システムソフトウェアと
オペレーティング・システム 67－6
(1994. 12. 8)

分散共有メモリ型並列計算機上のセグメントによるメモリ保護と例外処理機構

田村仁　冨澤眞樹　阿刀田央一

東京農工大学工学部電子情報工学科

〒184　東京都小金井市中町 2-24-16

{tamura,tomisawa}@cc.tuat.ac.jp

あらまし

発注受領型並列計算機上で、マルチタスクを実現する例外処理機構と、セグメントを用いたメモリ保護のためのモデルを提案する。この計算機はハードウェアによる並列手続き呼出しを持ち、手続きのインスタンスの並行実行機能を有する。制御機構が使用するデータ構造を仮想化し、例外によって切り替えるように変更を加えマルチタスク化をおこなった。また共有の論理アドレス空間をセグメンテーションし、タスクに対して与え、タスク境界でメモリ保護をおこなう方式を検討した。これらタスク間のインタラクションについても検討し、タスク境界を越えた発注命令を用意した。これらを現在インテル社の i486 上にエミュレーションで実装中である。

和文キーワード　並列計算機，分散共有メモリ，メモリ保護，セグメンテーション，例外処理，マルチタスク

Memory Protection and Exception Mechanism for Distributed Shared Memory Parallel Computer Depend on Segmentation of Linear Address Space

Hitoshi TAMURA Masaki TOMISAWA Oichi ATODA

Division of Electronic and Information Engineering,
Faculty of Technology,
Tokyo University of Agriculture and Technology
2-24-16 Nakamachi Koganei-shi Tokyo,184 Japan

{tamura,tomisawa}@cc.tuat.ac.jp

Abstract

We propose a exception processing model and a memory protection method, that depend on segmentation of a linear address space for the Demand-Accept model parallel computer. This parallel computer has a control mechanism based parallel procedure call, that executes instances of procedures concurrently. We design a multitask system on the parallel computer, by duplicating data structures of the control mechanism. When some exception occurs in execution on the parallel computer, a exception mechanism switch the data structure to other. We also design a memory protection method, depend on segmentation of linear address space to each tasks. We are implimenting the exception mechanism and the memory protection method on intel i486 processors.

英文 key words　parallel computer, memory protection, segmentation, exception processing, multitask

1. はじめに

並列計算機を汎用的に使用するためには、多様なニーズに応え、計算機資源が効率よく提供されなければならない。このため並列計算機のOSは、動的な資源管理をはじめとして、プログラムのチューニングをおこなうために性能評価ツールの提供、またプログラムの開発効率をあげるためのデバッガ等プログラミング開発環境を提供する必要がある。

このようなOSを開発するためには、デバッグの支援や性能評価のための機能を考慮した例外処理モデルを構築する必要がある。また信頼性の高いシステムプログラムのためにも、このモデルは保護について十分考慮する必要もある。そしてOSの開発を容易にするために、このモデルを例外処理機構としてハードウェアで実現すべきである。

このような例外処理機構上で並列プログラムのデバッグを考えた場合、システム保護の観点から、デバッガのプロセスとデバッグ対象のユーザプロセスの実行を明確に分離できないといけない。これから、限定された意味ではあるが両者のプロセスをマルチタスクで実行できることが望ましい。またそれぞれのプロセスのメモリ空間も分離されている必要が有り、メモリのセグメンテーションなどが必要である。さらにマルチタスク環境の構築を考えた場合、十分なメモリ空間が必要であり、仮想記憶のための機構を提供することも必要である。

本稿では、「発注受領型並列制御機構」を2次元アドレスに対応させ、例外処理のモデルを設計する。そして発注受領型並列制御機構を拡張し、次のモデル化をおこなう。

1) マルチタスクモデルの構築と実現
2) 並列例外処理モデルの構築と例外機構の実現

2. 発注受領型並列計算機

2.1 並列制御機構

筆者等はこれまで、並列計算機は並列制御のためのプリミティブをハードウェアで提供するべきだと主張し、分散共有メモリ型並列計算機に対して発注受領型並列並列制御機構を提案してきた。これは並列制御命令とそのためのデータ構造からなり、fork-join形式の制御をおこなう。

まずサブルーチン呼出しのように、発注(demand)命令を実行する。これで他プロセッサエレメント(PE)の手続きを呼び出す。受注側は、呼び出されるたびに新たな環境を生成し、要求された手続きを走らせる。これを手続きのインスタンスと呼ぶ。受注側は納品(deliver)命令によって値を返し、消滅する。発注側は受領(accept)命令によって、この返り値を受け取る。この3命令によって並列手続呼出しを実現する。

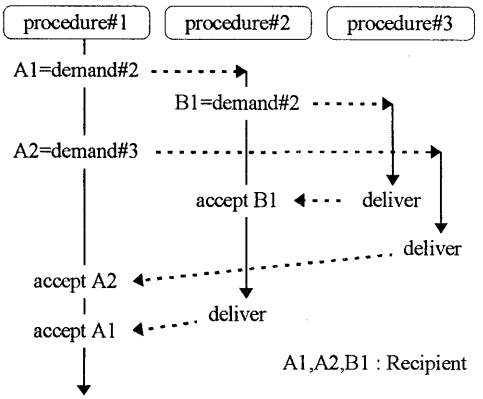

図1 並列手続き呼出しの制御フロー

発注命令と受領命令の対応づけは受信点(recipient)によっておこなう。 受信点は発注命令の中で動的に取得され、受領命令や納品命令の中で伝票として使用される。この受信点は、要求する手続きの識別子、引き数や返り値の格納場所などをもっている。発注命令はこの受信点を発注先のDQ(Demand Queue)に積む。

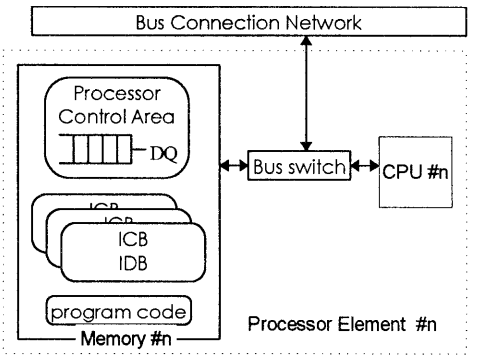

図2 プロセッサエレメントとデータ構造

各PEはそれぞれのホームメモリに、並列制御命令用のデータ構造を構築する。各PEにDQと、PEの状態フラグなど制御情報と、インスタンス切替え用の構造をおく。各インスタンスには受信点管理領域などを含むICB (Instance Control Block)、また

スタック領域としてIDB (Instance Data Block)を割り当てる。

各PEはこのデータ構造により、自律的に動作する。発注要求がDQに積まれると、PE内部で動的にICBとIDBを割り当て、インスタンスを生成する。このインスタンスは、基本的に同期待ちが生じるまで実行される。例えば、受領しようとしたときにまだ返り値がなければ、DQ中の別の実行可能インスタンスに切り替えて実行する。インスタンスは納品命令の時にICBとIDBを解放して消滅する。

このように、発注受領型並列制御機構ではハードウェアが用意した並列制御命令とデータ構造を用いて、手続きのインスタンスを各PEが自律的に手続きのインスタンス群を並列・並行実行する。

2.2 動作モード

発注受領型並列計算機は、並列手続き呼出しを基本の計算モデルとし、ハードウェアレベルで手続きインスタンスの並行実行を実現した。これは多数の仮想PEをソフトウェアに与え、物理的なPE数やインスタンスの並行実行のスケジューリングを意識させないプログラミングを可能にしている。

しかし例外処理では、物理的なPEやデバイスに対する処理が必要であり、勝手にコンテキストスイッチされては不都合が生じる。つまりシステムプログラムに対してはハードウェアによる並行実行機能はいらず、仮想化しない裸のPEが必要とされる。そこで動的なインスタンス環境とは別に、静的な環境で動作し、並列制御命令中でコンテキスト切替えも起こさない動作状態を導入した。

動的にインスタンスを生成しながら計算を進める通常の状態を M(Menber)モード、例外処理状態を S(Singular)モードとよび明確に区別する。動作モードの遷移は、例外の発生によって行われる。

さらに並列計算機の例外処理を考えた場合、各PE で局所的に処理できる例外と、計算機全体に対して影響を与える例外が存在する。局所的な例外は逐次計算機の例外と変わらない。Sモードに遷移して処理すればよい。しかし致命的な例外に対しては、一箇所で集中処理をおこなうほか手段がない。

発注受領型並列制御機構ではこのような大域的な例外が発生すると、凍結割込みと呼ぶ専用の割込みによって即座に全PEを凍結し、KPE(Kernel PE)が集中処理する。これを特に「凍結例外」と呼ぶ。

このKPEは系のなかでただ一つ存在する特殊なPEであり、凍結割込みの制御回路を有する。KPEが凍結例外を処理する時には、当然Sモードで実行されるが、計算機中でただ一つ存在することを明確にするために、この状態を特にK(kernel)モードと呼ぶ。

3. プロセスモデルの拡張

3.1 制御モデル

従来の発注受領型並列計算機では、手続きのインスタンス群を各PEが分散実行し、並列プログラムを実行していた。これを複数の並列プログラムを同時に実行できるように拡張する。一つの並列プログラムに対して一つの仮想的な(従来の)発注受領型並列計算機を与えるのである。

この仮想計算機を提供するには、計算機を静的にグ分割してしまう方法と、各PEを仮想化して切り替える方法がある。前者の方が実現容易であるが、本稿では対象プログラムに最大の並列度を与えてもデバッグできるように、後者の方法を選択する。

ここで本稿における『タスク(Task)』を、並列プログラムとそれを実行する仮想的な並列計算機をもつ処理単位として定義する。ここでいう仮想的な計算機は、一つの並列プログラムを実行するためのデータ領域、そしてDQなど従来の発注受領型並列制御機構をもった、タスクの実行環境をもつ。

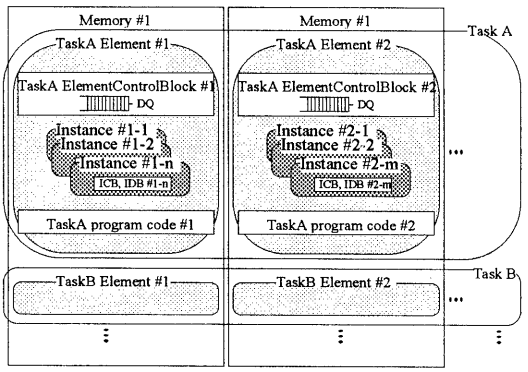

図3 タスク

従来の制御機構で各PEが自律的におこなっていたインスタンスの実行管理を、並列プログラム実行の一単位と考え、この拡張モデルにおける仮想PEの処理単位とする。これを、タスクを構成する『タスク要素(Task Element)』と呼ぶ。タスクスイッチは、タスク要素制御部にコンテキストを保存して、

別のタスク要素に切り替えることで行われる。

このタスク要素を物理的な PE に複数個割り当て、PE を仮想化する。ただし発注受領型並列制御機構では、すでにインスタンスの並行実行による PE の仮想化がある。これに対し、タスクは各並列プログラムを独立に実行させ、タスク間の保護をおこなうために存在する。このためあるプログラムのタスク要素を複数も、一つの PE に割り当てない。

逆に、並列度の低いプログラムでは各 PE 分のタスク要素を用意できない。例えば集中管理用のモニタタスクは一つのタスク要素で構成される。各 PE が受け持つタスク要素の数は、同数とは限らない。

タスクスイッチは各 PE が実行するタスク要素を、時間的に切り替えておこなうが、

①同期型タスクスイッチ方式

各 PE が同期して一斉に、別タスクへ切り替え

②非同期型タスクスイッチ方式

各 PE が任意のタスク要素を任意時間で切り替え

という二つがある。①は②を制限したもので、各 PE に割り当てたタスク要素数が異なると、仕事のない PE が発生する。また②でデバッグや性能評価を考えると、タスク全体の状態を把握できず、その時のタスクの数や負荷などによって振舞いも変わり、実行の再現性がなくなる恐れがある。

これから、通常のタスクは効率の点を考え②で動作させ、デバッグ対象タスクや性能評価用のタスクに関しては①に制限したタスクスイッチもおこなえるようなシステムプログラムが必要になる。

3.2 例外処理モデル

ここで並列計算機で必要な例外処理について論じる。まず発注受領型並列計算機の命令レベルで生じる例外について列挙し、次のように分類する。ここで「エラー」を致命的な例外として、また「フォールト」を回復可能な例外を指す用語とする。

表1 例外の種類

エラー (致命的例外)	ダブルエラー 並列制御機構が生じた致命的例外
フォールト (回復可能な 例外)	バスエラー、不正セグメントなどメモリ機構の不正例外 不正命令、保護違反、ページフォルト 数値演算用の例外　0除算やオーバーフロー例外 コプロセッサ不在例外
割込み	タスク・インスタンス切替え用タイマ割込み デバイス制御用の割込み
ソフトウェア 割込み	デバッグ用の命令による例外 ブレークポイント例外　トレース例外 システムコール用の例外

これらを並列計算機上でどう処理すべきか考えなければならない。2.2で簡単に触れたが、発注受領型並列計算機では、並列計算機の例外処理を

1） 計算機全体に影響する例外処理(凍結例外処理)

2） 例外発生 PE だけに影響する例外処理

の二つに分け、エラーを前者、残りを後者で処理した。凍結例外は集中処理のため KPE だけが凍結例外用ベクタをもっていた。これに対しデバッグ目的のフォルトやソフトウェア割込みは、2)として各 PE が自分の例外ベクタを参照して処理を行った。しかしタスクが複数存在する場合は、

3） 例外発生タスクだけに影響する例外処理

4） 例外発生タスク要素だけに影響する例外処理

5） 例外発生インスタンスに影響する例外処理

のように概念を拡張する必要があり、各例外をそれぞれどのモデルで処理すべきか議論する。

従来の 1)をタスクに制限したものが 3)である。これは例外が発生したときの凍結割込みに対して特定タスクに含まれるタスク要素だけを凍結し、タスク別に用意された例外ベクタに従い、KPE が処理する。しかし無数に存在しうるタスクに対してハードウェア的な凍結割込みはひとつだけである。ソフトウェアによって実現すべき例外処理であろう。

次に局所的な例外は、従来の 2)の他に 4)5)が拡張される。この例外は主にデバッグ目的で使われる。例えば実行履歴の取得は 4)を使ってタスク別に行いたい。しかしタイマ割込みによるタスク切替えなど、タスクにまたがる 2)の例外処理も必要である。このため 4)とは独立なハンドラの環境が必要になる。また 5)もありうるが、軽量さが優先されるインスタンス切替えを考え、制御機構では支援しない。

これより制御機構に拡張する例外処理方式を、改めて次のように呼ぶことにする。

① 凍結例外処理　並列計算機全体に影響する

② PE 例外処理　例外発生 PE に影響する

③ 解析例外処理　例外発生タスク要素に影響する

①は、KPE 上のカーネルタスクが、PE からの凍結例外処理の発注を受けて、処理をするモデルである。このときの発注は、タスク間にまたがる発注になり、特に「凍結例外発注」とよぶ。②は、各 PE がそれぞれの S モードタスクへスイッチし例外を処理する。その S モードタスクはタスク要素一つだけから成る。③は、実行中のタスク要素が S モードに遷移し、タスク要素内でそのまま処理される。

ここで改めて表 1 の例外を考える。エラーは凍結

例外処理である。フォルトも、通常は凍結例外処理で扱う。例えばページフォルトは凍結例外を利用し、分散共有メモリ上での仮想記憶の実現に応用できる。しかし、保護違反を外部アクセスの履歴を取得に利用したりするなど、必要に応じてPE例外で処理したほうが都合がよい場合もある。また、デバッグ用や数値計算用のフォルトに関しては、タスク内で処理できるため解析処理として扱うべきである。割込みやシステムコールは、その目的がPE別のものか、計算機全体のものかによってそれぞれPE例外か凍結例外かが決定される。

これらから、デバッグ目的や数値計算用のフォルトは解析例外処理をおこなう。それ以外を、PE例外、凍結例外のどちらで処理するかはシステムプログラム側で選択できる機構を提供する。

PE例外用のベクタに、解析例外を除いたすべての例外ベクタを持たせ、そこにハンドラのアドレスがあればPE例外、なければ凍結例外を引き起こすように設計する。無論、制御機構内部のエラーや例外処理中のエラーなど致命的な事態に対しては、即時に凍結例外が生じる。

3.3 メモリモデル

3.3.1 ホームメモリ複合体

共有メモリを前提にした発注受領型並列計算機では、タスクに参加するどのPEも同じメモリ空間を参照する必要がある。3.1で述べた非同期式のタスクスイッチでは同じ時刻でも各PEが別々のタスクを走らせているが、常にどのPEからも、どのタスクのメモリ空間へもアクセスできなければならない。メモリ管理の負荷を考え、全PE全タスクで共有の空間を切り分けて使用することを考える。

このメモリモデルでは、全PEで共有する一次元の論理アドレス空間を複数のタスクによって分割することになる。そこで本稿では、セグメントアドレスから一次元の論理アドレス(以後リニアアドレスと呼ぶ)へのマッピング、リニアアドレスから物理アドレスへのマッピングの2段階のアドレス変換機構を有するような2次元アドレッシング機構をハードウェアが提供していることを前提とする。

共有リニアアドレス空間をセグメンテーションし、あるタスクへ割り当てる。この領域はさらに、そのタスクの各タスク要素を担当するPEのホームメモリへ物理的に分割して配置されるべきである。この一つのタスクに対して割り当てられた領域を、各PEのホームメモリを複合したという意味で、『ホームメモリ複合体(Home Memory Compound)』と呼ぶことにする。

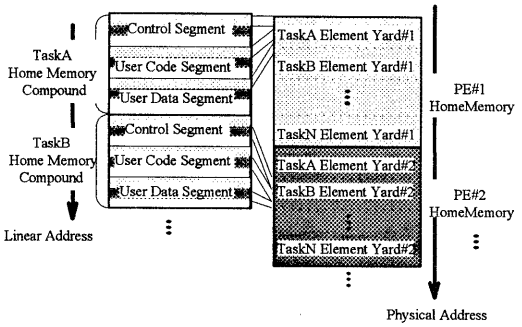

図4 ホームメモリ複合体とセグメント

3.3.2 メモリの保護

前節より、タスク別にセグメントを与える。並列計算機ではさらに、物理的にPE間に保護の境界が存在しうる。しかしここにセグメント境界をおくと、他PEへは常にセグメントを越えた参照となる。またPE数が大きいと一タスク当たりのセグメント数も増大する。これはタスク切替や発注命令など制御機構の基本機能に重大な負荷をもたらす。これから、セグメントはタスク別に与え、PE間の境界は考慮しない。もし他PEから保護したい部分があれば、自PEだけのタスク要素からなるタスクを生成し、その中に閉じこめればよい。

次に一つのタスクに対して、ユーザ用のセグメントと、発注受領機構が使用するための制御用セグメントが必要である。発注受領型並列計算機では、

①ユーザ状態のMモードのインスタンス
②解析例外処理中のSモード状態
③PE例外処理中のSモード状態
④KPEのKモードタスク処理状態

が存在する。①は通常のICBとIDBを環境に実行される。また②は各タスク要素に対して、DQなどの制御部やコンテキストの待避領域を静的に確保したTECB(Task Element Control Block)とTEDB(Task Element Data Block)を環境にする。①と②は各タスク要素に対して存在する。そこであるタスクに含まれるタスク要素がもつすべてのIDBを、まとめて一つのユーザデータセグメントに割り当てる。同様に全PEのICBやTECBもひとつの制御データセグメントに割り当てる。

③、④は、ユーザプログラムを実行しているタスクとは独立したシステムタスクで実行されるべき処理である。これらはそれぞれのSタスクやKタスクのTECBとTEDBを環境に実行される。

また①から④のそれぞれの例外処理の目的から考えて、システムプログラムはこの順にPEの特権レベルを設定すべきである。これでアクセス可能なセグメントを制限する。①の状態では、自分のタスクのデータセグメントだけをアクセスできればよい。他PEへのアクセスでも同じセグメント内だけで処理が行われる。②ではさらに制御セグメントが読めて、また③では自PE内のすべてのセグメントが読めればよい。

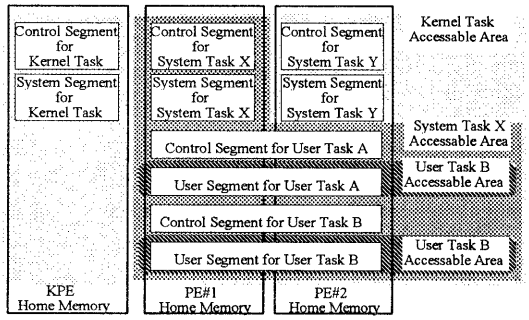

図5　メモリ保護概念図

Sモードとは物理的PEをプログラムに与えるものであり、②では自PEの中だけに参照を制限した方がよい。しかしセグメントはタスクを実行するPE群で共有されており、他PEとの間では保護されない。ただし他PEの参照から保護する必要のある③と④のための環境は、SタスクやKタスクとして独立させれば、タスク境界で保護が可能である。

図4はタスク内のセグメントを三つ用意した例である。セグメント内は論理アドレス上で連続領域にとられなければならないが、物理アドレスでは各ホームメモリへ分散する必要がある。

3.4　タスク間、タスク要素間の遷移と通信

3.4.1　例外によるタスク要素切替え

すべての例外は発生すると、例外を起こしたPEをSモード状態へ遷移させる。このとき同時に、例外を処理するために、ユーザタスクから例外処理ハンドラの環境へ切り替えなければならない。

3.2でのべた三つの例外処理の中で、解析例外処理はタスク切替えをおこさない。しかしスタックやICBは、タスク要素内の静的に用意されているTECBとTEDBへ切り替えられる。

TECBには、発注命令で使用する受信点の管理のために、ICBと同じ構造が含まれている。発注命令からみれば、解析例外処理状態でもTECB内の専用のICBに切り替えられただけにみえる。

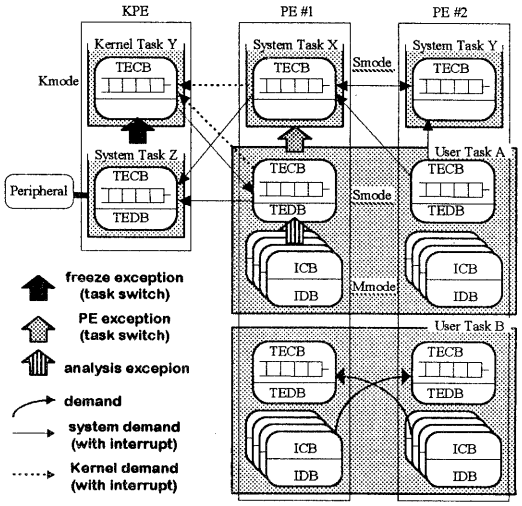

図6　タスク間の遷移と通信

PE例外処理ではシステムタスクへ切替えがおこなわれる。このシステムタスクはSモードで動作するためインスタンスは生成されず、解析例外処理と同様に静的なTECBとTEDBで実行される。

凍結例外処理ではPEは凍結され、解凍されるのを待つだけである。ただしKPEだけは凍結割り込みをうけ、カーネルタスクへタスクスイッチが発生し、凍結例外処理をおこなう。

タスク要素の切替え先は、各PEに割り当てられているタスク要素である。他PEのタスク要素へコンテキストが移行させることは、ホームメモリのアクセス等を考えれば、おこなうべきではない。

また、複数のユーザタスクが存在する場合は、タイムスライスによる割込みを利用し、タスク要素の切替えを行わなければならない。これは割込みに対するPE例外処理で一度システムタスクを起動し、スケジューリングさせることで実現できる。

さらにタイムスライス割込みがくる前でも、DQ中に何も要求がない場合は、タスク要素を切り替えるべきである。従来はこのような場合、制御機構中でDQの監視ループがまわっていただけだった。そこでスケジュール用の例外を発生してシステムタ

スクへ遷移する手段を与えなければならない。

3.4.2 タスク間発注

タスク間のメモリ保護のため、タスク間での発注は通常認めない。相手先の DQ の位置を間違えると、相手タスクへの書き込みで障害を引き起こすおそれがある。また、発注時には手続きのアドレスを指定しなければならないが、他のタスクからはみえないため、そもそも手続きを発注できない。

しかしシステムコールにあたるシステムタスクへの発注は必要である。例えばある PE にだけ接続されたプリンタへは、その管理タスクへ発注しなければならない。そこでシステム発注(System Demand : SDMND)命令を使って、システムタスクが定義する機能を発注できるようにする。これはセグメント境界を越える命令である。オペランドとして手続きアドレスではなく機能番号を指定する。

しかしそれでも相手先タスクの DQ の位置は必要である。これはシステムが握っている情報なため、SDMND 命令は、特権命令として、システムプログラムだけが実行できるようにする。 ユーザタスクは、PE 例外や解析例外によって一度特権状態に遷移してから他タスクへの発注をおこなう。

システムタスクは要求がなければ、休眠させておくべきである。すると割込みをかけないと実行されないため、SDMND 命令は発注だけでなく割込みもおこなう。発注受領型並列計算機では、凍結割込みの他に PE 別に割込みをかける機構が必要になる。

システムタスクから他のシステムタスクへさらに SDMND される場合、自分へ発注が返ってくる可能性もある。すると、双方とも S モード状態のため、受領待ちのまま飢餓状態が生じる。システムプログラマは注意深くこのような発注の巡回を避けるか、受領命令の失敗に対し、タイムアウト処理をおこなうようにプログラムするべきである。

SDMND 命令は発注先がシステムタスク、つまり S モードで動作するタスクだけに対し認められ、ユーザタスクへは認められない。ただし最初に一回だけ外からメイン手続きへの発注が必要である。このため K モードで動作しているカーネルタスクからは、ユーザタスクへのタスク間発注を認めることにし、SDMND 命令をかけられるようにする。

タスク要素間のインタラクションは、発注命令や受領命令など並列制御命令をとおして発注動作によるものと、共有メモリ上のデータへのアクセスによりおこなわれる。このとき DQ などが存在する制御セグメントはユーザプログラムから保護されている。ユーザプログラムは並列制御命令を実行することで DQ などを操作できる。

3.4.3 システムタスクの構築

タスク間発注の特殊なものが凍結発注である。これは凍結例外が発生した時にハードウェアにより自動的に K モードタスクに対して行われる。これを計算機全体の資源に対する要求としてシステムコールのように使うことも考えられるが、次の問題点がある。一つは凍結例外処理は負荷が大きすぎる点、要求がバーストした場合 KPE の DQ があふれて肝心な致命的例外を処理できなくなる点である。そこで凍結例外ではなく SDMND を KPE 上の別のタスクへおこない、カーネルへのシステムコール用の DQ を分離する。このタスクはカーネルタスクではなく S モードのシステムタスクである。K モードは凍結割込みに対する特権を有するべきであり、その特権は物理的にただ一つでなければならない。必要ならば、このシステムタスクがソフトウェア的に凍結例外を引き起こし、K モードへ遷移すればよい。このようにカーネルタスクは、凍結例外処理のためだけに存在するタスクとする。

システムタスクを構築する方法には、S モードタスクを用意する他に、常に S モードで動作するタスク要素をタスクに組み込む方法がある。つまり各ユーザタスクに S モードタスク要素を用意し、このタスク要素間で資源管理をおこなうものである。

S モードは物理的な PE をプログラムに与えるためのものであるが、S モードタスクや S モードタスク要素は PE に対して一つだけとは限らない。管理すべき独立した資源や事象があれば、その数だけ複数存在しても問題はない。この各ユーザタスクに用意された S タスク要素は、あくまでそのタスク自体の管理をおこなうものである。これによりシステムコールのためにタスクスイッチをおこさなくてよい。ただし、タスク内ではメモリ保護ができないため、物理的な PE やデバイスなどの管理には、やはり独立の S モードタスクを用意するべきである。

3.4.4 タスクの生成と消滅

タスクの生成には、まずメモリ領域の確保が行われる。共有の論理アドレス空間から領域を確保し、空き物理メモリに割り付ける。この割り付けは、うまくホームメモリへ分散するように行われる。この

ようなメモリ領域の操作は、計算機全体で整合する必要があるため、カーネルタスクがおこなう。

次にカーネルタスクは、確保した領域中の各TECB の部分に初期化用のパラメータをうめこんでいく。例えばDQの長さや、生成可能なインスタンス数などである。そして各PEのシステムタスクに対して SDMND 命令を発し、パラメータに従ったタスク要素の初期化を要求する。このときタスク要素のコンテキストには、DQのスケジューリング例外処理をおこなうように設定される。

最後にカーネルタスクは生成したタスクのメイン手続きを SDMND する。このメイン手続きは最後に凍結例外を発生して終了させればよい。この凍結例外をもってカーネルタスクが起動され、タスクの消滅処理をおこなう。タスクの消滅処理は、各PEへタスクの消滅を知らせるSDMNDと、メモリの解放をおこなうことである。

4. 例外処理モデルの実装

これまで述べてきた発注受領型並列計算機の例外処理機構を、インテル社のi486をもちいてエミュレーションによる実装をおこなっている。ここではi486固有のアーキテクチャと、どのように例外処理機構をのせるかについて議論する。

i486のメモリ管理機構は、セグメントとページングを独立に使用している方式である。つまりセグメントアドレスから一度論理的な一次元アドレスに変換する。これをリニアアドレスと呼ぶ。このリニアアドレスからさらにページングにより物理アドレスに変換する方式である。

まずリニアアドレスを全PE、全タスクで共有するために、ページテーブル群を論理的にひとつ用意する。各PEはこれを自分のホームメモリへコピーして使用する。コピーの一貫性管理について問題が生じるが、自ホームメモリ上に存在するページフレームへのアクセスを高速にできなければ意味がないので、そのフレームへのポインタを有したページテーブルはホームメモリ上にコピーしておく必要がある。現段階ではスワップはおこなわず、ページテーブル書き換えはタスクの生成消滅意外に発生しないことにして実装する。

i486の例外処理機構は、割込みテーブルにコールゲート、トラップゲート、タスクゲートが登録できる。コールゲートはタスク切替えなしで、単にハンドラへ飛ぶ処理をする。トラップゲートも同様であるが、割り込みをマスクしてくれる。タスクゲートはタスクスイッチを伴うものである。

実装するエミュレータは、このうちトラップゲートを用いておこなう。発注命令などはソフトウェア割込み命令を用いて実現する。

5. まとめ

本稿では、発注受領型並列制御機構をもった分散共有メモリ型並列計算機の例外処理機構とメモリ保護について述べた。 発注受領型並列計算機は、ハードウェアによる並列手続き呼出しを持ち、手続きのインスタンスの並行実行機能を有する。これをマルチタスク化するために、制御機構が使用するデータ構造を仮想化し、例外によって切り替えるように変更を加えた。

その上で共通の論理アドレス空間を切り分け、タスクに対してセグメントを割り当て、タスク境界でメモリ保護をおこなう方式を検討した。この方式ではPE間の不正アクセスから保護できないが、システムタスクを各PE内に用意することでPE固有の情報の保護をおこなえる。また、これらタスク間のインタラクションについても検討し、タスク境界を越えた発注命令を用意した。これらを現在インテル社の 486 プロセッサ上にエミュレーションで実現中であり、その実装について述べた。

参考文献

[1] 田村ほか:発注／受領型並列計算機の例外処理機構とデバッグ支援機構,情処研報,ARC-89-18,pp.127-134(1991).

[2] 田村ほか:発注／受領型並列計算機の例外機構と保護機構,情報処理学会第42回全国大会講演論文集(6),pp.11-12(1991).

[3] 冨澤ほか:手続きフロー型並列計算機における分散型組み込み制御機構,信学論(D),J71-D,10,pp.1921-1030(1988).

[4] 冨澤:発注受領型並列計算機の制御機構とハードウェア,JSPP'91 論文集,pp.221-228(1991).

[5] 冨澤ほか:発注受領型分散制御機構をもつ密結合並列計算機用言語 C//の設計と実現,信学論(DI),J75-D-I,11,pp.1025-1036(1992).

[6] 星野ほか:『発注／受領型』並列計算機上の並列手続型言語・実行系・応用例,情処研報,ARC-93-26,pp.193-199(1991).