(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4596488号
(P4596488)
(45)発行日　平成22年12月8日(2010.12.8)　(24)登録日　平成22年10月1日(2010.10.1)

(51)Int.Cl.　F I
G06F 11/14 (2006.01)　G06F 11/14 310K

請求項の数 17 (全 17 頁)

(21)出願番号	特願2007-537272 (P2007-537272)	(73)特許権者	501125231
(86)(22)出願日	平成17年10月19日 (2005.10.19)		ローベルト　ボッシュ　ゲゼルシャフト　ミット　ベシュレンクテル　ハフツング
(65)公表番号	特表2008-518292 (P2008-518292A)		ドイツ連邦共和国　７０４４２　シュトゥットガルト　ポストファッハ　３０　０２２０
(43)公表日	平成20年5月29日 (2008.5.29)		
(86)国際出願番号	PCT/EP2005/055390		
(87)国際公開番号	WO2006/045733		
(87)国際公開日	平成18年5月4日 (2006.5.4)	(74)代理人	100095957
審査請求日	平成19年4月24日 (2007.4.24)		弁理士　亀谷　美明
(31)優先権主張番号	102004051967.6	(74)代理人	100096389
(32)優先日	平成16年10月25日 (2004.10.25)		弁理士　金本　哲男
(33)優先権主張国	ドイツ(DE)	(74)代理人	100101557
			弁理士　萩原　康司

最終頁に続く

(54)【発明の名称】コンピュータプログラムを処理する方法、駆動システム、および計算装置

(57)【特許請求の範囲】

【請求項１】

計算装置（２０）上でコンピュータプログラム（２３）を実行する方法であって、

前記コンピュータプログラム（２３）が複数のプログラムオブジェクトを有し、前記計算装置（２０）上で前記コンピュータプログラム（２３）を実行中にエラーが検出され、

前記プログラムオブジェクトが少なくとも２つのクラスに区分され、前記エラーが検出された場合に第１のクラスのプログラムオブジェクトの実行が繰り返され、第２のクラスのプログラムオブジェクトの実行が繰り返されず、

実行のために供給された前記第１のクラスのプログラムオブジェクト内でエラーが検出された場合に、前記第１のクラスのプログラムオブジェクトが所定の状態に移行されて前記所定の状態から再開され、後続する前記第１のクラスの他のプログラムオブジェクトの実行がシフトされることを特徴とする、コンピュータプログラムを実行する方法。

【請求項２】

再開されるクラスのプログラムオブジェクトについてのみエラーが検出されることを特徴とする、請求項１に記載のコンピュータプログラムを実行する方法。

【請求項３】

すべてのプログラムオブジェクトのためのランスルーに全計算時間が設定され、エラーが検出された場合に再開される前記第１のクラスのプログラムオブジェクトがエラーのないランスルーで前記全計算時間の最大５０％を得るように、前記全計算時間が区分されることを特徴とする、請求項１に記載のコンピュータプログラムを実行する方法。

【請求項４】
前記異なるクラスのプログラムオブジェクトが交互に実行されることを特徴とする、請求項１に記載のコンピュータプログラムを実行する方法。
【請求項５】
エラーを伴う前記第１のクラスのプログラムオブジェクトが、前記第１のクラスのプログラムオブジェクトに続く前記第２のクラスのプログラムオブジェクトの代りに実行されることを特徴とする、請求項１に記載のコンピュータプログラムを実行する方法。
【請求項６】
前記プログラムオブジェクトが前記コンピュータプログラム（２３）のタスクとして形成され、エラーが検出された場合に少なくとも１つのタスクが再び実行されることを特徴とする、請求項１に記載のコンピュータプログラムを実行する方法。
【請求項７】
エラーが検出された時に実行されていたプログラムオブジェクトが再び実行されることを特徴とする、請求項１または６に記載のコンピュータプログラムを実行する方法。
【請求項８】
前記プログラムオブジェクトの実行中に、前記プログラムオブジェクトの少なくとも１つの所定の状態が生成されて記憶されることを特徴とする、請求項１に記載のコンピュータプログラムを実行する方法。
【請求項９】
前記プログラムオブジェクトの実行開始時に、前記プログラムオブジェクトの少なくとも１つの所定の状態が生成されて記憶されることを特徴とする、請求項８に記載のコンピュータプログラムを実行する方法。
【請求項１０】
前記計算装置（２０）が自動車内で使用されることを特徴とする、請求項１に記載のコンピュータプログラムを実行する方法。
【請求項１１】
前記計算装置（２０）が自動車制御装置内で使用されることを特徴とする、請求項１０に記載のコンピュータプログラムを実行する方法。
【請求項１２】
前記コンピュータプログラム（２３）の実行中で前記プログラムオブジェクトの実行前に、前記プログラムオブジェクトの実行に必要な変量の値が記憶されることを特徴とする、請求項１に記載のコンピュータプログラムを実行する方法。
【請求項１３】
前記コンピュータプログラム（２３）が期間内で周期的に実行される場合において、
エラーが検出された場合に、前記コンピュータプログラム（２３）の前記期間内の予め設定可能なジャンプバック点の特定のプログラムオブジェクトにジャンプバックされることを特徴とする、請求項１に記載のコンピュータプログラムを実行する方法。
【請求項１４】
計算装置（２０）上で実行可能な駆動システム（２５）であって、
前記駆動システム（２５）が請求項１から１３のいずれか１項に記載の方法を実行するためにプログラミングされ、
前記計算装置（２０）上で実行された場合に請求項１から１３のいずれか１項に記載の方法を実行することを特徴とする、駆動システム。
【請求項１５】
複数のプログラムオブジェクトを有するコンピュータプログラム（２３）を実行する計算装置（２０）であって、
前記計算装置（２０）が、前記計算装置（２０）上で前記コンピュータプログラム（２３）の実行中にエラーを検出するエラー検出機構を有し、
前記コンピュータプログラム（２３）が複数のプログラムオブジェクトを有し、前記計算装置（２０）上で前記コンピュータプログラム（２３）を実行中に前記エラーが検出さ

れ、
前記プログラムオブジェクトが少なくとも２つのクラスに区分され、前記エラーが検出された場合に第１のクラスのプログラムオブジェクトの実行が繰り返され、第２のクラスのプログラムオブジェクトの実行が繰り返されず、
前記計算装置（２０）は、実行のために供給された前記第１のクラスのプログラムオブジェクト内でエラーが検出された場合に、前記第１のクラスのプログラムオブジェクトを所定の状態に移行して前記所定の状態から再開し、かつ後続する前記第１のクラスの他のプログラムオブジェクトの実行をシフトするエラー処理機構を有することを特徴とする、計算装置。
【請求項１６】
前記エラー処理機構は、前記エラーが検出された場合に前記第１のクラスの少なくとも１つのプログラムオブジェクトを再開するトリガー論理を有していることを特徴とする、請求項１５に記載の計算装置。
【請求項１７】
前記計算装置（２０）上で、リアルタイム駆動システム（２５）が実行されることを特徴とする、請求項１５に記載の計算装置。
【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、計算装置上、特にマイクロプロセッサ上でコンピュータプログラムを処理する方法に関する。本発明に係るコンピュータプログラムは、複数のプログラムオブジェクトを有している。また、本発明に係るコンピュータプログラムを処理する方法は、計算装置上でコンピュータプログラムを処理する間にエラーを検出することができる。
【０００２】
本発明は、さらに、計算装置上、特にマイクロプロセッサ上で遂行可能な、駆動システムに関する。
【０００３】
そして、本発明は、複数のプログラムオブジェクトを有するコンピュータプログラムを処理する計算装置に関する。本発明に係る計算装置は、計算装置上でコンピュータプログラムを処理する間にエラーを検出するためのエラー発見機構を有している。
【背景技術】
【０００４】
計算装置上でコンピュータプログラムを処理する場合に、いわゆる過渡的エラーがもたらされることがある。ここで、過渡的エラーは、半導体モジュール（いわゆるチップ）上の構造が微細化されるのに対して、信号のクロックレートが増大し、また、信号の電圧は低下されることによって、より頻繁に生じるようになる。また、過渡的なエラーは、恒久的なエラーとは異なり、一時的にだけ発生し、通常はいくらかの時間が経過した後に自ら再び消えてしまう。過渡的なエラーにおいては、個々のビットが歪曲されるだけで、計算装置自体が恒久的に損傷されることはない。過渡的エラーは、例えば、電磁的な影響、アルファ粒子または中性子のような、種々の原因により生じる可能性がある。
【０００５】
通信システムにおいては、今日すでに、エラー処理における重点は、過渡的なエラーにある。通信システムにおいて（例えば、コントローラエリアネットワーク；ＣＡＮにおいて）、エラーが検出された場合には、誤りをもって仲介されたデータを新たに送信することが知られている。さらに、通信システム内でエラーカウンタを使用することが知られており、そのエラーカウンタはエラーが検出された場合に増分され、正しく送信された場合には減分され、所定の値を上回るとすぐに、データの送信が停止される。
【０００６】
しかしながら、コンピュータプログラムを処理する計算装置においては、エラー処理は、大体において、恒久的なエラーについてのみ行われる。過渡的エラーの考慮は、エラー

カウンタの増分と、場合によっては減分とに限定される。これが、メモリ内に格納されて、オフラインで、すなわち、例えば、計算装置が自動車制御装置として形成されている場合に、修理工場滞在の間に、診断情報またはエラー情報として読み出すことができる。その場合に初めて、それに応じてエラーに反応することができる。

【０００７】

したがって、従来のエラーカウンタによるエラー処理は、１つには、特に安全上重要なシステムのために必要な短いエラー許容時間内のエラー処理を許さず、他方では、エラー許容時間内にコンピュータプログラムが再び秩序通りに処理されるような、構造的なエラー処理も許さない。従来技術においては、その代りに、コンピュータプログラムはエラーカウンタが所定の値を上回った場合に、例えば、非常走行駆動に切り替えられる。これは、コンピュータプログラムにおけるエラーのある部分の代りに、他のものが処理されて、そのようにして得られた代替値がそれ以降の処理に利用されることを意味している。代替値は、例えば、他の変量を用いてモデリングすることができる。代替的に、コンピュータプログラムのエラーのある部分によって計算された結果を、誤っているとして捨て去り、それ以降の計算のために、例えば、非常走行駆動のために設けられている標準値に代えることができる。したがって、計算装置上で遂行されるコンピュータプログラムの過渡的なエラーを処理する既知の方法は、多くのエラーの過渡的な性質と系統的、構造的につきあうことを許さない。

【０００８】

従来技術からは、さらに、計算装置上でコンピュータプログラムを処理する場合に、発生する過渡的エラーに、計算装置の完全な新規スタートによって対処することが知られている。この解決も、実際には満足のゆくものではない。というのは、コンピュータプログラムの処理のそれまでの推移において得られた変量が失われてしまい、計算装置は新規スタートの期間の間その規定通りの機能を満たすことができないからである。これは特に、安全上重要なシステムにおいて、受け容れられない。

【０００９】

また、計算装置上で処理されるコンピュータプログラムの過渡的エラーのためのエラー処理として、コンピュータプログラムを数クロック戻してコンピュータプログラムの個々の機械指令を繰り返すことも知られている。この方法は、ミクロロールバックとも称される。この既知の方法において、オブジェクトだけ機械レベル（クロック、機械指令）へジャンプバックされる。ここで上記ミクロロールバックは、機械レベル上のそれに応じたハードウェア支援を必要とし、それが計算装置の領域における著しい手間および費用と結びついている。また、純粋にソフトウェアによって制御される、既知の方法の実施は、実質的に不可能である。

【００１０】

したがって、上述したように、従来技術から知られたエラー処理機構は、計算装置上でコンピュータプログラムを処理する場合に発生する過渡的エラーに、適切な方法で反応することはできない。

【００１１】

しかしながら、過渡的エラーは、まさに未来のテクノロジーにおいて極めて頻繁に生じうる。したがって、例えば、デュアルコア機構を介して、それを発見した場合に、正しい結果を識別するためには、エラー位置測定の問題に応えることが必要となる。特に、過渡的エラーが生じたとしても、計算器の新規スタートを行わないこと目標とする場合には、エラー位置測定を行うことが必要となる。エラー位置測定は、上述したように、典型的には、比較的複雑な方法を介してのみ達成される。

【発明の開示】

【発明が解決しようとする課題】

【００１２】

本発明の課題は、計算システム内でコンピュータプログラムを処理する際に過渡的エラーが発生した場合において、それを構造的に、できるだけ短いエラー許容時間内に処理し

、計算器システムの機能を回復させ、および機能安全性を再形成することが可能な、新規かつ改良されたコンピュータプログラムを処理する方法、駆動システムおよび計算装置を提供することにある。

【課題を解決するための手段】

【００１３】

この課題を解決するために、冒頭で挙げた種類の方法に基づいて、エラーが検出された場合に、すでに処理へ供給されている少なくとも１つのプログラムオブジェクトが定められた状態へ移行され、この状態から新たに開始されることが、提案される。

【００１４】

もちろんシステムレベル上で、さらに、どのようにしてタスク繰返しのこの種のコンセプトをきちんと使用することができるか、という問題が生じる。これは、通常、任意の誤りのあるタスクを単純に再度計算できる、というわけにはいかない。というのは、付加的に必要な計算時間と、そのために使用される時点も、システムの観点から別に用途が計画されているからである。プロセッサの負担がすでにほぼ１００％である（そして通常そうである）場合に、この種の計画されない付加時間（クロック繰返しのような）によって、システムの過負荷が生じ、それが典型的なシステムの崩壊をもたらす可能性がある。これは、例えば、少なくとも部分的に自らを押し通す、時間制御されるシステムを考える場合に、さらにはっきりとする。上記のような時間制御されるシステムにおいては、デッドライン侵害は、多くの他のリアルタイムコンセプトの場合と同様に、許容できない。

【００１５】

したがって結果として、システムの視点から、潜在的なタスク繰返しによって生じる可能性のある、付加的な負荷を計算に入れなければならない。ここで、各タスクの後に、タスク繰返しを必要とする計算時間を見込む場合には、確実に機能し得るが、エラー処理しないシステム１００％と比較すると、付加的なパフォーマンスの代償を支払わなければならない。したがって、コストの観点からは、受け容れられない。

【００１６】

したがって本発明の課題は、さらに、例えば、タスクの二倍の計算をもはや計画に入れず（そしてそれに伴って恒久的かつ非常に大きいオーバーヘッドを発生させず）、同時にどのようにして時間制御される考えと結合するか、の問題を解決する、最適なシステムストラテジーを提供することである。

【００１７】

したがって、本発明に基づくシステムストラテジーの枠内で、最小限のパフォーマンスオーバーヘッドで、あるいはパフォーマンスオーバーヘッドなしで、タスク繰返しのコンセプトを組み込むことを許容する、コンピュータプログラムを処理する方法、駆動システムおよび計算装置が提案される。その場合にこの種のシステムストラテジーは、タスク、課題、プログラムまたは、以下においてプログラムオブジェクトと称する、プログラム部分のスケジューリング方法のための、周辺条件である。

【００１８】

新たに開始されるプログラムオブジェクトは、エラーが検出された時点で、完全に処理されている必要はない。本発明の主旨において、エラー検出の時点でまだ完全に処理されていないが、その処理がすでに開始されているプログラムオブジェクトも、エラーが発生した場合に新たに開始させることができる。したがって本発明によれば、過渡的または恒久的なエラーが発生した場合に、少なくとも１つの駆動システムオブジェクトが新たに実施される。ミクロロールバックに対する利点は、特に、プログラムオブジェクトの繰返しを極めてわずかなハードウェア支援で実現することができることにある。プログラムオブジェクトの新たな実施のために必要な情報（例えば、プログラムオブジェクトの入力量）を格納することができるようにするために、付加的なメモリが必要となるだけである。本発明に基づく方法の本来の管理は、計算装置の駆動システムによって実施することができる。すなわち、本発明に基づく方法は、付加的なハードウェアを必要とすることなしに、市場で一般的な従来のプロセッサによって実現することができる。なお、本発明に係るコ

ンピュータプログラムを処理する方法をハードウェア支援で実現することも、もちろん可能である。

【００１９】

したがって、エラーのあるタスク、エラーのある課題、エラーのあるプログラミングまたはプログラミング部分、もしくはプログラムオブジェクト、あるいは少なくとも駆動システムオブジェクトを再度計算する従来の技術と比較して、本発明に係るコンピュータプログラムを処理する方法は、効果的である。もともとエラーが過渡的であった場合には、新しく計算する場合に２つの出力は等しく、したがってエラーは消滅し、新規計算によって処理される。

【００２０】

したがって好ましくは、計算装置上、特にマイクロプロセッサ上でコンピュータプログラムを処理する方法が示され、上記コンピュータプログラムは複数のプログラムオブジェクトを有しており、かつ計算装置上でコンピュータプログラムを処理する間にエラーが検出され、エラーが検出された場合には、すでに処理へ供給されている、少なくとも１つのプログラムオブジェクトが定められた状態へ移行されて、この状態から新たに開始され、後続の他のプログラムオブジェクトがずらされる。

【００２１】

また、好ましくは、プログラムオブジェクトは少なくとも２つのクラスに分割され、エラーが検出された場合には、第１のクラスのプログラムオブジェクトが繰り返され、第１のクラスの、すでに処理へ供給されているプログラムオブジェクト内でエラーが検出された場合には、第１のクラスのこのプログラムオブジェクトが第２のクラスのプログラムオブジェクトの代りに新たに開始される。上記場合においてエラーの検出は、新たに開始されるクラスにおいてのみ実施される。

【００２２】

また、好ましくは、すべてのプログラムオブジェクトのためにランスルー内に全計算時間が設けられており、かつ全計算時間が次のように、すなわち、エラーが検出された場合に新たに開始されるプログラムオブジェクトが、エラーのない場合においてはランスルー内の全計算時間の最大５０％を有するように、分割され、その場合に異なるクラスのプログラムオブジェクトが交互に処理され、第１のクラスのエラーのあるプログラムオブジェクトが、すぐ後に続く第２のクラスのプログラムオブジェクトの代りに処理され、もしくは新たに開始される。

【００２３】

また、エラー検出自体は、任意の方法にしたがって行うことができる。エラー検出手段としては、例えば、コンピュータプログラムの処理の間にエラーを検出する（いわゆるconcurrent checking；並行チェック）ことなど、各種類のエラー発見機構の使用が考えられる。例えば、デュアルコアアーキテクチャにおいては、全計算機コアが二重に形成されている。計算機コアがロックステップモードで駆動される場合に、各インストラクションについて、２つの計算機コアが同一の結果を供給するか、結果を比較することができる。その場合に結果の差は、確実にエラーを推定させる。したがってこのエラー発見機構は、プログラムオブジェクトを処理する場合にエラーをリアルタイムで発見することができる。同様なことが、プログラムアーキテクチャ内で一貫して使用されるエラー発見コードについても、あるいは計算装置の二重にされた部分コンポーネントについても言える。これらすべてのエラー発見機構に共通なことは、過渡的エラーを極めて迅速に発見し、エラーが検出された場合には、エラー信号を供給することができることである。

【００２４】

この種のエラー信号には、例えば、プログラムオブジェクトを繰り返すエラー処理機構を合わせることができる。プログラムオブジェクトを新たに実施したとき同一のエラーが再度発生した場合には、恒久的エラーを推定することができ、あるいはエラーカウンタが増分され、その場合にエラーカウンタが所定の値を上回った場合に初めて、恒久的エラーを推定することができる。それに対してプログラムオブジェクトを新たに実施した場合に

エラーがもはや発生しない場合には、エラーが過渡的なエラーであった、と言うことができる。プログラムオブジェクトのエラーがまだ生じていない新たな実施の間に、コンピュータプログラムは、その規定通りの機能を実現するために再び準備を行うことができる。したがって極めて短い時間の後に、可用性を再び高めることができる。したがって少なくとも１つのプログラムオブジェクトの繰返しは、過渡的エラーに対処するための、良好な手段である。

【００２５】

本発明の好ましい展開によれば、プログラムオブジェクトがコンピュータプログラムのランタイムオブジェクトとして形成されており（以下においては特に「タスク」と称する。）、エラーが検出された場合に少なくとも１つのタスクが新たに実施されることが、提案される。タスクは、特に駆動システムレベル上の、典型的なオブジェクトである。タスクの繰返しは、最小限の手間とコストで、望ましい場合には特に純粋なソフトウェア制御で、実現するこができる。

【００２６】

本発明の好ましい実施形態によれば、エラーが検出された時点で実施されていたプログラムオブジェクトが新たに開始されることが、提案される。しかしその代りに、あるいはそれに加えて、エラーの検出の時点ですでに完全に処理されていたプログラムオブジェクトも開始し、新たに処理することができる。

【００２７】

また、プログラムオブジェクトの処理の間、特にプログラムオブジェクトの処理の開始時に、プログラムオブジェクトの少なくとも１つの定められた状態が生成されて、記憶されることが、提案される。これは、例えば、、プログラムオブジェクトの状態にとって重要なすべての変量の値が格納されることによって、行うことができる。

【００２８】

さらに、エラー検出のために、複数のプログラムオブジェクトを有するコンピュータプログラムが処理されている計算装置に対して冗長に作業する、他の計算装置が使用されることが提案される。もちろん、エラー検出のために１つより多い冗長な計算装置を使用することもできる。

【００２９】

好ましくは、本発明に基づく方法は、自動車内、特に自動車制御装置内で、コンピュータプログラムを処理する場合に不可避の過渡的エラーにもかかわらず、コンピュータプログラムの確実かつ信頼できる処理を保証するために、使用される。これは特に、自動車内の安全上重要な適用における開ループ制御および／または閉ループ制御プログラムを処理するために、重要である。

【００３０】

さらに、少なくとも１つのプログラムオブジェクトを新たに実施した場合に同一のエラーが新たに発生した場合に、恒久的エラーが推定されることが、提案される。また、エラーが、プログラムオブジェクトの予め定められた数の繰返し後に相変わらず発生した場合に初めて、恒久的エラーが推定されることも、考えられる。したがって、この場合においては、プログラムオブジェクトが３回あるいはさらに繰り返された後に初めてなくなった場合でも、まだ過渡的エラーが推定される。本発明のこの展開によって、重要なプログラムオブジェクトを、例えば、２回だけでなく３回繰り返すことができる。

【００３１】

本発明の他の好ましい展開によれば、少なくとも１つのプログラムオブジェクトの繰返しの数が、予め定めることのできる値に限定されることが、提案される。それによって、恒久的エラーの場合に同一のプログラムオブジェクトが任意に何回も繰り返されることが、防止される。少なくとも１つのプログラムオブジェクトの繰返しの数の制限は、例えばカウンタを用いて、あるいはタイムバリアを介して行うことができる。タスクに依存する繰返し値を予め定めらることによって、さらに、重要なタスクをそれより重要でないものより多く繰り返し、それによって重要なタスクにより多く、もしくはより長い間、過渡的

エラーなしでエラーのない遂行を行う可能性を与えることが、可能となる。また、それより重要でないタスクの場合には、比較的迅速に恒久的エラーが推定されて、他のシステムリアクションが導入される。

【００３２】

本発明の他の好ましい実施形態によれば、少なくとも１つのプログラムオブジェクトの繰返しの数が予め定めることのできる値に動的に制限されることが、提案される。好ましくは少なくとも１つのプログラムオブジェクトの繰返しの数が、スケジューリングのために残っている残留時間にしたがって、予め定めることのできる値に動的に制限される。このようにして、例えば、第１のタスクと第２のタスクは通過することができ、第３のタスクは何回も繰り返すことができる。

【００３３】

本発明に基づく方法を実現するために、コンピュータプログラムを処理する間、プログラムオブジェクトを実施する前に、プログラムオブジェクトの実施に必要な変量、もしくはプログラムオブジェクトの状態を定める変量の値が記憶されることが、提案される。したがってこの実施形態によれば、すべてのプログラムオブジェクトの変量が格納される。

【００３４】

代替的に、周期内に周期的に処理すべきコンピュータプログラムにおいて、エラーが検出された場合に、コンピュータプログラムの周期内の予め定めることのできるジャンプバック点の所定のプログラムオブジェクトへジャンプバックすることが、提案される。したがってこの実施形態によれば、エラーが発生した場合に常に周期内の同一の箇所へジャンプされる。その場合に好ましくは、コンピュータプログラムの処理の間、ジャンプバック点でプログラムオブジェクトを実施する前にだけ、プログラムオブジェクトの状態にとって重要なすべての変量の値が記憶される。したがってサイクルまたは周期当たり一回だけ、ジャンプバック点でプログラムコードの重要な変量の値だけを格納すれば済む。それによって記憶のための時間とメモリスペースを省くことができる。

【００３５】

エラーが検出された後に、プログラムオブジェクトを新たに実施する場合に、格納されている入力量が呼び出されて、新たに実施すべきプログラムオブジェクトに入力量として提供される。

【００３６】

本発明の他の実施形態として、１つのプログラムオブジェクトに複数のジャンプバック点があてがわれることが、提案される。エラーが発生した場合に、プログラムオブジェクト全体ではなく、プログラムオブジェクトの一部を新たに処理すれば済む。エラーが発生した場合に単に、そこまではプログラムオブジェクトの処理にエラーがなかった、先行するジャンプバック点へジャンプバックされる。例えば、ｎ番目のジャンプバック点までプログラムオブジェクトがエラーなしに処理された場合に、この点と（ｎ＋１）番目のジャンプバック点との間でエラーが発生した場合に、ｎ番目のジャンプバック点へジャンプバックすることができる。その場合にプログラムオブジェクトは、ｎ番目のジャンプバック点から処理される。それによって時間の節約が可能である。好ましくはプログラムオブジェクトの処理の間に各ジャンプバック点を通過する場合に、それぞれ少なくとも１つの定められた状態が生成されて、格納される。

【００３７】

特に重要なことは、本発明に基づく方法を駆動システムの形式で実現することである。その場合に駆動システムは、計算装置上、特にマイクロプロセッサ上で遂行可能であって、計算装置上で遂行された場合に、本発明に基づく方法を実施するようにプログラミングされている。したがって、この場合において本発明は、駆動システムによって実現されるので、この駆動システムは、駆動システムが実施するのに適している方法と同様に、本発明を表す。駆動システムは、好ましくはメモリ素子上に格納されており、処理するために計算装置へ引き渡される。メモリ素子として、とくに任意のデータ担体または電気的なメモリ媒体、例えば、ランダムアクセスメモリ（ＲＡＭ）、リードオンリーメモリ（ＲＯＭ

）またはフラッシュメモリ、を使用することができる。

【００３８】

本発明の課題の他の解決として、冒頭で挙げた種類の計算装置に基づいて、計算装置がエラー処理機構を有しており、そのエラー処理機構はエラー発見機構によってエラーが検出された場合に、少なくとも１つのプログラムオブジェクトの新たな実施を促すことが、提案される。

【００３９】

本発明の好ましい展開によれば、エラー処理機構が、エラーが検出された場合に少なくとも１つのプログラムオブジェクトを新たに開始させる、トリガー論理を有していることが、提案される。

【００４０】

好ましい実施形態によれば、計算装置上でリアルタイム駆動システム、例えばＯＳＥＫ（Open Systems and the Corresponding Interfaces for Automotive Electronics）が遂行されることが、提案される。そして、計算装置がマイクロプロセッサを有してることが、提案される。

【発明の効果】

【００４１】

本発明によれば、計算システム内でコンピュータプログラムを処理する際に過渡的エラーが発生した場合において、それを構造的に、できるだけ短いエラー許容時間内に処理し、計算器システムの機能を回復させ、および機能安全性を再形成することができる。

【発明を実施するための最良の形態】

【００４２】

以下に添付図面を参照しながら、本発明の好適な実施の形態について詳細に説明する。なお、本明細書および図面において、実質的に同一の機能構成を有する構成要素については、同一の符号を付することにより重複説明を省略する。なお、本発明は、以下に説明する実施の形態に限定されないことは言うまでもない。当業者であれば、特許請求の範囲に記載された範疇内において、各種の変更例または修正例に想到し得ることは明らかであり、それらについても当然に本発明の技術的範囲に属するものと了解される。

【００４３】

本発明に係る実施形態は、計算装置上、特にマイクロプロセッサ上でコンピュータプログラムを処理する方法に関する。コンピュータプログラムは、複数のプログラムオブジェクトを有しており、それらは好ましくはタスクとして形成されている。本発明の実施形態に係る方法では、計算装置上でコンピュータプログラムを処理する場合に、エラーが検出される。検出されたエラーは、過渡的な（したがって一時的な）種類のエラー、あるいは恒久的な種類のエラーであり得る。

【００４４】

過渡的なエラーは、計算装置上でコンピュータプログラムを処理する場合に発生する可能性がある。ここで、過渡的エラーは、半導体モジュール（いわゆるチップ）上の構造が微細化されるのに対して、信号のクロックレートが増大し、また、信号の電圧は低下されることによって、より頻繁に発生するようになる。また、過渡的エラーは、恒久的エラーとは異なり、一時的にだけ発生し、通常いくらかの時間が経過した後に自ら再び消えてしまう。過渡的なエラーにおいては、個々のビットが歪曲されるだけで、計算装置それ自体が恒久的に損傷されることはない。過渡的エラーは、例えば電磁的な影響、アルファ粒子または中性子のような、種々の原因により生じる可能性がある。

【００４５】

“過渡的なエラーの発生を予測することはほぼ不可能であり、したがって再現できない”という事実に基づいて、従来の技術を用いた計算装置においては、エラー処理はほぼ恒久的エラーについてだけ行われている。過渡的エラーの考慮は、エラーカウンタのインクリメントと場合によってはデクリメントに限定される。これがメモリ内に格納されて、オフラインで、すなわち、例えば、修理工場滞在の間に、診断情報またはエラー情報として

読み出すことができる。その場合に初めて、診断情報またはエラー情報に応じて反応することができる。したがって、従来のエラー処理は、特に安全上重要なシステムにおいて重要な短いエラー許容時間内のエラー処理を行うことはできず、他方ではまた、エラー許容時間内にコンピュータプログラムが再び秩序どおりに処理されて、計算装置がその規定どおりの課題を満たすことができるような、構造的なエラー処理を行うこともできない。

【００４６】

従来のエラー処理方法に対して、本発明の実施形態に係る方法は、多くのエラーの過渡的な性質に対して、系統的、構造的に対応することによって、計算装置上で遂行されるコンピュータプログラムの過渡的なエラーの処理を行うことができる。ランタイムオブジェクト、いわゆるタスクの例における、本発明の実施形態に係る方法のフローチャートが、図１に示されている。他のタスクの存在は、原理的なシーケンスに影響を与えないため、図１では考慮を省く。すなわち、図１に示すシーケンスにしたがってタスクが処理され、したがって本発明の実施形態に係る方法によれば、複数のタスクも処理することができる。並列に作業するエラー発見機構（いわゆるconcurrent　checking）が特に効果的である。なお、エラー発見機構について、図１のフローチャートでは、それに応じた箇所にシリアルのモジュールとして挿入されている。

【００４７】

本発明の実施形態に係る方法は、機能ブロック１で開始される。機能ブロック１において、計算装置上でタスクの処理が開始される、すなわち、タスクが呼び出される。機能ブロック２において、ジャンプバック点が生成される。ジャンプバック点を生成するために、タスクを新規スタートのために定められた状態へ移行させ、タスクを再度始動させるのに十分な、安全で重要なタスク入力量が、計算装置のメモリ素子内に格納される。好ましくはタスクのすべての入力量が格納される。その後機能ブロック３において、タスクが再び処理される。処理は、他のジャンプバック点で行われるか、あるいはタスクの最後まで行われる。その後、エラー発見機構が実施される。ここで、エラー検出は、任意の方法にしたがって行うことができる。例えば、エラーは、コンピュータプログラムの処理の間に検出される（いわゆるconcurrent　checking）。すなわち、例えば、いわゆるデュアルコアアーキテクチャにおいては、計算機コア全体が二重に形成されている。計算機コアがいわゆるロックステップ（同期並行処理）モードで駆動される場合に、各インストラクションについて、２つの計算機コアが同一の結果を供給するか、結果を比較することができる。その場合に結果が異なることは、確実にエラーを推定させる。したがってこの種のエラー発見機構は、タスクを処理する場合のエラーを実時間で発見することができる。同様なことが、プロセッサアーキテクチャ内で広く使用されるエラー発見コードについて、あるいは、計算装置の二重にされた部分コンポーネントについて言える。好ましくは、過渡的エラーを極めて迅速に発見し、エラーが検出された場合に、エラー信号を出力することが可能な、この種のエラー発見機構が使用される。

【００４８】

照会ブロック４において、エラー、すなわち過渡的エラーまたは恒久的エラーが発見されたか、が調べられる。エラーが検出された場合には、他の照会ブロック７へ分岐し、そこでエラーカウンタ論理の現在の値が検査される。エラーカウンタがまだ予め定めることのできるカウンタ状態を下回っていない場合（デクリメントするエラーカウンタの場合）、あるいは上回っていない場合（インクリメントするカウンタの場合）には、その処理の間にエラーが発生したタスク、もしくはエラーの発生前に処理された所定数のタスクをもう一度実施することができる。タスクの処理の新規スタートが可能である場合には、機能ブロック８へ分岐し、エラーカウンタ論理のステータスが、他のエラーが発生している、という情報によって更新（デクリメントまたはインクリメント）される。そこから機能ブロック５へ分岐して、その中で機能ブロック２で格納された変量がロードされて、タスクが定められた状態を発生させるために処理の開始へ供給される。その後機能ブロック３へ分岐して、そこで繰り返すべきタスクが部分的に、すなわち例えばすでに処理されたジャンプバック点から、あるいは全体として、すなわちタスクが最初からもう一度開始される

ように、再度処理される。

【００４９】

照会ブロック４において、機能ブロック３内でタスクが処理される間に何らエラーが発生していないことが明らかにされた場合に、機能ブロック９へ分岐して、そこでエラーカウンタ論理のステータスが、“エラーが検出されなかった”という情報によって更新される。次に判断ブロック１１へ移行して、そこでコンピュータプログラムが最後まで処理されたか否かが調べられる。コンピュータプログラムが最後まで処理された場合には、機能ブロック６のコンピュータプログラムの最後に分岐する。他の場合には、機能ブロック１２へ分岐して、そこで実際のタスクステータスに応じて、タスクを再度開始させるのに十分な、安全で重要なタスク入力量が定められ、かつ格納されることにより、他のバックジャンプ点が発生される。機能ブロック１２から再び機能ブロック３へ分岐して、そこで繰り返すべきタスクが開始されて、部分的に、あるいは全体として、もう一度処理される。

【００５０】

照会ブロック７において、エラーカウンタ論理の状態に基づいてタスクを新たに処理するためのこれ以上の試みがもはや不可能であることが明らかになった場合には、機能ブロック１０へ分岐する。照会ブロック７においては、このタスクのためにエラーカウンタ論理の値が、タスクに依存する繰返し値よりも大きいか否かが調べられる。このタスクに依存する繰返し値は、種々のタスクについて等しく設定することができ、あるいはまた各タスクについて個別に設定することができる。このようにして、本発明の実施形態に係る方法は、例えば、恒久的なエラーが報告される前に、特に重要なタスクをまず何回か繰り返すことが可能となる。タスクに依存する繰返し値が１として設定される場合には、恒久的なエラーが検出される前に、タスクは１回だけ繰り返される。また、タスクに依存する繰返し値が２または３に設定される場合には、恒久的なエラーが検出される前に、タスクは２回または３回繰り返される。したがってこの場合においてタスクは、過渡的なエラーがもはや発生しなくなるまで、より長い時間、もしくは多くのランスルーを使用することができる。また、機能ブロック１０において、恒久的なエラーが検出されて、それに応じた措置が導入される。この措置は、例えば、コンピュータプログラムを非常走行駆動へ移行させ、あるいはとりあえず行わず、その後コンピュータプログラムの遂行を終了することにある。

【００５１】

なお、本発明の実施形態に係る方法は、図１に示し、かつ上述したすべての機能ブロックおよび照会ブロックを有する必要はない。すなわち、例えば、エラーカウンタ論理に関する、ブロック７から９を省くことができる。エラーが検出された場合に、新たに開始して実施すべき１つまたは複数のタスクは、エラーがもはや発生しなくなるまで、繰り返すことができる。恒久的なエラーは検出されないので、機能ブロック１０を省くことができる。代替的に、タスクに依存する繰返し値を１と設定することができるので、エラーカウンタを更新するための機能ブロック８と９を省くことができる。そして、唯一のジャンプバック点を有する唯一のタスクが実施される場合には、ブロック１１と１２を省くこともできる。

【００５２】

図２には、コンピュータプログラムを処理するための本発明の実施形態に係る計算装置が示されている。本発明の実施形態に係る計算装置は、その全体を符号２０で示されている。計算装置は、メモリ素子２１を有しており、そのメモリ素子は例えば電子的なメモリとして、例えば、特にフラッシュメモリ（Ｆｌａｓｈ）として形成されている。さらに、計算装置２０はマイクロプロセッサ２２を有しており、そのマイクロプロセッサ上でコンピュータプログラムを処理することができる。コンピュータプログラム（ＣＰ）は、例えば、電子的なメモリ媒体２１上に格納されており、参照符号２３で示されている。マイクロプロセッサ２２上でコンピュータプログラムを処理するために、コンピュータプログラムは全体として、あるいは部分的に、例えば指令単位で、データ接続２４を介してマイクロプロセッサ２２へ転送される。データ接続２４は、例えば、１本または複数本のデータ

線として、あるいはデータ伝送のためのバスシステムとして形成することができる。メモリ媒体２１上には、例えば、さらに、駆動システムが格納されており、その駆動システムは計算装置２０を立ち上げる場合に少なくとも部分的にメモリ２１からマイクロプロセッサ２２へ転送されて、そこで実施される。ここで、駆動システム（ＢＳ）は、参照符号２５で示されている。駆動システムは、例えば、マイクロプロセッサ２２上のコンピュータプログラム２３の処理と計算機２０に接続されている周辺機器を制御し、かつ管理する課題を有している。本発明の実施形態によれば、駆動システム２５が特別な方法で形成されているので、同システムは本発明の実施形態に係る方法を実施するようにプログラミングされており、かつマイクロプロセッサ２２上で遂行される場合に、本発明の実施形態に係る方法を実施する。特に、駆動システム２５は、マイクロプロセッサ２２上でコンピュータプログラム２３を処理する間にエラーを検出するためのエラー発見機構へのアクセスを有している。さらに、駆動システム２５はエラー処理機構を有しており、そのエラー処理機構はエラーが検出された場合にコンピュータプログラム２３の少なくとも１つのプログラムオブジェクト（タスク）の新たな実施を促すことができる。

【００５３】

したがって、本発明の実施形態に係る方法、駆動システムおよび計算装置は、本発明に基づくシステムストラテジーの枠内で、最少のパフォーマンスオーバーヘッドで、あるいは特にパフォーマンスオーバーヘッドなしで、タスク繰返しのこのコンセプトを組み込むことが可能となる。

【００５４】

その場合に、それぞれ異なる前提においてパフォーマンスオーバーヘッドとそれに伴ってコストを最小限に抑える、システムストラテジーが基礎とされる。その場合に一般的に、上述したタスクの遂行の際にエラーを発見することのできる（例えば、冗長な処理を有する、デュアルコア機構）エラー発見機構が提供されることが前提とされる。さらに、ここで特に、過渡的エラーについて詳しく説明する。また、恒久的エラーを発見するためには、拡張、例えば、上述したエラーカウンタが必要である。

【００５５】

図３で説明するように、タスク繰返しを組み込むための本発明に基づくストラテジーにおいては、幾つかの前提に注意しなければならない。

【００５６】

タスクの少なくとも２つのクラス（例えばクリティカルなクラスとクリティカルでないクラス）が区別される。その場合にすでに、エラー発見機構はタスクのすべてのクラスのために使用されず、および／または、タスクのすべてのクラスについて実施されない。

【００５７】

図３の例において、タスクの２つのクラスが区別され、一方のクラスにおいてだけタスク繰返しが実施され、および／または、一方のクラスにおいてだけエラー発見機構が始動される。クリティカルなクラスとクリティカルでないクラスに区別する場合には、クリティカルなクラスにおけるエラーだけが本発明に基づいて検出され、特に第１のクラスのためだけに、エラー発見機構が使用される。その場合にクリティカルなクラスは、システムの全機能または基礎となる機能性のために正確な処理が必要であって、この機能を得るためには、遅くとも所定の時点までに行われなければならないタスクである。クリティカルでないタスクは、全システム機能あるいはまた基礎となる機能が該当しないか、あるいは本質的に制限されない、タスクである。その場合に特に、システムにおいては、安全上重要な機能と安全上重要でない機能の間で区別される。

【００５８】

第２のクラス、すなわちクリティカルでないクラス２のタスク内の過渡的エラーは、例えば、無視することができる。さらに、上述したように、第２のクラスのタスクは、「順番が来なくても」よく、すなわち、この第２のクラスのタスクがタスク処理サイクル内で呼び出されない場合に、システムの観点から重大な結果をもたらさないと言える。さらに、第１のクリティカルなクラス１のタスクの全ランタイムは、好ましくはランスルーない

しタスク処理サイクルの全計算時間の、システムに基づく所定のパーセンテージ（例えば５０％）より多くを要求しない。その場合にクリティカルなタスクには、クリティカルなタスクとクリティカルでないものとに二分する場合に、全計算時間の最大で５０％が割り当てられるので、ワーストケースにおいて、すべのクリティカルなタスクにエラーがある場合に、これらを新たに開始させ、もしくは計算することができる。

【００５９】

その場合には、図３に示すシステム考えが可能であって、それにおいては種々のクラスのタスクが、タイムスケジュール内でタスクＴ１（クラス１）の「後任」Ｓ１が少なくとも、Ｔ１のＷＣＥＴ（worst case execution time）と同じ大きさの時間を有し、タスクＴ１の置き換えを可能とする。

【００６０】

その場合に、基本的な考えは、Ｔ１内に過渡的なエラーが発生した場合に、Ｓ１の代りに再度Ｔ１が計算されることである。それによって、Ｔ１内のエラーがＴ２の計算前に除去されることが、保証される。したがって、タスクの計算時間内で、エラー認識および処理（エラー許容特性を維持しながらの極めて確率の高い回復を含む）が行われる。

【００６１】

そのために図３において、クラス１のタスクＴ１、Ｔ２およびＴ３と、クラス２のタスクＳ１、Ｓ２およびＳ３の間で区別される。エラーのない実施において、図３ａにタスク順序Ｔ１、Ｓ１、Ｔ２、Ｓ２、Ｔ３、Ｓ２が例示されている。Ｔ１の間に過渡的エラーが発生した場合に、図３ｂに示すように、Ｔ１がＳ１内で再度計算されて、次にＴ２、Ｓ２、Ｔ３、Ｓ３が計算されるので、過渡的エラーが補正される。

【００６２】

タスク繰返しを組み込むための他のシステム考えが、図４ａと４ｂからなる図４に示されている。その場合にシステムは、比較的小さいジッタ（タスク計算時間の規模の中で）にうまく対応することができる。これは、好ましくは、プラットフォーム機構（特に駆動システム）に適用されうる。その場合に図４ａは、エラーのない実施におけるタスクＴ１、Ｔ２、Ｔ３、Ｔ４およびＳを有するランスルーを示している。

【００６３】

ここでもタスクの少なくとも２つのクラス（例えば上述のようにクリティカルとクリティカルでない）とを存在させることができ、ここでは例えば、一方のクラスにおいてだけタスク繰返しが実施され、および／または、一方のクラスにおいてだけエラー発見機構が始動される。クリティカルなタスクとクリティカルでないタスクを区別する場合には、本発明に係る実施形態にしたがってクリティカルなクラス内のエラーのみがキャッチされ、特に第１のクラスのためにだけ、エラー発見機構が使用される。これは、図３の場合と同様である。

【００６４】

すなわち、ここでも第２のクラスのタスク内の過渡的エラーは、無視することができる。さらにここでも、第２のクラスのタスクは「順番が来なくても」よく、すなわちサイクル内でこのクラスのタスクが呼び出されない場合に、システムの観点からは重要な結果はもたらされない、と言える。そして、ランスルーまたはサイクル当たり、したがってタスク計算サイクル当たり、クリティカルなタスクの最も長いものより長く続く、少なくとも１つのクリティカルでないタスクが存在しなければならない。この少なくとも１つのクリティカルでないタスクは、アイドルタイムとして実現することもでき、もちろんそれは、図３に示す例にも当てはまる。

【００６５】

例において、ランスルーまたはサイクル内にＴ１、…、ＴｎのタスクＴｉ（自然数としてのｎ、ここでは、例えば、５とする。）が存在し、それらはすべてクリティカルであって、および、ほぼこの順序で処理されると仮定する。さらに、Ｔｎの後に、クリティカルでなく、かつ各個々のＴｉよりも長い、少なくとも１つのタスクＳが存在する。図４ｂにおいては、タスクＴ３内で過渡的エラーが発生している。Ｔｉ内のエラーが検出された場

合に、タスクＴｉが単純に即座に繰り返され、残り、したがってＴ４とＴ５は、図４ｂに示すように、後方へずらされる。ランスルー内で過渡的エラーが１つしか発生しない間は、クリティカルなタスクのどれも話題にならず、実施においてどれも最大のタスク長さより多くは後方へずれない。したがってその場合に、エラーが複数になった場合にクリティカルなタスクが実施できないか、はクリティカルでないタスクないしアイドルタイムの長さに直接依存する。安全上の理由からは、ここでも、第１のクリティカルなクラス１のタスクの全ランタイムは、それが好ましくはランスルーないしタスク処理サイクルの全計算時間のシステムに基づく所定のパーセンテージ（例えば５０％）より多く要求しないように、選択することができる。その場合にクリティカルなタスクとクリティカルでないタスクに二分する場合に、クリティカルなタスクには、ここでも安全コンセプトから、全体で全計算時間の最大５０％が割り当てられているので、最悪の場合に、すべてのクリティカルなタスクにエラーがある場合に、これらを新たに始動させ、もしくは計算することができる。

【００６６】

図５ａと５ｂからなる図５は、他の考え方を示しており、それにおいてはほぼ一致する前提を用いることができる。ここではシステムは、少し大きいジッタ（ランスルーの規模において）に対応することができる。これは、プラットフォーム機構、特に駆動システムに適用されうる。ここでもタスクの少なくとも２つのクラス（例えばクリティカルなクラスＴ１－Ｔ５とクリティカルでないクラスＳ）が存在する。ここでも、第１のクラス（Ｔ１－Ｔ５）のためにだけエラー発見機構が使用される。第２のクラスのタスク内の過渡的エラーは、ここでも無視することができる。さらに第２のクラスのタスクは、「順番が来なくても」よく、すなわちサイクル内でこのクラスのタスクが呼び出されない場合に、システムの観点から重大な結果をもたらさないことが、成立しなければならない。そして、ランスルーまたはサイクル当たり、クリティカルなタスクの最も長いものより長く続く、少なくとも１つのクリティカルでないタスクが存在しなければならない（ここでも、アイドルタイムとして実現することができる）。（上の図３および図４の場合と同様の考え方である。）

【００６７】

ランスルー内に、このランスルー内で処理されるべき、タスクＴ１、…、Ｔｎ（すべてクリティカル）が存在すると、仮定する。さらに、Ｔｎの後ろに、クリティカルではなく、かつ各個々のＴｉよりも長い、少なくとも１つのタスクＳが存在するものとする。そのために図５ａには、１つのランスルーのために順序Ｔ１、Ｔ２、Ｔ３、Ｔ４、Ｔ５およびＳを有する、エラーのあるタスクのない、シーケンスが示されている。

【００６８】

例えば図５ｂのＴ３におけるように、Ｔｉ内でエラーが検出された場合に、Ｔｉ（Ｔ３）の結果は重要でないと説明されて、前の例とは異なり、他のタスク（Ｔ４、Ｔ５）が処理される。Ｔｎ（Ｔ５）が処理された後に、Ｔｉ（Ｔ３）がもう一度計算される。その場合の前提は、サイクルのタスクの結果が、同じサイクルの他のタスクによってさらに（必ず）必要とされることを、要求しないシステムデザインである。というのは、その場合にはタスクの順序が維持されないからである。

【００６９】

このシステムの考え方は、図１と２の説明にしたがって、それに応じた装置によって実施される。したがって、本発明の実施形態は、上記に限られず、例えば、各実施形態は、本発明に基づく他の各々の実施形態と組み合わせることができる。

【００７０】

それによって本発明に係る実施形態に基づいて、特にＦＯ特性（FO:Fail Operation(al),Fault Operation(al)）自体を極めて短い回復時間で再び回復させることが可能な、過渡的エラーに関する最適なＦＯ特性を得ることができる。この考え方は、時間制御されるシステムにおいても極めて良好に使用可能であり、および、それに向けて最適化することができる。

【図面の簡単な説明】

【００７１】

【図１】本発明の実施形態に係るコンピュータプログラムを処理する方法を示すフローチャートである。

【図２】本発明の実施形態に係る計算装置を示す説明図である。

【図３】本発明の実施形態に係る、タスク繰返しを組み込むための第１の実施例を示す説明図である。

【図４】本発明の実施形態に係る、タスク繰返しを組み込むための第２の実施例を示す説明図である。

【図５】本発明の実施形態に係る、タスク繰返しを組み込むための第３の実施例を示す説明図である。

【図１】

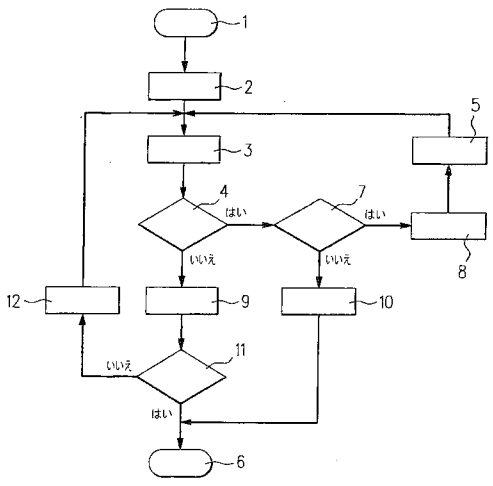

Fig. 1

【図２】

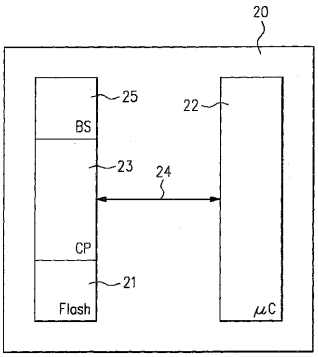

Fig. 2

【図３ａ－ｂ】

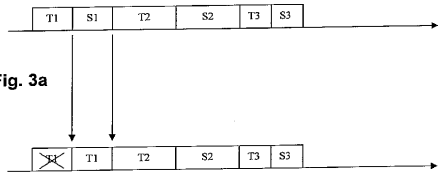

Fig. 3a

Fig. 3b

【図４ａ－ｂ】

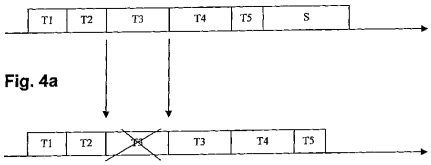

Fig. 4a

Fig. 4b

【図５ａ】

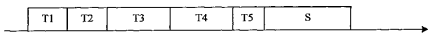

Fig. 5a

【図５ｂ】

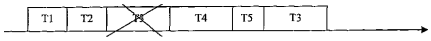

Fig. 5b

フロントページの続き

(72)発明者　ヴァイベルレ、ラインハルト
ドイツ連邦共和国　７１６６５　ファイヒンゲン／エンツ　カルケッカーシュトラーセ　１０
(72)発明者　ミュラー、ベルント
ドイツ連邦共和国　７０８３９　ゲルリンゲン　シュターラー　シュトラーセ　３８
(72)発明者　ハルター、ヴェルナー
ドイツ連邦共和国　７５４２８　イリンゲン　フンメルベルク　４
(72)発明者　アンゲルバウアー、ラルフ
ドイツ連邦共和国　７１７０１　シュヴィーバーディンゲン　クララ－シューマン－シュトラーセ　４
(72)発明者　コトゥケ、トーマス
ドイツ連邦共和国　７１１３９　エーニンゲン　ライメンタールシュトラーセ　１３／１
(72)発明者　コラーニ、ヨルク
ドイツ連邦共和国　７１７１７　バイルシュタイン　リスツヴェーク　９
(72)発明者　グメーリッヒ、ライナー
ドイツ連邦共和国　７１２５４　ディツィンゲン　ヘーエンヴェーク　２

審査官　塚田　肇

(56)参考文献　特開平１１－００７４３１（ＪＰ，Ａ）
特開平１０－３２６２２０（ＪＰ，Ａ）

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 11/00-11/20
G06F 9/46- 9/54