウンタと呼ばれ、ディスパッチ・プロセス６０によってディスパッチされた共有リソースへの最後の（又は最新の）アクセス要求の識別子を格納するために用いられ、典型的にはディスパッチ・プロセスが最後に通知された

最後の完了カウンタ６６の値に設定される。カウンタ６２、６４、６６及び６８を識別するために用いた特定の名称は、単に便宜的に選定されたのであって、本発明の範囲を制限するものではない。

【００２９】

図示された実施形態においては、各カウンタ６２、６４、６６及び６８がゼロに初期化され、カウンタ６２及び６４が１だけ増分され、その結果、各カウンタは、常に増加する整数で常に更新される。しかしながら、本発明に係る要求を固有に識別するために、他の単調な順番番号又は他の識別子を用いることもできることが分かるであろう。さらに、特定の数の要求が処理された後にカウンタがロール・オーバーすることを可能にするカウンタ・サイズを用いることができるが、実行可能な時間内でカウンタがロール・オーバーすることを可能にしないカウンタ・サイズ（例えば、８バイトカウンタ）を用いることによって、処理が実質的に簡単化されることが見出された。また、システムの始動の間に各カウンタを再び初期化して、実際問題として、コンピュータ・アーキテクチャ５０の動作の通常の過程ではカウンタのロール・オーバーが起こらないようにすることが望ましい。

【００３０】

ここで図２に移ると、本発明に係るクライアント要求を処理するのに用いられる例示的なクライアント要求処理ルーチン７２が、かなり詳しく示されている。ルーチン７２は、各要求に応答してサーバ・プロセス５８によって実行され、したがって、多数のクライアントが同時に要求をサブミットするときには、ルーチン７２の多数のインスタンスがあらゆる所与の時点で並列に実行される。

【００３１】

ルーチン７２は、ブロック７４において、カウンタ・ロックを得ることによって、すなわち、カウンタ６２－６８に関連するロック又はセマフォーを得ることによって始まる。図示された実施形態においては、こうしたセマフォーは、全てのカウンタ６２－６８に割当てることができ、又は代替的に、こうしたカウンタの各々に別のロックを与えることができ、或いはカウンタのサブセットに所与のロックを割当てることができる。

【００３２】

ロックが多数のカウンタによって共有されるか、又はカウンタの１つ又はサブセットにのみ割当てられるかどうかに関係なく、ブロック７４においてロックが得られると、制御がブロック７６に移って要求カウンタ６２を増分する。次に、ブロック７８が、要求に関連付けられることになるＭＹ＿ＲＥＱＵＥＳＴ変数７０を要求カウンタ６２の現在値に設定する。その後、ブロック８０がカウンタ・ロックを解除する。その結果、ブロック７４－８０が、要求カウンタにおけるアトミック増分動作を有効に実行し、また、要求カウンタの増分後の値に基づいて要求に固有の要求識別子を割当てる。他の実施形態においては、要求カウンタの増分前の値を用いることができる。さらに、ロックの下で要求カウンタの有効なコピーが得られる限り、ブロック７８のＭＹ＿ＲＥＱＵＥＳＴ変数への値の割当ては、ロックの外部で行われても良い。しかしながら、用いられる方法に関係なく、要求は、複数のクライアントによって発せられた他の要求に対して要求が受け取られる順番に基づいて、固有の識別子を有効に割当てられる。

【００３３】

次に、ブロック８２が、クライアントに代わって要求を準備し、サブミットすることによって要求の処理を完了する。しかしながら、注目すべきは、ブロック８２は、ロック又はセマフォーの下では実行されず、したがって、共有リソースにアクセスする際の対立の結果として現在のプロセス又は別のプロセスを停止させることがある、あらゆる競合の問題を伴わない。要求を準備し、サブミットすることに関連するステップは、典型的には、特定の要求のタイプ及びアクセスされるリソースのタイプに応じて変化することになる。例えば、ブロック８２において、入力／出力アダプタへの要求をサブミットする目的のために、直接メモリ・アクセス（ＤＭＡ）動作を介した伝送のためにネットワーク・フレームを準備するステップといった種々のステップを行うことができる。本発明に係るブロック８２においては、要求を準備し、要求をサブミットし、及び／又は、要求の処理を、例

えば、メモリ割当て、データのコピー又は伝送、プールされたリソースの取得などといった他の方法で完了することに関連する他の動作を実行することもできる。

【００３４】

次に、ブロック８４はまた、カウンタ・ロックが得られるのに必要なだけ待って、カウンタ・ロックを得る。カウンタ・ロックが得られると、制御がブロック８６に移って完了カウンタ６４を増分し、それにより現在の要求の処理の完了を確立する。

【００３５】

次に、ブロック８８で、現在の要求の識別子が最後の完了カウンタ６６に格納された値より大きいかどうかを判別する。大きい場合には、現在の要求は、完了した最新の又は最後の要求であることが分かる。したがって、制御がブロック９０に移って、最後の完了カウンタ６６の値が現在の要求の識別子と同値に設定される。次に、制御がブロック９２に移る。さらに、ブロック８８に戻り、現在の要求の識別子が最後の完了カウンタ６６に格納された値より大きくない場合には、ブロック８８は、制御を直接ブロック９２に移し、それによりブロック９０をバイパスする。

【００３６】

ブロック９２は、典型的には、完了カウンタ６４に格納された値が最後の完了カウンタ６６に格納された値に等しいかどうかを判別することによって、最近完了した要求より前に発せられた他の要求の処理が完了したかどうかを判別する。完了カウンタ６４の値がより小さいときには、前の要求の少なくとも１つがまだ完了していないことを示すので、この条件が満たされているときには、最近完了した要求（カウンタ６６において識別された）より時間が先行する全ての要求が完了したことがわかる。

【００３７】

ブロック９２における条件が満たされた場合には、制御がブロック９４に移って、最後のディスパッチ・カウンタ６８の値が最後の完了カウンタ６６に格納された値と同値に設定され、それは、最後の完了カウンタにおいて識別された要求とそれまでの要求を含む全ての要求が完了したことを示す。次に、制御がブロック９６に移って、全ての今のところまだディスパッチされていない要求（典型的には、最後の完了カウンタ６６の値と、ブロック９４を実行する前の最後のディスパッチ・カウンタ６８に格納された前の値との間の要求）をディスパッチするようにディスパッチ・プロセスに通知する。次に、制御がブロック９８に移って、ブロック８４において得られたカウンタ・ロックを解除し、それによりルーチン７２が完了する。本発明に係る他の実施形態においては、ブロック９６の後にブロック９４が実行されることを認識されたい。

【００３８】

ブロック９２に戻り、全ての先行の要求が完了していない場合には、ブロック９２が制御をブロック１００に移し、典型的には現在の要求の識別子が最後のディスパッチ・カウンタ６８に格納された値より大きいかどうかを判別することによって、現在の要求が、ディスパッチ・プロセスによる要求の最後のディスパッチの後に始められた最初の要求であったかどうかを判別する。

【００３９】

そうである場合には、制御がブロック１０２に移って、カウンタ６８の値が現在の要求識別子と同値に設定される。次に、制御がブロック９６に移って、ディスパッチ・プロセスに現在の要求をディスパッチするように通知する。そうでなければ、ブロック１００に戻り、ブロックにおいて示された条件が満たされない場合には、後で実行される他の作業が要求を最終的にはディスパッチさせるので、このルーチン７２のインスタンスを通して実行されるべき更なる作業は要求されない。したがって、ブロック１００は、制御を直接ブロック９８に移してカウンタ・ロックを解除し、ルーチンを終える。

【００４０】

ブロック１００において行われる決定、及び結果としてのブロック１０２の流れは、幾つかの実施形態においては省略されても良い。上記の機能は、バッチでディスパッチした後に始められた最初の要求の待ち時間を減らすという観点での利点を与える。幾つかの環

境においては、最初の要求の待ち時間は大きな懸念とはならず、そのため上記の機能は、こうした実施形態においては省略されても良い。こうした事例においては、ブロック９２は、そこで示された条件が満たされてない場合には、制御をブロック９８に直接移すことができる。

【００４１】

上記の要求を取り扱う技術は、多数のクライアントによる共有リソースのアクセスを管理する観点で多くの固有の利点を与える。多数のアクセス要求が、要求の生成の観点から同期化又は逐次化を必要とすることなく生成され、保持されることができる。例えば、要求が、制限された数のブロック又はスロットに区画化されたバッファに組み込まれる場合には、クライアントは、こうしたスロットの完全な消費を同期させる必要なしに特定のスロットに同時に割当てられる。さらに、１つの共有リソースに対する多数の要求を、１つずつではなくバッチでディスパッチすることができ、それにより各ディスパッチに関連するオーバーヘッド（例えば、割り込みを介してクライアントに通知することに関連するオーバーヘッド）の多くが排除される。さらに、共有リソースへのアクセスを保護し、管理するのに用いられるロックにおける競合が減少される。ロックは、前述のカウンタの比較的短い更新の間の特定の要求のためにのみかけられ、何らかの同期化機構が得られることを必要とすることなく、そのディスパッチが行われる前の要求の作成及び準備に関連するオーバーヘッドの多くをイネーブルにする。

【００４２】

上述のように、共有リソースへの同時アクセスを管理するための上記の技術は、多様な用途に用いることができる。図３－図５と関連して後で詳しく説明される１つの特定の用途は、図３のコンピュータ１０のような論理的に区画化されたコンピュータにおける物理的又は仮想イーサネット（登録商標）・アダプタへのアクセスを管理することである。特に、イーサネット（登録商標）・アダプタ、トークン・リング・アダプタなどのような入力／出力アダプタは、この事例においてはフレーム・バッファのスロットである有限の数のリソースのみを有することが良く知られている。典型的には、多数のクライアントがスロットを使用することを許される。さらに、スロットの消費は、長時間にわたって該スロットを拘束して、その間、他のクライアントが該スロットを使用できなくなることがある。多数のクライアントが、他のクライアントを妨害することなくこうしたスロットを同時に消費できるようにすることにより、コンピュータ・システムにおけるより有効な並列実行が可能になり、それによりシステム性能が向上する。後述するように、コンピュータ１０は、本発明に係るこうしたスロットの同時消費をサポートする。

【００４３】

コンピュータ１０は、通常は、例えば、ネットワーク・サーバ、中間価格帯のコンピュータ、メインフレーム・コンピュータなど、例えば、ＩＢＭ　ｅＳｅｒｖｅｒ（ｅＳｅｒｖｅｒはＩＢＭの商標）コンピュータといったあらゆる数のマルチユーザ・コンピュータを表す。しかしながら、本発明は、他のコンピュータ及びデータ処理システム、例えば、ワークステーション、デスクトップ・コンピュータ、ポータブル・コンピュータなどといった単一ユーザ・コンピュータ、又は他のプログラム可能電子デバイス（例えば、内蔵コントローラなどを組み入れる）において実装されてもよいことを認識されたい。さらに、本発明はまた、論理的に区画化されていないマルチスレッド化コンピュータと組み合わせて用いることもできる。

【００４４】

図３に最も良く示されるように、コンピュータ１０は、一般に、バス１６を介してメモリ１４に結合された１つ又はそれ以上のプロセッサ１２を含む。各プロセッサ１２は、シングルスレッド化プロセッサとして、又は、複数のハードウェア・スレッド１８を組み入れていることが示されたプロセッサ１２ａのようなマルチスレッド化プロセッサとして実装することができる。大部分は、マルチスレッド化プロセッサ１２ａにおける各ハードウェア・スレッド１８は、コンピュータに常駐するソフトウェアによって、独立プロセッサと同様に処理される。これに関しては、本開示の目的上、シングルスレッド化プロセッサ

は、シングル・ハードウェア・スレッド、すなわち単一の独立型実行ユニットを組み入れると考えられる。しかしながら、コンピュータにおける多数のタスクの並列性能をさらに支援するために、ソフトウェアベースのマルチスレッド化又はマルチタスク化を、シングルスレッド化プロセッサ及びマルチスレッド化プロセッサの両方と組み合わせて用いることができる。

【００４５】

さらにまた、図３に示されるように、１つ又はそれ以上のプロセッサ１２（例えば、プロセッサ１２ｂ）は、システムの初期プログラム・ロード（ＩＰＬ）を管理するために、及びシステム・ハードウェアを監視し、診断し、構成するために特殊ファームウェア・コードを実行するのに用いられるサービス・プロセッサとして実装することができる。一般に、コンピュータ１０は、オペレーティング・システム及びコンピュータに常駐するアプリケーションを実行するのに用いられる１つのサービス・プロセッサと多数のシステム・プロセッサを含むが、本発明はこの特定の実装に限定されない。幾つかの実装においては、サービス・プロセッサは、バス１６を通す以外の形で、コンピュータにおける種々の他のハードウェア・コンポーネントに結合することができる。

【００４６】

メモリ１４は、１つ又はそれ以上のレベルのデータ、命令及び／又は組み合わせキャッシュだけではなく、例えば、ＤＲＡＭベースの主ストレージといった１つ又はそれ以上のレベルのメモリ・デバイスを含むことができ、特定のキャッシュは、当該技術分野では周知のように個々のプロセッサ又は多数のプロセッサと関連して動作する。さらに、メモリ１４は、バス２０、例えば１つ又はそれ以上のネットワーク・アダプタ２２（コンピュータとネットワーク２４をインターフェースするための）、１つ又はそれ以上のストレージ・コントローラ２６（コンピュータと１つ又はそれ以上のストレージ・デバイス２８をインターフェースするための）、及び１つ又はそれ以上のワークステーション・コントローラ３０（複数のワークステーション・アダプタを介して１つ又はそれ以上の端末又はワークステーション３２とインターフェースするための）を介して多くのタイプの外部デバイスに結合される。

【００４７】

図３はまた、区画マネージャ又はハイパーバイザー３６によって管理される複数の論理区画３４を含む、コンピュータ１０上の論理的に区画化されたコンピューティング環境を実装するのに用いられる主ソフトウェア・コンポーネント及びリソースをかなり詳細に示す。当該技術分野では公知のあらゆる数の論理区画をサポートすることができ、あらゆる時点でコンピュータに常駐する論理区画の数は、区画がコンピュータに加えられ、又はコンピュータから除去される際に動的に変化することができる。

【００４８】

図示されたＩＢＭ　ｅＳｅｒｖｅｒベースの実装においては、区画マネージャ３６は、２つのプログラム・コードの層からなる。ここではディスパッチ不可能部分３８と呼ばれる第１の層は、コンピュータ１０のファームウェア又はライセンス内部コード（ＬＩＣ）内で実装され、それは、ハードウェア・アクセスの詳細から例えばオペレーティング・システムのようなより高次の層を分離しながら、種々のハードウェア・コンポーネントとの低レベルのインターフェースを与えるために用いられる。ファームウェアはまた、サービス・プロセッサ１２ｂのようなサービス・プロセッサと通信することができる。ディスパッチ不可能部分３８は、コンピュータ１０のための多くの低レベルの区画管理機能、例えば、ページ・テーブル管理などを与える。ディスパッチ不可能部分３８はまた、タスクの概念をもたず、主に、ソフトウェアのより高次の層からの機能コールを介してアクセス可能である。

【００４９】

区画マネージャ３６におけるプログラム・コードの第２の層は、ここではディスパッチ可能部分４０と呼ばれる。再配置オフの状態で実行され、ソフトウェアのより高次の層からの機能コールを介してアクセス可能な、タスクの概念をもたないディスパッチ不可能部

分３８とは対照的に、ディスパッチ可能部分４０は、（あらゆるオペレーティング・システムと同様に）タスクの概念を有し、再配置オンの状態で実行される。ディスパッチ可能部分は、典型的には、それがユーザから隠れていること以外は、区画とほぼ同じような形で実行される。ディスパッチ可能部分は、一般に、区画の作成及び削除、同時Ｉ／Ｏメンテナンス、プロセッサ、メモリ及び他のハードウェア・リソースの種々の区画３４への割当てといったより高いレベルの区画管理動作を管理する。

【００５０】

各論理区画３４は、典型的には、コンピュータ１０における利用可能なリソースの一部に静的に及び／又は動的に割当てられる。例えば、各論理区画は、１つ又はそれ以上のプロセッサ１２、及び／又は、１つ又はそれ以上のハードウェア・スレッド１８、並びに、利用可能なメモリスペースの一部に割当てられる。論理区画は、プロセッサのような特定のハードウェア・リソースを共有して、所与の１つのプロセッサが、１つより多い論理区画によって用いられるようにすることができる。別のハードウェアにおいては、リソースは、一度に１つだけの論理区画に割当てられることができる。

【００５１】

付加的なリソース、例えば、大容量ストレージ、バックアップ・ストレージ、ユーザ入力、ネットワーク接続、したがってＩ／Ｏアダプタは、典型的には、当該技術分野では周知の形で１つ又はそれ以上の論理区画に割当てられる。リソースは、例えば、１つ１つのバスに基づいて、又は１つ１つのリソースに基づいて、といった多くの形で割当てられ、多数の論理区画は、同じバスにおけるリソースを共有する。幾つかのリソースは、一度に多数の論理区画に割当てられることもできる。

【００５２】

各論理区画３４は、区画化されていないコンピュータのオペレーティング・システムと同じ形で論理区画の主動作を制御するオペレーティング・システム４２を用いる。例えば、各オペレーティング・システム４２は、インターナショナル・ビジネス・マシーンズ社から入手可能なＯＳ／４００オペレーティング・システムを用いて実装することができる。

【００５３】

各論理区画３４は、別々の、すなわち独立したメモリスペースにおいて実行され、それにより各論理区画は、そうした各論理区画において実行される各ユーザ・アプリケーション（ユーザアプ）４４の見地から、独立した区画化されていないコンピュータとほぼ同様に働く。したがって、ユーザ・アプリケーションは、典型的には、区画化環境において用いるためのあらゆる特別な構成を必要としない。

【００５４】

別々の仮想コンピュータとしての論理区画３４の性質から、論理区画があたかも別の物理マシン上にあるようにして別の論理区画と通信することを可能にする区画間通信をサポートすることが望ましい。したがって、幾つかの実装においては、ディスパッチ不可能部分３８において、論理区画３４がイーサネット（登録商標）・プロトコルのようなネットワーキング・プロトコルを介して別の論理区画と通信することを可能にする仮想ローカル・エリア・ネットワーク（ＬＡＮ）４６をサポートすることが望ましい。区画間の通信をサポートする別の方法も、本発明に従ってサポートすることができる。

【００５５】

本発明に従って他の論理的に区画化された環境を用いることもできることが認識されるであろう。例えば、あらゆる区画３４から分離されたディスパッチ可能部分４０を用いるのではなく、ディスパッチ可能部分の機能を、１つ又はそれ以上の論理区画に代替的に組み入れてもよい。

【００５６】

一般に、本発明の実施形態を実装するために実行されたルーチンは、オペレーティング・システム又は特定のアプリケーション、コンポーネント、プログラム、オブジェクト、モジュール又は命令のシーケンス、或いはそのサブセットの一部のいずれとして実装され

るかに関係なしに、ここでは「コンピュータ・プログラム・コード」又は単に「プログラム・コード」と呼ばれる。プログラム・コードは典型的に、コンピュータの種々のメモリ及びストレージ・デバイスに種々の時点で常駐する１つ又はそれ以上の命令を含み、それは、コンピュータの１つ又はそれ以上のプロセッサによって読み取られ、実行されたときに、本発明の種々の態様を具体化するステップ又は要素を実行するのに必要なステップをコンピュータに実行させる。さらに、本発明は、満足に機能するコンピュータ及びコンピュータ・システムとの関連で以下に説明されるが、本発明の種々の実施形態は、種々の形態のプログラム製品として配布されることができ、そして、本発明は、実際に配布を行うのに用いられる信号伝達媒体の特定のタイプに関係なく等しく適用されることを当業者であれば認識するであろう。信号伝達媒体の例は、この限りではないが、揮発性及び不揮発性メモリ・デバイス、ディスケットその他のリムーバル・ディスク、ハードディスクドライブ、磁気テープ、光ディスク（例えば、ＣＤ－ＲＯＭ、ＤＶＤなど）などのような書き込み可能型媒体、並びに、ディジタル及びアナログ通信リンクのような伝送型媒体を含む。

【００５７】

さらに、以下に説明される種々のプログラム・コードは、本発明の特定の実施形態において実装されるアプリケーション又はソフトウェア・コンポーネントに基づいて識別することができる。しかしながら、つけられたあらゆる特定のプログラム名は、単に便宜のために用いられたのであり、したがって本発明は、そうした名称によって識別され及び／又は示される何らかの特定のアプリケーションにおける使用にのみ限定されるものではないと認識されたい。さらに、コンピュータ・プログラムを、ルーチン、プロシージャ、メソッド、モジュール、オブジェクトなどに構成する無数の典型的な方法、並びに、典型的なコンピュータ内に常駐する種々のソフトウェア層（例えば、オペレーティング・システム、ライブラリ、ＡＰＩ、アプリケーション、アプレットなど）の間にプログラム機能を割当てる種々の方法があり、本発明は、ここで説明されたプログラム機能の特定の構成及び割当てに限定されるものではないと認識されたい。

【００５８】

図３に示された例示的な環境は、本発明を制限することを意図されたものではないことを当業者は理解するであろう。実際は、本発明の範囲から逸脱することなく、他の代替的なハードウェア及び／又はソフトウェア環境を用いることができることを当業者は理解するであろう。具体的には、本発明は、論理的に区画化された環境における使用に限定されるものではなく、共有リソースへの同時及び非同期アクセスをサポートする多くの他の環境において用いることができる。

【００５９】

ここで図４に移ると、ここでは仮想入力／出力アダプタにおけるフレーム・バッファである特定のタイプの共有リソースへの同時アクセスを実行することに関連する主ソフトウェア・コンポーネントが示されている。この実施においては、複数のクライアント１２０は、仮想ローカル・エリア・ネットワーク（ＶＬＡＮ）を介して区画３４間の区画間通信をサポートする目的のために、種々の区画３４におかれる。図示された実施形態においては、各クライアントは、例えば、区画に常駐するデバイス・ドライバとして実装することができる。

【００６０】

区画マネージャ３６内には、複数の仮想入力／出力アダプタ（ＩＯＡ）１２２が配置され、それらは、ＶＬＡＮ上での通信のためにクライアント１２０によってアクセスされることができる。こうした構成の下では、１つの区画のクライアントが別の区画のクライアントにデータを通信する必要があるときにはいつも、データは、非仮想ＩＯＡにデータが転送されるのとほぼ同じように、区画マネージャ３４において送信側クライアントから送信側クライアントに割当てられた仮想ＩＯＡ１２２に転送される。データは、仮想ＩＯＡによって従来のネットワーク・パケット（例えば、イーサネット（登録商標）互換パケット）と同様にフォーマットされ、異なる区画３４の受信側クライアント１２０に割当てら

れる別の仮想ＩＯＡ１２２に通信される。このデータ通信は、物理的ＬＡＮにおいて用いられるような物理的ケーブル上ではなくソフトウェアにおいて行われるが、その他の点では、使用される通信プロトコルは同じである。次に、受信側クライアントに割当てられた仮想ＩＯＡ１２２が、受信したデータを、非仮想ＩＯＡとほぼ同じようにして受信側クライアントに渡す。したがって、各クライアント１２０の見地から、データが或るクライアントによって送信され、別のクライアントによって受信される方法は、データが物理的ＬＡＮ上で通信されるのと同じである。

【００６１】

この方法で仮想ＬＡＮを実装することによって、あたかも他の区画は異なる物理的コンピュータ上に常駐するかのようにして、或る区画において実行されるアプリケーションが別の区画の別のアプリケーションにデータを通信するので、論理的に独立したコンピュータのような区画の論理構成が維持される。多くの事例においては、これは、論理的に区画化されていないコンピュータ上で実行されて、論理的に区画化されたコンピュータとまさに同じように動作するように構成されたアプリケーションを可能にする。

【００６２】

各仮想ＩＯＡ１２２は、１つ又はそれ以上のフレーム・バッファ１２４と関連し、それらは、非仮想ＩＯＡと同様にイーサネット（登録商標）と互換性のあるデータ・フレームを送受信するのに用いられる。各フレーム・バッファ１２４は、バッファにおける別個のデータエリアを表す複数のスロット１２６を含む。各スロット１２６はまた、関連するクライアント１２０によって用いられるために、区画におけるローカル・バッファ（例えばバッファ１２８）へのポインタを含む。

【００６３】

図示された実施形態においては、フレーム・バッファ１２４におけるスロット１２６は、種々のクライアント１２０によって順番に用いられることができる共有リソースを表す。多数のクライアント１２０がスロットを消費することができ、スロットは要求の処理の間にクライアントによって消費されるので、これらのリソースが消費される際にクライアントに通知する必要がある。図示された実施形態においては、例えば、仮想ＩＯＡからクライアントに割り込みを発行することによってクライアントに通知される。以下でより明らかとなるように、本発明によれば、これらの試みは、要求ごとにではなくバッチで取り扱って、ディスパッチ・プロセスの効率を改善することができる。

【００６４】

図４に示された実施形態においては、各仮想ＩＯＡ１２２は、図１のリソース・マネージャ５２と類似したリソース・マネージャとして動作する。したがって、各仮想ＩＯＡ１２２は、複数のサーバ・プロセス１３０、ディスパッチ・プロセス１３２、要求カウンタ１３４、完了カウンタ１３６、最後の完了カウンタ１３８、及び最後のディスパッチ・カウンタ１４０を含む。さらに、各サーバ・プロセス１３０は、関連するサーバ・プロセスによって発せられた各要求を固有に識別するために、ＭＹ＿ＲＥＱＵＥＳＴ変数１４２を格納する。本発明に係るリソース管理は、典型的には、仮想ＬＡＮからのデータの受信と関連して起こり、結果として、クライアント１２０は、データが関連する仮想ＩＯＡによって受信される際の割り込みを介して通知される。

【００６５】

ここで図５に移ると、図３及び図４の環境のクライアント要求を取り扱うのに特に適したクライアント要求処理ルーチン１７２が、かなり詳しく示されている。ルーチン１７２は、多くの要求においてルーチン７２と類似しているが、特に、仮想入力／出力アダプタのフレーム・バッファのスロットを消費するためのクライアント要求を取り扱うように構成されている。一般に、各クライアント要求は、フレーム・バッファにおける利用可能なスロットをアトミックに取得し、新しい固有の識別子が、要求カウンタをアトミックに増分することによってこうした要求に割当てられる。フレーム・バッファにおけるスロットの数を法とする要求識別子は、使用されるべき特定のスロットを得るために用いられる。この実装のためには、ここでは使用済みスロットカウンタと呼ばれる付加的なカウンタが

、消費されたスロットの総数を維持するために用いられる。このカウンタは、クライアントがリソースを解放するときにはいつも、典型的にはルーチン１７２以外のルーチンに実装された機能を介して減分される。

【００６６】

ルーチン１７２は、使用済みスロットの総数が利用可能なスロットの最大数（総スロット定数又は変数によって表される）に等しいかどうかを判別することによって、ブロック１７４において始まる。等しい場合には、クライアントの要求が不履行となり、ブロック１７６に示されるようにエラーが戻される。そうでなければ、ブロック１７４は、制御をブロック１７８に移して、使用済みスロットの数を増分させる。次に、ブロック１８０、１８２、１８４及び１８６は、要求カウンタをアトミックに増分し、ルーチン７２のブロック７４－８０と同様にして要求識別子を取得する。

【００６７】

次に、制御はブロック１８８に移って、フレーム・バッファにおけるスロットの総数を法とする現在の要求識別子をとることによって、現在の要求のための現在のスロット（ＭＹ＿ＳＬＯＴ）を設定する。次に、ブロック１９０において、現在のスロットへの要求が準備され、サブミットされ、それにより要求の処理が完了する。したがって、制御がブロック１９２に移って、カウンタ・ロックが取得され、次いで、ブロック１９４に移って、ルーチン７２のブロック８４及び８６と同じようにして完了カウンタを増分させる。特に、ブロック１９６、１９８、２００、２０２、２０４、２０６、２０８及び２１０に示されるようなルーチン１７２における残りの動作は、ルーチン７２のブロック８８、９０、９２、９４、９６、９８、１００及び１０２と関連して上記で開示されたのと同一であり、ブロック２０４における要求のディスパッチは、ディスパッチ・プロセスへの割り込みを介して起こり、通知される必要がある区画における種々のクライアントに付加的な割り込みを生成する。さらに、本発明に係る幾つかの実施形態においては、ルーチン７２のブロック１００及び１０２と同様に、ブロック２０８及び２１０は、ルーチン１７２から省略することができる。

【００６８】

上記の実装における本発明の使用の実際的な例として、４つの論理区画が仮想ＬＡＮ上で互いに結合され、仮想ＩＯＡが各論理区画に割当てられ、３つの論理区画が第４の区画に送信される必要があるデータを有するケースを考える。第１の区画が送信されるべき６４ＫＢのデータを有し、第２の区画が送信されるべきたったの２ＫＢのデータを有し、第３の区画が送信されるべき６４ＫＢのデータを有すると考える。このケースでは、第１、第２及び第３の区画は、それぞれの仮想ＩＯＡに要求を発し、そして、これらの仮想ＩＯＡが、第４の区画の仮想ＩＯＡに送信されるべきデータのフレームをフォーマットするとみなす。さらに、仮想ＩＯＡが、番号順に、すなわち第１、第２及び第３の区画からの要求が第４の区画の仮想ＩＯＡによって順番に受信されるように、第４の区画の仮想ＩＯＡに要求を発する。したがって、ルーチン１７２を用いて、要求に順番に固有の識別子が割当てられ、第４の区画の仮想ＩＯＡのフレーム・バッファのスロットが順番に消費される。

【００６９】

しかしながら、要求に識別子が割当てられると、仮想ＩＯＡにおける各サーバ・プロセスは、例えば、そのプロセスによって消費されるスロットによって示されるクライアント・バッファへのそれぞれのフレームのＤＭＡのために、その要求の取り扱いを並列に進める。しかしながら、図示された例においては、第２の区画からのデータ・フレームは第１の区画のデータ・フレームよりかなり小さい（２ＫＢｖｓ．６４ＫＢ）ので、第１の区画の要求が完了する前に第２の区画の要求が完了することが起こりうる。そうした場合、ルーチン１７２は、完了カウンタを増分させ、最後の完了カウンタを、第２の区画からの要求の固有の識別子に設定する。しかしながら、ルーチンはまた、ブロック２００において、全ての先行の要求が完了したわけではないことに気付き、その結果、第１の区画からの要求が完了するまで第４の区画のクライアントへの通知を据え置くことになり、この時点

でバッチでディスパッチが行われて、両方のデータ・フレームの受信が第４の区画に通知されることになる。さらに、第３の区画からの要求がかなり遅れて完了する場合には、別のディスパッチが起こって、第４区画に第３データ・フレームの受信が通知される。

【図面の簡単な説明】

【００７０】

【図１】本発明に係る形で複数のクライアントによる共有リソースへの同時アクセスをサポートする汎用コンピュータ・アーキテクチャのブロック図である。

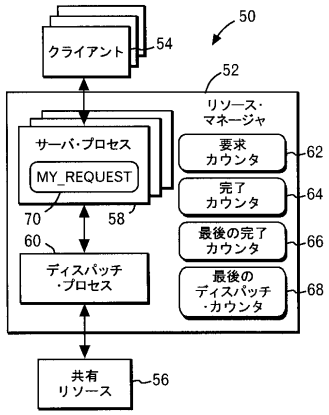
【図２】クライアント要求を処理するために図１のコンピュータ・アーキテクチャのサーバ・プロセスによって実行されるクライアント要求処理ルーチンのプログラム・フローを示すフローチャートである。

【図３】本発明に係る共有リソースの同時アクセスを実行する論理的に区画化されたコンピュータの基本ハードウェア・コンポーネントのブロック図である。

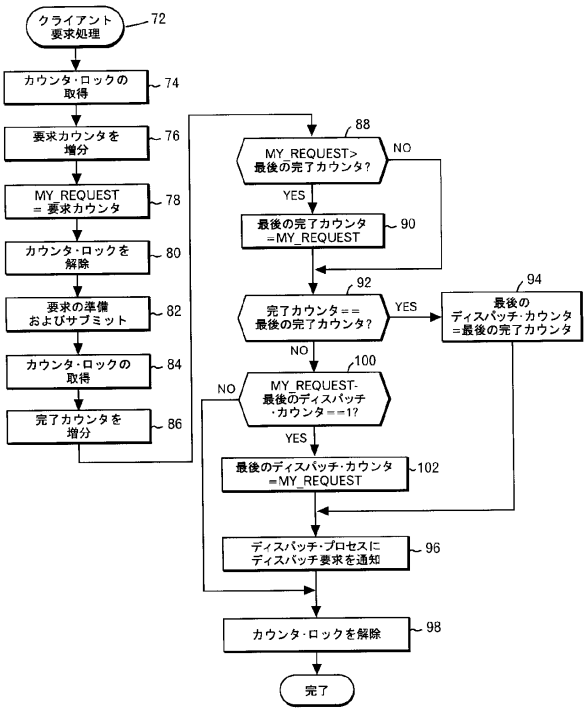
【図４】コンピュータの共有リソースへの同時アクセスを管理することに関係する図３のコンピュータの基本コンポーネントのブロック図である。

【図５】図４に関連するサーバ・プロセスによって実行されるクライアント要求処理ルーチンのプログラム・フローを示すフローチャートである。

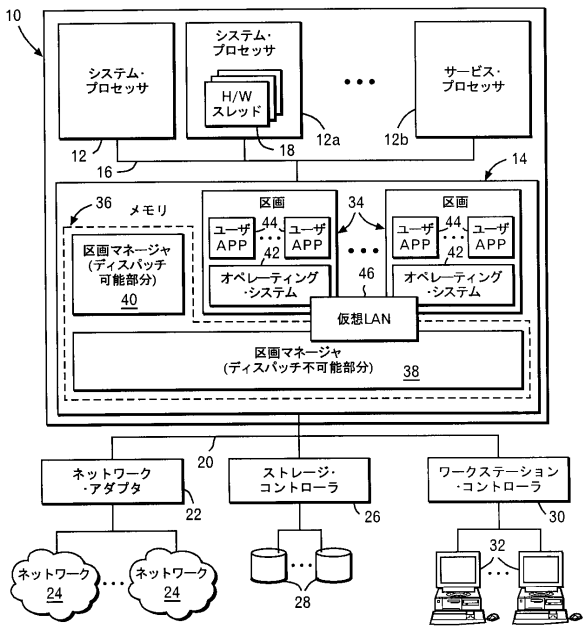
【図１】



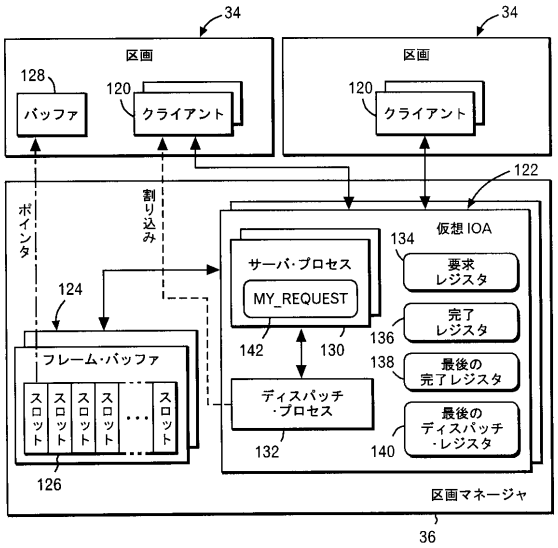
【図２】



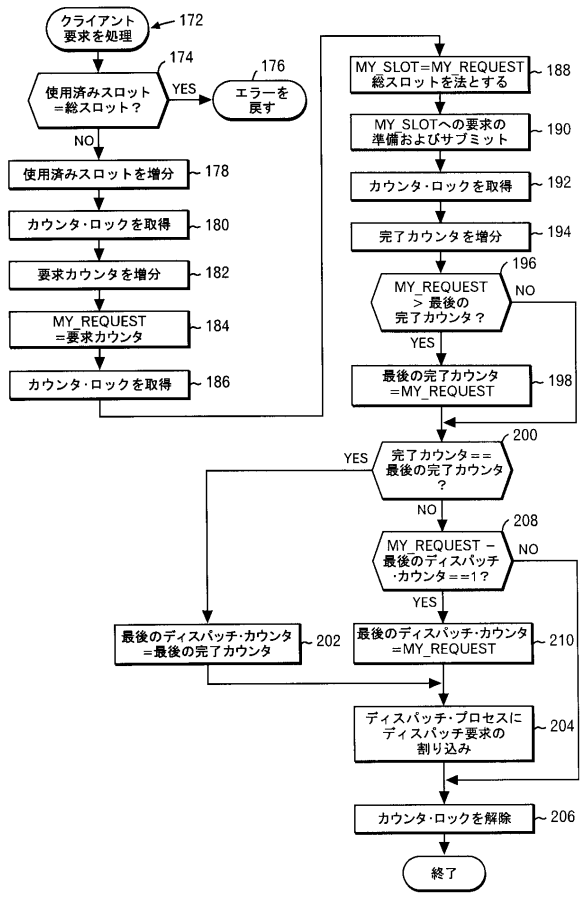
【図３】


10
システム・プロセッサ
システム・プロセッサ
H/W スレッド
サービス・プロセッサ
12
16
18
12a
12b
14
36
メモリ
区画
区画
34
ユーザ APP
ユーザ APP
ユーザ APP
ユーザ APP
44
42
オペレーティング・システム
オペレーティング・システム
46
区画マネージャ（ディスパッチ可能部分）
40
仮想LAN
区画マネージャ（ディスパッチ不可能部分）
38
20
ネットワーク・アダプタ
22
ストレージ・コントローラ
26
ワークステーション・コントローラ
30
32
ネットワーク
24
ネットワーク
24
28

【図４】


34
34
区画
区画
128
バッファ
120
クライアント
120
クライアント
122
ポインタ
割り込み
仮想 IOA
134
サーバ・プロセス
MY_REQUEST
要求レジスタ
完了レジスタ
136
最後の完了レジスタ
138
124
フレーム・バッファ
スロット
スロット
スロット
スロット
スロット
142
130
ディスパッチ・プロセス
最後のディスパッチ・レジスタ
140
132
126
区画マネージャ
36

【図５】


クライアント要求を処理
172
使用済みスロット＝総スロット？
174
YES
176
エラーを戻す
NO
使用済みスロットを増分
178
カウンタ・ロックを取得
180
要求カウンタを増分
182
MY_REQUEST＝要求カウンタ
184
カウンタ・ロックを取得
186
MY_SLOT=MY_REQUEST 総スロットを法とする
188
MY_SLOTへの要求の準備およびサブミット
190
カウンタ・ロックを取得
192
完了カウンタを増分
194
MY_REQUEST ＞ 最後の完了カウンタ？
196
NO
YES
最後の完了カウンタ＝MY_REQUEST
198
YES
完了カウンタ＝＝最後の完了カウンタ？
200
NO
MY_REQUEST − 最後のディスパッチ・カウンタ＝＝1？
208
NO
YES
最後のディスパッチ・カウンタ＝最後の完了カウンタ
202
最後のディスパッチ・カウンタ＝MY_REQUEST
210
ディスパッチ・プロセスにディスパッチ要求の割り込み
204
カウンタ・ロックを解除
206
終了

フロントページの続き

(74)代理人 100086243
弁理士 坂口 博
(72)発明者 アームストロング、トロイ、デビッド
アメリカ合衆国 ５５９０２ ミネソタ州 ロチェスター アシュレイ・レーン・サウスウェスト １３１０
(72)発明者 リュッケ、カイル、アラン
アメリカ合衆国 ５５９０６ ミネソタ州 ロチェスター キャシディ・ドライブ・ノースイースト ３０２３

審査官 鈴木 修治

(56)参考文献 特開２０００－０９００５７（ＪＰ，Ａ）
特開２００１－０１４１７８（ＪＰ，Ａ）
特開平１１－３２７９３１（ＪＰ，Ａ）
特開平０９－１９８２６５（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 9/46-9/54