ＴＯＰ－１　オペレーティングシステム

穂積　元一

日本アイ・ビー・エム

高性能並列処理ワークステーションＴＯＰ－１の概要、および、スレッドを持ったそのＯＳについて解説する。ＴＯＰ－１システムは８０３８６マイクロプロセッサー１０台を持つ密結合型共有メモリタイプのワークステーションで、そのオペレーティングシステムはＡＩＸをベースとした並列処理用のＯＳである。
ＴＯＰ－１アーキテクチャの特徴、特にシステムソフトに関連する部分、および、我々のオペレーティングシステムの特徴であるスレッドの考え方に関して述べる。スレッドとは従来のプロセスに比べてスイッチングの軽いものであり、ある意味で実行コンテクストと考えられるものである。

ＴＯＰ－１　Ｏｐｅｒａｔｉｎｇ　Ｓｙｓｔｅｍ

Ｍｏｔｏｋａｚｕ　Ｈｏｚｕｍｉ
IBM Japan Ltd. Tokyo Research Lab.
5-19 Sanban-cho Chiyoda-ku Tokyo Japan

This paper describes the overview of TOP-1 system which has 10 80386 micro-processors and its operating system. This operating system is based on AIX and has thread mechanism for supporting MP functions. Thread is a light weighted process and can be regarded as an executing context.

１．はじめに

我々は高速並列処理ワークステーション（ＴＯＰ－１）の上にＡＩＸをベースとした並列処理オペレーティングシステムを設計作成中である。第１版は既に稼働中である。

このオペレーティングシステムを解説するには、まずＴＯＰ－１のアーキテクチャの概要から示さなければならない。

ＴＯＰ－１プロジェクトの目的とは、密結合共有メモリ型の並列処理ワークステーションを実現することで、汎用の高いコストパフォーマンスの実現と、それに伴う各種ソフトの研究にある。

２．ＴＯＰ－１アーキテクチャの概要

ＴＯＰ－１アーキテクチャの詳しい内容については、本年秋の情報処理学会全国大会に於て６件の発表が予定されており、そちらを参照されるのが良いが、現在まで新聞発表以外に詳しい情報がほぼ公表されていないので、オペレーティングシステムの概要を理解する上で必要な部分、および重要な特徴についてのみここでは述べる。

２．１　ＴＯＰ－１システムの構成

図１に示されるとうり、ＴＯＰ－１のシステム構成は、１０台のインテル８０３８６プロセッサーを使用した、共有メモリ型マルチプロセッサーシステムである。このシステムはローカルメモリを使用しておらず、すべてのＣＰＵから完全に均質に共有された共有メモリのみからできている。従来はローカルメモリを使うことで、各ノードに固有なデータを割りつけ、これによって共有メモリとバスの競合を軽減するのが普通であった。しかし本システムはそのシステムソフトと相俟って汎用性を大きな特徴としているため、複雑なメモリ管理を要求するローカルメモリを使用せず、一切を共有メモリとしている。ただし共有メモリ型アーキテクチャの欠点であるバス競合を大幅に低下させるべき工夫としてスヌープキャッシュと高速バスを採用して、この問題に対処している。

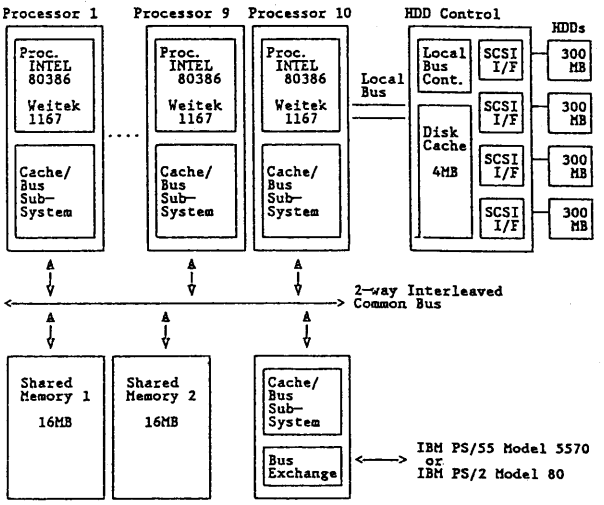

図１　ＴＯＰ－１のハード構成

浮動小数点プロセッサーとしては、８０３８７より高速のＷｅｉｔｅｃ１１６７を採用している。

又、Ｉ／Ｏに関しては、図１にも示されるとうりハードディスクとしてＳＣＳＩの３００ＭＢのものを４台実装しており、その他のＩ／Ｏについては弊社の製品ＩＢＭ　ＰＳ／５５　Model　５５７０とＩ／Ｏカードを通じて直接接続することで行っている。

以下にＴＯＰ－１の重要な特徴であるスヌープキャッシュとＭＰ－ＯＳを作成する上で基本的なプロセッサー間非同期通信機構について述べる。

２．２　スヌープキャッシュ

バス及びメモリの競合を低減するために、上に述べたようにＴＯＰ－１では各ＣＰＵにキャッシュを用いている。これはデータと命令を区別するものでなく、大きさは１２８Ｋバイトのものである。

キャッシュのサイズは大きければ、それだけヒット率が高くなるが、逆にコストも高くなるわけで、このかめあいで決められるものであるが、我々は高速ＳＲＡＭを用いることで１２８Ｋバイトのキャッシュを実現している。

マルチプロセッサーでのキャッシュの最大の問題は同一のメモリロケーションに対して複数のコピーがキャッシュに存在してしまう点で、いかにしてこれら複数のコピーの矛盾をなくし、一致性を保証するかということである。この問題を解決するために、ＴＯＰ－１ではスヌープキャッシュ方式を採用している。
我々は４つの状態を導入することで、できる限り各キャッシュ内でローカルに処理できる比率を高く保ち、共有バスに対する要求回数を低く抑える事を計っている。

２．３　プロセッサー間非同期通信機構

マルチプロセッサーにおいて効率の良い並列処理を行うためには、プロセッサー間の同期機構が大切である。ＴＯＰ－１システムは共有メモリ方式であるので、共有メモリ上に共有変数を置くことで、プロセッサー間の同期を計る事ができる。しかし共有変数だけでは非同期に発生するイベントをプロセッサー間で効率よく伝達できない。このためにハードとしてのプロセッサー間非同期通信機構が必要になる。

もちろん通常のセマファは８０３８６のＬＯＣＫ＃信号を用いる事でTest-and-Setが実現できプロセッサー間の排他制御を行っている。

イベントドリブンな通信を行うために、ＴＯＰ－１では図２に示すような、Everybody Broadcast，Anybody Broardcastという２種類のプロセッサー間非同期通信機構を用意している。

Everybody Broardcastとは指定したすべてのプロセッサーがメッセージを受信できたとき、成功と見なされ、一つでも受取に失敗するとこのメッセージパッシングは失敗となるものである。

Anybody Broardcastとは指定した宛先のうち一つ以上のプロセッサーが受信できれば成功と見なされ、全ての宛先が受取に失敗したときに、失敗と見なされるものである。例えば、Everyboby BroardcastはＴＬＢのコンシステンシーのプログラムで用いられ、Anybody BroardcastはＩ／Ｏ割りこみに用いられる。

３．ＴＯＰ－１オペレーティングシステム

上に述べたＴＯＰ－１システム上にＡＩＸをベースとしてＭＰ－ＯＳ　ＴＯＰ－１オペレーティングセステムを作成中であるが、まず並列処理オペレーティングシステム作成のアプローチを述べ、次にどのような研究課題があり、更にそれらをどのように解決しているかについて述べる。

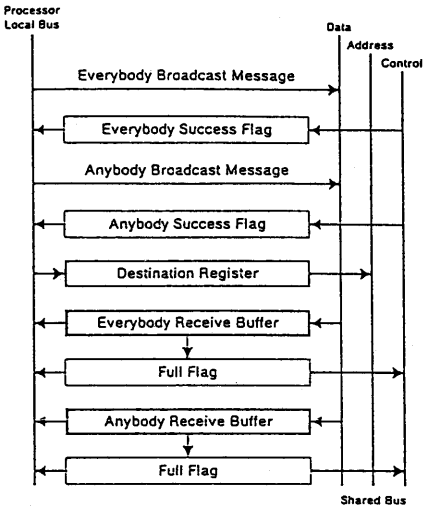

図 2: Message Passing Hardware

1. プロセッサがある I/O アドレスに出力命令を開始する。
2. キャッシュ・バス・コントローラがバスを獲得する。
3. 宛先の受信バッファが空のとき，プロセッサの出力データがバッファに書かれる。
4. バスが解放され，プロセッサの I/O 命令が終了する。

３．１　ＴＯＰ－１オペレーティングシステムのアプローチ

ＴＯＰ－１オペレーティングシステムはシングルプロセッサー用のＯＳであるＡＩＸをベースとしてＭＰ－ＯＳを作成しようとしている。そこで次のようなアプローチが考えられる。

ａ．ＯＳのカーネルをできるだけリエントラント化する。
ｂ．ＯＳのカーネルはどれか一つの特定のＣＰＵでのみ実行させ、ユーザプロセスを他のＣＰＵで並列動作させる。
ｃ．ＯＳのカーネルはシリアリリユーザブルのまま、任意のＣＰＵで動作させるようにする。

ｂは概念が明確である分作成は容易であるが、よりよいパフォーマンスを求めると、どうしてもａを実現しなければならない。ただしａはｂに比べて作成は困難であり、又幾通りかの実現方法があり、将来的な並列処理に耐えられるカーネルの構造をも実現しようとすると、どのようなアプローチでも良いというわけではない。

更に作成上のことを考慮に入れると、並列処理ＯＳのデバッグはかなり困難なものであり、初期の設計が余程完全であるか、ステップ　バイ　ステップに作成するかのいずれかと思われる。我々は既存のＯＳの並列かを行っているので、一からの完全な設計は望めない。そこでｂ－＞ｃ－＞ａのアプローチで作成することとし最終段階のａを実現する段で十分な将来的な機能分割を達成する事を目指している。

３．２　ＴＯＰ－１オペレーティングシステムの研究課題

このような並列処理オペレーティングシステムを作成する上で、どのような研究課題を考えて行って要るかを述べる。

１．　ＯＳのカーネルの機能分割
２．　ユーザに対する並列処理のより良いモデルの提供
３．　並列処理特有の問題の解決

１については、今後の並列オペレーティングシステムを考える上で本質的な問題であり、現時点では多くを論文にできないが、概念のみ説明する。

２については、我々は一般にスレッドと呼ばれているモデルを導入しているが、これについては幾通りかの考え方があり、そのうちの一つについて以下に説明する。

３については、ＴＬＢフラッシュの問題を含めた並列処理特有の資源管理の問題解決の事であり、知られている事も多いのでここには述べない。

３．２．１　カーネルの機能分割

カーネルのできるだけ多くの部分をリエントラントにして、並列度を上げ性能向上を計ることは、並列処理オペレーティングシステムの作成上の一つの眼目である。もちろん必要なクリティカルセクションをきちんと認識し、ロックを取る事によってこれらは達成される。しかしいかにしてデッドロックを回避するか、更には将来的な展望を考えた場合望まれるカーネルの構造との調和が問題として残る。

そこで我々はマルチサーバモデルというものを採用しており各サーバは一つのまとまった機能単位として、サーバ単位の並列性の実現を考えている。これによってサーバの入れ替えや追加更には機能分割型のアーキテクチャにも柔軟に対処できることを目指している。

３．２．２　スレッドモデル

一般にスレッドと呼ばれているモデルのインプリメントには幾通りか存在し、もちろんそれぞれに意味が異なっている。例えばＳＵＮのlight-weight-processはある意味のコルーチンとして実現されている。又ＣＭＵで開発されたオペレーティングシステムＭＡＣＨにもスレッドモデルは採用されている。我々はスレッドを共有メモリを持つ並列計算のモデルとして実現している。この共有メモリのもたらす実行環境をプロセスといい、その中の実行文脈（コンテクスト）をスレッドと言っている。

この両者の関係を図３に示す。

我々がこのようなスレッドを導入した意味は大きく言って次の２点から成る。

１．もしコルーチンのままだと、マルチプロセッサー上でスレッドによる実時間並列処理ができない。メモリを共有する＇軽いプロセス＇としてはコルーチンも意味があるが真の並列処理が実現できない。

２．ＵＮＩＸのSystem Ｖにはメモリセグメントの概念があり、複数のプロセスでメモリを共有できる。ただしこの場合はプログラムコードやデータ全般が共有されるわけではなく、従ってより密な相互作用が不可能になっている。更に我々のスレッドでは全空間を図４のように共有しているので、同一プロセス内のスレッドスイッチの場合にＴＬＢフラッシュの必要がなく効率的である。System Ｖのプロセスの場合は当然これをＭＰ化するとプロセススイッチ時にＴＬＢフラッシュが必要になる。

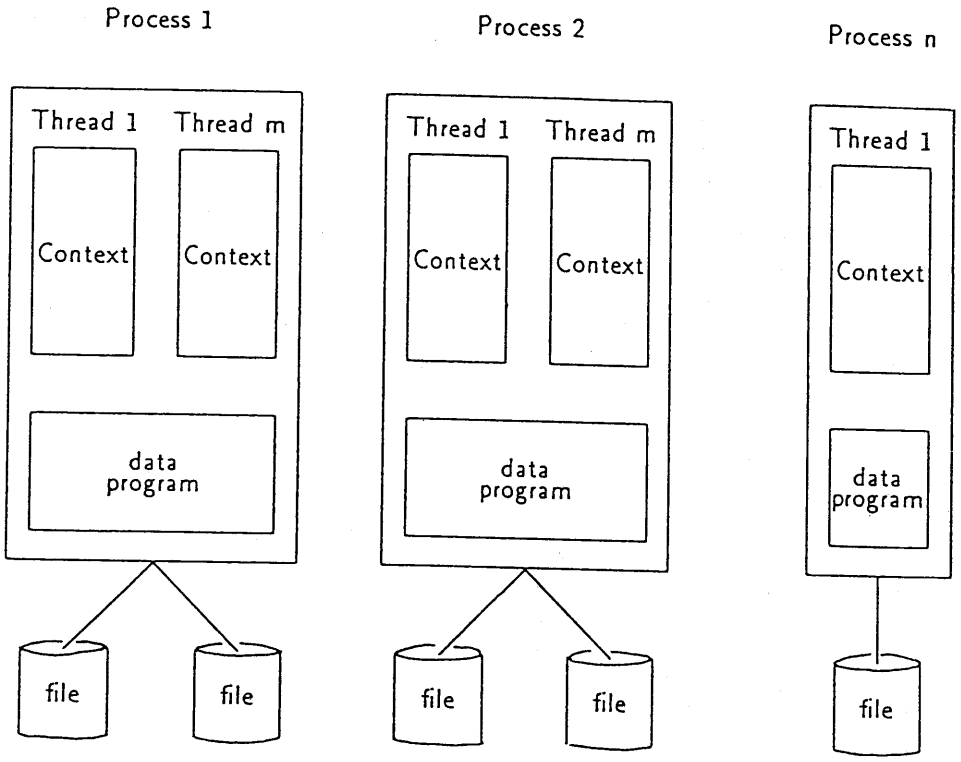

図3　プロセスとスレッド

次に同一プロセス内の各スレッドのスタックについて述べる。これは言語処理系をスレッドを用いてインプリメントする場合のスコープルールと深く関係する問題である。

基本的にスタックのインプリメントには２通りの考え方がある。

ａ．スタック領域は同一プロセス内の全てのスレッドで共有し、それらの間のチェインニングや保護は何もサポートしない。すべて言語処理系に任せる。

ｂ．ハードのセグメント機構等を用いて各フレーム毎の保護をサポートする。更に親子関係といったチェインニングもＯＳで行う。

ａの方法は言語処理系が自由にスタックの使用法を定義できる利点があるが、保護や空フレームの回収等も自分でやらなければならなくなる欠点がある。

- Virtual memory

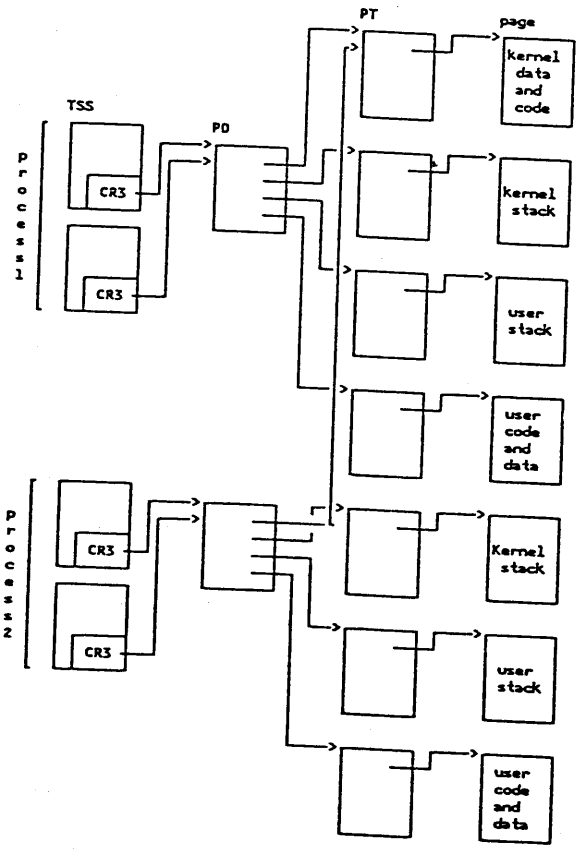

図4　プロセスとスレッドの仮想記憶機構

これに対してｂでは、ＯＳやハードである程度の事をしてくれる代りに、自由なスコープルールや親子関係は定義できなくなる。ＬＩＳＰやＣのような比較的自由な言語ではａのほうが向いているといえ、Ａｄａのような厳密な言語ではそれにあったスタックインプリメントがｂの方法で求められるだろう。将来的な互換性や言語の移植性といった事まで含めると軽々には決定できない問題といえる。

しかしＴＯＰ－１は研究用のマシンなので、言語処理系にとって自由度の高い方法ａを採用している。もちろんこのほうが言語処理系の研究にとって都合が良いからである。

次にスレッドスタックと言語の変数スコープの関係について、図５に従ってもう少し詳しく眺めてみる。左がわのツリーはスレッドの生成関係を示しており、右側のボックスは言語の構造（プロシージャ，ファンクション）による変数のスコープを示している。つまり、左側でいうとスレッドＡがスレッドＢとＣを生成し、更にＢがＤとＥをＣがＦとＧを生成したことを、しめしている。右側でいうと、構造ＢとＣからは構造Ａの変数は参照可能である事を、ＢとＣでは互いに参照不可能の事を示している。又右と左で同一の名前は構造Ａがスレッドで実行される事を示している。この様な場合のスレッドスタックのインプリメントは参照可能方向のチャインと参照不可能間の保護を同時にサポートしなければならない。これには動的関係つまりスレッドの生成関係と静的関係つまり言語構造の入れ子関係の二つが必要になる。ＯＳだけでこれを全て処理すると、一義的なインプリメントとなって、微妙な言語仕様（ＬＩＳＰのSpecial Variables等）のインプリメントを困難なものとしてしまう。

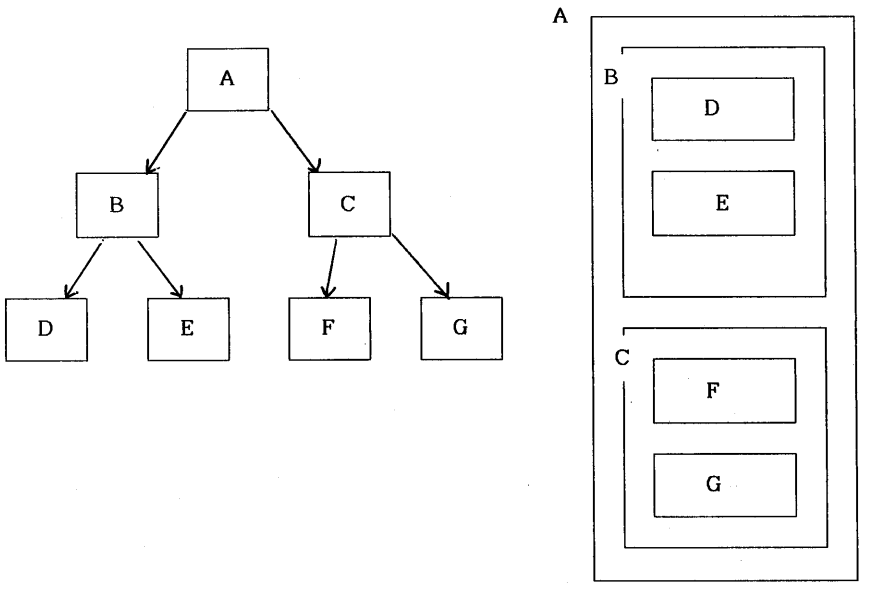

図5　スタックとフレームの関係

4．おわりに

現在ＴＯＰ－１は稼働中であり、我々は新しいバージョンのＭＰ－ＯＳを作成中であり、同時にCommon-LISPの並列化にも取り組んでいる。いくつかの並列アプリケーションを走行させる予定である。もちろん我々のＯＳはＡＩＸの完全上方互換であるので、ＡＩＸのアプリケーションはそのまま並列に走行させる事ができる。今後これら並列アプリケーションから有用なデータの収集が期待される。

5．参考文献

- "System Service Overview"　Sun Microsystem Inc.
- "MACH Kernel Interface Manual"　Robert V baron, David Black et al　Carnegie-Mellon Univ. 1987
- "MACH: A Basis for Future UNIX Development"　Avadis Teranian Jr. Richard F. Rashid
  Carnegie-Mellon Univ. 1987
- "MACH Threads and UNIX Kernel"　Avadis Teranian Jr. Richard F. Rashid et al.
  Carnegie-Mellon Univ. 1987