オペレーティング・システム 58－4
（1993. 3. 19）

超並列計算機のためのオペレーティングシステムの構想

萩野 達也, 徳田 英幸, 斎藤 信男 (慶應義塾大学)
松岡 聡, 米澤 明憲 (東京大学)
砂原 秀樹, 多田 好克 (電気通信大学)
柴山 悦哉 (龍谷大学)
篠田 陽一 (北陸先端科学技術大学院大学)

超並列計算機の利用においては，いくつかの異なる種類のユーザを取り扱うことができなくてはならない．通常のオペレーティングシステムが提供する機能で満足するユーザ，オペレーティングシステムによって提供されるいくつかの機能を使って実行環境を自分で作るユーザ，ハードウェアが提供する機能を最大限に利用しようとし，オペレーティングシステムが邪魔になるユーザ，などが考えられる．CPU，メモリ，補助記憶装置などのハードウェア資源はpartitionと呼ばれる単位に分けられ，オペレーティングシステムの最下層部分によってサポートされる．それぞれのpartitionは完全に保護され，別のpartitionの影響を受けることはない．このようにpartition使うことにより，異なるレベルのユーザの処理を行なうことができる．本論文では，このような超並列計算機のための新しいオペレーティングシステムの構想について述べる．

Design of Operating System for Highly Parallel Machine

Tatsuya Hagino, Hideyuki Tokuda, Nobuo Saito (Keio University)
Satoshi Matsuoka, Akinori Yonezawa (University of Tokyo)
Hideki Sunahara, Yoshikatsu Tada (University of Electro-Communications)
Etsuya Shibayama (Ryukoku University)
Yoichi Shinoda (Japan Advanced Institute of Science and Technology)

An operating system for highly parallel computer system should support several different kinds of users. Some users will satisfy with the usual operating system service; some will pursuit to construct his own execution environment using a certain amount of existing functionality provided by the operating system; some will investigate to fully utilize the given hardware, and therefore the functionality of operating system may be an obstacle for his purpose. A given hardware resource such as processing element, local memory, auxiliary storage and so on are aggregated into a subset of resources called a partition. The lowest kernel of the operating system should manage the secured partition which are completely protected from the disturbance caused by other partitions. By using partitions, it is possible to provide several different supports for users. This paper describes a new operating system for highly parallel computer system which addresses these problems.

この研究は文部省科研費重点領域「超並列原理に基づく情報処理基本体系」(領域番号211)の援助を受けて行なわれた．

1 はじめに

現在，文部省科研費重点領域「超並列原理に基づく情報処理基本体系」において，1000台から10000台を越えるPE (Processing Element) を持った超並列計算機システムの研究を行なっている．4つのPEが共有バスで結合され，1つのclusterを形成し，メモリを共有する (図1)．そのようなclusterがネットワークを使って結合され，全体のシステムを作り上げる．各PEは直接メモリをアクセスするのではなく，MBP (Memory Based Processor) を介してアクセスする．MBPは，ローカルなメモリをアクセスするだけでなく，ネットワークを介して別のclusterのメモリのアクセスも行ない，分散共有メモリをサポートする．したがって，PEから見れば，巨大な共有メモリ計算機とみなすことができる．さらに，MBPは，FIFOやバリア同期などの機能もメモリアクセスという形で提供する．本論文では，このようなハードウェア (以下，Dマシーンと呼ぶ) 上のオペレーティングシステムCOS (Collaborative Operating System) の構想について述べる．ただし，COSの設計はDマシーンのハードウェアアーキテクチャと独立に行なっている．したがって，COSは他の超並列計算機に実装することも十分意義のあることである．

以下には，COSの設計方針，基本アーキテクチャ，各種インタフェース，開発計画などを述べる．

2 COSの設計方針

1. **多様なエンドユーザの要求に対応**: 超並列システムのエンドユーザが記述する応用ソフトウェアは，コンピュータハードウェアの使い方への多様な要求を持っている．例えば，従来の逐次処理用プログラミング言語 (代表例はFortran) で記述したものを超並列処理用に変換したもの，および新しいプログラミングパラダイムで超並列用応用ソフトウェアを記述したもの (代表例は並列オブジェクト指向言語) は，それぞれ超並列マシーンの使い方に関して，全く正反対の要求を持っている．前者は，出来るだけハードウェアを生の形で使い効率を上げることを最大目標にする (データパラレル) のに対し，後者は出来るだけハードウェアを仮想化して新しい実行環境 (オブジェクトパラレル) を論理的に実現することを要求する．
2. **マルチユーザの実行環境を提供**: 超並列計算機は，大規模なコンピュータシステムであり，そのユーザはさまざまな要求を抱えている．そのような状況で効率良くシステムを利用していくためには，オペレーティングシステムがマルチユーザの環境を提供する必要がある．
3. **独立かつ完全なオペレーティングシステムであること**: 超並列計算機システムのマシーンは，それ自体で閉じたシステムであり，バックエンドプロセッサとして働くという形態はとらない．それを制御するCOSは，オペレーティングシステムとして，閉じていることが望まれる．それは，ファイルシステム，ユーザインタフェース，プログラムの起動，各種デバイスの制御などのサービスをCOSは包括的に提供する必要がある．
4. **Fault Toleranceを実現**: 超並列計算機は，ハードウェアの部品の量だけでも，膨大なものになる．これらのハードウェアに基づいて動作するシステムは，その信頼性を高める機能を支援しておかなければならない．
5. **柔軟な実現法を採用**: 超並列システムのオペレーティングシステムには，多様な要求が与えられるが，それらの実現に十分に対応するためには，柔軟なオペレーティングシステムの実現法を採用しておかなかければならない．

3 ユーザに対するサービスインタフェースの構造

設計方針にも述べたように，超並列計算機システムに対してはさまざまな異った要求をもったエンドユーザの応用ソフトウェアが与えられる．これらは，ハードウェア資源あるいはソフトウェア資源に対して，異った利用要求を持っている．オペレーティングシステムをユーザの立場で見た時には，どんなサービスをシステムが提供するかが最も重要な問題になる．

そこで，COSがサービスを提供するユーザを，お任せ型ユーザ，協調型ユーザ，独立自尊型ユーザの3種類に分類し，それぞれのサービスクラスが提供し易いように基本アーキテクチャを設計する．

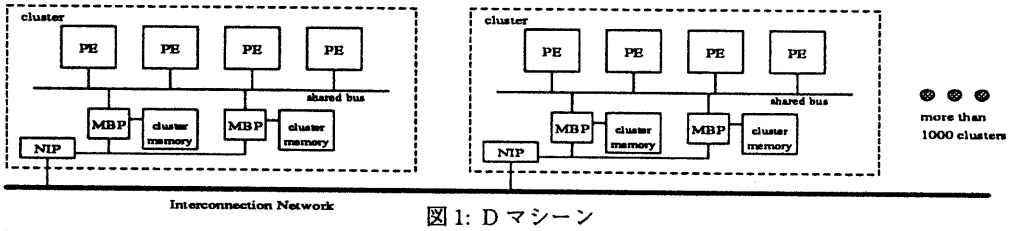

図 1: D マシーン

従来，オペレーティングシステムの設計方法として，Policy と Mechanism の分離という原理があった．ここでは，上記のユーザのクラスの動作に関して，この原理を持って説明できる．COS の基底には，いわゆる Mechanism のモジュールが共通にある．これに対して，Policy のモジュールは，ユーザのクラスのよって，以下のように選択する．

1. お任せ型ユーザ: すべての Policy モジュールは，既定のものを使う．従って，ユーザも対するサービスも既定のものになる．
2. 協調型ユーザ: COS の基本的な Policy モジュールは，既定のものを使い，それ以外の Policy モジュールは自分自身で提供するものを使う．従って，ある程度ユーザの要求ににカスタマイズしたサービスを受けることが出来るが，全体的な協調性を外れることはない．
3. 独立自尊型ユーザ: COS の基本的な Policy モジュールも含めて，すべての Policy モジュールをユーザの提供したものにする．従って，全体的な協調性は，他のユーザの割り当てられた資源を破壊しないという最低の条件以外には求めない．完全にユーザの要求にカスタマイズされたサービスを受けることが出来る．

以上のサービスの構造を図 2に示す．

4 COS のソフトウェアアーキテクチャ

前節に示したようなサービスを各クラスのユーザに提供するのが，COS の主な役割である．これを実際に実現するために設計したのが，以下に述べる COS ソフトウェアアーキテクチャである．ここでは，超並列のハードウェア資源の管理をきちんと実施する部分と，ユーザの応用ソフトウェアに最終的に有益なサービスを与える部分に大きく分けられる．

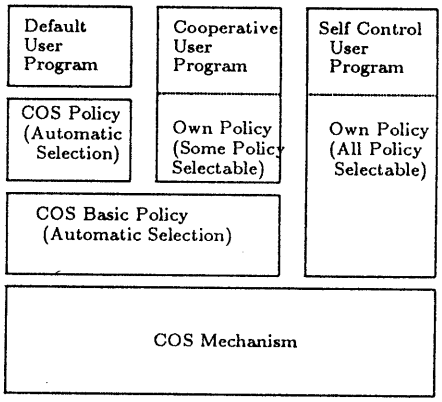

図 2: COS のサービスインタフェースとその構造

4.1 基本概念の定義

ここで，COS が提供する抽象化の実体を表す基本概念をいくつか定義する．

1. パーティション (partition): 超並列計算機のハードウェアおよびソフトウェアの資源の集合体で，他のパーティションとの間に十分堅牢な保護が施されている．超並列システムがハードウェアの実装単位である cluster にわかれている場合には，cluster を単位として partition を設定する．すなわち，原則として cluster の集合が partition となる．
2. ユーザ (user): 超並列コンピュータを利用する主体で，いくつかのコンピュータの資源の割り当てや応用ソフトウェアの実行などを要求する．
3. ジョブ (job): 一つの応用ソフトウェアを実行するときの管理の主体．資源割り当ての主体となったり，ユーザとの対応の主体となる．
4. プロセス (process): 独立したプログラム実行の制御の主体で，自分固有のアドレス空間を持

つ．実際の実行はいくつかのスレッドの集合が行なう．

5. **スレッド**(thread): 独立したプログラム実行の制御の主体で，具体的なPEに割り付けられる単位である．このアドレス空間は，それを含むプロセスと同一のものとなるので，複数個のスレッドが同一のアドレス空間を共有する．
6. **プロセス間通信**(inter-process communication): プロセス間の情報の交換を行なう動作とその機能．
7. **スレッド間通信**(inter-thread communication): プロセス間の通信は，実際にはそのプロセスが持つスレッドによって実施される．
8. **通信チャネル**(communication channel): プロセス間通信あるいはスレッド間通信を行なう際に必要な通信経路．
9. **ボリューム**(volume): COSに於けるファイル空間の最下層の抽象化の単位．このowner以外，そのvolumeへのアクセスは許可を必要とするので，良く保護されたファイル空間となる．
10. **仮想記憶**(virtual memory): 仮想的にアドレスの付けられたまとまった記憶の実体．
11. **アドレス空間**(adress space): 一連の連続したアドレスを持った記憶空間．

4.2 ソフトウェアアーキテクチャ

COSのソフトウェアアーキテクチャは，以下の通りであり，次の基本構成要素からなる．

1. **POS (Partition OS)**: COSに於けるpartitionの管理を行なう制御システム．partitonの定義，割り当て，不正なアクセスからの保護，partition間の情報交換の制御，volumeの管理，I/O操作の管理などを行なう．COS全体で，一種類のPOSがある．実際には，POSは更に以下の要素から成る．
   (a) **MMK (Meta Micro Kernel)**: 超並列計算機システムの各PEに一つずつ常駐し，ハードウェアの基本機能の制御を行なう．割り込み制御，スレッド制御，コンテクスト切替え，記憶管理の一部の機能などを行なう．
   (b) **CLM (Cluster Manager)**: 超並列システムのハードウェアの実装単位であるclusterに一つづつ存在し，clusterに特有のハードウェアの制御管理を行なう．たとえば，入出力機能の制御，partition間の通信の管理，partitionの動的な設定(拡大縮小，共有)の制御などを行なう．
   (c) **PTS (Partition Server)**: 一つのpartitionに原則として一つ存在し，partition全体に関する管理，他のpartitionとの通信制御，partitionの動的拡大縮小の管理などを行なう．
2. **SOS (Service OS)**: COSに於ける具体的なシステム機能のサービスを実現する制御システム．partition割り当ての要求，プログラムの起動や停止，高度な通信，入出力操作起動，ファイルシステムの実現，ユーザインタフェースなどを行なう．COS全体では，複数種類のSOSが稼働している．

このソフトウェアアーキテクチャを，図3に示す．図2に示した3種類のユーザのクラスと，図3に示したソフトウェアアーキテクチャとの関係は，まだ厳密には仕様を決めていないが，おおよそ表1のようになる．

すなわち，お任せ型ユーザは，POSは既定のものを使い，またユーザの要求に応じてシステムで提供するSOSの一つを選択して使う．このSOSのサービスた動作は，既定のものである．COSが豊富なサービスを提供したければ，SOSを多数開発しユーザに提供すればよい．

一方，協調型ユーザは，一部のpolicyを自前のものにしたい．ここでは，最終的なサービスを決めるSOSあるいは，partitionの動作を管理するPOSの構成要素であるPTSの一部を自前にして実行をするとよい．

さらに，独立自尊型ユーザでは，サービスを決定するSOSは勿論のこと，POSの構成要素であるPTSもすべて自前のものを使う．また，partitionの動的な制御に関係してくるCLMの一部も自前のもので置き換えて実行をするとよい．

ここで，協調型ユーザに対するPTSの一部を公開したり，独立自尊型のユーザにCLMの一部を公開するのは，あくまでpartitionの基本的な機能や動作を変更しないという前提で行なわれる．従って，そこで公開されるシステム機能あるいは置き換えるモジュールの選択薬，慎重に行なわなければならない．

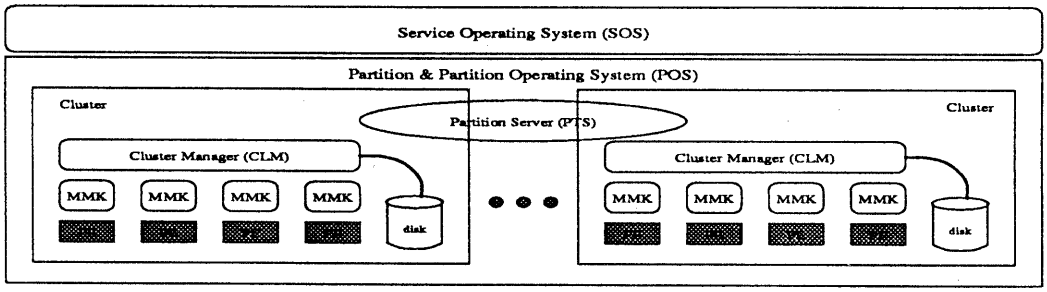

図 3: COS のソフトウェアアーキテクチャ

	POS			SOS
	MMK	CLM	PTS	
お任せ型ユーザ	既定	既定	既定	既定
協調型ユーザ	既定	既定	既定/自前	自前
独立自尊型ユーザ	既定	既定/自前	自前	自前

表 1: サービスインタフェースとソフトウェアアーキテクチャとの関係

このような保護を厳密に行なうためには，ハードウェア機能のサポートが必要である．上記の場合を考えると，ハードウェアでサポートされる実行モードは，少なくとも 4 レベルあることが望ましい．

5 POS 構成要素の機能

前節で述べた POS の構成要素の機能について考える．

5.1 Meta Micro Kernel

MMK の機能は PE 単位で必要な機能と共有資源へのアクセスなど CLM の介入を必要とする機能に関する例外処理機能を提供する．MMK の機能は以下の通りである．

1. **初期化**: 計算機システム起動時の初期化作業を行なう．
2. **プログラムの起動**: 割り当てられたメモリへ実行イメージのロードを行ない，実行を開始する．
3. **CLM に対する例外処理**: Disk I/O など共有資源へのアクセスや PE の獲得など CLM の介入を必要とする機能について CLM に対し通知を行ない処理を依託する．
4. **プロセッサの障害検知**: 要求にしたがってプロセッサを調べ，障害が発生しているか否かを調べる．

5.2 Cluster Manager

CLM は，各 cluster に 1 つずつ存在し，他の cluster の CLM と協調し資源の管理を行なう．実際の CLM は，cluster 内の PE のいずれかを用い，仮想プロセッサとして実行される．CLM の機能は以下の通りである．

1. **初期化**: 超並列計算機システムの最初の起動時のブートストラッピング，各種オペレーティングシステム構成要素のローディングと起動，システム停止時の処理を行なう．
2. **システム管理**:
   - ◇ **障害管理**: 各 MMK の障害検知機能，MBP およびメモリ，I/O の障害検知機能を用いクラスタの障害管理を行なう．
   - ◇ **システム更新**: 各オペレーティングシステムの構成機能要素の変更/更新を行なう機能．このため，資源毎に処理を一時停止し再開する機能を持つ．
3. **Partition 管理**: cluster の partition への割り当て，管理を行なう．
4. **メモリ管理**: メモリの partition への割り当て，管理を行なう．

5. **プロテクション管理**: パーティションを越えたメモリやCPUへのアクセス，I/Oへの直接アクセスを検知し管理を行なう．
6. **Partition 間通信**: Partition を越えた通信機能を提供する．
7. **I/O 管理**: 原則としてI/O処理はCLMを介して行なわれる．ただし，スワップ領域や一時記憶用の領域(例えば，動画像の格納に用いられるような領域)については割当を行なった後，各partitionが自己管理することが可能である．

5.3 Partition Server

Partitionの管理機能およびpartition間通信に関する機能は，CLMによって提供されるが，実際にはPTSによってpartitionは管理されている．PTSは，各partitionに1つずつ存在するサーバである．PTSは自分のpartition内の管理を行なうだけでなく，他のPTSと協調し，partitionの拡大縮小，partition間通信などを実現する．CLMのpartitionに関する機能は，実際にはPTSが行なう．

また，各partitionにはユーザがあり，ユーザの管理もPTSが行なう．

6 SOSの1つPUNIX

POSの上には複数のSOSが存在することができる．ユーザはシステムの提供するSOSを使うこともできるし，自分自身でSOSを書くこともできるし，POSの機能を直接使うこともできる．システムの提供するSOSはお任せユーザのためのものであり，ここでは，その1つであるPUNIXについて説明する．

6.1 PUNIXの基本方針

PUNIXはUNIXインターフェイスを提供するSOSである．UNIXインターフェイスを介するため超並列計算機を生で使う効率には比べられないが，UNIXインターフェイスがあることによって，これまでのコンパイラ，エディタ，デバッガなどの開発用ソフトウェアの移植が非常に容易になる．また，UNIXの部品の概念を超並列計算機に応用することも可能になる．UNIXでは単機能のプログラムを多数用意しておき，それらをパイプやファイルを使って組み合わせることによって複雑な処理を行なう．超並列

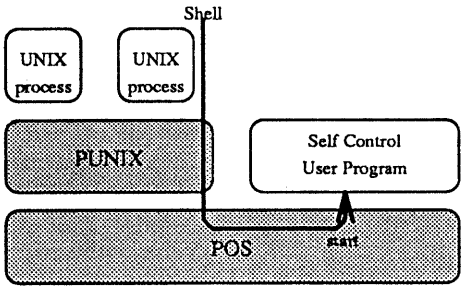

図4: PUNIXと独立自尊ユーザプログラム

計算においても，千台のPEを使って数値計算を行なうプログラムと，千台の計算機でビジュアル化するプログラムをパイプのようなもので結合することによって，計算と同時にビジュアル化を行なったりすることができると便利である．さらに，SOSを使わない独立自尊ユーザにとっても，そのアプリケーションを最初に起動するには，POS以上ものが必要であり，UNIXのシェルを利用することができる(図4)．

PUNIXはUNIXとコンパチブルなインターフェイスを提供するだけでなく，それを拡張し，超並列をあるていどサポートする．したがって，PUNIX上で実験，デバッグを行ない，その後で独自のSOSを作り動かすことも可能である．

6.2 プログラミングモデル

システムコールを設計する上で，超並列計算機上で，ユーザがどのようなイメージでプログラムすれば良いのかという，プログラミングモデルを決める必要がある．次のような概念を導入する．

- **プロセス**: 従来のUNIXと同じように処理の単位である．資源の管理などはプロセス毎に行なわれる．
- **スレッド**: 並列動作の単位である．1つのプロセスは1つ以上のスレッドを持つ．同一プロセスのスレッドが複数のPE上に分散して動作することがある(その場合が多い)．
- **仮想アドレス空間**: 通常，スレッドは共有メモリの形で並列実行するが，スレッド毎に別々の仮想アドレス空間を持たせることも可能である．これにより，オブジェクト指向のようにそれぞれのPE上で別々のことを行ないながら並列実行するモデルを実現することができる．

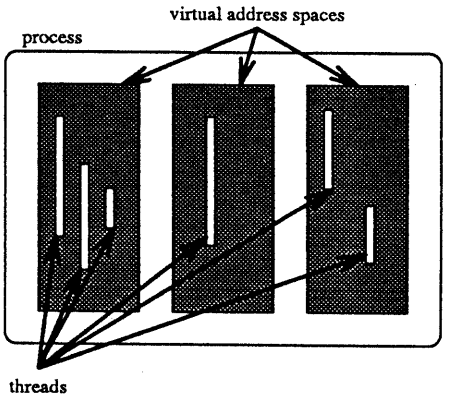

図5: PUNIX プログラミングモデル

- ユーザ: 従来のUNIXと同じ概念のユーザである．プロセス，ファイルの保護の単位である．POSのユーザとは異なるが，独立自尊ユーザの取り扱いから，必ずしも独立したものではなく，何らかの関係が必要である．

6.3 システムコール

UNIXと同じ名前のシステムコールは，コンパチビリティのため同じ引数を持つ．その機能は，UNIXと同じ使い方をする限り，UNIXと同じであるが，超並列のために拡張されているシステムコールがあり注意する必要がある．

6.3.1 プロセス

PUNIXでプロセスを作る唯一の方法は`fork`することである．これにより，新しいプロセスが元のプロセスの子プロセスとして作られる．新しいプロセスは，1つのスレッド，1つの仮想アドレス空間を持つ．仮想アドレス空間は`fork`を実行したスレッドの仮想アドレス空間をコピーしたものである．親プロセスの別の仮想アドレス空間はコピーされない．また，親プロセスにスレッドが複数あった場合にも，子プロセスのスレッドは1つである．

`exec`した場合には，そのスレッドの仮想アドレス空間を置き換え，エントリ・ポイントから実行を開始する．プロセス内の他のスレッドの仮想アドレス空間は変更しない．

`exit`すると，他のスレッドが動作している場合であっても強制的にプロセスを終了させる．

6.3.2 スレッド

スレッドには，スレッドIDがある．ただし，これはプロセスにローカルな番号である．`fork`で最初に作られたプロセスのスレッドIDが1である．

`thread_fork(n)`とすることによって，スタック以外の仮想アドレス空間を共有したn個の新しいスレッドが作られる．

`thread_exit`したばあいには，そのスレッドだけを停止させる．もし，そのスレッドがプロセス内の唯一のスレッドであった場合には，プロセス自身も停止させる．

スレッド関係のシステムコールとプロセス関係のシステムコールを組み合わせることにより，共有メモリマルチスレッド，非共有メモリマルチスレッドなどを実現することができる．`exec`した後に`thread_fork`すると共有メモリになり，`thread_fork`した後に`exec`すると非共有メモリになる．

6.3.3 ファイル

従来のUNIXと同じようにファイルデスクリプターがプロセス内に存在し，そのプロセスのスレッドが共有する．通常，ファイルへのポインタは1つであるが，別々のスレッドで書かれた内容は必ずしも時間通りに並ぶとは限らない．システムの都合により並び変えられる可能性がある．並び変えられては困る場合には，スレッド間で通信を行なう必要がある．

また，複数のスレッドが同時に入出力できるように，ファイルポインタをスレッド毎に持たせることも可能であり，1つのファイルを複数のスレッドで同時に別々の所を書くことができる．このようなデスクリプターをストライプデスクリプターと呼ぶ(図6)．

ストライプを設定するには`stripe_set(d, tid, n, size)`を使う．これにより，デスクリプターdはストライプとして設定され，tidで指定されたスレッドからn個のスレッドがストライプを行なう．sizeはそれぞれのストライプの大きさである．

パイプに対してもストライプを指定することができる．これにより2つのプロセス間で高速通信を行なうことができる．また，入力と出力でストライプの設定は異なってもかまわない．

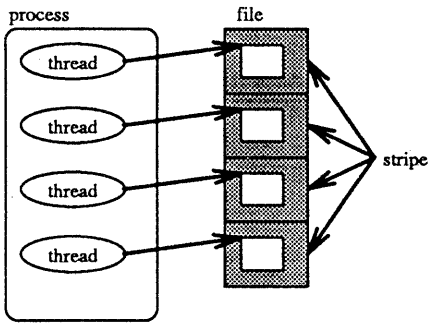

図6: ストライプファイルデスクリプター

6.3.4 同期

Dマシーンでは高機能メモリが提供され，単にこのメモリに読み書きすることにより種々の同期をとることができる．ここでは，高機能メモリをメモリオブジェクトとして取り扱い，それを仮想アドレス空間にマップすることにより利用する．

実際には直接メモリを読み書きせず，ライブラリを用意しておいて，それを介して利用した方が一般性がある．バリア同期，その他のライブラリを用意する予定である．ライブラリにせず，すべてをシステムコールとして実現することも可能であるが，それではDマシーンの性能を出すことができない．

7 開発環境

COSの開発環境では，いくつかの問題点があるが，以下にその概要を示す．

7.1 シミュレータとデバッギング

一般にOSの開発には，開発マシーンのシミュレータが役に立つ．また，新しいハードウェアアーキテクチャに対する開発ではなおさら必要となる．しかし，どの程度の詳細さでシミュレーションができれば良いかは，かならずしも決まっていない．

たとえば，簡単な並列計算機を利用して，POSのかなりの部分をシミュレートすることは出来よう．

また，並列プログラミングのデバッギングは，かなり困難な問題を引き起こす可能性がある．そこで，いろいろなレベルでのデバッギングツールの開発を行なわなければならない．これについては，既存のマイクロカーネルの技術が役立つと思われる．

7.2 システムの記述とその言語

システムの記述には，仕様記述と実際のプログラム記述がある．仕様記述に関しては，本来プログラミング言語と独立にあるべきで，そのような形式的な仕様記述言語があれば，試用してみる．

プログラミング言語としては，当面，C言語で行なうことが最も妥当位であると判断された．将来，オブジェクト指向を強く出す必要があれば，C++との混在も考えられる．

8 おわりに

本論文では，現在開発中の超並列オペレーティングシステムCOSについて，その構想を述べた．今後，POSおよびSOSのシュミレータ上での開発，Dマシーン上への移植などを行なっている予定である．

参考文献

[1] 文部省重点領域「超並列原理に基づく情報処理基本体系」シンポジウム予稿集，東京大学，9月，1992.

[2] Wulf, W. et al. HYDRA: The Kernelof Multiprocessor Operating System, CACM, Vol. 17, No. 6, pp.337-345, 1974.

[3] Tokuda, H. Evaluation of Real-Time Synchronization in Real-Time Mach, Proceedings of 2nd Mach Symposium, November 1991.