並列処理用OSカーネル "OMICRONV3" の開発とハイパOSによる共有メモリ型マルチプロセッサへの実装†

岡野裕之††* 横関隆††**
並木美太郎†† 高橋延匡††

共有メモリ型マルチプロセッサを対象にしたオペレーティングシステム，OS/omicron 第3版（以下 OMICRONV3）を開発した．OMICRONV3 は，タスクフォースをマルチプロセッサ環境で有効に機能させるために，タスクフォースを1つのプロセッサに束縛せず，タスクから見てプロセッサをアノニマスにする設計とした．このとき，対象ハードウェアがプロセッサに関して不均質であったため，ハードウェアに直接 OMICRONV3 を実装すると，OS の構成が複雑になることが予想された．そこで，ハードウェアと OS の間にハイパ OS 層を導入し，ハードウェアを均質化した．これによって，OMICRONV3 の内部でプロセッサをアノニマスに扱うことが可能となった．OMICRONV3 は，プロセッサをアノニマスとして実装した結果，シングルプロセッサ用の OS と比べて数か所の変更でマルチプロセッサに対応できた．したがって，ハイパ OS を使った実装が，OS の記述性，保守性を高める上で非常に有効であることが判明した．OMICRONV3 の設計は，OS 内の統一した資源環境，マルチタスクのデバッガ，OS の変更の容易性などを考慮して行った．本論文は，OMICRONV3 カーネルの設計と実現，マルチプロセッサへの実装方式などについて述べる．

1. はじめに

計算機の処理能力を向上する手段として，マルチプロセッサによる並列処理がある．マルチプロセッサを有効に利用するには，ハードウェアの方式と，その能力を引き出すオペレーティングシステム（以下 OS）をはじめとするシステムソフトウェアに関する研究が必要である．これらの研究分野では，システムの実現を通して得られる知見が重要であり，実際のシステムを評価することが成功への近道と考えられる．

そこで我々は，中粒度から粗い粒度の問題を対象とした並列処理用の OS を設計し，共有メモリ型マルチプロセッサ上に実現した．この OS を，OS/omicron 第3版（以後 OMICRONV3 と略す）と呼ぶ．

OMICRONV3 は，次のような特徴を持っている．

（1） プロセッサをアノニマス（無名，均質）とするタスク管理を行い，資源割付け，負荷分散を容易とする．

（2） OS 自体を，プロセッサをアノニマスとして記述し，上記(1)を実現している．

（3） カーネルが統一的に資源を管理し，確実な資源解放と高い拡張性を実現している．

（4） マルチタスクのデバッガを実現するためのスーパバイザコール（以下 SVC）を備えている．

共有メモリ型マルチプロセッサ上への実装の際には，上記(2)を実現するために，ハードウェアと OS の間にハイパ OS 層を導入して，ハードウェア資源をプロセッサに関して均質化する方法を提案し，その有効性を考察した．この結果，ハイパ OS を用いたアプローチは，次のような点で有効であることが分かった．

（1） OS に均質な仮想マシン・インタフェースを提供できる．

（2） OS に安全な割込みを提供できる．

（3） OS の移植性を向上できる．

（4） OS のデバッグを支援できる．

（5） ハードウェアの欠陥を補える．

本論文では，OMICRONV3 の構成，カーネルの設計について述べ，共有メモリ型マルチプロセッサへの実装と，そのときに用いたハイパ OS の設計などについて述べる．

---

† Development of an OS Kernel "OMICRONV3" for Parallel Processing and Its Implementation on a Shared-memory Multiprocessor Using a Hyper OS by HIROYUKI OKANO, TAKASHI YOKOZEKI, MITAROU NAMIKI and NOBUMASA TAKAHASHI (Department of Computer Science, Faculty of Technology, Tokyo University of Agriculture and Technology).

†† 東京農工大学工学部電子情報工学科
* 現在 (株)日本アイ・ビー・エム
** 現在 (株)ソニー

2. 基本設計

システム設計において，対象アプリケーションを明確にすることは重要である．これは，設計者にとって対象の不明確な“汎用”を追求すると，問題が発散してしまうからである．そこで，本章ではまず，OMICRONV3 が対象とするアプリケーションプログラムを示し，さらにそれを実行する並列処理モデルを示す．次に，OMICRONV3 を実装するハードウェアの概要を示し，実装方式を含めた，本システムの方式設計を示す．

2.1 対象とするアプリケーションプログラム

OMICRONV3 は，中粒度から粗い粒度の問題を対象とする．具体的には，我々の研究室で研究・開発を行っている，次のようなプログラムである．

（1） 日本語オンライン手書き文字認識システム
（2） 日本語形態素解析，推敲システム
（3） 囲碁プログラム

これらのプログラムには，次のような性質がある．

（a） プログラムを複数のフェーズ（前処理部，認識部など）に分割できる．また，フェーズ間のデータは順々に直列に流れる．

（b） フェーズ内の辞書検索などを並列化できる．このときの並列プロセス間の通信は疎であり，粒度は関数単位である．

これらの性質から考えて，**図1**のような並列処理モデルを採用した．これは，プログラムを複数のタスクフォースにより構成し，各タスクフォースをパイプラインで結んで並列に実行するという方式である[1),4)]．タスクフォースとは，1組の手続き領域と静的データ領域からなるロードモジュールであり，複数のタスクにより並列に実行される．

2.2 対象とするハードウェア

我々が理想とするマルチプロセッサシステムは，システムを構成する全プロセッサが協調して，1つの問題を解決するというものである．このときの処理モデルとして，前節のように，共有データを持つ並列タスク群（タスクフォース）を実現したい．したがってハードウェアとしては，均一なアドレス空間が各プロセッサに提供されるものがよい．そこで OMICRONV3 は，単一バス結合方式の共有メモリ型マルチプロセッサを対象にした．

このハードウェアは，VMEbus を用いて，MC 68020 によるプロセッサエレメント（以下 PE）を4台結合したものである（**図2**）．ハードウェアの仕様を**表1**に示す．表中にあるローカル・バスタイムアウトとは，バスが混み合って一定時間以内に VMEbus マスタになれない場合に起こるもので，これによって

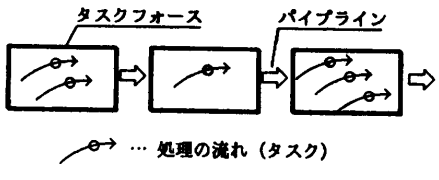


図1 並列処理モデル
Fig. 1 Model of parallel processing.

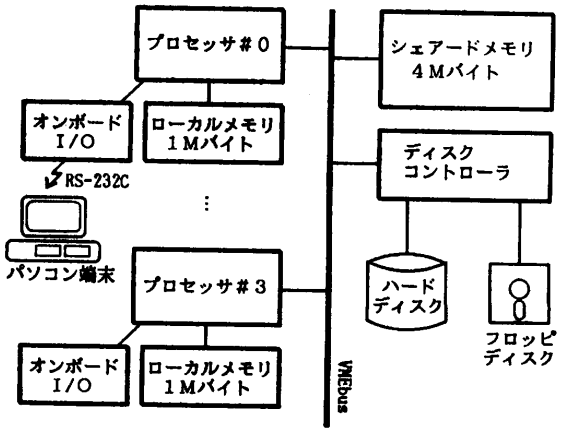


図2 対象ハードウェア
Fig. 2 Target hardware.

表1 対象ハードウェアの仕様
Table 1 Specification of the target hardware.

システムの仕様

バス	VMEbus
バス調停方式	シングルレベル方式
モジュール	PE 4枚 共有メモリ（4Mバイト） ディスクコントローラ
ディスク	フロッピディスク1台（5インチ HDD） ハードディスク1台（40Mバイト）
端末	PC-9801 VM

PE の仕様

プロセッサ	MC 68020 (20 MHz)
コプロセッサ	MC 68881 (20 MHz)
ローカルメモリ	1Mバイト
グローバル・バスタイムアウト	50～57 μs
ローカル・バスタイムアウト	13～15 ms

プロセッサはバスエラーとなる．本システムではこれが悪影響を及ぼしたが，ハイパ OS でバスアクセスをリトライすることにより，OS には完全な仮想マシン・インタフェースを提供した．これについては第6章で述べる．

このシステムでは，各 PE のローカルメモリがグローバルなアドレス空間にマッピングされており，互いのローカルメモリにアクセスできる．ただし，各 PE に付属しているローカル I/O は，その PE しかアクセスできないという制限がある．このため，このシステムでは，ハードウェアの資源がプロセッサについて不均質である．

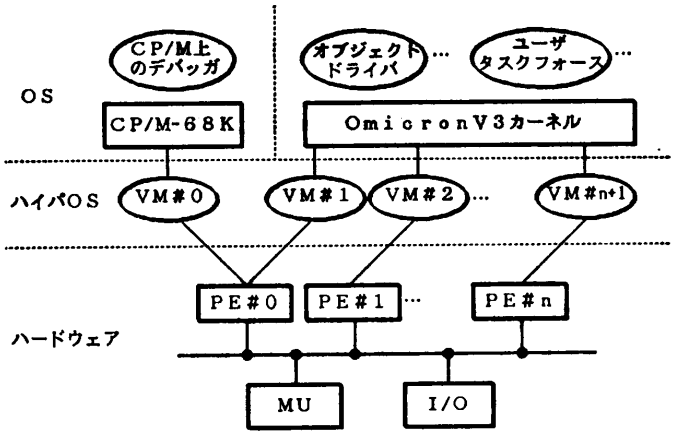
2.3 システムの方式設計

ハードウェアに直接 OS を載せる場合，OS の設計はハードウェアの仕様に少なからず影響を受ける．特にマルチプロセッサシステムではその傾向が強い．例えば，前節で示した我々の対象ハードウェアでは，ハードウェア資源がプロセッサに関して不均質であるため，OS 自体も不均質とならざるを得ない．すなわち，あるプロセッサでしかディスパッチできないタスクを設けてそれらをローカルメモリに置くなど，不均質な設計を行うことになる．この場合，負荷分散を行うにはタスクの移動 (migration) が必要となる．あるいは，タスクから見てプロセッサをアノニマス（どのプロセッサでもディスパッチ可能）とする場合，OS 内でプロセッサ間通信が必要となる上に，I/O 処理の手続きをすべて OS に組み入れなくてはならず，OS は大規模で複雑になる．これに対し我々は，ハードウェアに合せるのではなく，あくまでもアプリケーション指向で OS を設計したい．また，OS の実現に関しても，ハードウェアに依存した問題で OS の構造を複雑にしたくないと考える．

OMICRONV3 では並列処理モデルとしてタスクフォースを採用しており，1つのアプリケーションプログラムが3，4個のタスクフォースに分かれることを想定している．したがって，タスクフォースをある PE のローカルメモリに置くなどして1つのプロセッサに束縛する設計にすると，高々3，4倍の並列性が限界となってしまう．そこで OMICRONV3 は，タスクから見てプロセッサがアノニマスな設計とする．こうすると，タスクがプロセッサに依らずにスケジューリングされ，プロセッサの負荷が均一になり，全体として効率がよい．

このような設計の OS を不均質なハードウェアに直接実装すると，先に述べたように OS が複雑になる．原理的に，プロセッサに関して均質な共有メモリ型マルチプロセッサがありうるにもかかわらず，対象ハードウェアに固有の問題で OS を複雑にするのは好ましくない．そこで筆者らは，OS とハードウェアの間にハイパ OS 層を導入し，ハイパ OS でハードウェアを均質化することを考えた．この均質な仮想マシン・インタフェースの上で，OMICRONV3 自体を，プロセッサをアノニマスとして記述する．OS 内部でプロセッサ番号に依存した処理を行わなければ，上記のタスクレベルのアノニマス性* は自然に実現できる．

この方式ではハイパ OS が，プロセッサに1対1で対応する仮想マシンを OS に提供する（図3，詳細は第5章で述べる）．仮想マシンには，プロセッサ (MC 68020) の機能のほかに，ハイパ OS への SVC として I/O へアクセスする機能を持たせる．このときの SVC には，プロセッサ番号による制限を加えない．このような設計とすることによって，プロセッサ間通信の手続きを OS から排除し，ハイパ OS にそれを任せることによって，OS の構造を簡単化することができる．また，第6章でも述べるとおり，ハードウェアの不完全さをハイパ OS で隠ぺいでき



VM : Virtual Machine
PE : Processor Element
MU : Memory Unit

図 3 システムの構成
Fig. 3 System structure.

* アノニマス性とは，プロセッサをアノニマス（無名）に扱うことを意味する．

るなどの利点が得られる．さらに，OS のデバッグ，マルチプロセッサ用 OS の性能測定などの観点からも，ハイパ OS を用いたアプローチは有効である．

本システムの方式設計を，次にまとめる．

（1） タスクから見てプロセッサをアノニマスとする．

（2） OS 内部でプロセッサをアノニマスに扱う．

（3） ハイパ OS でハードウェアを均質化する．

3. OMICRONV3 の設計

筆者らは，OS/omicron の第3版である **OMICRONV3** を設計した．OS/omicron とは，我々が1980年から研究・開発を行っている，日本語情報処理用の実記憶系 OS である[1),2)]．**OMICRONV3** の設計に当たっては，シングルプロセッサ上に実現されていた OS/omicron 第2版[3),4)] の問題点を明らかにして設計方針を立てた．さらに，OS に拡張性を持たせること，OS 内部に並列性を持たせることなどに着目して設計を行った．

3.1 設計方針

設計に先立って OS/omicron 第2版を評価した結果，次のような問題が明らかになった．

（a） OS 内の資源解放を確実に行えない．

これは，OS を動的に拡張する機構であるユーザ拡張部が，カーネルとは独立に資源管理を行うためである．

（b） タスクに親子関係があるためプログラミングスタイルが制限される．

（c） マルチタスクのデバッガの構築が困難である．

これらの問題を受けて，**OMICRONV3** は次のような設計方針を採った．

（1） OS 内の資源をカーネルが統一的に管理する．

ただし，OS の記述性，保守性の点から，動的に拡張できる資源管理モジュールを実現したい．これは，後で述べるように，OS 内部に並列性を持たせる点においても有効である．そこで，各資源管理モジュール（オブジェクトドライバと呼ぶ）が管理するファイル，ウィンドウ，かな漢字変換機能などの資源を，どのタスクフォースで使用しているかをカーネルが集中管理することにした．これによって，タスクフォース間の資源の引継ぎ，タスクフォースがアボートした時の資源解放などが確実に行える．

（2） タスクを，親子関係のないフラットなものとし，プログラミングスタイルに柔軟性を持たせる．

（3） 親タスクフォースが子タスクフォースを監視できるようにし，マルチタスクのデバッガを構築可能とする．

さらに，OS 内のモジュラリティを高めるために，次のような方針を採った．

（4） 1つのカーネルと複数のオブジェクトドライバによって構成する．

（5） ファイルシステム[5)] を1つのオブジェクトドライバとして実現する．

このオブジェクトドライバを，特化したタスクフォースとして実現する．したがって，これらはマルチタスクで動作し，並列に実行される．OS の中で重要な位置を占めるファイルシステムを並列に実行することは，性能向上に有効である．

また，並列処理 OS として，次の基本方針がある．

（6） タスクから見てプロセッサをアノニマスとする．

つまり，OS 内のレディリストを1本とし，すべてのプロセッサでタスクをディスパッチ可能とする．このほかに，タスクを特定のプロセッサに固定し，プロセッサ間の負荷が不均一になるとタスクを移動 (migrate) する方式がある．しかし，この方式ではタスクフォースを有効に機能することができない．タスクから見てプロセッサをアノニマスとする方式は，スケジューリングが一意になるため，並列アプリケーションの記述性の点でも優れている．

3.2 OMICRONV3 における資源管理

OMICRONV3 は，カーネルとオブジェクトドライバ群によって構成される（**図4**）．ユーザから OS への呼出し (SVC) は，いったんカーネルが受け取り，カーネル内で完結する処理であればそれを実行する．オブジェクトドライバへの呼出しが必要な場合は，カーネルが，対応するメッセージをオブジェクトドライバへ渡して処理を要求する（**図5**）．

各オブジェクトドライバが提供する SVC セット

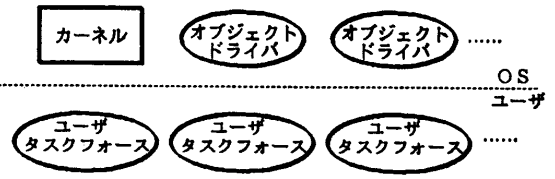


図 4 OMICRONV3 の構成
Fig. 4 Structure of OS/omicron version 3.

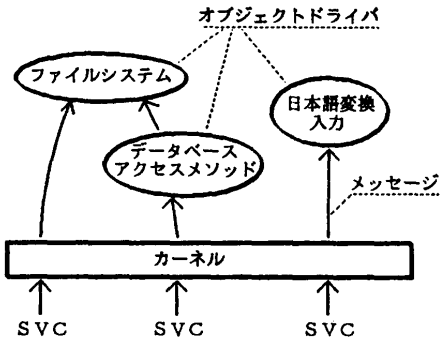


図 5 オブジェクトドライバの呼出し
Fig. 5 Calls to object drivers.

は，OS 内の一貫性のために，open, read, write, close などのプリミティブで統一した．このインタフェースで管理される資源をオブジェクトと呼ぶ，オブジェクトは，任意のバイト幅を単位としてアドレス付けされる資源であり，例えばファイルならば，1バイトを単位とする配列とみなされる．read/write の SVC インタフェースは，並列処理を考慮して，次のように必ずアドレスを指定する仕様とした．

read (obj_id, address, buffer, size);

これは，ランダムアクセスファイルあるいはメモリマップド I/O と同様の，自由度の高いインタフェースである．

さらに，オブジェクトドライバだけでなく，子タスクフォースのメモリ，タスクフォース間通信経路もオブジェクトとして扱い，オブジェクト ID (obj_id) を通じてこれらにアクセスする仕様とした (**図 6**)．これによって，タスクフォースから外部に見える資源をすべてオブジェクトで統一し，カーネルの内部構造を簡潔にすることができる．

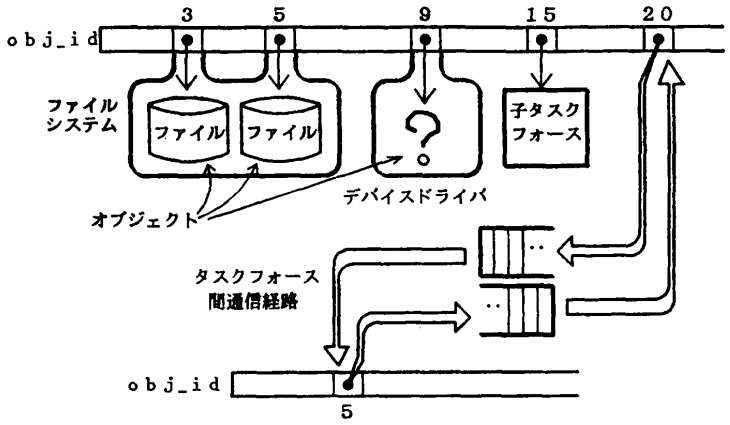


図 6 資源とオブジェクト ID のバインド
Fig. 6 Binding of object ID's and resources.

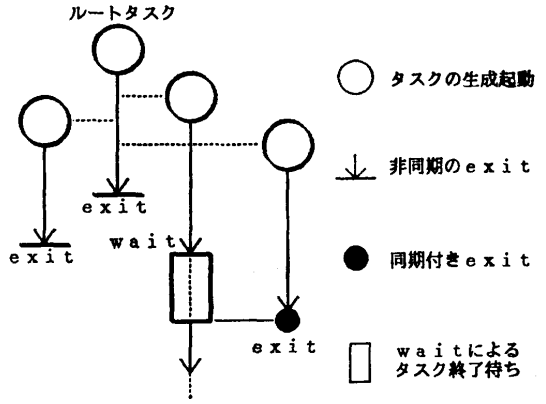


図 7 タスクの実行形態
Fig. 7 Execution of tasks.

3.3 タスクの実行形態

OMICRONV3 では，タスクに親子関係を設定せずフラットに扱うことにした．ただし，この仕様においても木構造の同期関係を作ることができ，自由度は高い．この仕様を示す．

タスクフォースが生成されるとリーダタスクが起動される．リーダタスクへの引数やスタックサイズはタスクフォース起動時の SVC で指定する．リーダタスクは，同一タスクフォース内にメンバタスクを起動でき，このとき関数へのポインタ，引数，スタックサイズを指定する．タスクは完全にフラットに扱われ，リーダタスクが先に消滅することも可能である (**図 7**)．

タスク間の同期は，PV セマフォと終了待ち SVC で行う．タスクの終了待ちは次のように行う．

wait (tid);

このとき，tid には自分以外のすべてのタスクを指定でき，親子関係などの制約は受けない．タスクの終了は次のように行う．

exit (retval);

retval は整数値であり，−1 を指定する場合とそうでない場合で別の処理が行われる．retval に −1 を指定した場合，自分を wait しているタスクがいなくても即終了処理が行われ，消滅する．retval に −1 以外を指定した場合，自分が wait されていなければ，誰かに wait されるまで待ってから終了処理が行われる．このときの retval の値が wait SVC の戻り値となる．

exit SVC の仕様が上記のようになっているのは，タスク間の完全性を保つためである．タスク間に親子関係の木構造が存在する場合，親が wait するまで子を残すことによって完全性を保証できる．これに対し，タスクをフラットに扱うと，同期が必要なのかどうかをユーザが明示的に指定しなければならないからである．

3.4 タスクフォースのデバッグ

OMICRONV3 は，デバッグ対象タスクフォースの親タスクフォースとしてデバッガを構築できるようにした．この仕様を示す．

マルチタスクのデバッガを構築する場合，子タスクフォース内のタスクあるいは子タスクフォースのメモリについて，次のような操作ができる必要がある．

（1） 任意のタスクを停止する．
（2） 任意のタスクを消去する．
（3） 任意のタスクをトレースで実行し，トレースストップ状態を知る．
（4） 停止しているタスクのレジスタ値を知り，それを変更する．
（5） タスクの生成，消滅の状態を知る．
（6） メモリにアクセスする（ブレークポイントの埋込み，メモリダンプ）．

これを実現するために，子タスクフォースで起こるイベントをメッセージとして親タスクフォースへ送り，親タスクフォースがそのメッセージに対して操作を加えるというモデルを採った（図8）．タスクフォース内のイベントには次のものがある．

- タスクの生成
- タスクの停止
- タスクの終了
- タスクのアボート
- タスクフォースの終了
- タスクフォースのアボート

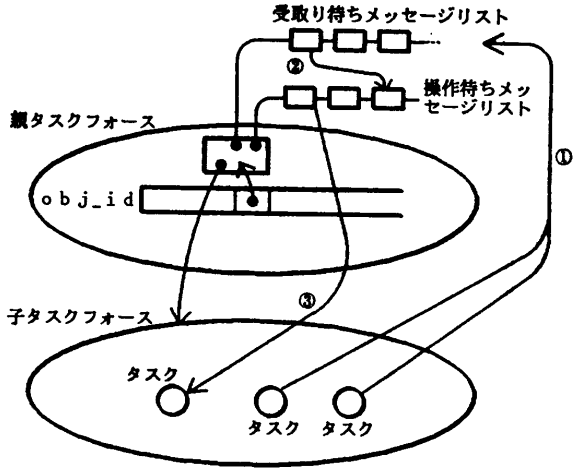
ただし，タスクに関するメッセージは，子タスクフォースがデバッグモードの時に届けられる．デバッグモードのタスクフォースがタスクを起動すると，親タスクフォースにタスク生成のメッセージが入り，タスクは停止状態となる．タスクがトレースストップなどで停止した場合は，タスク停止のメッセージが入る．親タスクフォースはタスク生成，タスク停止のメッセージに対して，

- タスクの実行再開（通常モードかトレースモードを選択）
- タスクのレジスタ状態の変更
- タスクの消去

という操作を加えることができる．また，タスク終了，タスクフォース終了のメッセージに対しては，

- タスクの消去
- タスクフォースの消去

という操作が可能である．親タスクフォースが実行再開の操作を加えるまで，デバッグ対象のタスクが同期して停止しているので，デバッガはタスクの実行順序を自由に変えられる．したがって，デッドロックの起こりそうなクリティカルなタイミングをテストすることなどができる．



① … タスクでイベントが発生した．
② … メッセージを受け取った．
③ … メッセージに対して操作を加えた．

図 8 子タスクフォースの監視
Fig. 8 Monitoring of tasks in a child task force.

4. OMICRONV3 の実現

OMICRONV3 は，タスクから見てプロセッサをアノニマスにする設計とした．しかし，対象とするハードウェアが不均質であるため，ハードウェアに直接 OS を実装すると OS の構造が複雑になる．そこで，ハイパ OS によってハードウェアを均質化することにより，OS 自体を，プロセッサをアノニマスとして記述した．これにより，必然的に OS 上のタスクのアノニマス性が得られる．

4.1 実現の概要

OS/omicron 第2版はカーネルを1つのタスクフォースとみなし，ユーザタスク1つにカーネルタスク1つを用意するというモデルを採用した[1),4)]．この方式はコーディングを容易にした反面，カーネルタス

クの異常終了の処理が困難であり，カーネルタスクごとにスタックを用意するため多くのメモリを必要とした．そこで OMICRONV3 では，カーネルをタスクとして実行するのではなく，不可分の処理として実行するようにした．

また，カーネル全体をクリティカルレジョンとし，処理を直列化した（第2版ではカーネルタスクがマルチタスクで動いていた）．これは記述性，保守性を考慮した方針である．カーネルを直列化しても，オブジェクトドライバ群が並列に実行されるので，OS 内部には並列性がある．なお，カーネル処理はすべてのプロセッサ上で実行され，特定のプロセッサに固定されることはない．

4.2 実現の結果

実現の手順として，マルチプロセッサへの拡張を考慮した上で，まずシングルプロセッサで動作させることを目標にカーネルを実現する．これをシングルプロセッサ上でデバッグしてから，マルチプロセッサ版への拡張を行うという方針を採った．実際の実現結果では，この拡張に伴う変更は次の3つであった．

（1） 初期化を1つのプロセッサだけで行うようにする（早いもの勝ちで初期化ルーチンを実行）．

（2） レディリストなどの処理待ちリストへタスクをつないだあと，アテンション割込みを発行する．

（3） 割込みエントリルーチンでアテンション割込みを無視し，直接ディスパッチルーチンへ抜ける．

アテンション割込みとは，ハイパ OS が提供するプロセッサ間割込み機能である（次章参照）．アテンション割込みは，各仮想マシン（プロセッサ）に注意をうながすための割込みであり，対応する割込み処理タスクがないため，上記(3)の処理を行う．

このように，シングルプロセッサ用からマルチプロセッサ用への拡張において3つの変更だけで済んだのは，本システムが OS レベルでプロセッサをアノニマスとする設計を採ったからである．現存する他の OS は，ユーザプログラムのレベルでプロセッサがアノニマスであっても，OS 内ではプロセッサ番号を意識した処理を行うものが多い．この場合，ターゲットマシン固有の局所性（ローカルメモリ，ローカル I/O）に起因する複雑な処理を，すべて OS 層で吸収しなければならない．本システムの場合，この処理をすべてハイパ OS で行い，プロセッサに関して均質な仮想マシン・インタフェースという断面で，ハイパ OS と OS を切り分ける．この方式により，OS の負担が大幅に軽減できることが判明した．

5. 共有メモリ型マルチプロセッサへの実装

OMICRONV3 を，VMEbus による共有メモリ型マルチプロセッサ上へ実装した．このハードウェアは，2.2 節で示したようにローカルメモリとローカル I/O を持ち，プロセッサに関して不均質である（図2）．そこで，OS 層とハードウェアの間にハイパ OS 層を導入し，ハードウェアを均質化した．

5.1 ハイパ OS の設計

本ハイパ OS は次のことを目的とする．

（1） ハードウェアの均質化

（2） ハードウェアの仮想化

（3） OS のデバッグの容易化

上記(1)の均質化には，メモリと I/O の2つの側面がある．メモリに関しては，対象ハードウェアの仕様上，均質化するのは困難である．また，スヌープキャッシュ[6]によってバス競合を減少する技術が確立されており，ローカルメモリを積極的に利用するメモリ管理システムを作成するのは無駄である．したがって，ローカルメモリは読出し専用とし，ハイパ OS および OMICRONV3 の手続き領域を置くことにする．また，I/O の均質化に関しては，ハイパ OS 内でプロセッサ間通信を行い，I/O を実行できるプロセッサに処理要求を出すことで実現する．

このときの I/O のインタフェースには，我々が開発したワークステーション用ハイパ OS，多重 OS「江戸」[7]との互換性を考慮した．本ハイパ OS が提供する仮想マシン・インタフェースの仕様（SVC セット）を「江戸」とコンパチブルにすれば，全く同じ OS を2つのシステムで動かせることになる．しかし，対象ハードウェアにはビットマップディスプレイ，マウスが付いておらず，「江戸」とコンパチブルな仮想マシン・インタフェースにすることができない．そこで，本ハイパ OS は，「江戸」からウィンドウ関係の SVC を除いた，サブセットの仮想マシン・インタフェースを提供することにした．

上記(3)については，「江戸」を開発した経験上，動作の安定した OS をデバッグ中の OS と並行に動かすことが，デバッグに有効であるという認識がある．そこで，本ハイパ OS では，CP/M-68K と OMICRON V3 を並行に動かすことにした．デバッグ用の OS として OS/omicron 第2版でなく，CP/M-68K を選んだ理由は，CP/M-68K が少ないメモリ，ディスク領

域で動作できることによる．CP/M-68K が 0.5M バイト足らずで動作できるのに対し，第2版は少なくとも 2M バイト必要である．

本ハイパ OS は，「江戸」と同様に特権命令エミュレータと割込みエミュレータを用いることによって，図3のような仮想マシンを提供することにした．PE #0 は VM #0, #1 を提供し，これらを割込み駆動でスイッチ（OS スイッチ）しながら，擬似的に並列実行する．VM #2〜#n+1 は，OS スイッチなしで実行する．

5.2 ハイパ OS の動作原理

本ハイパ OS が対象とするプロセッサは MC 68020 である．MC 68020 は，ユーザ状態，スーパバイザ状態という2つの状態を持っており，いくつかの特権化された命令（特権命令）はユーザ状態で実行不可となっている[8]．特権命令をユーザ状態で実行しようとすると，特権違反例外が起こり，スーパバイザに制御が移るようになっている．ハイパ OS はこの機能を利用して，PE #0 における VM #0, #1 の擬似的な並列実行を実現する．つまり，仮想マシンをユーザ状態で実行し，特権命令をエミュレートすることによって，プロセッサ資源を各仮想マシンに分配する[9]．VM #2〜#n+1 も同様に，ユーザ状態で実行する．

5.3 ハイパ OS の実現

本システムのメモリ配置を図9に示す．各 PE から見て，それぞれのローカルメモリが0番地から始まるので，これを利用して手続き領域と割込みベクタテーブルをローカルメモリに置いた．

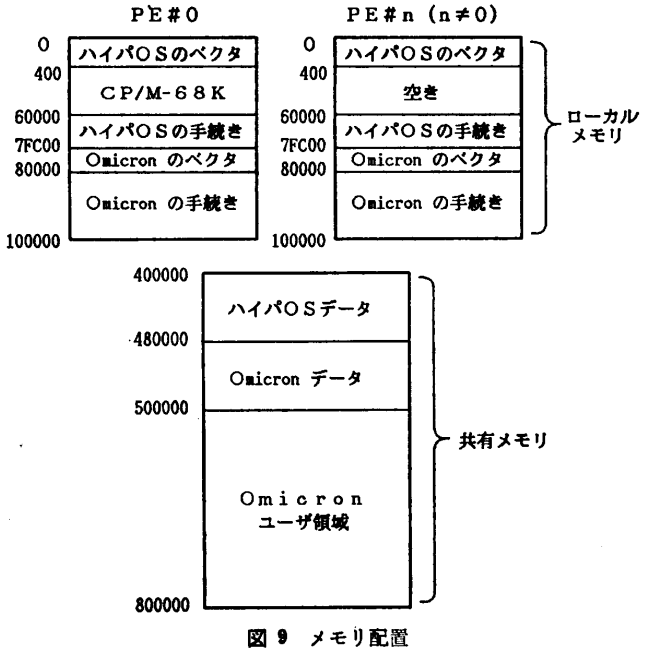


図 9 メモリ配置
Fig. 9 Memory allocation.

カーネルの主要部分である特権命令エミュレータ，割込みエミュレータは，多重 OS「江戸」のものを利用した．これは，マルチプロセッサでの実現を考慮して「江戸」を設計したために可能となった．ハイパ OS の SVC セットはほぼ「江戸」のサブセットとし，アテンション割込みなど，いくつかの SVC を追加した．アテンション割込みとは，各仮想マシンに一斉に割込みをかける機能である．仮想マシン上の OS は，タスクのレディキューをスキャンするために，このアテンション割込み機能を利用する．アテンション割込みは，ハイパ OS がエミュレートする仮想割込みであり，このベクタ，優先レベルは OS 側から自由に設定できる．

6. 議 論

6.1 他のマルチプロセッサ OS との比較

共有メモリ型マルチプロセッサを対象にした他の国内の OS と，**OMICRONV3** を比較した．

SKY-1[10]

SKY-1 は特化したプロセッサを扱うことができる．スレッド単位にディスパッチ可能なプロセッサの属性を持たせて，これを管理する．また，キャッシュのヒット率を考慮し，なるべく同じプロセッサにスレッドを割り当てる．

前者は，**OMICRONV3** にない特徴である．SKY-1 はハイパ OS 層を持たないので，この設計は有効だろう．これに対し **OMICRONV3** では，特化したプロセッサをハイパ OS 層で仮想化し，SVC あるいは拡張命令セットなどの仮想マシン・インタフェースによって OS に提供する方針である．後者に関しては，**OMICRONV3** の対象ハードウェアにキャッシュがないため，今回は考慮しなかった．

TOP-1 OS[11]

TOP-1 OS は，1つのプロセッサをカーネル用，他のプロセッサをユーザ用とし，カーネル用プロセッサだけが OS のコードを実行する．これは，ハードウェア資源が不均質であることによる．また，カーネルの機能分割を進め，マルチサーバモデルとすることを指向している．

これに対し **OMICRONV3** は，OS から見てプロセッサをアノニマスとしたため，OS のコードを全プロセッサで実行できる．このとき，カーネル部分を直列化しているが，タスクフォースとして動作するオブジェクトドライバが並列に実行される．**OMICRONV3** は，マルチサーバモデルになっていると言える．

MUSTARD[12]

MUSTARD は，OS をリアルタイムカーネル(RK)と UNIX カーネル (UX) の2層に分けており，リアルタイム性が要求されるタスクを RK が直接扱う．また，UX は RK の資源（ディスパッチャなど）を利用して実現されており，この層ではプロセッサをアノニマスとしている．基本的に均質なハードウェアを仮定しているが，プロセッサにローカルな資源をアクセスする機能も用意されている．

OS を2層に分けている点で **OMICRONV3** と似ているが，その目的は異なる．MUSTARD が既存 OS にリアルタイム性を付加しようとしているのに対し，**OMICRONV3** はハードウェアの均質化，仮想化をねらっている．また，**OMICRONV3** のハイパ OS は，内部にハードウェアのハンドラを持っているため，リアルタイム性も達成できる．さらに，ハイパ OS は仮想マシンとして動作するため，タスクのディスパッチやスケジューリングがすべて OS 側に任され，OS の自由度が高い．OS の実験環境としては，**OMICRONV3** のアプローチの方が有効である．

6.2 ハイパ OS の有効性

本ハイパ OS の実現によって，次のような利点が得られた．

（1） OS に対して均質な仮想マシン・インタフェースを提供した．

ハイパ OS 上では，どの仮想マシンに対しても同じ仮想マシン・インタフェースが提供される．したがって，仮想マシンで実行される OS はプロセッサをアノニマスに扱うことができる．第4章で示したように，プロセッサをアノニマスとすると，シングルプロセッサ用の OS に数カ所の変更を加えるだけでマルチプロセッサ用に拡張できるため，OS の記述性の点で非常に有効である．また，OS の構造がシングルプロセッサ用のものと同様になるので，OS の保守性の点でも有効である．

（2） OS に対し安全な割込みを提供した．

ハイパ OS の提供する仮想マシン上で実行される OS には，仮想的な割込みが提供される．OS に対してエミュレートされる割込みはハイパ OS 内でキューイングされるので，OS が長期間割込み禁止状態になっても，OS は割込みを取りこぼすことがない．

（3） OS の移植性を向上した．

ハイパ OS 層で I/O を仮想化することにより，機種間のハードウェアの仕様の違いを吸収できる．また上記(2)のように，時間的な制約もハイパ OS で吸収できる．本システムのようなマルチプロセッサのハードウェアは，プロセッサにローカルな I/O が存在するため，シングルプロセッサと比べて機種間の違いが大きいと考えられる．したがって，上記(1)の特徴と合わせて，仮想マシン・インタフェースで機種間の差異を吸収できるハイパ OS は，マルチプロセッサの場合特に有効な手段となる．

（4） OS のデバッグを容易とした．

既存 OS と開発中の OS を同時に実行し，既存 OS 上のデバッガを利用した．OS をインクリメンタルに開発する場合，ある時点で不足している機能（ファイル入出力など）を既存 OS が補助することも可能である．また，ハイパ OS 内にデバッガを組み込むというアプローチも今後考えられる．

（5） ハードウェアの欠陥を補った．

本システムで使用した対象ハードウェアは，バス競合の度合いが大きくなると，プロセッサがバスエラーを起こすという欠陥がある．これは，一定時間以内に VMEbus マスタになれないと，ローカル・バスタイムアウトとみなされるからである（表1）．ハイパ OS では，この欠陥を補うために，バスエラーのハンドラでバスアクセスをリトライするようにした．この対処で，仮想マシン・インタフェースとしては欠陥のないハードウェアを OS に提供することができた．本ハイパ OS で採ったこの手法は，動作の不安定なハードウェア上に OS を実現する場合，一般に適応できると考えられる．また，コプロセッサ命令のソフトウェアエミュレートなど，ハードウェアにない機能を補うためにもハイパ OS を利用できる．

ハイパ OS の利用を一般に考えた場合，上記(4)，(5)はシステムの開発段階で得られる利点であるが，(1)，(2)，(3)は OS の信頼性，保守性を上げるために利用できる点である．(1)に関しては，対象ハードウェア自体を均質化すれば対処できる問題だが，それでは特化したプロセッサを扱うことができない．パーソナルユースのマルチプロセッサシステムの場合，個人のニーズに合わせて特化したプロセッサや

I/O を付加できるのが望ましい．これを，OS の記述性，保守性を低下することなく行うには，本システムのようなハイパ OS によるアプローチが有効である．

6.3 OMICRONV3 の成果

OMICRONV3 の実現によって，次のような成果が得られた．

（1） 並列処理の研究基盤を提供した．

実際のマルチプロセッサで並列アプリケーションを実行し，実行時間，プロセッサの台数効果などを計測することが可能となった．これによって，タスクフォースを用いた並列プログラミングの研究，それを支援する言語処理系の研究などの基盤を提供した．

（2） 資源を統一的に管理する，信頼性の高い OS カーネルを提供した．

資源をオブジェクトという形で統一し，オブジェクトドライバで管理する機構を実現した．このとき，資源の所在をカーネルで集中管理することにより，システムの信頼性を確保することができた．また，オブジェクトというモデルを設定することにより，OS を拡張する際のインタフェースの設計に，統一性のある指針を与えることができた．

（3） マルチタスクのデバッガを構築できる SVC セットを提供した．

親タスクフォースから子タスクフォースを監視するための機構，SVC セットを設計し，実現した．これによって，タスクの実行順序を制御するなど，タスクフォースプログラミングを支援するデバッガの構築を可能とした．

7. おわりに

本研究によって次の成果が得られた．

（1） ハイパ OS により，プロセッサに関して均質な仮想マシン・インタフェースを実現し，マルチプロセッサ用 OS の記述性，保守性を高めた．

ハイパ OS でハードウェアを均質化することにより，OS 自体を，プロセッサをアノニマスとして記述した．この結果，シングルプロセッサ用 OS を数カ所変更するだけでマルチプロセッサに対応できることが分かり，OS の記述性，保守性を向上できることが分かった．

（2） カーネルが統一的に資源を管理する OS，OMICRONV3 を設計・実現した．

これにより，カーネルの内部構造を簡潔とし，OS の信頼性，記述性，拡張性を向上した．

（3） タスクフォース機構を持ち，タスクのアノニマス性，タスクのフラットな扱いなどを特徴とするタスク管理を設計・実現した．

（4） さらに，マルチタスクのデバッガを実現するための機構，SVC を設計し，実現した．

これらにより，並列処理プログラミングを容易とし，並列アプリケーションの実験環境を提供すると共に，デバッガを含むプログラミング環境の研究基盤を提供した．

今後の課題としては，OMICRONV3 のオブジェクトドライバとして，ウィンドウシステム，ネットワークシステムなどを構築すること，並列アプリケーションの研究を通じて，OS の仕様を評価することなどが残されている．

参考文献

1) 鈴木，小林，田中，中川，高橋：OS/omicron における日本語プログラミング環境，情報処理学会論文誌，Vol. 30, No. 1, pp. 2-11 (1989).
2) 並木，屋代，田中，篠田，藤森，中川，高橋：OS/omicron 用システム記述言語 C 処理系 Cat のソフトウェア工学的見地からの方式設計，電子情報通信学会論文誌 (D)，Vol. J71-D, No. 4, pp. 652-660 (1988).
3) 鈴木，田中，岡野，堀，横関，並木，高橋：OS/omicron 第2版の実現と評価，情報処理学会 OS 研究会資料，42-1 (1989).
4) 並木，鈴木，岡野，堀，横関，中川，高橋：マルチプロセッサ向けの OS/omicron タスク管理の設計と実現，情報処理学会論文誌，Vol. 31, No. 6, pp. 894-905 (1990).
5) 横関，岡野，並木，高橋：OS/omicron 第3版ファイルシステム内部アーキテクチャの設計と実現，情報処理学会 OS 研究会資料，47-2 (1990).
6) 鈴木，多田，梅村：マルチプロセサー用キャッシュメモリーの設計，電子情報通信学会，EC 82-63 (1982).
7) 岡野，堀，中川，高橋：多重 OS「江戸」の設計と実現，情報処理学会論文誌，Vol. 30, No. 8, pp. 1012-1023 (1989).
8) MC 68020 ユーザーズ・マニュアル，CQ 出版社，Motorola Inc. (1986).
9) Buzen, J. P. and Gagliadi, U. O.: The Evolution of Virtual Machine Architecture, *National Computer Conference*, Vol. 42, pp. 291-299 (1973).
10) 山口，上脇，斎藤，小林，坂東：並列処理用 OS SKY-1 開発構想，第39回情報処理学会全国大会論文集，3P-1 (1989).
11) 穂積，白鳥，吉永，大澤，山崎，河内谷，森山：TOP-1 オペレーティングシステム(1)基本方針，

第 39 回情報処理学会全国大会論文集，3P-4 (1989).
12) 広屋，桃井，宮地：マルチプロセッサ・リアルタイム UNIX MUSTARD，並列処理シンポジウム JSPP '90 論文集，pp. 241-248 (1990).

(平成 2 年 7 月 12 日受付)
(平成 3 年 2 月 12 日採録)

岡野　裕之（正会員）

昭和 63 年東京農工大学工学部数理情報卒業．平成 2 年同大学院修士課程修了．同 4 月(株)日本 IBM に入社．在学中．仮想マシン，マルチプロセッサ用オペレーティングシステムの研究に従事．

横関　　隆（正会員）

昭和 63 年東京農工大学工学部数理情報卒業．平成 2 年同大学院修士課程修了．同 4 月(株)ソニーに入社．在学中，追記型光ディスクを用いた世代管理ファイルシステム，マルチプロセッサ用オペレーティングシステムの研究に従事．

並木美太郎（正会員）

昭和 59 年東京農工大学工学部数理情報卒業．昭和 61 年同大学院修士課程修了．同 4 月(株)日立製作所基礎研究所入社．昭和 63 年より東京農工大学工学部数理情報助手．平成元年 4 月より電子情報助手．並列処理，日本語情報処理のソフトウェア／ハードウェアアーキテクチャに興味を持ち，コンパイラ，オペレーティングシステムなどシステムプログラムの研究・開発に従事する．

高橋　延匡（正会員）

昭和 8 年生．昭和 32 年早稲田大学第一理工学部数学卒業．同年(株)日立製作所中央研究所入社．HITAC 5020 モニタ，TSS の開発に従事．昭和 52 年より東京農工大学工学部数理情報教授．平成元年電子情報教授．オペレーティングシステム，日本語情報処理，パターン認識の研究に従事．電子情報通信学会，ソフトウェア科学会，計量国語学会，ACM 各会員．理学博士．