オペレーティング・システム 43 － 8
（１９８９．６．９）

可変構造型並列計算機の並列／分散オペレーティング・システム

福田晃† 福澤祐二† 廣谷良彰† 村上和彰† 末吉敏則‡ 富田眞治†
（†九州大学大学院総合理工学研究科，‡九州工業大学情報工学部）

「可変構造型並列計算機」は，(1) ネットワーク／メモリ・アーキテクチャと各種並列処理形態との親和性の検討，(2) 解くべき並列処理形態にネットワーク／メモリ・アーキテクチャを適合させることによる応用分野の拡大，を主な目的とする．本システムは密／疎結合型マルチプロセッサのいずれの形態も実現できる．密結合型マルチプロセッサの場合，メモリは多重バンク構成をとり，さらに均一／不均一メモリ・アクセス構成のいずれの形態をもとることができる．我々はまず，共有メモリという魅力的で基本的なメモリ・アーキテクチャをもつ密結合型マルチプロセッサ用OSを開発する．密結合型マルチプロセッサに適した並列処理モデルとして，タスク／スレッド・モデルを導入する．タスクは計算機資源割当ての単位であり，スレッドはその中で動く活動体（制御フロー）である．プロセッサの割当てをタスク単位に行うことによって，スレッド切り換えに伴うオーバヘッドを小さくできる．

A Parallel／Distributed Operating System for the Reconfigurable Parallel Processor

Akira FUKUDA†, Yuji FUKUZAWA†, Yoshiaki HIROTANI†, Kazuaki MURAKAMI †, Toshinori SUEYOSHI‡, and Shinji TOMITA†

† Interdisciplinary Graduate School of Engineering Sciences, Kyushu University, 6－1 Kasuga－Koen, Kasugashi, Fukuoka, 816 JAPAN
‡ Faculty of Computer Science and Systems Engineering, Kyushu Institute of Technology

A reconfigurable parallel processor system is under development at Kyushu University. The main research objects of the system is ; 1) to investigate the problem of the degree of match between algorithms and architectures, and 2) to construct a high－performance multiprocessor system which can be tailored to a broad range of applications. The system provides two types of multiprocessor configurations : a tightly coupled multiprocessor or a loosely coupled multiprocessor. First, we develop a operating system for a tightly coupled multiprocessor, which has a attractive and basic memory architecture. A task－thread model is employed as a well matched parallel processing model to a shared memory architecture. A task is the basic unit of resource allocation. Threads are control flows in a task. Overhead during context switching can be reduced by employing a scheme in which processors are assigned to a task.

1. はじめに

マルチプロセッサ・システムは，絶対性能，価格性能比，信頼性などを向上させる可能性を秘めており，現在盛んに研究され，商用化されているシステムもある[1]．マルチプロセッサ・システムの可能性をうまく引き出せるかどうかは，対象とする処理形態がハードウェア・アーキテクチャにうまく適合しているかどうかに大きく依存する．特に，ハードウェア・アーキテクチャの主要な構成要素であるメモリ・アーキテクチャ，ネットワーク・アーキテクチャによって，それに適した応用分野が制限されがちである．しかし，これらのアーキテクチャと各種応用分野との親和性の定量的検討は必ずしも十分に行なわれていない．また，実行する処理形態にメモリ・アーキテクチャ，ネットワーク・アーキテクチャを動的に適合させることができれば，応用分野の裾野を広げることができる．これらの研究には，並列／分散オペレーティング・システム（OS）が深く係わっており，並列／分散OSの研究は不可欠である．現在，様々なOSが研究・開発されている[2]－[6]．我々は，このような目的からメモリ・アーキテクチャ，ネットワーク・アーキテクチャを可変構造にした汎用のマルチプロセッサ「可変構造型並列計算機」の開発，並列／分散OSおよび並列プログラミング言語の研究を進めている[6]．本稿では，そのOSの概要を述べる．

2. OSの研究・開発方針

本システム開発の主な目的を以下に示す．

（1）ネットワーク／メモリ・アーキテクチャと各種並列処理形態との親和性の検討．

（2）解くべき並列処理形態にネットワーク／メモリ・アーキテクチャを適合させることによる応用分野の拡大．

（1），（2）を遂行するには，並列／分散OSの研究を含めた広範囲な並列処理の研究が不可欠である．（2）を実現するには，まず（1）の検討すなわち，ハードウェア・アーキテクチャと各種並列処理形態との親和性を定量的に検討する必要があると考える．そこで，我々は（2）を実現するOSの開発に先立って，（1）を対象としたOSを開発する．

マルチプロセッサはメモリ・アーキテクチャによって大きく分類される．OS構築の立場からみると，各プロセッサからのメモリの見え方によって以下の分類が考えられる．

（1）アクセスできるメモリ範囲による分類（共有メモリ構成vs. 私的メモリ構成）

共有メモリ構成とは，メモリの任意の場所に任意のプロセッサからアクセス可能なメモリ構成法である．私的メモリ構成とは，アクセスできるメモリ範囲がプロセッサごとに異なるメモリ構成法である．すなわち，メモリのアドレスとそこにアクセスできるプロセッサが1対1対応をなすものである．もちろん，これらを組合せたものも考えられる．

（2）メモリへのアクセス距離による分類（均一メモリ・アクセス構成（UMA：Uniform Memory Access）vs. 不均一メモリ・アクセス構成（NUMA：Non－Uniform Memory Access）

均一メモリ・アクセス構成は，プロセッサから等距離でメモリにアクセスできる構成法である．不均一メモリ・アクセス構成は，メモリのアドレスによって距離が異なる構成法である．

通常，私的メモリ構成では均一メモリ・アクセス構成をとる．一方，共有メモリ構成は，均一／不均一メモリ・アクセス構成の2つのものが考えられる．プロセッサ台数の増加にともなって，均一メモリ・アクセス構成から不均一メモリ・アクセス構成に移行すると考えられる．なお，本稿で以後用いる密結合，疎結合という単語は，各々共有メモリ構成，私的メモリ構成と同義である．密結合方式は，プロセッサ間でメモリを共有しているので，プロセッサ間の情報交換は一般に疎結合方式よりも速い．しかし，メモリへのアクセス競合が生じやすく，それによって性能が抑えられるので，プロセッサ台数の少ないシステムに向いていると考えられる．一方，疎結合方式はプロセッサ間の結合形態にもよるが密結合方式よりも大きくとることができる．プロセッサ台数の増加にともなって，システムのメモリ・アーキテクチャは密結合方式から疎結合方式に移行するが，この定量的な評価は必ずしも十分でないように思える．我々は，密結合方式は魅力的であり，かつ基本的なメモリ・アーキテクチャであると考え，まずこれを対象としたOSを研究・開発する．したがって，以下のアプローチをとる．

（1）密結合型マルチプロセッサ用のOSを研究・開発し，各種並列処理形態との親和性などの定量的検討を行う．また，これらを通じてOSの研究を含んだ並列処理の研究を行う．

（2）上記（1）を疎結合型マルチプロセッサを対象として行う．

（3）（1），（2）の検討を踏まえて，本システムが提供するネットワーク・アーキテクチャ，メモリ・アーキテクチャの可変性を活かしたOSを研究・開発する．

3. メモリ・アーキテクチャの概要[6]

密結合型マルチプロセッサ用OSの説明に先立って，本システムのメモリ・アーキテクチャの概要を示す．

（1）アドレス変換機構

仮想アドレス空間，実アドレス空間，物理アドレス空間の3つの空間を提供する．実アドレス空間をプライベート空間とコモン空間の2つに分けることによって，私的／共有メモリ構成の両方をサポートする．それらの変換過程を

述べる．まず，仮想アドレスは，MMU（Memory Management Unit）によって実アドレスに変換される．実アドレスの最上位ビットが0であれば，プライベート空間へのアクセスとなり当該プロセッサのローカル・メモリへ直接アクセスされる．一方，実アドレスの最上位ビットが1であれば，各プロセッサ対応に256個の共有メモリ・ウインド（SMW：Shared Memory Window，現在128個のSMWは未使用）に分割されたコモン空間へのアクセスとなる．コモン空間へのアクセスは当該SMWを提供するプロセッサにおいて，実アドレスをSMW変換表を用いたページングにより物理アドレスに変換して，当該プロセッサのローカル・メモリをアクセスする．これらの機構により，密結合／疎結合型マルチプロセッサ，またはその混合形態を提供している．すなわち，コモン空間（プライベート空間）だけをアクセスさせるようにすれば，密結合型（疎結合型）マルチプロセッサを実現できる．

（2）均一／不均一メモリ・アクセス機構

あるプロセッサからコモン空間を介してメモリをアクセスする場合，自分が提供するSMWと異なるSMWへのアクセスは，ネットワークを介したものとなる．一方，同じ場合，すなわちプロセッサが自分に対応するSMWをアクセスした場合，1）ネットワークを介するアクセス，2）ネットワークを介さないアクセス，の2つのアクセス方法をハードウェア・レベルで提供できる．すなわち，1）均一メモリ・アクセス構成，2）不均一メモリ・アクセス構成の両方を実現できる．したがって，対象とする密結合型マルチプロセッサのメモリ・アーキテクチャは，多重バンク構成および均一／不均一メモリ・アクセス構成をとる（図1）．

4．密結合型マルチプロセッサ用OS

4.1 開発目的

開発目的を以下に示す．

（1）各種並列処理形態と密結合型マルチプロセッサとの親和性の定量的検討

マルチプロセッサの効率は，並列処理形態，プロセス間の通信パターン，およびメモリ参照パターンなどに大きく依存する．これらの定量的評価を行う．また，台数効果などについて定量的評価を行う．

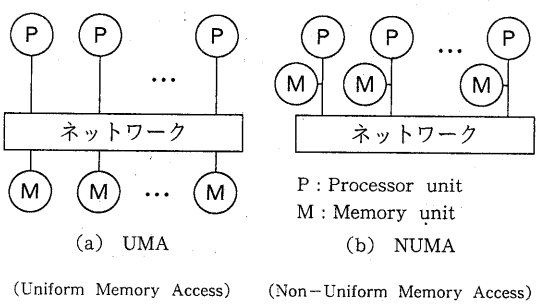


図 1　ハードウェアが提供する密結合型のメモリ・アーキテクチャ

（2）密結合型マルチプロセッサ用OSの研究

並列処理を効率よく行うためには，以下の点が重要となる．

（a）問題の分割

マルチプロセッサの可能性を十分引き出すように，解くべき問題を複数の副問題に分割しなければならない．このとき，分割する副問題の大きさ（粒度）が問題となる．分割法には，機能分割法と負荷分割法が考えられる．機能分割法は，異なる機能からなる副問題に分割する方法である．負荷分割法は，同一機能で扱うデータが異なる副問題に分割する方法である．これらは問題の性質によって自動的に決まることもある．これらを組合せたものも考えられる．

（b）スケジューリング

分割された副問題をいつ，どのプロセッサでどのような順序で実行させるかのスケジューリングが重要となる．これらは，プロセッサの負荷状況，副問題間の通信パターン，およびメモリ参照パターンなどを考慮して決定する必要がある．

（c）資源管理

各資源をどのように管理，利用するかが問題となる．メモリにおいては，アクセス競合を軽減させることが重要である．特に，メモリがバンク化されている場合や不均一メモリ・アクセス構成の場合には，データ，コードなどのメモリへの配置方法が重要となる．

4.2 設計方針

開発にあたっての設計方針を以下に示す．

（1）密結合方式に適した並列処理モデルをユーザに提供する．

仮想メモリ，ファイルなど同一の計算機資源の中で複数の活動体（制御フロー）が協調しながら仕事を進めるモデルは，密結合方式に適したモデルと考え，このモデルをユーザに提供する．したがって，使用する計算機資源を規定するタスクと，その中で動く活動体スレッドの概念を導入する．言い換えれば，タスクは資源割当ての単位であり，タスク対応に仮想アドレス空間を持つ．スレッドは，プログラム・カウンタ，レジスタの内容などを持つ．タスク内のスレッドはコード，データを共有するが，スタック領域はスレッドごとに持つ．これらは，Mach[2]におけるそれと同様である．

（2）カーネルのデータ構造を隠蔽化する．

カーネルは種々の操作を実行する機構とデータ構造からなるが，実現を容易にするため，データ構造を各機構内に隠蔽化し，他の機構から直接アクセスできないようにする．

(3) 多様なシステム・コールを提供する.

ユーザが多様な並列処理問題を扱えるようにするため, 多様なシステム・コールを提供する. 可能なかぎりUNIX*のシステム・コールも提供する.

4.3 カーネルの構成方法

カーネルの構成方法には, 種々のアプローチが考えられる. 以下に, その分類と我々が採用する構成方法を示す.

(1) マスタ・スレーブ型かシンメトリ型か

カーネルを実行するプロセッサに注目して, 以下の分類が考えられる

(a) マスタ・スレーブ型 : (特定の) 1つのプロセッサでカーネルを実行する. したがって, 一時にカーネル・モードで実行しているプロセッサは高々1台となる.

(b) シンメトリ型 : 全てのプロセッサがカーネルを実行できる.

(a) は実現が容易であるが, カーネル内の処理は逐次的になるので, 効率が悪くなる. 一方, (b) は (a) に比べて実現が困難であるが, カーネル内の処理が並列に実行されるので効率がよい. 我々は, (b) を採用する. また, カーネルのコードはプロセッサ固有の私的メモリ上におく. 一方, カーネルのデータは共有メモリ上におき, 複数のカーネルからアクセスできる.

(2) 機能分散方式か負荷分散方式か

カーネルの機能とそれを実行するプロセッサとの関係から以下の2つがある.

(a) 負荷分散方式 : 各プロセッサ上のカーネルに同一機能を持たせて負荷を各プロセッサに分散させる方式である.

(b) 機能分散方式 : カーネルの機能を分割してそれらを各プロセッサで分担させる方式である.

機能分散方式も魅力的な方式であるが, 我々は均一なカーネルを作りたいので, 基本的には負荷分散方式を用いる.

4.4 スケジューリング

システム内には, 同一タスク内の複数のスレッドや, タスクの異なるスレッドが存在するので, これらのスレッドをいつどのような順序でどのプロセッサで実行させるかが重要である.

これをプロセッサからみると, プロセッサの割当てをどのレベルで行うかが問題となる. これには, 以下の2つが考えられる.

(1) タスクを考慮しないでスレッド単位で割当てる.

(2) タスク単位で割当てる.

(1) は, プロセッサごとに異なるタスクのスレッドが実行されることになる. また, タスクの異なるスレッドが入り混じって1つのプロセッサで実行される. 一般に, 同一タスク内のスレッドはお互に協調しながら動作することが多い.

したがって, (1) の方法をとると, スレッドの切り換えはタスクの切り換えをともなうことが多くなるので, アドレス空間切り換えのオーバヘッドが生じる. また, 同一タスク内におけるスレッドの局所性を活かしきれず, TLB (Translation Look-aside Buffer), キャッシュのヒット率が低下すると考えられる. また, ページフォルト頻度が増加する.

一方 (2) は, タスク単位にプロセッサを確保し, それらのプロセッサで同一タスク内のスレッドを並列に実行する.

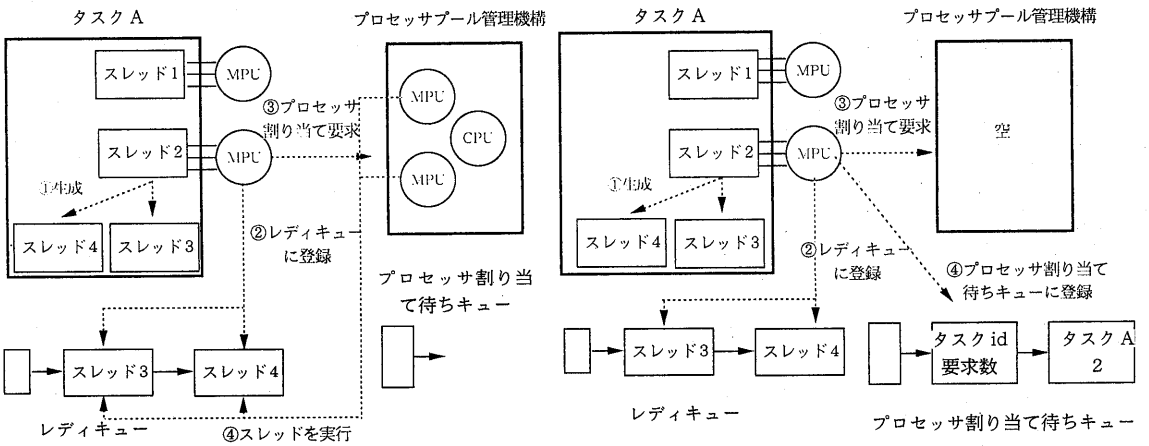
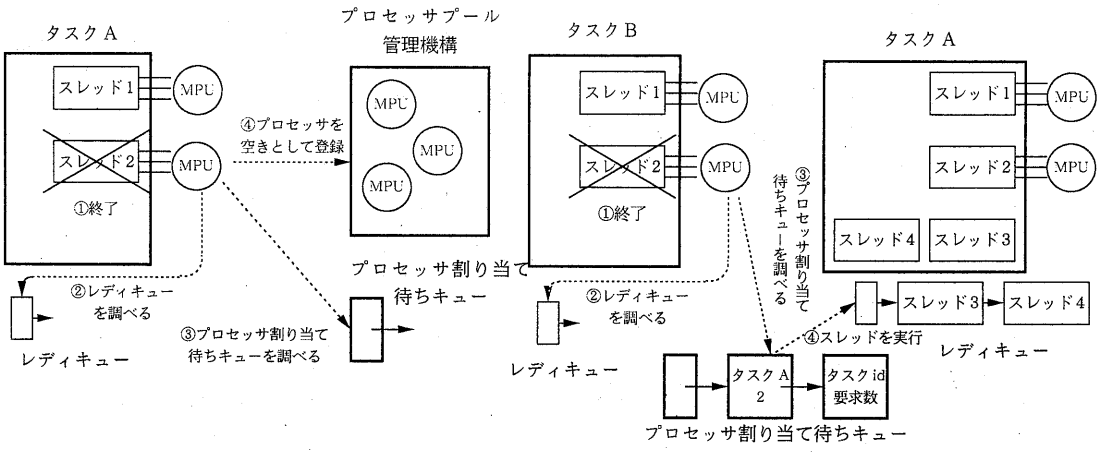


図2 スレッドの実行

*UNIXは米国bell研究所の登録商標である.



(a) プロセッサ割り当て待ちタスクが存在しない場合　　(b) プロセッサ割り当て待ちタスクが存在する場合

図3 スレッドの終了

したがって，スレッド切り換えのオーバヘッドが（1）よりも小さくなる．また，局所性を利用できるので，TLB，キャッシュのヒット率が高くなり，効率的である．以上の理由により，方式（2）を採用する．実現方法の概要を以下に述べる（図2，3）．

（1）スレッドが複数のスレッドを生成すると，カーネルは当該タスクに対応したレディキューに生成したスレッドをつなぐ．さらに,プロセッサ・プールからそれらを実行するためのプロセッサを確保する．

（2）生成されたスレッド数が空きプロセッサ数よりも大きい場合は，残りの数を要求するため，プロセッサ割当て待ちキューに要求数を登録する．

（3）スレッド生成を発行したプロセッサ上のカーネルは，プロセッサ間割り込み機構によって，確保したプロセッサに当該レディキュー内のスレッドを実行させる．

（4）事象待ちなどによりスレッドがサスペンドされると，当該プロセッサは当該レディキューからスレッドを取り出し，実行する．

（5）タスクが終了する場合，タスクがサスペンドする場合，およびスレッド終了時に実行可能なスレッドがない場合は，当該プロセッサを解放する．

タスク内のスレッドのスケジューリングとして，FCFS（First Come First Served）方式，ラウンドロビン，優先度方式，またはそれらを併用したものなど種々の方式が考えられる．タスク内のスレッドのスケジューリングを複雑なものにしてもさほど意味がないと考えられるので，我々は，FCFS方式か優先度方式を採用する．

スレッドの制御はカーネル・レベルで行う．他の方法として，タスクの仮想アドレス空間内に実行時ルーチンの形で埋めこんで行う方法が考えられる．このようにすると，スレッド切り換えのオーバヘッドをさらに小さくできるので非常に魅力的な方法であるが，これは今後の検討課題としたい．

また，空きプロセッサの数だけのスレッドを生成し，それらを同時に実行する方法も考えられる．これも今後の検討課題としたい．

4.5 仮想メモリの管理

スレッドが新たなスレッドを生成した場合や，新たなタスクを生成した場合における仮想メモリの管理方法について述べる．

4. 5. 1 スレッドを生成する場合

仮想アドレス空間はタスク対応に存在し，同一タスク内のスレッドはコード，データを共有する．我々が対象とするメモリ・アーキテクチャ（均一／不均一メモリ・アクセス構成，多重バンク構成）を考慮すると，同一タスク内の複数のスレッドを複数のプロセッサで実行させる場合，スレッドを実行するプロセッサごとにアドレス変換表（セクション表およびページ表），およびコードをコピーするかしないかの選択が考えられる．

以下，これらの選択にともなう問題点を考える．なお，データに関してはコピーするとデータの一貫性を保つためのオーバヘッドが大きくなると考えられるので，データはコピーしないとする．また，スタック領域はアクセス競合の点から分散させてメモリ・バンクにマッピングする．

（1）アドレス変換表をコピーしない場合

アドレス変換表が1つであるので，プロセッサごとに存在するMMUからアドレス変換表へのアクセス競合が生じやすい．また，不均一メモリ・アクセス構成の場合，リモート・アクセス（ネットワークを介したアクセス）となるプ

ロセッサからは変換速度が遅くなるという欠点がある．しかし，これらはプロセッサごとに存在するTLBの効果によって，その軽減が期待できる．また，アドレス変換表が1つであるので，ページアウト，ページインに伴う変換表のエントリの一貫性の問題が生じないという利点がある．

(2) アドレス変換表をコピーする場合

スレッドを割当てられたプロセッサの数だけメモリ・バンク対応にアドレス変換表をコピーする．このようにすると，MMUからのアクセス競合は生じない．また，不均一メモリ・アクセス構成の場合，プロセッサがアクセス距離の近いメモリ・バンク（以後ローカル・メモリとよぶ）内のアドレス変換表を使用すれば，変換が速くなる．しかし，当然コピーするオーバヘッドが生じる．

コピーの場合はさらに，コードをコピーする場合としない場合の2つが考えられる．以下，それぞれの問題点を示す．

(a) コードをコピーしない場合

コードをコピーするオーバヘッドは生じない．しかし，複数のアドレス変換表が1つのコード領域を指すことになるので，コードのページアウト，ページインが生じるとすべてのアドレス変換表のエントリの一貫性を保つ必要がある．これを実現するには，基本的には全アドレス変換表のエントリを書き替えなければならない．また，コードへのアクセス競合が生じる．さらに，不均一メモリ・アクセス構成の場合は，リモート・アクセスするプロセッサからのアクセスが遅くなる．しかし，これらはキャッシュの効果によりある程度の軽減が期待できる．

(b) コードをコピーする場合

メモリ・バンクごとにコピーされたコードを持たせることができるので，コードへのアクセス競合は生じない．不均一メモリ・アクセス構成の場合は，ローカル・メモリへコピーすることによって，コピー以後のコードへのアクセスが速くなる．また，コードのページアウト，ページインは個別的に行なわれるので，これに伴う他のアドレス変換表への影響はない．しかし，当然コードのコピーに伴うオーバヘッドが生じる．

以上各々の利点，欠点をみてきた．アドレス変換表をコピーした場合はそのオーバヘッドが大きいと考えられる．また，コピーしない場合に生じるリモート・アクセスおよびアクセス競合の問題は，TLBおよびキャッシュによりその軽減が期待できる．したがって，我々はアドレス変換表をコピーしない方式を採用する．

4. 5. 2 タスクを生成する場合

(1) 親子関係の定義

タスクは仮想アドレス空間や，ファイルなどの計算機資源を割当てる単位である．タスクを生成する側のタスクを親タスク，生成されたタスクを子タスクと呼ぶ．一般に，親子にどのような関係を持たせるかが問題となる．この関係には，まず大きく分けて，(1) 親のイメージを子が引き継ぐ，(2) 引き継がない，の2つに大別できる．さらに (1) は，親のイメージを引き継いだ後の関係を定義する必要がある．これには，(1−a) 引き継いだ後もそのイメージを親子の間で一貫性を保つ，(1−b) 一貫性を保たない，の2つに分けることができる．

本稿では，上記3つの関係をそれぞれ，(1−a) coherence, (1−b) incoherence, (2) noneとよぶ．本OSでは，これら3つのイメージ継承をページ単位で行うことにより，柔軟な親子関係が定義できる．

(2) 子タスクの仮想アドレス空間生成の実現方法

親タスク内のスレッドがタスク生成を行うと，子タスク用のアドレス変換表を作成する．子タスクの仮想アドレス空間から実アドレス空間へのマッピングは，親タスクのイメージ継承属性 (coherence, incoherence, none) にしたがって行う．リードオンリであるコードは親子で共有するが，メモリ・アーキテクチャを考慮すると，この実現方法として以下の2つが考えられる．

1) 異なるメモリ・バンク上にコードをコピーする．このとき，効率の点からオンデマンドで行う．

2) 親タスクのコードへリモート・アクセスする．

1) の場合は，コードへのアクセス競合が生じない．また，不均一メモリ・アクセス構成の場合は，子タスクを実行するプロセッサのローカル・メモリ上にコピーすることによって，コードへ速くアクセスできる．さらに，アドレス変換表とコード領域が1対1に対応するので，親（子）タスクのコードのページアウト，ページインが生じても，子（親）タスクのアドレス変換表へ影響を与えない．しかし，当然コードをコピーするためのオーバヘッドが生じる．コードのコピーはオンデマンドで行うにしても最低1ページサイズ（4KB）はコピーする必要がある．

2) の場合は，コピーのオーバヘッドは避けられるが，コードへのアクセス競合が生じる．また，不均一メモリ・アクセス構成の場合，リモート・アクセス遅延が問題となるが，これらはキャッシュによってその軽減が期待できる．しかし，ページアウト，ページインに伴うマッピングの一貫性をとる必要がある

ここで，forkシステム・コールの後にすぐexecveシステム・コールがくる場合を考える．execveは元のアドレス空間の上に新たなイメージを上書する．したがって，1) の場合のオーバヘッドは1ページサイズをコピーする時間である．2) の場合は，おもに数回のリモート・アクセス遅延である．この場合，オーバヘッドは1) の方が大きい．通常，forkはexecveを伴うことが多いので，我々は方法2) を採用する．

4.6 カーネルの構成

現在検討を進めているカーネルの構成について述べる．カーネルは，種々の管理機構から構成されるが，このとき，データ構造を各機能ごとに隠蔽化して他の管理機構から直接にはアクセスできないように構成する．以下の機能などから構成される．

（1）物理メモリ管理機構：各プロセッサ対応の物理メモリ（4MB）をページ・フレーム単位で管理する．本機構はコアマップ表を保持する．ページ単位の物理メモリの割当て，解放などの操作を提供する．

（2）実プロセッサ管理機構：スレッドに実プロセッサを割当てる管理を行う．ディスパッチの操作などを提供する．

（3）プロセッサ・プール管理機構：各プロセッサの状態（idle／busy）およびプロセッサ対応に実行されているタスク識別子，スレッド識別子の情報を管理する．状態の書き替えなどの操作を提供する．

（4）プロセッサ割当て待ちキュー管理機構：プロセッサを割当てられていないタスク，および割当てられたプロセッサ数よりもスレッド数が多いタスクを管理する．エンキュー，デキュー，スケジュールなどの操作を提供する．

（5）タスク管理機構：タスク・コントロール・ブロック（TCB）を保持する．タスクの生成，起動，励起，中断，終結，消滅，などに応じて該当するTCBの書き替え，TCBの確保，解放などの操作を提供する．

（6）スレッド管理機構：スレッド・コントロール・ブロック（THCB）を保持する．スレッドの生成，起動，励起，中断，終結，消滅，などに応じて該当するTHCBの書き替え，THCBの確保，解放などの操作を提供する．

（7）仮想メモリ管理機構：仮想アドレスから実アドレスへの変換を行うアドレス変換表を保持する．仮想メモリの生成，消滅 割当て，解放などの操作を提供する．

（8）共有メモリ管理機構：共有メモリに関するアドレス変換表を保持する．共有メモリの生成，消滅，割当て，解放などの操作を提供する．

（9）レディキュー管理機構：実行可能なスレッドを管理するもので，レデイキューを保持する．タスク対応に1つのレディキューを保持する．本機構は，エンキュー，デキュー，スケジューリングなどの操作を提供する．

（10）セマフォ管理機構：セマフォリスト，セマフォ値を保持する．セマフォの生成，消滅，P／V操作を提供する．

（11）メイルボックス管理機構：メイルボックス用のバッファを保持する．メイルボックスの生成，消滅，書きこみ，読みだしなどの操作を提供する．

（12）プロセッサ間割り込み機構

その他，タイマ，割込み処理，事象待ちなどに関する管理機構がある．

5．並列プログラミング言語

現在，C言語に並列性とスレッドの記述を導入した言語と，メッセージ指向のプログラミング言語SERVEを開発中である．本稿では，SERVEについて簡単に述べる．

並列プログラミング言語SERVE[7]の設計にあたっての方針を以下に示す．

1）実行効率よりもむしろ記述性の向上に主眼をおく．

2）メッセージ指向の言語とする．

3）モジュール化を支援する．

5.1 プログラム構成要素

プログラムは1個以上のモジュールから成る．さらに，図4に示すようにモジュールは複数の並列プロセスから成る．

（1）プロセス

プロセスはユーザが意識する並列実行の単位であり，データとそれに対する操作をカプセル化したものである．多次元のプロセス配列を静的，動的に生成できる．

（2）モジュール

密接に関係した複数のプロセスは，ある1つの機能を遂行するものとしてモジュールにまとめられる．規模の大きなプログラムを構築するためには，その機能ごとにコーディング，コンパイル，テストできるような構成要素に分割することが望ましい．モジュールはその単位となるものである．また，モジュールは他のモジュールとの間で名前の可視性を表現するための仕様部を持つ．ここでは，手続き名やプロセス名およびエントリ名が宣言される．

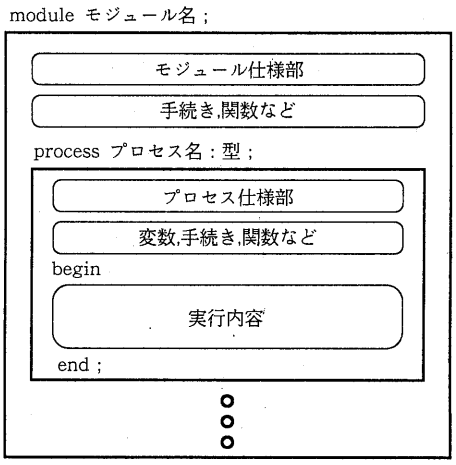


図4 モジュールとプロセスの構造

5.2 プロセス間通信

柔軟なプロセス間通信を提供する．

(1) 通信プリミティブ

SERVEの提供する通信プリミティブは，基本的に非同期式メッセージ通信である．非同期式は，同期式に比べ効率的であり，不必要な同期を排除できる．メッセージの送受信は以下のように行われる．

送信：
send ポート名（実引数リスト）[to 受信プロセス]
受信：
receive ポート名（仮引数リスト）[from送信プロセス]

ポート（port）は，メッセージを蓄えるバッファであり，メッセージの構成を指定する．ここで，ポートが空の場合は，受信操作はメッセージが到着するまで封鎖される．

またSERVEは，プロセス間のクライアントーサーバ関係をプログラムするために，send／waitおよびreceive／reply構文により同期式メッセージ通信も行える．

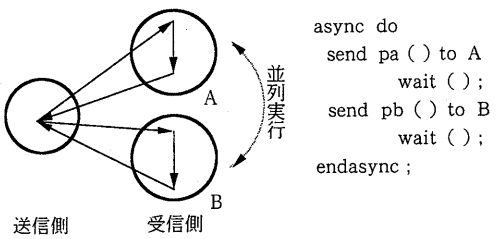
(2) 対多メッセージ通信

ポートの宣言は，プロセス内部でローカルに，またはプロセス外部でグローバルに行える．ローカルに宣言された場合，ポートからのメッセージの受信はそのプロセスによってのみ行われる．これに対し，グローバルに宣言されたポートは，それを輸入（import）した複数のプロセスにより共有される．したがって，グローバルなポートへのメッセージは，それを輸入した任意のプロセスへの通信となる．

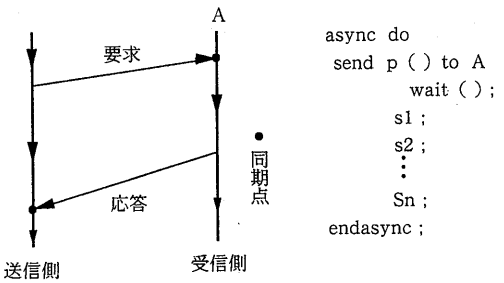
また，グローバルなポートはモジュー間にまたがる通信にも使用される．

(3) 非同期文async／endasync

非同期文は，同期式メッセージ通信とともに用いる．これは，同期式メッセージ通信の応答待ちのための同期点をendasyncのところまで移項する．図5の(a)，(b)に示すような並列手続き呼出しあるいは非同期メッセージ交信の使い方ができる．



(a) 並列手続き呼び出し



(b) 非同期メッセージ交信

図5 非同期文

6．おわりに

我々が開発を進めている「可変構造型並列計算機」のOSの開発計画，および第一次OSの概要を述べた．現在，カーネルの各機構の詳細化を進めている．第一次OSのファイル・システムについては，本システムのフロントエンド・プロセッサであるSun－4のものをできる限り流用したいと考えており，その詳細を検討しなければならない．今年度末を目標に第一版を完成させたい．

謝辞

我々と共に開発を進めている森，蒲池，岩田，甲斐，草野，恒冨の各氏，および，日頃ご討論頂く富田研究室の皆様に感謝致します．また，貴重な御意見を頂いた東京大学，田胡和哉助手に感謝致します．

参考文献

(1) 富田眞治：並列計算機構成論，昭晃堂（1986）．

(2) M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, and M. Younge：”Mach：A New Kernel Foundation for UNIX Development”, Proceedings of USENIX 1986 Summer Conference, pp. 93－112 (1986).

(3) D. R. Cheriton：”The V Distributed System”, Communication of the ACM, vol. 31, no. 3, pp. 314－333 (1988).

(4) 高野，田胡，益田：”プロセス・ネットワークにおける分散型オペレーティング・システムの設計”，情報処理学会論文誌，vol. 29, no. 4, pp. 359－367 (1988).

(5) 並木，高橋：”OS／omicronのマルチプロセッサ化の構想”，信学技報，CPSY88－38 (1988).

(6) 森，濱口，村上，福田，末吉，富田：”可変構造型並列計算機のPE間メッセージ通信機構”，情報処理学会「並列処理シンポジウム JSPP'89」，pp.123－130 (1989).

(7) 福澤，廣谷，福田，村上，末吉，富田：”可変構造型並列計算機のソフトウエア”，情報処理学会九州支部第3回研究会，pp.29－38 (1989).