オペレーティング・システム　45 － 8
（１９８９．１１．１１）

分散並列システムの構成とその実現方式

田胡　和哉　　中山　泰一
東京大学工学部

小型軽量なフロントエンド計算機と、高い処理能力をもつサーバ計算機をネットワークによって結合したシステムを制御するオペレーティング・システムの実現方式につい述べる。この形式のシステムでは、オペレーティング・システムは、異機種間結合による機能分散を図る機能、および、計算機内部において分散／並列処理を行うことによって高い処理性能を実現する機能が必要である。機能分散機能を実現するために、資源アクセスに関する共通プロトコルを用いる方式を提案する。さらに、汎用の並列処理環境を実現するうえで問題となる、プロセス管理のオーバヘッドを削減するために、プロセスの遅延生成方式を提案する。

DESIGN OF A DISTRIBUTED SYSTEM WITH PARALLEL PROCESSING NODES

Kazuya Tago and Yasuichi Nakayama

Departement of Mathematical Engineering and Information Physics,
Faculty of Engineering, University of Tokyo

4-3-1 Hongo, Bunkyoku, Tokyo 113, Japan

Design of the distributed system which consists of lightweight frontend machines and powerful server machines is stated. The operating system of the system must implement cooperation between frontend and server machines and parallel processings within server machines. The cooperation is implemneted on the basis of the common protocol which prescribes the access method of remote resources. An effective parallel processing environment is realized by the lightweght process kernel which implements delayed process creation mechanism. The mechanism does not create a process at requested time but re-allocate an idle process to the request to redue process creation overhead.

1．まえがき

マイクロプロセッサの出現により、計算機システムの形態が大きく変化している。特に、処理の分散／並列化の傾向が著しい。たとえば、大学の研究室においても、ワークステーションを結合したLANシステムや、多数のノードからなるWANが盛んに利用されている。また、従来の小型機に代わってマイクロプロセッサを用いたマルチプロセッサ・システムの利用が広まりつつある。オペレーティング・システムに関しても、このような、計算機環境の変化に対応することが求められている。

従来、分散／並列処理システムは、LAN，WAN等の結合媒体の種類によって分類されてきた。オペレーティング・システムも、この分類に基づいて機能設計が行なわれてきた。しかしながら、最近では、このような分類が実情にそぐわなくなりつつある。たとえば、高速度のWANはLANと同様の目的に利用されている。また、階層構造を持つきわめて大規模なLANが出現している。そこで、個々の実現技術のみならず、システム全体の構成をも含めて検討してみることが有用である。

本稿では、現在の、ワークステーションを主体とした分散システムではなく、小型軽量のフロントエンド計算機と、計算機内部での分散／並列処理により高い処理能力をもつサーバ計算機からなるシステムを実現する方式について検討する。このようなシステム用のオペレーティング・システムは、フロントエンド／サーバ間での分散処理を制御する機能、および、サーバ計算機内での分散／並列処理を制御する機能を備えていなければならない。ネットワーク上での分散処理の制御方式、および、細粒度の汎用並列処理環境の実現方式を提案する。

2．分散システムの設計

現在のハードウェア技術の水準を考慮に入れて計算機環境の設計を行ない、その実現に必要なシステム技術について検討する。

（1）分散システムの形態

会話型の計算機システムを構成する方式として、高機能の等質な計算機を結合する方式と、フロントエンド／サーバ構成をとる方式がある。一般には、等質な計算機を用いる方式の方が良好なマンマシン・インタフェースを得やすい。一方、この方式では、処理能力に無駄が生じやすく、高い分散透明性を実現してネットワーク上で負荷の分散を図る必要がある。

ネットワークを利用して負荷分散を実現するためには、プログラムの実行環境の分散透明性を高めること、および、通信効率を高めることが必要になる。これに対応して、すべての計算機上に同一の実行環境を実現する機能を持つ、分散型のオペレーティング・システム[2]が開発されている。分散型オペレーティング・システムは高い分散透明性を実現しているために、ネットワーク中の各計算機には同一仕様のオペレーティング・システムが動作していることを仮定している。また、効率の点から、専用の通信プロトコルを用いることが多い。

フロントエンド／サーバ構成をとる方式では、計算機間で機能の分担が図られているために、ネットワークを経由した負荷分散の必要性は少ない。そこで、完全な分散透明性が実現されていなくともよい。また、通信機構の性能に関する要請も、よりゆるやかである。その一方において、サーバ計算機が単一とは限らないこと、フロントエンド計算機用のオペレーティング・システムとサーバ計算機用のシステムがソフトウェア資産の関係から同一とは限らないこと等の理由により、異機種間結合を実現している必要がある。

（2）システム構成の検討

メモリ容量の増加、表示装置の改良、カスタムLSIの普及等により、超小型の計算機を用いても良好な使い勝手が得られるようになっている。利用者が直接用いる計算機は、使い勝手が同一であれば、より小型軽量であることが望ましい。また、高い処理性能を得るためには、ネットワークによる分散構成をとるよりも、集中的な構成をとる方が有利である。そこで、図1に示すように、フロントエンド計算機と、複数のプロセッサからなる高能力のサーバ計算機からなる系を用いて計算機環境を実現することを考える。

フロントエンド計算機は、たとえば、表1に示すようなハードウェアを備えていることが求められる。メモリ容量、および、表示装置の表示密度は、できる限り高くとる。これと同時に、持ち運びが可能であることが望ましい。サーバ計算機と結合されて運用されることを仮定すれば、多様な入出力装置、および、インタフェースを備えている必要はない。また、計算能力についても、特に高くなくともよい。さらに、二次記

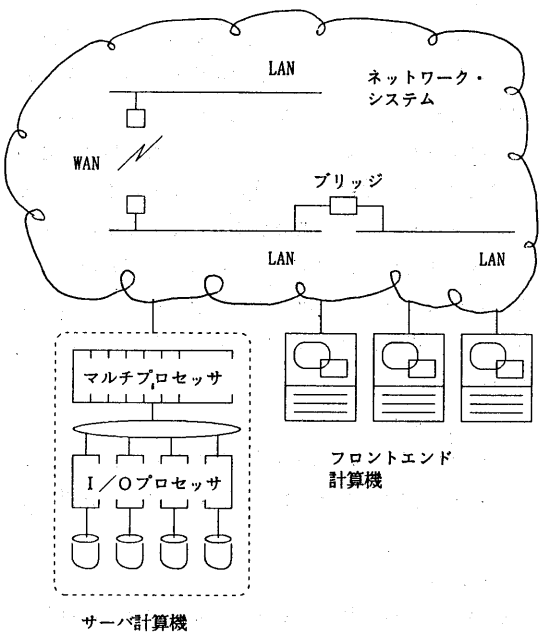

図1　フロントエンド／サーバ構成による計算機環境

億装置も不要であり、バッテリ・バックアップ付きのメモリ、および、ネットワーク機能によって代えることができる。

表1　フロントエンド計算機の仕様

メモリ容量	4Mバイト 半分バッテリバックアップ
表示密度	1000＊1000
周辺機器	マウス LANインタフェース モデム

この規模の計算機は、ノート型の形状をとることができる。

サーバ計算機は、高い処理能力を提供することを目的としている。これは、入出力専用プロセッサの利用、マルチプロセッサによる負荷分散および並列処理によって達成される。並列処理による改善は、計算処理の並列化と、入出力処理の並列化による。

サーバ計算機とフロントエンド計算機は、LANおよびWANから構成されるネットワーク・システムを経由して結合する。

(3) オペレーティング・システムの機能

図1に示したシステムは、多種の通信媒体から構成される多種の計算機から構成されており、各計算機のオペレーティング・システムは通信媒体の種類に依存しない分散処理環境を実現する機能を備えている必要がある。たとえば、フロントエンド計算機を屋外に持出しても、公衆回線を経由して、同一の環境が実現できるようにする。

フロントエンド計算機上では、ウインド管理、エディタ、仮名漢字変換、デバッグ環境等、会話的なジョブを実行する。ここでは、コマンドに対するレスポンスが最も重要である。一方、複数の利用者が利用することがないので、保護機能、スケジューリング機能等は簡略なものでよい。したがって、オペレーティング・システムの規模をできうる限り縮小し、効率の改善を図る必要がある。たとえば、画面操作、および、キーボード入出力管理は、ウィンドウ・システムの存在を仮定し、オペレーティング・システムは最小限の機能のみを実現すればよい。

サーバ計算機のオペレーティング・システムは、計算機内での機能分散、負荷分散、および、利用者プログラムの並列処理の制御を実現している必要がある。また、システム機能自体も、並列処理を利用して実現されていることが望ましい。

オペレーティング・システムに求められる機能について概観した。これらのなかでも、特に、サーバ計算機内の分散／並列処理の実現方式、および、異機種の計算機を結合したネットワークの制御機構の実現方式の開発が重要な課題である。

3．OSOM

ネットワークの制御方式について検討する。多種の通信媒体からなるシステムでは、専用の通信プロトコルを用いるのではなく、汎用のプロトコルを用いる必要がある。現在、通信プロトコルの統合化を目的とする整備が続けられている。これにより、ネットワークの規模や通信媒体によらず、単一の通信プロトコルを用いて通信が行なえるようになると期待できる。フロントエンド／サーバ構成用のオペレーティング・システムの分散処理機能は、この上に構築する。

異機種の計算機間の結合においては、オペレーティング・システム核の構造が同一であることは期待できない。そこで、分散環境の実現において、核の実現方式ではなく、資源アクセスのプロトコルに基づいた設計をとる必要がある。このプロトコルは、プログラム間通信、ファイル・アクセス、および、コマンドの実行方式を含む。このプロトコルは、汎用の通信プロトコルを利用して実現される。これを、OSOMA(Open Systems Object Management Architecture)と名付け

る。

OSOM機能は、核内部、もしくは、デーモンプロセスによって実現される。たとえば、図2に示すように、フロントエンド計算機上では核内で実現し、サーバ計算機上ではデーモン・プロセスを用いる方式が考えられる。これにより、既存の計算機も比較的容易にサーバ計算機として利用することができる。デーモン・プロセスは、フロントエンド計算機のオペレーティング・システムからの要請により、サーバ計算機のファイル・アクセス、および、コマンドの実行の制御を代行する。

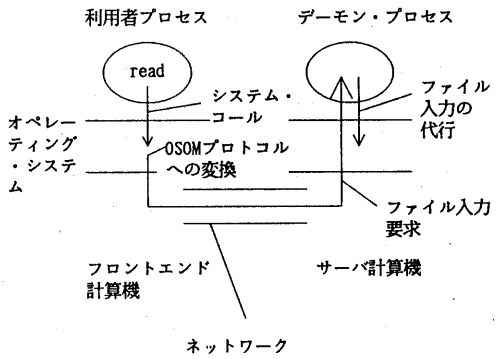

図2　OSOMの実現

4．サーバ計算機における並列処理の管理

4．1　マルチプロセッサにおける並列処理

サーバ計算機は、マルチプロセッサ構成をとる。最近では、数十個程度のプロセッサからなる共有メモリ型のシステムが実現できるようになっている。このようなハードウェアを利用して、利用者プログラムにおける並列処理を実現する方式について述べる。

複数のプロセッサを有効に利用するためには、プログラムの並列度が、常に、割当てられているプロセッサ数より多い必要がある。このため、高並列のシステムでは、平均すると、10^2オーダの並列度を保つ必要がある。通常の規模の計算では、これは、ほぼ手続きの粒度に対応する。すなわち、並列型オペレーティング・システムの機能として、限られた資源の範囲内で、手続きと同程度の粒度による並列処理を効率良く実行できるようにする機能が重要である。

マルチプロセッサを用いた並列処理において利用されているプログラムの記述方式として、プリミティブを用いてプログラマが直接制御するもの、並列処理用の言語構造を用いるもの[1]、既存の言語を自動的に変換するもの、および、プログラムを大域的に複数の単位に分割して複数の利用者プログラムを用いて並列処理を行うものに大別できる。上記の粒度による並列処理を構造的に記述するためには、並列処理用の言語構造を用いる方式が最も適している。言語構造、および、その実現法として、以下のようなものがあげられる。

1)　par_begin／par_end

par_beginからpar_endの間の文を並列に実行する。各々の文にプロセスを割当てることによって実現できる。

2)　do_all

繰返し実行のかわりに並列実行を行なう。繰返しの各々を、別個のプロセスを用いて実行する。

3)　並列オブジェクト指向

図3に示すように、従来のオブジェクト指向言語では、メッセージの受渡において、手続呼出しと同様に、同期的な機構を用いていた。これに、非同期的なメッセージの受渡機構を付加することにより、処理の並列化が図れる。同期的なメッセージ呼出し系列は、全体で一つのプロセスによって実現できる。非同期的なメッセージの発行を行なった時点で新たなプロセスを割当てればよい。

4)　並列Lisp

Lispでは、関数の評価の並列実行が比較的容易であ

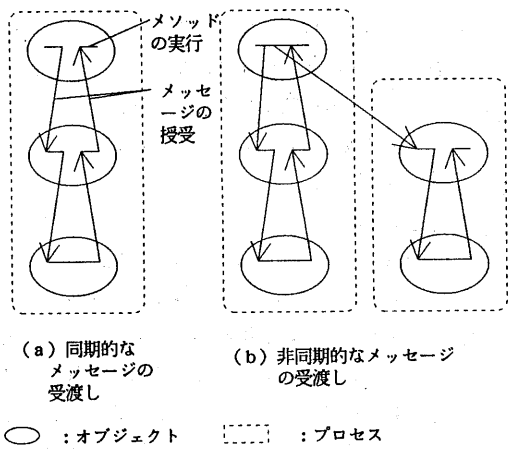

図3　オブジェクト指向言語の実行環境

る。たとえば、並列実行の制御のために、future型等の方式の提案が行なわれている。並列に評価可能な関数の各々にプロセスを割当てることにより、並列処理が実現される。

4．2　核

汎用の並列処理環境を実現する方法として、プロセスを単位とする機構が用いられている。プロセス・ディスパッチ、および、プロセス生成、同期等のプリミティブを実行する核を実現し、記述言語の実行時環境として利用する。

プロセスの生成においては、pcb(process control block)等の制御情報を保持する領域の割当て、初期化、および、スタック等の作業領域の割当てが行なわれる。同期においては、待ちキューの操作等、管理情報の操作、および、状態の切り換えが行われる。

並列処理環境を実現するうえでの今一つの問題として、記憶領域の管理があげられる。並列度が高く、多数のプロセスを用いるプログラムでは、作業領域、特に、スタックに割当てられる領域が無視できなくなる。また、オーバフローのチック等、管理のオーバヘッドが少なくない。

手続と同程度の粒度による並列処理を実現するためには、プロセスの生成、同期が、手続の呼出と同程度の時間で実行できる必要がある。マルチプロセッサ用のオペレーティング・システムでは、効率的な並列処理環境の実現を目的として、軽量なプロセス(light-weight process)を実現する核を提供している。たとえば、MACHシステムでは、記憶領域の割当ての単位であるタスクと、プロセッサの割当て単位であるスレッドを別個に実現しており、スレッドのスケジューリング、および、同期をシステム核内で実現している[5]。単一の論理アドレス空間を用いることにより、プロセス管理のためのオーバヘッドの削減に努めている。しかしながら、手続呼出し機構に較べれば、そのオーバヘッドは遙かに大きく、この粒度の並列処理には適していない。

4．3　プロセスの遅延生成方式

プロセス生成、消滅が頻繁に行われる環境において、プロセス管理のためのオーバヘッドを削減する方式を提案する。プロセス生成の要求時に実際にプロセスを生成するのではなく、生成要求だけを保存し、少数のプロセスを繰返し用いることにより、実際にプロセスを生成する機会を減ずる。

図4に、提案方式による並列処理環境を示す。複数のプロセスと、プロセスの生成要求を保持する生成要求キューから構成されている。生成要求には、実行すべき手続きの開始番地、および、引数が記入されている。プロセス生成を要求するプリミティブにより、生成要求キューに要求が一つ登録される。プロセスは、生成要求キューから要求を一つ取り出し、そこに記入されている手続きを実行する。手続きの実行が終了してもプロセスは消滅せず、次の要求を取り出してそれを実行する。

もし、同期命令によって関数の実行が中断することがなければ、この系における必要なプロセス数は、プログラムの並列度の最大値かプロセッサ数のどちらか

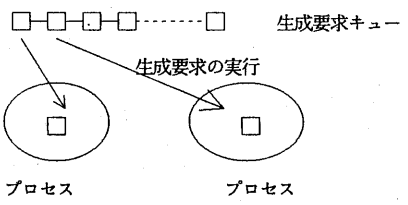

図4　遅延生成方式による並列処理

小さい方の値に等しい。たとえば、単一プロセッサ・システムでは、1つのプロセスを繰返し利用すればよい。プログラム中の同期命令によってあるプロセスの実行が中断したときには、新たなプロセスを生成する。実行を中断したプロセスは、内部状態を保持するために、実行を再開するまで放置する。プロセッサの割当て数が増加した場合にも、プロセスを生成する。

プロセスの生成消滅は、プログラムからの要請ではなく、仮想記憶システムにおけるページ枠の割当てと同様に、プログラムの特性、および、システムの負荷状態からシステムが決める。実際には、あらかじめいくつかのプロセスを割当て、さらに実行時に必要に応じてまとめてプロセスの割当てを行なう。実行すべき仕事が無くなった場合には、プロセスの実行を中断するか、あるいは、システム負荷が高い場合には消滅させる。

以上のような方法により、プロセス生成要求が、生成要求キューの登録、および、削除のオーバヘッドのみで実現できる。また、実際のプロセス生成の頻度と領域の消費の削減が図れる。また、並列処理にともなう資源の消費を、システムの負荷状態にあわせて制御することができる。

4．4　軽量核の設計

遅延生成方式に基づいた、軽量なプロセスの実行環境について述べる。この実行環境を、軽量核と名付ける。軽量核の機能は、利用者プログラムにリンクされて利用者プログラム・モードで動作する部分と、オペレーティング・システム核の機能を総合して実現され、共有メモリ型高並列マルチプロセッサ・システム用のオペレーティング・システムの一部として利用できるように設計されている。これとは別に、既存の共有メモリ型プロセッサ上で、利用者モードで動作する部分のみからなるより簡略な実現法による軽量核を実現した。

軽量核は、利用者プログラムに対して、表2に示すプリミティブを提供する。これらのプリミティブは、C言語から手続き呼びだしの形式によって利用することができる。また、並列記述言語の実行時環境として利用できる。

Pallocによって、生成要求キューに並列実行要求が登録される。引数のflagによって、割当てるプロセスの数を指定する。flagが0のときには、1つだけ割当てられる。flagが1のときには、割当てたプロセスのうちのいずれか1つの実行が終了するまで、可能な限りのプロセスが割当てられる。起動された手続きが終了すること、あるいは、実行している手続きが陽にPexitを発行することによって並列実行が終了する。Ppop, Pvopにより、並列実行単位間のPV命令が実現される。

表2　軽量核のプリミティブ

名前	機能	引数
Pinit	全体の初期化	
Palloc	並列実行の開始要求	手続きの先頭番地 手続きへの引数 flag
Pexit	並列実行の終了	セマフォ番号
Ppop	P命令の実行	セマフォ番号
Pvop	V命令の実行	セマフォ番号

さらに、Pexit命令の引数にセマフォ番号を指定することによってもV命令が実行される。これによって、fork-join形式の制御を実現する。

オペレーティング・システムは、単一の論理アドレス空間を共有する複数の利用者プロセスを生成する。これらの利用者プロセスが、分担して軽量なプロセスを並列に実行する。複数の利用者プロセスが関連して単一のプログラムを実行するので、以下では、全体をジョブとよぶことにする。軽量核の実現において、利用者プロセスの生成、および、軽量なプロセスの生成を遅延実行する。

軽量核は、ジョブの論理アドレス空間に配置される、生成要求キュー、実行すべき仕事が割当てられていない軽量なプロセスを保持する空きプロセス・キュー、軽量なプロセスの制御情報等を中心として構成されている。軽量なプロセスのスタック領域も、この論理アドレス空間上に割当てられる。軽量なプロセスの切り換えは利用者プログラム・モードで行なわれる。

利用者プロセスは、生成要求キューを参照し、要求があれば、軽量なプロセスを空きプロセス・キューから割当て、それを実行する。割当てるべき軽量なプロセスが存在しないときには、新たに生成する。実行が終了すると、新たな生成要求を取り出し、要求を実行する。生成要求がなければ、利用者プロセスは、軽量なプロセスを空きプロセス・キューにもどして実行を中断する。

軽量なプロセスは、それぞれ、システム・コールを発行することができる。単一プロセッサ・システムでは、同期的なシステム・コールが用いられてきた。このインタフェースをそのまま継承すると、システム・コールの発行時に利用者プロセスが停止し、プロセッサの割当て数が減少する。これを防ぐために、利用者プロセスの遅延生成を行う。オペレーティング・システムがジョブあたりの実効的なプロセッサの割当て数を測定し、それが低下した場合には新たな利用者プロセスの生成を行うことにより、利用者プロセス数の最適化を図る。単一の並列処理プログラムを実行する利用者プロセスは、単一の論理アドレス空間を共有するため、システム内の制御表の割当てのみが行われる。

利用者プロセスの遅延生成を実現するために、dforkシステム・コールを設ける。dforkシステム・コールによって、オペレーティング・システムに、複数の利

用者プロセスの割当てを要求するとともに、割当てられたプロセスの実行開始番地を指定する。このシステム・コールは、処理の開始時に1回だけ発行する。実際の割当ての時期は、利用者プログラムからは予想できない。

利用者プロセスは、実行すべき仕事が無くなると、dslpシステム・コールを発行する。これにより、プロセッサ割当てのスケジューリング・リストからはずされる。また、システムの負荷によっては、そのままプロセスが消滅する。

4．5　スケジューリング方式

サーバ計算機は、並列処理を行なう複数の利用者プログラムを同時に処理する。このため、ジョブに対するプロセッサ割当てのスケジューリングを行なわなければならない。プロセッサ割当ての制御方式として、単一プロセッサ・システムと同様に、タイムスライス・スケジューリングと、イベント・ドリブン・スケジューリングが考えられる。並列処理プログラムでは、同一のジョブに属する利用者プロセスのスケジューリングを別個に行うことには問題が多い。たとえば、際どい領域にあるプロセスの実行をタイムスライス・スケジューリングによって中断すると、際どい領域に入ろうとして待ちあわせを行っている他のプロセスの実行を阻害する。TS命令を用いたポーリングによって排他制御が実現されていれば、多量のプロセッサ時間を浪費する。そこで、並列処理全体の実行を一度に中断する。

プログラムの実行全体を中断するのではなく、プロセッサの割当て数を一部削減する場合には、プログラムの同期点にあわせてプロセッサのとりあげを行なわなければならない。このために、利用者プロセスは、同期点をオペレーティング・システムに知らせる必要がある。このような制御は、2つのカウンタを利用者プロセスとオペレーティング・システム間で共有することによって実現できる。それぞれ、RカウンタとAカウンタと名付ける。Rカウンタは、オペレーティング・システムがジョブにプロセッサの開放を要求するためのカウンタである。軽量なプロセスの実行の中断、終了ごとに、このカウンタの値を参照する。カウンタの値が負であれば値が1増加するとともに、トラップが発生する。これによって、オペレーティング・システムはプロセッサを得ることができる。Aカウンタは、ジョブがオペレーティング・システムに新たなプロセッサの割当てを要求するためのものである。Pallocのたびにカウンタを1増加し、その実行を開始するたびにカウンタの値を1減ずる。Rカウンタの値が0でなければ、オペレーティング・システムは、利用者プロセスの起動、もしくは、新たな割当てによってプロセッサの割当てを行なう。pslpコールは、これ以上プロセッサの割当てを必要しないことを意味するので、これが発行されるとオペレーティング・システムによってカウンタの値が0にもどされる。

オペレーティング・システムは、Aカウンタの値を参照して、プロセッサの割当て量を決める。また、Rカウンタの値を時間によって積分した値が、プロセッサの開放の要求と実際の開放とのずれを表わしている。ずれの大きいものに対しては、ジョブ全体に対するタイムスライス・スケジューリングを行なう。

4．6　アーキテクチャによる支援

軽量核を実現するうえで、スケジューリング用のカウンタをハードウェアによって実現するとともに、記憶管理に関するアーキテクチャ上の支援を実現することが望ましい。プロセスを多数生成する場合、スタック領域の管理が重要な問題となる。たとえば、軽量なプロセスの実現において、スタックを個別に保護することは、記憶管理ハードウェアのマッピング・テーブルが複雑になり、管理のオーバヘッドの増大をまねく。また、プロセスの切り換えに、オペレーティング・システムの介入が必要になる。この管理は、図5に示すように、利用者プログラム・モードで利用することができる前置的なアドレス・マッピング・ハードウェアを設け、論理アドレス空間の一部を他の論理アドレス空間にマッピングすることによりより容易に実現できる。プロセスの切り換えごとにスタック領域のマッピングを変更することにより、スタック領域の保護が実現できる。また、領域の拡張による再配置も容易に実現できる。さらに、プロセスごとのヒープ領域の管理にも利用できる。このハードウェアは、論理アドレス空間内のマッピングを変更するだけなので、オペレーティング・システムの記憶管理機構とは独立に利用することができる。

5．現状

(1)　多段結合ネットワークを用いた共有メモリ型マルチプロセッサ・システムの開発

現在、筆者らの属する東京大学工学部森下研究室では、多段結合ネットワークを用いた共有メモリ型マルチプロセッサ・システムの開発を行っている。これは、オメガ・ネットワークによってプロセッサとメモリ・バンクを結合したシステムであり、高い並列度を持つ共有メモリ型マルチプロセッサ・システムを実現できる特徴をもつ。

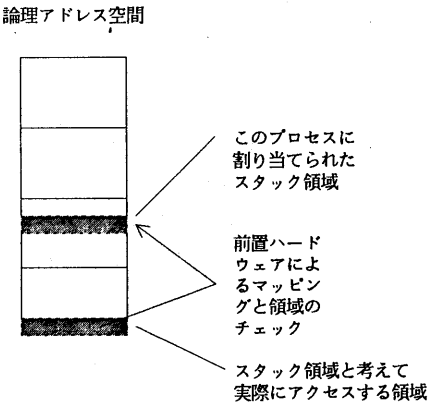

図5　マッピング・ハードウェアを用いたスタックの管理

このシステムは、画像処理、および、画像認識処理を行う専用サーバとして動作するように設計されている。雑音除去等の画素単位の前置処理、特徴抽出等の数値計算、抽出された特徴に基づく画像認識等の記号処理を、この計算機上で一貫して実行することを目的としている。このため、異なる原理に基づいて設計された複数の並列記述言語を実行することが必要である。これらの言語の並列処理環境として、遅延生成方式による核を利用することを検討している。

(2)　マルチプロセッサ用オペレーティング・システム

現在、低並列の共有メモリ型マルチプロセッサ・システム上で、プロセス・ネットワーク方式によるオペレーティング・システムの開発を行っている[6]。プロセス・ネットワーク方式[3]は、通信で結合された軽量なプロセスの多数の集合によってシステムを構成する方式である。これまでに、この方式を用いて、実用的な分散型オペレーティング・システムが実現できることが検証されている。

プロセス・ネットワーク方式は、構造的な記述が可能であること、システム内部の並列度を高くできること、機能分散型のハードウェア容易に対応できること等、サーバ計算機用のオペレーティング・システムの実現に適した性質を持つ。

開発しているシステムは、UNIXシステムとシステム・コール・レベルで同一の外部使用を持ち、従来のアプリケーション・プログラムをそのまま実行できる。この上に、軽量核を実現する予定である。

6．むすび

分散／並列を指向したオペレーティング・システムの構成、および、実現方式の提案を行った。現在、既存のマルチプロセッサ上で、遅延生成方式による軽量核の評価を行なう準備を進めており、定量的な評価を行なうことが今後の課題である。

参考文献

1)　Filman, R.E. and Daniel P.Friedman(雨宮、尾内、高橋訳)：協調型計算システム、マグロウヒル(1984).

2)　Tanenbaum, A.S. and R.V.Renesse : Disributed Operating Systems, ACM Cmputing Survey, Vol.17, No.4, pp.419-470(1986).

3)　田胡，益田：オペレーティング・システムの論構造記述に関する一試み、情報処理学会論文誌、Vol.25, No.4, pp.524-534(1981).

4)　田胡、出口、森下：多段結合ネットワークを用いたマルチプロセッサ・システムの実現方式の提案、信学技報、Vol.89、No.167、pp.13-18 (1989).

5)　Tevanian, A.et al. : MACH Threads and the Unix Kernel: The Battle for Control, CMU Technical Report(1986).

6)　中山、田胡：軽量なプロセスの集合を用いたマルチプロセッサ用オペレーティング・システムの設計、情報処理学会第38回全国大会講演論文集、pp.809-810 (1989).