データベース・システム 73－2
オペレーティング・システム 44－2
（１９８９．９．８）

データベースの並列処理を支援するオペレーティング・システムの基本機能

新城 靖，清木 康
（筑波大学 電子・情報工学系）

バス共有型マルチプロセッサやシングルプロセッサが高速ＬＡＮで結合された並列・分散処理環境において、データベース処理のように、高い並列性が内在し、独自の資源割当ての最適化を行う必要があるアプリケーションを対象にしたオペレーティング・システムの基本機能について述べる。本システムの提供する遠隔カーネル・コール機能により、各アプリケーションは、ネットワーク上の資源を利用することができる。アプリケーション・レベルの軽量プロセスであるマイクロプロセス、および、システム・レベルの軽量プロセスである仮想プロセッサの２種類の軽量プロセスの利用により、高速なプロセス間通信とコンテキスト切換えを実現する。さらに、データベース処理において重要となるディスク・ブロックの先読み、および、入出力とデータ処理の重ね合わせも軽量プロセスにより実現する。ネットワーク通信機能として、大量データの転送に向いたデータグラム・サービスを提供する。

Basic Functions of an Operating System for Supporting Parallel Processing of Databases

Yasushi SHINJO and Yasushi KIYOKI
Institute of Information Sciences and Electronics,
University of Tsukuba

In this paper, we present basic functions of an operating system which supports parallel processing of databases in parallel and distributed processing environments. In those environments, shared-bus multiprocessors and single-processor workstations are connected to a high speed network. In database processing, it is important to realize the optimal resource allocation and to exploit parallelisms. In our system, a remote kernel call function is provided to support utilization of remote resources in the network. We introduce two kinds of lightweight processes (lwp's) which are called microprocesses and virtual processors. The microprocesses are used as lightweight processes of the application level, and the virtual processors are used as lightweight processes of the system level. By using those lwps, high speed interprocess communication and context switching can be implemented. Those lwps make it possible to prefetch disk blocks and to overlap I/O and CPU processing. Furthermore, as a network communication facility, this system provides a datagram service which is suitable for massive data transfer.

1．はじめに

マルチプロセッサや高速のLAN(Local Area Network)の普及にともない、並列・分散処理環境を利用する試みが数多く行われてきた。例として、プログラミング言語により並列処理や分散処理を記述し、利用する方法や、分散型オペレーティング・システムを構築することにより利用する方法があげられる。前者は、例えばＡｄａやＣＯＮＩＣ［13］のような言語を用いて、明示的に並列処理・分散処理を記述する方法である。関数型言語、論理型言語、オブジェクト指向言語などによりプログラムを記述し、コンパイラにより並列性を抽出する方法も提案されている。プログラミング言語による方法は、１つのアプリケーション内の処理の記述には優れているが、他の言語で記述されたアプリケーションとの接続には向いていない。

並列・分散処理環境を利用する他の試みとして、分散型オペレーティング・システムを構築し、その上で従来の逐次的なアプリケーションを実行することで利用する方法があげられる。分散型オペレーティング・システムの上で動作するアプリケーションは、ネットワークの構造を意識することなく、ネットワーク上に分散した資源を利用することができる（分散透明性）［16］。ゆえに、従来のシングルプロセッサ用のアプリケーションを変更することなく分散環境を利用することができる。分散型オペレーティング・システムの上で並列処理を実現する場合、コマンド・レベルの並列処理は容易に実現される。しかしながら、より細かいレベルの並列処理の記述を行う場合、分散透明性や負荷分散など分散型オペレーティング・システムの提供する高度な機能は不用となるばかりでなく、かえって並列性の抽出の妨げになる場合もある。なぜならば、ネットワーク上の資源の位置やプロセッサの境界は、並列処理を行う上で重要な情報となるからである。

このように、上記のいずれの方法も並列・分散処理環境の利用に関して不十分な点がある。そこで、我々は、これらの点を解決するオペレーティング・システムの設計を行っている。本システムの対象とする並列・分散ハードウェアは、図１に示すように、バス共有型マルチプロセッサやシングルプロセッサが高速ＬＡＮで結合されたものである。本システムでは、アプリケーションを次の２つに分類し、上記の問題を解決することを試みる。

（１）データベース処理のように、負荷が大きく、高速化のために並列処理が有効なもの。内部にそれ自身として高い並列性をもつ。

（２）処理時間が短く、並列処理の効果が小さいもの。内部の並列性は低い。

本システムは、（１）のアプリケーションに対して、下位層の並列・分散ハードウェア、すなわち、バス共有型

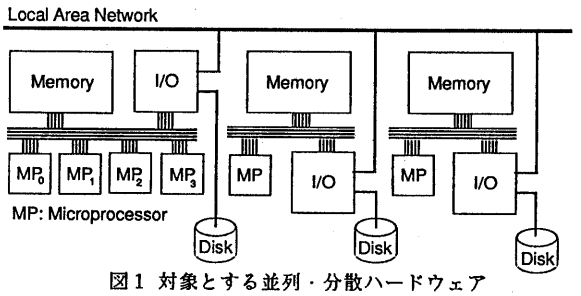


図１ 対象とする並列・分散ハードウェア

マルチプロセッサやＬＡＮを、ほとんどそのままの形で提供する。これにより、データベース処理のようなアプリケーションは、自ら並列・分散ハードウェアを観察し、資源割当てを行うことで、並列・分散ハードウェアを効率的に利用することができる。また、（２）のアプリケーションと組合せて使うことも可能になる。

本稿では、（１）を支援するためのオペレーティング・システムの機能について述べる。２章では、我々の提案しているデータベースの並列処理方式とその並列アプリケーションとしての特徴について述べる。３章では、データベースの並列処理を支援するためのオペレーティング・システムの基本機能について述べる。

2．データベースの並列処理の特徴

本章では、まず、データベース処理の一般的な特徴について述べる。次にデータベースの並列処理方式の例として、我々が開発しているＳＭＡＳＨにおける並列処理方式をとりあげ、その特徴について議論する。

2．1．データベースからオペレーティング・システムへの一般的な要求

シングル・プロセッサにおいてデータベース処理を行う場合、従来のオペレーティング・システムの問題点として次のようなことが指摘されている［15］［14］。

（１）プロセス管理

プロセス間通信やプロセス切換えのオーバヘッドが大きい。マルチユーザのデータベース・システムを実現する場合、各ユーザ毎にプロセスを割当てる方法と、１個のプロセスで全ての問い合せ処理を行う方法が考えられる。前者の方法を採用すると、ユーザに割当てられたプロセス間に並列性が存在する。しかしながら、データベースのロック操作のためのプロセス間通信処理のオーバヘッドが生じる。後者の方法を採用すると、ロック操作の処理は軽くすることができる。しかしながら、重い問い合せ処理のために、軽い問い合せ処理の応答時間まで長くなるという問題が生じる。

（2）ディスク・ブロックの先読み

例えばＵＮＩＸでは、ファイルをシーケンシャルにアクセスする場合に限って１ブロックの先読みを行っているが、このような単純な先読みは、データベース処理では不十分である。なぜならば、データベース処理の場合、ファイルをランダムにアクセスする場合においても次に必要となるディスク・ブロックを予め知ることができる場合が多いからである。

２．２．ストリーム指向型並列処理方式の概要

ＳＭＡＳＨは、我々が開発してるデータベースや知識ベースを対象とした並列処理システムである［３］［４］［５］［７］［８］。ＳＭＡＳＨは、関数型プログラミングの概念をデータベース処理に適応したシステムで、任意のデータベース演算を関数として記述し、システム内に組み込み、任意の関数間に存在する並列性を抽出することを可能にしている。ゆえに、データベースの多様な応用分野に柔軟に対応することができる［９］［10］。関数間のデータの受け渡しは、ストリームにより行われる。ＳＭＡＳＨに組み込まれた関数は、関数型計算の枠組みの中で並列に実行される。関数の評価方式として要求駆動型評価を用いている。これにより、関数間で次の２種類の並列性が抽出される。

（１）関数の各引数を同時に評価することにより引き出される並列性。

（２）関数引数の適用側とその引数の生成側との間で抽出されるストリーム型並列性。これは、ストリームを生産する関数がストリームの一部を生成した時点で、ストリームを消費する関数の実行を開始することができることにより抽出される。

２．３．シングル・プロセッサにおけるＳＭＡＳＨの実現

シングル・プロセッサにおいて、ＳＭＡＳＨを以下のように実現した。関数として記述されたデータベース演算をコンパイルして、ストリーム入出力プリミティブを含むオブジェクト・コードに変換する。利用者から、データベースへの問い合せコマンドが発せられると、ＳＭＡＳＨのコマンド解釈・実行系は、関数インスタンスの木を生成する。すなわち、コンパイルして得られたオブジェクト・コードに対して、固有のデータ領域、および、スタック領域を割当て、関数のインスタンスを生成し、各関数インスタンスの間を、チャネルとよばれるストリーム・データの通信路で結合する。次のような関係データベースに対する検索要求に対して、生成される木を図２に示す。

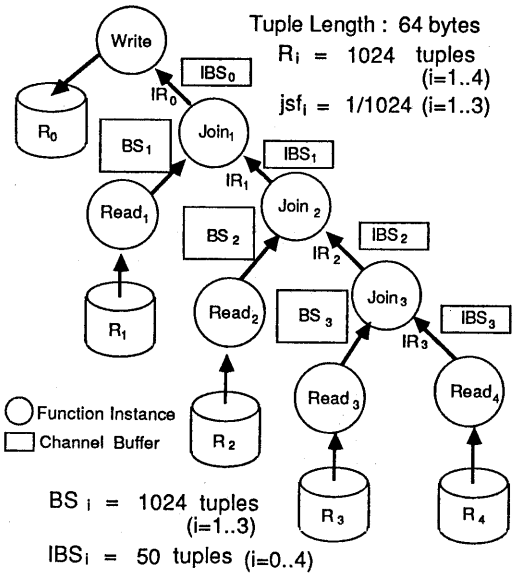


図２ 問い合わせに対して生成される木の例

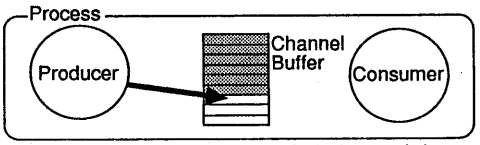
```
(write (join (read R1)
             (join (read R2)
                   (join (read R3) (read R4)))))
```

シングル・プロセッサにおいて、関数インスタンスを、コルーチンとして実現した。これは、ＵＮＩＸのプロセス間通信やプロセス切換えのオーバヘッドを避けるためである。関数インスタンス間のストリーム・データの受け渡しは、図３(a),(b)に示すように、関数インスタンスが同一アドレス空間上に存在することを利用して、メモリ上のバッファを介して行われる。

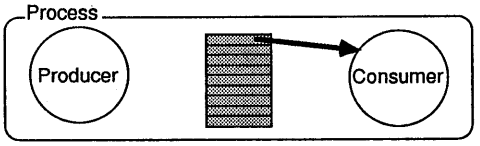
関数インスタンスのＣＰＵバースト時間の分布を調べるために、Sequent社のBalanceシステム（次節参照）において、図２に示した問い合せについて実験を行った。結果を表１に示す。小さい（平均数ミリ秒）のバースト時間が非常に多いことがわかる。ＳＭＡＳＨの関数インスタンスをＵＮＩＸのプロセスとして実現すると、プロセス切換えのオーバヘッドが大きく、処理効率が悪くなることが予想される。

表１　関数インスタンスのＣＰＵバースト時間の分布

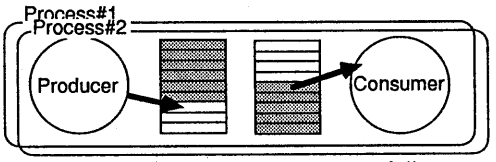
バースト時間	個数
０～　２０ｍsec	４８７
４０～１００ｍsec	３４
１００～２００ｍsec	１４
２．５～３ sec	３



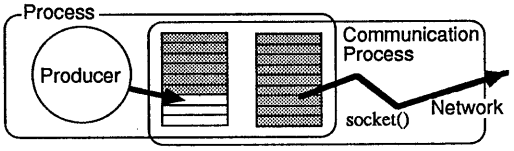
(a) シングルプロセッサにおける実現(データの生産)



(b) シングルプロセッサにおける実現(データの消費)



(c) バス共有型マルチプロセッサにおける実現

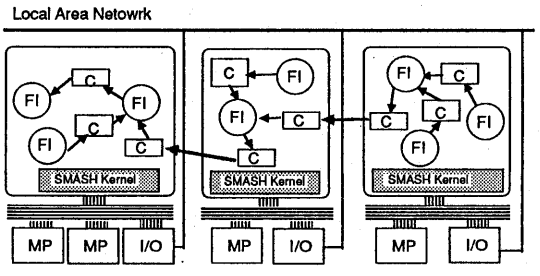


(d) ＬＡＮ環境における実現

図３ チャネルの実現

２．４．バス共有型マルチプロセッサにおける ＳＭＡＳＨの実現

ＳＭＡＳＨの実現を行ったバス共有型のマルチプロセッサ Sequent社 Balanceについて簡単に説明する［１］。Balanceは、２個から１２個の３２ビット・マイクロプロセッサ、１個の共有メモリ、入出力装置がバスで結合されている構成を取っている。各プロセッサは、８ｋバイトのライト・スルー型のキャッシュを備えている。Balanceは、パイプライン可能な（１度に複数の読み書き要求を受け付けることができる）メモリ・システムを備えており、１０ＭＨｚのバス・クロックにおいて、最大毎秒２６．６７Ｍバイトのデータ転送能力を有する。Balanceのオペレーティング・システム Dynixは、基本的にＵＮＩＸ４．２ＢＳＤと同一の外部仕様をもっている。但し、mmap()システム・コールを用いてプロセス間共有セグメントをもつことを可能にしている。これを利用して、高速なプロセス間通信を実現することができる。さらに、システム・コールを用いることなくプロセス間の同期を行うために、スピン・ロック・ライブラリを提供している。これを利用すると、セマフォ等のシステム・コールを用いる場合に比べて、はるかに小さいオーバヘッドでプロセス間の同期をとることができる。



FI: Function Instance
C: Channel
MP: Microprocessor

図４ バス共有型マルチプロセッサ およびＬＡＮを利用した並列処理

Balance上でＳＭＡＳＨの実現を以下のように行った［11］。シングル・プロセッサ用のＳＭＡＳＨカーネルから出発し、レディ・キューやメモリ管理表など、複数のプロセッサで共有すべきデータ構造や、各関数インスタンスの固有領域やスタック領域など、ほとんど全てをプロセス間共有セグメントに置いた。そしてfork()システム・コールを発行し、複数のＵＮＩＸのプロセスを通じて複数の Balanceのプロセッサを利用した。レディ・キューやメモリ管理表など、排他制御を必要とするデータ構造に対してロックを設けた。１個のチャネル（関数インスタンス間のストリーム・データの通信路）に対して、図３(c)に示すように、複数のバッファを設定した。こうすることにより、１つのバッファに対する処理を行っている間は、他の関数インスタンスと同期処理を行う必要がなくなるので、同期のオーバヘッドを軽減することができる。各関数インスタンスは、特定のＵＮＩＸのプロセスと対応しないように設定したので、ＵＮＩＸのプロセス切換えなしに動的負荷分散を実現することができた。

Dynix上でＳＭＡＳＨを実現するに際し、ファイルやメモリ等の資源割当てに関して不自由な点があった。複数の（ＵＮＩＸの）プロセスで並列処理を行っている場合、あるプロセスで新たなメモリの確保やファイルのオープンを行ったとしても、それは他のプロセスには伝達されない。これを回避するために、fork()システム・コールを発行して複数のプロセスによる処理を開始する前に、まずメモリの確保とファイルのオープンを行った。

２．５．ネットワーク環境におけるＳＭＡＳＨの実現

ＳＭＡＳＨのネットワーク環境での実現は、次のようになっている（図４）［６］。同一アドレス空間内に存在する関数インスタンス間の通信は、図３の(a),(b)に示した、シングル・プロセッサにおける実現と同じ方法により実現されている。ネットワークを介する関数インスタン間の通信は、図３(d)に示すように、ＵＮＩＸの socket()システム・コールを通じて Internet Protocol のＵＤＰ（User Datagram Protocol）を利用した。これ

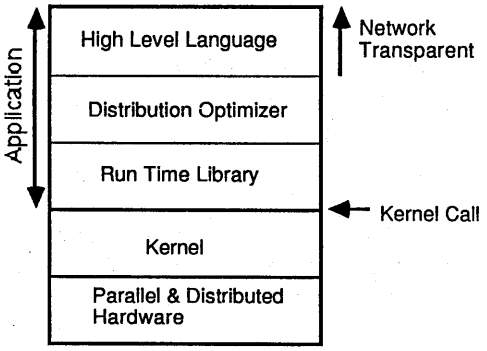


図5 システムの階層構造

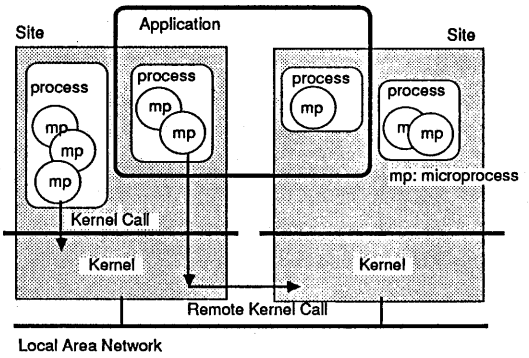


図6 サイトと遠隔カーネル・コール

表2 カーネル・コール

名称	機能
process_create(site,param,&pid)	プロセスを site上に生成する。
process_exit(code)	プロセスを終了する。
virtual_processor_allocate(n)	n個の仮想プロセッサを要求する。
virtual_processor_sleep(t)	アイドル状態になったプロセッサを t μsec の間システムに返す。
send(site,pid,port,buff,size)	ネットワークを介してデータを送信する。
receive(&site,&pid,&port,buff,&size)	ネットワークを介してデータを受信する。

は、Internet ProtocolのＴＣＰ(Transmission control protocol)と比較して、プリミティブに近い機能を提供しているので、オペレーティング・システムのオーバヘッドが小さく、データベース処理で必要となる大量データの転送に向いている。ＵＤＰは、メッセージ到着を保証しないので、独自のアルゴリズムによりエラー回復を行っている。ネットワーク通信処理とデータベース処理を重ね合わせるために、通信処理をデータベース処理とは別のプロセスで行っている。これらのプロセスは、プロセス間共有セグメントにより結合されている。

３．基本機能

本章では、データベース処理のように、内部に高い並列性をもつアプリケーションに対して、システムが提供する機能を述べ、その機能のデータベース処理における利用法を示す。

本システムの階層構造を図５に示す。アプリケーション、カーネル、並列・分散ハードウェアの３層から構成される。カーネルは、アプリケーションとハードウェアの中間に位置する。特権モードで動作し、下位層のハードウェアを管理する。ここで、並列・分散ハードウェアとは、図１に示したバス共有型マルチプロセッサと高速ＬＡＮである。カーネルは、カーネル・コールとよばれるインタフェースを通じて、上位層のアプリケーションに対して、プロセス生成、仮想プロセッサ生成、ネットワーク通信などの機能を提供する。仮想プロセッサについては３．２、ネットワーク通信については３．４で詳しく説明する。カーネル・コールは、下位層の並列・分散ハードウェアがもつ能力のレベルに設定されている［２］［12］。ゆえに、アプリケーションは、これらの情報を利用して最適化を行うことができる。

アプリケーションは、上位から、高級言語層、分散オブティマイザ層、実行時ライブラリ層の３層から構成される。高級言語層では、言語処理系により分散透明性が実現される。ゆえにこの層では、ネットワークの構造に独立なプログラムを記述することができる。次の分散オブティマイザ層は、下位層の並列・分散ハードウェアを観察し、資源割当てを行う。実行時ライブラリ層は、マイクロプロセスとよばれるアプリケーション・レベルの軽量プロセスを実現し、高速なプロセス間通信とプロセス切換えを実現する。マイクロプロセスについては、３．２で詳しく説明する。

本章では、主にカーネルの機能について述べる。本章で関係するカーネル・コールを表２にまとめる。

３．１．遠隔カーネル・コール

本システムの各アプリケーションは、図６に示すように、複数のプロセスから構成される。プロセスは、メモリ、ファイル等の資源が割当てられる単位であり、プロテクションの単位でもある。サイトは、ＬＡＮに結合さ

れた計算機である。バス共有型マルチプロセッサやシングル・プロセッサ・ワークステーションが１つのサイトとなる。

カーネルは、遠隔サイト上のカーネルに対してカーネル・コールを発行する機能を提供する。これを遠隔カーネル・コールとよぶ。この機能は、遠隔サイト上にプロセスを生成するために用いられる。データベースの並列処理を行う場合、この機能を用いて自由にネットワーク上の資源を利用することができる。

オブジェクトの識別

ファイルやプロセス等のオブジェクトは、次の３つ組により識別される。

＜サイト識別子、サーバ識別子、サーバ内の識別子＞

サイト識別子は、各サイトにユニークに与えられた整数である。サーバとは、オブジェクトを管理するプロセスである。例えばファイル・オブジェクトは、ファイル・サーバにより管理される。カーネル自身も、プロセスを管理するサーバとして扱われる。サーバ識別子は、サーバ・プロセスを識別するものである。サーバ内の識別子は、各サーバにより管理される。

常に、サイトを識別する情報を得ることができるので、並列・分散ハードウェアを意識した最適化を行うことができる。例えば、データベースの検索を行うことを考える。データベースが存在するサイト上にプロセスを生成し、検索作業を行わせ、ヒットしたデータのみをネットワークにより問い合せを発行した利用者のサイトへ転送すれば、パケット生成のコストやネットワークの負荷を引下げることができる。このような最適化を行う必要がない場合、遠隔カーネル・コールを用いて、分散透明にオブジェクトを扱うことも可能である。例えば遠隔サイト上のファイルを読み出す場合、そのサイトに遠隔カーネル・コールを発行することにより、ローカル・サイト上のファイルと同様にアクセスすることができる。なぜならば、通常のカーネル・コールは、ローカル・サイト上のカーネルに対する遠隔カーネル・コールと解釈することができるからである。

３．２．アプリケーション・レベルの軽量プロセスとカーネル・レベルの軽量プロセス

２．で述べたＳＭＡＳＨの実現において、関数インスタンスは、アプリケーション・レベルの軽量プロセス（lightweight processes）に対応する。関数インスタンス以外に、ＳＭＡＳＨカーネル自身も、通信処理のために軽量プロセスを利用する。このようなユーザ・レベルの軽量プロセスをマイクロプロセスとよぶ。プロセスとマイクロプロセスの関係を図７に示す。マイクロプロセスの生成・消滅、マイクロプロセス間の同期・通信など全ての制御は、アプリケーション・レベルで行われる。カーネルは一切介在しない。

複数のマイクロプロセスの並列実行を可能にするために、カーネルは、１個のプロセスに対して複数の仮想プロセッサを割当てる。この様子を図７に示す。バス共有型マルチプロセッサ上で実行されるプロセスの場合、複数の仮想プロセッサに実プロセッサ（ハードウェアのプロセッサ）を割当てることにより、並列処理が行われる。仮想プロセッサは、カーネル・レベルの軽量プロセスと見ることもできる。ただし、実プロセッサと同様に、並列処理の途中で仮想プロセッサの数を増したり減らしたりすることはできない。複数の仮想プロセッサ生成の要求は、プロセスが起動した直後に、カーネル・コール virtual_processor_allocate() を発行することにより行われる。一時的にアイドル状態になったプロセッサをカーネルに返す場合は、virtual_processor_sleep()を用いる。

マイクロプロセスは、従来のコルーチンと類似の機能を提供する。コルーチンとの違いは、マイクロプロセスの場合、複数の仮想プロセッサにより、並列に実行されることがあるという点である。コルーチンの場合、常に１度に１ルーチンしか実行さず、また、あるルーチンでシステム・コールを発行するとプロセス全体の実行が停止してしまう。

マイクロプロセスというアプリケーション・レベルの軽量プロセスの必要性は、２．３で述べた。その他に、マイクロプロセスを利用すると、次のような利点が生じる。マイクロプロセスのスケジューリングは、アプリケーション・レベルで行なわれるで、アプリケーションがもっている情報を利用することが可能である。例えば、次の様なスケジューリングを行うことが考えられる。

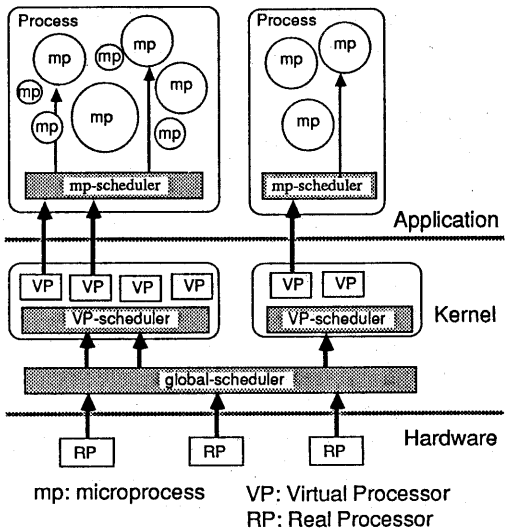


図７ プロセス、マイクロプロセス、仮想プロセッサの関係

・並列度を向上させるために、多くのマイクロプロセから待たれているマイクロプロセスの優先度を上げる。
・キャッシュのヒット率を高くするために、データの生成を行ったマイクロプロセスを実行した直後に、そのデータを消費するマイクロプロセスを起動する。

カーネル・レベルで軽量プロセスを制御する場合、このようなスケジューリングを行うことは難しい。

また、アプリケーション・レベルにおいて軽量プロセスを実現することにより、カーネルの実装が容易になるという利点も生じる。これは、軽量プロセスの生成・消滅をアプリケーション・レベルで行うので、カーネル・レベルにおいてその機能が不用になること、プロセス1個当りのカーネル・レベルの軽量プロセスの数を抑えることができること、カーネル・レベルにおける軽量プロセスのスケジューリングが簡単になること、などによる。

仮想プロセッサとプロセスのアドレス空間

仮想プロセッサのようなカーネル・レベルの軽量プロセスの実現において、プロセスの有するアドレス空間を共有する方法として、次のようなものが考えられる。

（1）全てのアドレス空間を共有する。
（2）スタックを固有領域とし、残りのデータ領域とテキスト領域を共有する。
（3）1ページの固有データ領域を除いて、全てのアドレス空間を共有する。
（4）共有・固有を自由に設定することを許す。

本システムでは、アプリケーション・レベルにおいてもコンテキスト切換えが行われるので、（2）の方法を採用することはできない。（1）の方式を採用すると、マイクロプロセスを実現する上で困難な点が生じる。例えば、現在実行中のマイクロプロセスの mpcb（マイクロプロセス・コントロール・ブロック）へのポインタを格納するために、プロセッサのレジスタが必要となる。しかしながら、mpcbへのアクセス頻度はかなり低く、また、レジスタにmpcbを保持するとアプリケーションによるレジスタ変数の使用を疎外することになる。カーネル・コールを発行して仮想プロセッサの識別子を得る方法も考えられるが、制御のためにカーネル・コールを必要としないというマイクロプロセスの利点が失われてしまう。そこで、本システムでは（3）の方式を採用した。仮想プロセッサ毎の固有領域は、mpcbへのポインタなど、仮想プロセッサ毎に必要なデータを格納する。固有領域が不足した時は、共有領域にメモリを確保し、固有領域にそのポインタを格納することで対応することができる。ゆえに多くの場合、（3）の方式で（4）の効果を得ることができる。

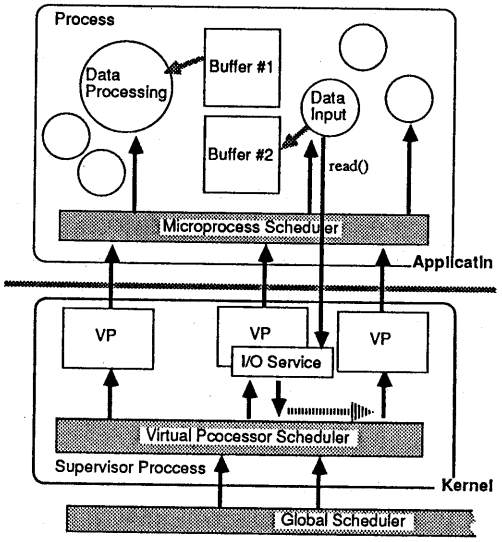


図8 仮想プロセッサによる先読みの実現

3．3．仮想プロセッサを利用した並列入出力

バス共有型マルチプロセッサ上で実行されるプロセスの場合、複数の仮想プロセッサに複数の実プロセッサが割当てられ、結果として並列処理が行われる。複数の入出力の並列実行や、ディスク・ブロックの先読みを実現するためにも、仮想プロセッサを利用することができる。

例えばＵＮＩＸでは、ファイルをシーケンシャルにアクセスする場合に限り、1ブロックの先読みを行う。データベース処理では、ファイルをランダムにアクセスする場合でも、次に必要となるディスク・ブロックを予め知ることができる場合が非常に多い。このような場合、以下に述べるような方法により、先読みを実現することができる。図8に示すように、データ処理を行うマイクロプロセスの他に、データ入力を行うマイクロプロセスとバッファを2個用意する。データ入力を行うマイクロプロセスは、データ処理を行うマイクロプロセスが一方のバッファについて処理を行っている間に、別のバッファに次に必要となるブロックの読込みを行う。

3．4．ネットワーク通信

本システムでは、同一のアプリケーションに属するプロセス間の通信と、異なるアプリケーションに属するプロセス間の通信を分離して扱う。前者のために、カーネルは、カーネル内のオーバヘッドが小さいデータグラム・サービスを提供する。この場合カーネルは、プロセス間のコネクションの形成、メッセージ到着の保証、メッセージのバッファリング、エラー回復を一切行わない。これはInternet ProtocolのＵＤＰと同様の機能である。

例えば、ＳＭＡＳＨの関数インスタンス間のストリームの受け渡しにおいて、ストリーム要素の順番に意味がないものが存在する。この場合、Internet Protocolの

ＴＣＰのように、ストリーム要素の順番に意味があるものに比べて、効率のよいエラー回復アルゴリズムを用いることができる。

カーネルは、マルチキャスト通信機能を提供する。これは、データベース処理において、同一データを複数のプロセスで利用する場合に用いられる。また、データベースのデータやデータベース演算の中間結果のキャッシングを行う場合、この機能を利用すれば、データの生産者やデータの消費者以外の第３のサイトにおいてキャッシングを行うことも可能になる。

データグラム・サービスの他に、カーネルは、信頼性のあるデータグラム（reliable datagram）や、順序付バケット（sequenced packet）のように、メッセージ到着の保証、および、メッセージのバッファリングを行う高いレベルの通信機能も提供する。これらの機能は、異なるアプリケーションに属するプロセス間の通信に用いられる。

４．むすび

本稿では、データベースの並列処理の特徴について、例として、我々が開発を行っているＳＭＡＳＨをとりあげて議論した。データベースの並列処理においては、高速なプロセス間通信、プロセス切換え、ネットワーク通信を必要とした。次に、このような特徴に対応するためのオペレーティング・システムの基本機能を示した。遠隔カーネル・コール、２レベルの軽量プロセス、高速ネットワーク通信機能について述べた。現在、マイクロプロセスを実現するライブラリを既存のオペレーティング・システム上で開発している。今後の課題は、システムの動的な状態をアプリケーションに伝え、資源の動的な割当てを行うための機能を実現することである。

謝辞

日頃から、筆者らがご指導いただいている東京大学理学部益田隆司教授に感謝致します。また、熱心にご討論していただいた東京大学工学部田胡和哉博士および東京大学理学部加藤和彦氏に感謝致します。

参考文献

[1] Balace 8000 Parallel Programming, Sequent Computer Systems,Inc. (1985)

[2] R.A.Finkel, M.L.Scott, Yeshayahu Artsy and Mung-Yang Chang: "Experience with Charlotte: Simplicity and Function in a Distributed Operating System", IEEE Trans. Software Eng., Vol.15,No.6,pp.676-685 (1989)

[3] Y.Kiyoki,K.Kato and T.Masuda: "A Relational Database Machine Based on Functional Programming Concepts", Proc. ACM-IEEE Computer Society Fall Joint Computer Conf., pp.969-978 (1986)

[4] Y.Kiyoki,K.Kato,N.Yamaguchi and T.Masuda: "A Stream-Oriented Approach to Parallel Processing for Deductive Database, Proc. 5th Int. Workshop on Database Machines, pp.102-115 (1987)

[5] 清木，加藤：関数型計算モデルのデータベース処理への適用，情報処理，Vol.29,No.8,pp.897-907 (1988)

[6] 黒沢，清木，加藤，益田：関数型計算モデルに基づく並列型問い合せ処理系の実現方式，情報処理学会データベース・システム研究会，68-7, Nov 17 (1988)

[7] P.Liu, Y.Kiyoki and T.Masuda: "Efficient Algorithms for Resouce Allocation in Parallel and Distributed Query Processing Environments", Proc. IEEE Int. Conf. on Distributed Computing Systems (1989)

[8] 劉，清木，益田：ストリーム指向型関係演算処理におけるバッファ資源割当ての計算方式，情報処理学会論文誌，Vol.29,No.8,pp.770-781(1988)

[9] 波内、清木、劉：演繹データベースの並列処理方式の実現と資源割当て方式、情報処理学会データベース・システム研究会、89-DBS-72,pp.105-112, July 20-21 (1989)

[10] 関、関島、清木：並列処理システムＳＭＡＳＨにおける機械系ＣＡＤデータベース実現のための一考察、情報処理学会データベース・システム研究会、89-DBS-72,pp.177-184,July 20-21 (1989)

[11] 新城、清木、劉、益田：データベースおよび知識ベースを対象としたストリーム指向型並列処理系の共有メモリ・マシン上への実現、アドバンストデータベースシステム・シンポジウム論文集, Vol.88,No.9,pp.117-126 (1988)

[12] 新城、清木：ＲｅＳＣ:並列アプリケーションのためのオペレーティング・システム、情報処理学会第３９回全国大会講演論文集 (1989)

[13] J.Magee,J.Kramer and M.Sloman: "Constructing Distributed Systems in Conic", IEEE Trans. Software Eng.,Vol.15,No.6,pp.663-675 (1989)

[14] T.W.Page,Jr., M.J.Weinstein, and G.J.Popek: "Genesis: A Distributed Database Operating System", Proc. ACM SIGMOD 1985,SIGMOD RECORD, Vol.14,No.4,pp.374-387 (1985)

[15] M.Stonebraker: "Operating System Support for Database Management", CACM,Vol.24,No.7,(1981)

[16] A.S.Tanenbaum and R.van Renesse:"Distributed Operating Systems", ACM Comp.Surveys, Vol.17, No.4,pp.419-470 (1985)