プロセス・ネットワークとして実現したUNIXカーネルの並列動作によるシステム・コール・レスポンス時間短縮の試み†

中山泰一†† 田胡和哉††* 森下巖††

マルチプロセッサ・システムを有効に利用するためには，利用者プログラムを並列化するのはもちろんのこと，OSが提供するシステム機能自体をもできる限り並列に処理できる構成とすることが要求される．システム内部で並列処理を行う並列型OSを，プロセス・ネットワーク方式を用いて設計し，疎結合型マルチプロセッサ・システム上で実現した．実現したシステムは広く実用されているUNIXと互換性をもつ．並列処理によって処理性能の改善を図るためには，OSをできるだけ多数の並列実行単位に分割すること，および，並列処理の実現コストを軽減することが必要である．本研究の採用したプロセス・ネットワーク方式では，相互排除アクセスされる資源の各々に軽量なプロセスを配置し，それらを同期式の通信で結合することによりシステムを実現した．プロセス・ネットワークのプロセッサへの分散配置を適切に行えば，システムの並列度を上げることが可能である．本論文では，第一ステップとして，システム・コールのレスポンスを向上させるための並列性の抽出について研究した．プロセス・ネットワークを各プロセッサに分散配置するための指針を考察し，この指針による分散配置が適切なものであることを実験により検証した．2台のプロセッサへの分散配置により利用者プログラムの処理時間が30%程度短縮されることが確認された．

1. はじめに

高性能のマイクロプロセッサが開発されている．これを用いてマルチプロセッサ・システムを実現することにより，高性能でかつ低価格な計算機システムが実現できるようになりつつある．

マルチプロセッサ・システムを有効に利用するためには，利用者プログラムを並列化するのはもちろんのこと，OSについてもマルチプロセッサ・システムに適合した構成とすることが要求される．

近年になり，並列／分散環境をサポートするOSの研究が盛んに行われている[1]．たとえばV-system[2]は，V-kernelとよばれる分散核による分散型OSである．分散核はプロセス間通信，記憶管理などを実現し，各プロセッサに同一のものが配置される．ファイル管理，プログラムの実行管理などは，核外の利用者プロセスであるソフトウェア・サーバによって実現されている．一般に階層的な依存関係を持つサーバによるシステム構成を，サーバ・クライアント・モデルとよぶ．

Mach[3]では，タスクとスレッドという概念を導入し，密結合型マルチプロセッサ・システムをうまく取り扱うことに成功した．プロセスをタスクとスレッドに分解し，複数のスレッドを1つのタスク内で走らせることにより，複数のプロセッサが同一のプログラムを効率よく実行することを可能としている．

本論文ではOSのシステム機能自体をマルチプロセッサ・システムの各ユニットに分散配置し，並列化することをめざす．これによりシステム機能がボトルネックとなることを防ぐ．具体的には，OSのシステム機能を機能ごとに多数の軽量なプロセスに分割し，それらをランデブ通信により結合する．すなわちプロセス・ネットワークを用いる方法によりシステムを実現する．本方式をプロセス・ネットワーク方式という．

プロセス・ネットワークを用いるOSの構成法には，

（1） 設計，保守が容易である，
（2） 異機種への移植が容易である，
（3） 疎結合型マルチプロセッサ・システムに自然な形で適用できる，

などの利点のほか，

（4） OSに内在する並列性を自然な形で抽出できる，

という利点がある．すなわち，OSは一群のシステム・プロセスのネットワークとして構成されているので，これらのシステム・プロセスをマルチプロセッサ・システムの各ユニットに適切に配置することにより，

（1） OSのシステム機能自体の並列実行，
（2） OSのシステム機能と利用者プログラムの並列実行，

† An Investigation for the Improvement of System Call Responese Times by Parallel Execution of a UNIX Kernel Implemented as a Process Network by YASUICHI NAKAYAMA, KAZUYA TAGO and IWAO MORISHITA (Department of Mathematical Engineering and Information Physics, Faculty of Engineering, University of Tokyo).

†† 東京大学工学部計数工学科
* 現在 日本IBM東京基礎研究所

が実現できる可能性がある．これにより，システム・コールの応答時間が短縮され，利用者プログラムの実行時間が短縮される可能性がある．

本論文では，まず，プロセス・ネットワークによるOS内部の並列性の抽出について考察する．ここでは，単一のシステム・コールのレスポンスを向上させることを目的として，プロセス・ネットワークの動作を考える．まず，プロセス・ネットワークの構造の位相的な情報，すなわちプログラム時にわかるプロセス間の通信参照関係だけから，マルチプロセッサ・システムの各ユニットへの配置法の指針を求める．そして，試作システム上において種々の配置について実験を行い，どの程度の時間短縮が可能であるかを明らかにする．

インプリメントに使用したマルチプロセッサ・システムは，少数のコンピュータ・ユニットが共有メモリで結合された方式のもので，ユニット間のメッセージ通信にはこの共有メモリを使用した．また，インプリメントに使用した OS は，田胡，益田[4]が報告した，プロセス・ネットワーク方式による UNIX である．

実験の結果，OS 内部での並列処理により処理性能が向上することが確認された．並列を意識せず設計されたOSをそのままプロセス・ネットワークによりインプリメントし，分散配置してやることにより性能の向上が可能であることが明らかになった．

2. プロセス・ネットワーク方式による OS の並列化

2.1 プロセス・ネットワーク方式

並列処理によって処理性能の改善を図るためには，プログラムをできるだけ多数の並列実行単位に分割すること，および，並列処理の実現コストを軽減することが必要である．プロセス・ネットワークは，この目的に適合する性能をもつ．

プロセス・ネットワークを用いたOSの構成を図1に示す．プロセス・ネットワーク方式では，次にあげる方針でOSを設計する．

（1） 相互排除アクセスされる計算機資源の管理機能を単位としてモジュール分割を行う．

（2） 軽量なプロセスによりモジュールを実現する．これらのプロセスを，システム・プロセスとよぶ．システム・プロセスの配置は固定している．

（3） ランデブ通信により，システム・プロセスを結合する．システム・コール，および周辺機器からの割込みも通信に変換する．

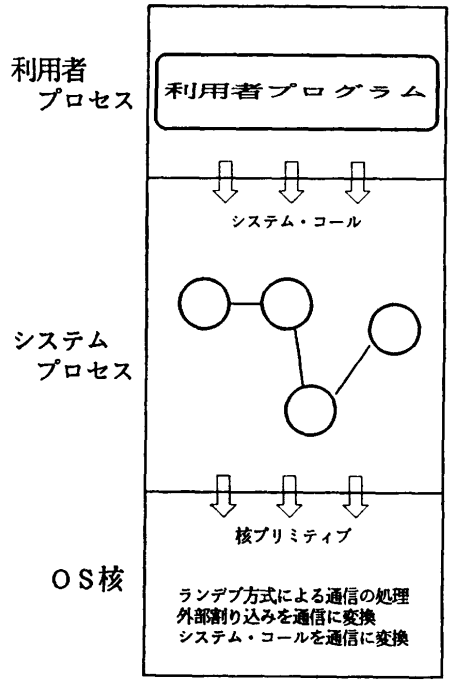


図 1 プロセス・ネットワーク方式による OS
Fig. 1 Basic structure of the operating system implemented as a number of light-weight processes interconnected by rendezvous communication.

システム・プロセス間の通信は，OS 核が提供する核プリミティブ (call, acc, endr) により実現される．これらの核プリミティブについては次章で詳しく述べる．

プロセス・ネットワーク方式により UNIX システムを再構成した．システム・プロセスを配置する資源管理機能の分割単位は，UNIXにおける資源アクセスの相互排除単位を踏襲し，UNIX システムの資源管理アルゴリズムをそのまま適用した．これまでに図2に示すような構造による OS が設計されている[4]．

2.2 OS の並列化

プロセス・ネットワークを用いて実現された OS を，複数のプロセッサ上に分散して配置することにより，OS 自体の並列化を試みる．密結合マルチプロセッサ・システムにおいて，低多重度時に利用者プログラムの処理時間を短縮すること，および，疎結合マルチプロセッサ・システム向き OS を実現することがその目的である．

システム・コールの処理時間が短縮されることによ

り，あらゆる既存の利用者プログラムの処理時間を短縮することができる．このため，OS 内部の並列度が低いものであっても，有用であることが期待できる．

本論文では以下の方針で OS の並列化を考える．

（1） 並列を意識していない従来の UNIX をそのままプロセス・ネットワークによりインプリメントする．このプロセス・ネットワークを分散配置させることにより並列化を試みる．

（2） システム動作時に，動的なプロセス生成や移動を行わない．プロセス・ネットワークの通信参照関係は定まっている．

（3） プログラム時にわかる情報のみから並列化について検討する．すなわち，ソース・コードを解析することによりプロセス・ネットワークの動作を考え，最適な配置の候補を得る．

（4） 実験により，候補の中から最適なものを選択する．

3. プロセス・ネットワークのプログラム解析による並列性の抽出

3.1 プロセス・ネットワークの動作

プロセス・ネットワーク方式による OS では図1に示したように，OS 核が通信によって結合されたプロセス集合体として OS を実現するための環境を実現する．その機能の1つであるランデブ通信は，**図3**（a）に示す，call，acc，endr の核プリミティブにより実現される．これらの核プリミティブは，通信に関する同期のみを実現し，データ転送は手続き呼出しと同一の機構により実現される．

call を，相手プロセス名，エントリ名，および，通信データを引数として呼び出すことにより，ランデブ呼出しが実現される．また，エントリ名を引数として acc を呼び出すことにより，ランデブの受付けが実現される．acc は，ランデブが成立すると，相手プロセスの call の引数列へのポイントを値として返すことにより，通信データの番地を知ることができる．データの返送，大量のデータ転送は番地呼出しによって実現する．ランデブは endr により終了する．

各システム・プロセスには，**図4**のように必ず通信のプリミティブが埋め込まれている．システム・プロ

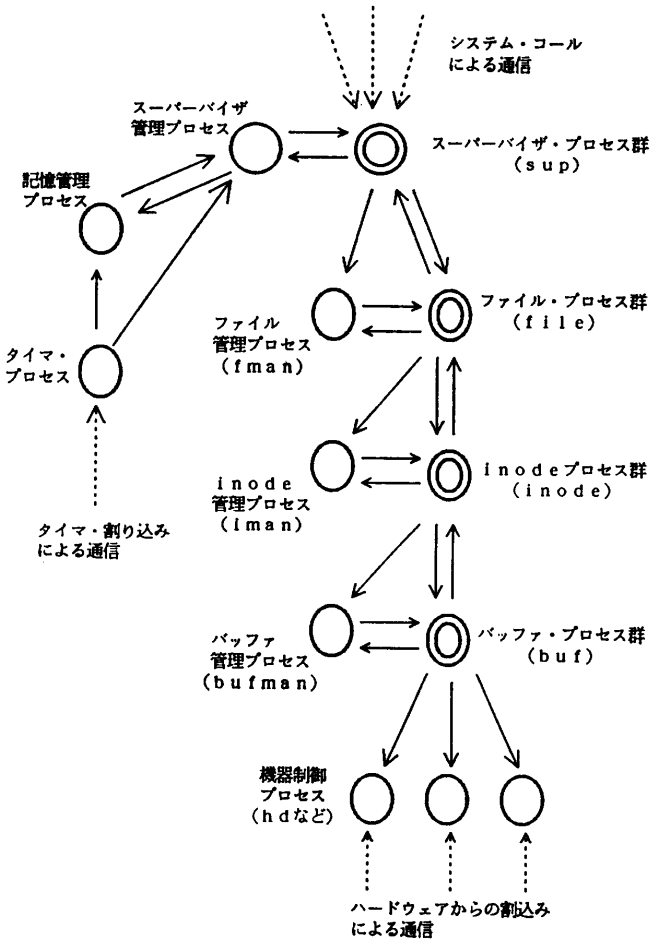


図2 プロセス・ネットワーク方式による UNIX の再構成
Fig. 2 Schematic diagram of the process network designed for the implementation of a UNIX system.

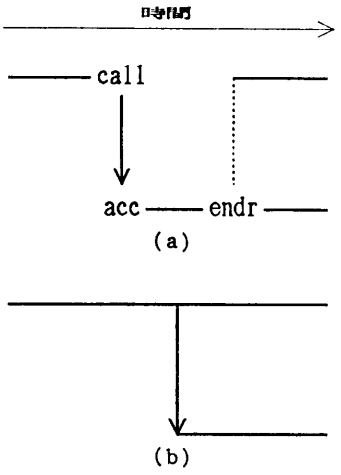


図3 （a）通信の核プリミティブ，（b）（a）の略記
Fig. 3 Kernel primitives employed for the rendezvous communication.

セスはそれぞれが無限ループの構造をしている．プロセス処理の最後まで行き着くとプロセスの最初に戻り，次の要求に対する処理を始める．再び他のシステム・プロセスとの通信を行うことを繰り返す．

プロセス・ネットワークの動作はシステム・プロセスの通信参照関係を追うことによりある程度まで推定することができる．これは，プロセス間の通信がランデブ方式なので，その実行順序に半順序関係があるためである．

図3(a)において，endr のあと call したシステム・プロセスと call されたシステム・プロセスの両者が並列に動作し，分岐が発生する可能性がある．簡単化のため，以下 call, acc, endr の組を図3(b)のように1本の矢印で表す．endr のあとシステム・プロセスの両者ともがシステム処理を行うか，あるいは通信を送った側は処理を終えて受け取った側だけがシステム処理を続けるかのどちらかで，**図5**に示す2通りの状態のどちらか一方となる．

並列を意識していない従来の UNIX をそのままプロセス・ネットワークによりインプリメントした場合，同期的なシステム・コールの処理は図5に示す2通りの状態がプロセス・ネットワークの動作の基本であり，全体としてはこれらの組合せとなる．この結果，分岐のあと処理が合流することは起こらず，プロセス・ネットワークの動作は木構造となる．

システム・コールの受付けから実際に利用者プロセスに結果を返すまでの処理の系列を主系列，それ以外の枝の系列を副系列とよぶことにする．プロセス・ネットワークの動作が木構造をしているためにこれら2種類の系列がはっきりと区別される．

3.2 プロセス・ネットワークのプログラム解析による並列化

1つのシステム・コールの処理時間，すなわちレスポンスを早くすることをめざすならば，プロセス・ネットワークを複数のプロセッサ上に分散して配置するときに副系列ができるだけ主系列と別のプロセッサで処理されることが好ましい．そのような場合にシステム内部の並列度がより抽出されるものと考えられる．システム・プロセス間の通信参照関係から，並列化についての検討を行うことができる．今後，システム・プロセス間の通信参照関係のことを，プロセス・ネットワークの位相の情報とよぶ．

システム処理の大半はファイル入出力によって費やされていることが知られている[5]．その中でも read システム・コールの頻度がもっとも高い．そこで，本論文では read システム・コールのレスポンスを向上させるためのシステム・プロセス配置法について，プロセッサ・ネットワークの位相の情報から考察する．同様の考察は，write システム・コールについて行うことが可能である．

read システム・コールにおける処理を見ると，プロセス・ネットワークの動作は**図6**のようになる．問題を簡単にするため，分割法としてア～エのラインによる2分割を考え，これを2台のプロセッサに配置する

```
process()
{
        for(;;){            /* for文による無限ループ */
                ...........
                call(put,c); /* ランデブ通信の呼出し  */
        }
}

process()
{
        for(;;){            /* for文による無限ループ */
                b = acc(get); /* ランデブ通信の受付け  */
                ...........
                endr();      /* 通信の終了 */
        }
}
```

図4 システム・プロセスの内部記述の例
Fig. 4 Examples of the internal structure of processes.

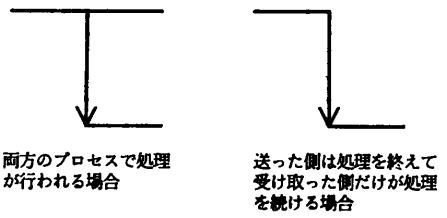


図5 通信が行われた時のプロセス・ネットワークの動作
Fig. 5 Behavior of a calling process and a called process when a rendezvous communication is executed.

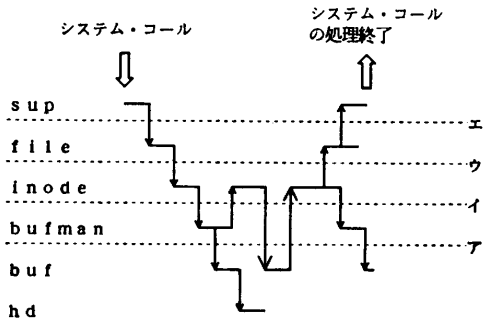


図6 read システムコールにおけるプロセス・ネットワークの動作
Fig. 6 Behavior of the process network when a read system call is invoked.

ものとする．

前節で主系列，副系列について定義したが，具体的に read システム・コールにおける主系列は，システム・コールを受け付け，データをファイルから読み込み，データをシステム・コールの引数により指定されたユーザ領域に格納し，読み込みバイト数を返り値としてユーザにわたす処理を行う系列である．これに対し，現在読み込んでいるブロックの次のブロックに対する先読み要求を出す系列や，次のシステム・コールの受付けの準備をする系列などが副系列である．

本節の初めに述べた，副系列をできるだけ主系列と別のプロセッサで処理することを考える．同期的なシステム・コールでは，副系列は非同期に動作可能なのに対し，主系列は終了するまで利用者プロセスの実行が待たされるので，できるだけ主系列には通信による遅延を入れたくない．そこで，プロセッサ分割により副系列を他のプロセッサに切り分けることを考える．

このとき，プロセッサ分割の指針として，

（1） 分岐の起きた通信の場所でプロセッサ分割する，

ことが導かれる．このことは各システム・プロセスの処理時間の情報がなくても，プロセス・ネットワークの構造の位相の情報，すなわちプログラム時にわかる通信参照関係だけからいうことができる．この指針によると，図6におけるイの分割は適切ではないことになる．

さらに，もう1つの指針として，

（2） （1）のように求められた分割点のうちでは，分割された主系列と副系列とが同一のシステム・プロセスへの通信を行わないものを選ぶ，

ことがある．図6において初めの分岐点の直後の主系列と副系列とは両方とも，同一のシステム・プロセス (buf) への通信を行っている．このことからアの分割は適切ではない．

これらの議論の結果，プロセス・ネットワークの位相の情報から推定できる最適配置の候補は，ウまたはエである．これは通信オーバヘッドや，プロセスの処理時間に依存しない．両者のいずれがより適切であるかは，実験によって確認する必要がある．

4. 試作システムにおける実験

4.1 試作システムの実現

プロセス・ネットワーク方式を用いて，システム機能自体を並列化した並列型 OS を沖電気製の ITC システム上に実現した[7]．ITC システムは図7に示すように，68020 マイクロプロセッサ，68851 MMU，4 MB の局所メモリなどから構成されるコンピュータ・ユニットを，共有バスを用いて最大7台まで実装することができる．これに 2 MB の共有メモリが装備されている．

ユニット間の通信用として，各ユニットにパイプ・レジスタが用意されている．送信先のユニット用のパイプ・レジスタにデータを書き込むことにより，送信先のプロセッサに割込みをかけることが可能である．この機能と共有メモリとを使って高速度のユニット間メッセージ通信機構を実現した[6]．共有メモリは通信路としてのみの利用であり，全体として ITC システムを疎結合型のマルチプロセッサ・システムのモデルとして用いた．

OS を構成するプロセス・ネットワークを複数のプロセッサ上に分散して配置することによりシステムを実現した．これは，標準の UNIX システムのアプリケーション・プログラムをそのまま実行することができる．

実現効率の点から，固定ディスクのかわりに RAM ディスクを用意した．システムの構成は，例として，図8のようになる．システム・プロセスのプロセッサへの配置は，ディスク制御プロセスなどデバイス・ドライバを除いて自由である．

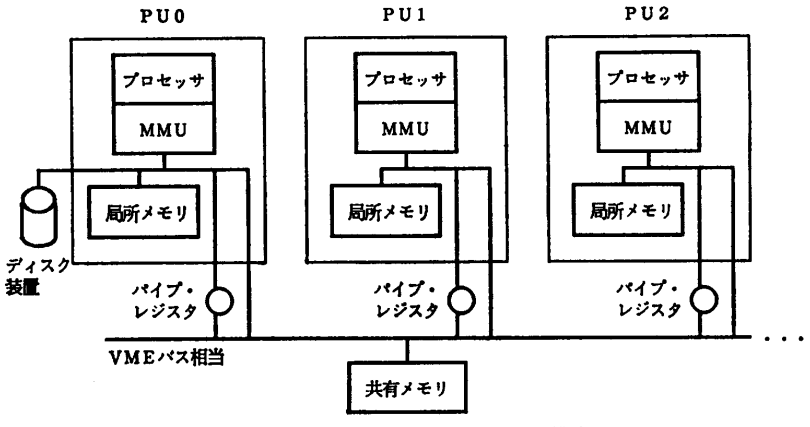


図7 ITC システムのハードウェア構成
Fig. 7 Hardware configuration of the ITC system.

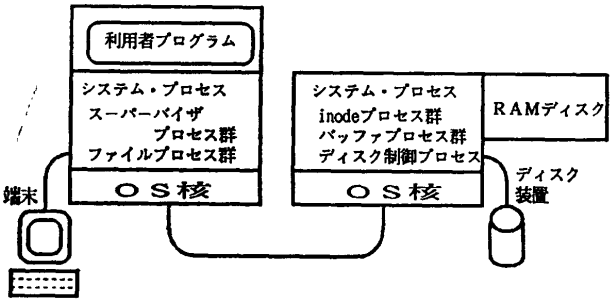


図 8 実現したシステムの構成の例
Fig. 8 An implementation of the operating system for the case of two processors.

4.2 実　　験

プロセス・ネットワークの2台のプロセッサへの分散配置の方法を変えながら，アプリケーション・プログラムを実行して，実行時間を測定した．これには，1つの利用者プログラムの開始から終了までに実際にかかる応答時間を測定した．

ここでは，システム処理の大半はファイル入出力によって費やされているので，ファイル・システムに関するシステム・プロセスの配置を変え，それ以外のシステム・プロセスは固定して実験を行った．OS 内部での並列処理の検討が目的であるので，利用者プロセスは1つのプロセッサ上にのみ配置した．

図9に示す7種類のシステムを実現して実験を行った．それぞれの構成をA～G構成と名づける．図9における破線は各構成におけるプロセッサ分割を示す．A構成が単一プロセッサ構成であり，B～G構成はシステム機能を2台のプロセッサに分散配置した構成である．

図9において hd 0 と hd 1 とは，ディスク制御を行うデバイス・ドライバのプロセスである．buf, inode, file の各プロセス群はそれぞれ，ファイル・システムを実現するデータ構造であるバッファ，iノード，ファイルの管理を行っている．bufman, iman, fman の各プロセスはそれぞれ buf, inode, file の割り当てを決める．以上のシステム・プロセスによりファイル・システムが構成されている．なお，前章で考えた read システム・コールの処理では iman, fman における処理を含まないために図6のア～エの4種類の分割を考えたが，実験では iman, fman の配置も変えて7種類のシステムを考えた．

利用者プログラムとしては，5種類のものを取り上げ，各利用者プログラムの実行が完了するまでの時間を測定した．

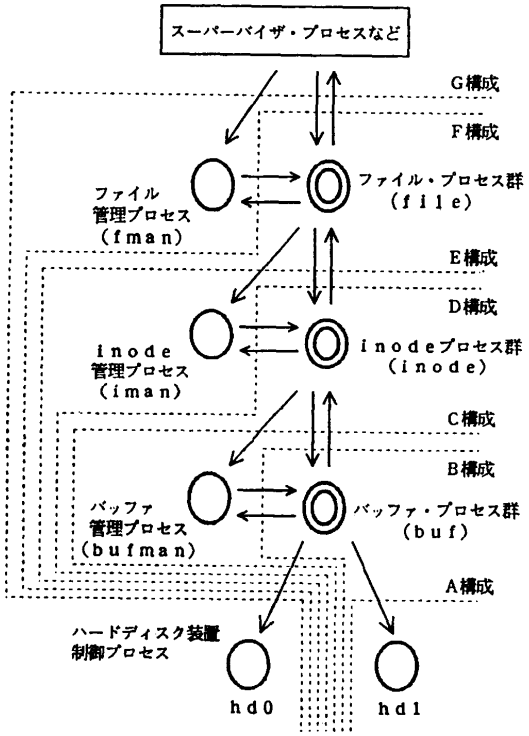


図 9 実験に用いた7種類のシステム構成（破線はそれぞれの構成でのプロセッサ分割を示す）
Fig. 9 7 types of process allocation used in the experiment.

4.3 実験結果と考察

測定結果を**表1**に示す．これによると，システム機能を2台のプロセッサに分散配置した B, D, E, F, G構成のもので単一プロセッサ構成のA構成のものに比べて性能が向上した．利用者プロセスの処理時間は一定であるので，システム・コールの処理時間が改善されていると言える．

最も性能が向上したのはE構成で，単一プロセッサ構成のA構成に比べて 30% 前後処理時間が短縮した．このE構成について，read システム・コールの処理を考えると，図6におけるウの位置でプロセッサが分割されるようにプロセス・ネットワークを配置したものに対応する．

逆にC構成（図6においてイの位置でプロセッサを分割したものに相当）では，単一プロセッサ構成のA構成に比べて処理性能が悪化した．

OS 内部の並列処理により，利用者プログラムの実行時間が短縮できることが確かめられた．また，この並列処理の効率は，システム・プログラムのプロセッサへの配置法によって大きく変わることも判明した．

システム・プロセス間で通信が行われる順序の情報

表 1 実行結果 (単位は sec)
Table 1 Experimental results.

		A	B	C	D	E	F	G
1つのファイルを読み込む (250 Kbytes)	i) hd0 上のとき	3.2	3.0	3.6	2.4	2.3	2.4	2.4
	ii) hd1 上のとき	3.4	2.6	3.4	2.1	2.0	2.5	2.5
複数のファイルを読み込む (7 files, 260 Kbytes)	i) hd0 上のとき	3.6	3.4	3.9	2.8	2.6	2.8	2.8
	ii) hd1 上のとき	3.6	3.0	3.6	2.3	2.2	2.7	2.7
(29 files, 450 Kbytes)	i) hd0 上のとき	6.5	6.0	7.2	5.1	4.9	5.2	5.2
	ii) hd1 上のとき	6.7	6.2	6.6	4.4	4.1	5.1	5.1
hd0 から hd1 へのコピー (260 Kbytes)		7.8	6.6	8.4	5.6	5.2	6.1	6.1
Cプログラムのコンパイル	プログラム a	13.9	12.8	15.1	11.6	11.1	12.4	12.5
	プログラム b	14.8	13.8	16.1	12.3	11.8	13.3	13.4

から，最適となる可能性のある配置の候補を限定した．プログラム記述から静的に抽出できる情報のみを用い，システムを動作させることなく，どの程度まで最適化が可能であるかを検討した．この情報からも多数の配置から最適配置の候補をしぼり込むことができることが，確かめられた．しぼり込まれた候補について実動テストを行うことにより，全体として，比較的容易に最適な配置を得ることができる．

5. おわりに

システム内部で並列処理を行う並列型 OS を，通信により結合された軽量なプロセス集合体であるプロセス・ネットワークを用いて設計し，疎結合型マルチプロセッサ・システム上で実現した．

プロセス・ネットワークのプロセッサへの分散配置を適切に行えば，システムの並列度を上げることが可能である．本論文では，システム・コールのレスポンスを向上させるための並列性の抽出について研究した．プロセス・ネットワークの構造の位相の情報から，プロセス・ネットワークの各プロセッサへの分散配置法の指針を求めた．そして，この指針による分散配置が適切なものであることを実験により検証した．

プロセス・ネットワーク方式を採用した場合，原理的には，各プロセスが並列に動作することが可能であり，単一システム・コールの実行時間の短縮だけでなく，多数のシステム・コールの並列実行の可能性が残されている．今後の検討課題である．

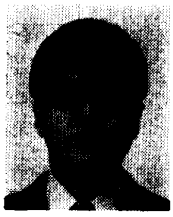
謝辞 本論文をまとめるにあたり，貴重なご助言をいただいた，東京大学工学部計数工学科出口光一郎助教授と，永松礼夫助手に深く感謝いたします．

また，今回の実験に用いた ITC システムを提供いただいた沖電気工業株式会社システム開発センターに感謝します．

参考文献

1) 清水謙太郎：分散オペレーティング・システムの研究調査，情報処理学会オペレーティング・システム研究会報告，89-OS-45-1 (1989).
2) Cherinton, D. R.: The V Distributed System, *Comm. ACM*, Vol. 31, No. 3, pp. 314-333 (1988).
3) Rashid, R. F.: Threads of a New System, *UNIX Review*, Vol. 4, No. 8, pp. 37-49 (1986).
4) 田胡和哉，益田隆司：OS の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 (1984).
5) 堀川 隆：ハイブリッド・モニタ手法を用いたシステム動作の測定・解析，情報処理学会オペレーティング・システム研究会報告，91-OS-50-10 (1991).
6) 田胡和哉，中山泰一：軽量なプロセスの集合によるマルチプロセッサ用OSの設計，第38回情報処理学会全国大会論文集，5N-1 (1989).
7) 中山泰一，田胡和哉，森下 巖：並列型OSの設計と実現，情報処理学会オペレーティング・システム研究会報告，90-OS-47-1 (1990).
8) 中山泰一，田胡和哉，森下 巖：プロセス・ネットワークによる OS 内部の並列実行，並列処理シンポジウム JSPP 91 論文集，pp. 317-324 (1991).

(平成3年 8 月2日受付)
(平成3年12月9日採録)



中山 泰一（正会員）

1965 年生．1988 年東京大学工学部計数工学科卒業．1990 年同大学大学院工学系研究科情報工学専攻修士課程修了．現在同博士課程に在学中．オペレーティング・システム，並列・分散処理などに興味をもつ．日本ソフトウェア科学会会員．

田胡 和哉（正会員）

昭和 31 年生．昭和 56 年筑波大学第 3 学群情報学類卒業．昭和 61 年同大学大学院工学研究科博士課程修了．工学博士．同年同大学電子・情報工学系助手．昭和 63 年東京大学工学部計数工学科助手．平成 2 年より日本 IBM 東京基礎研究所勤務．オペレーティング・システムの設計方式に興味を持つ．昭和 60 年本学会論文賞受賞．計測自動制御学会，ソフトウェア科学会各会員．

森下 巖（正会員）

1934 年生．1957 年東京大学工学部応用物理学科計測工学コース卒業．同年東レ(株)入社．計装制御システムの設計に従事．1966 年東京大学工学部計数工学科助教授，1980 年同教授，現在に至る．工学博士．1973 年 SRI 人工知能研究センタ滞在．パターン認識，信号処理，画像処理，マルチプロセッサシステムなどの研究に従事．著書「マイクロコンピュータのハードウェア」（岩波書店），「マイクロコンピュータの基礎」（昭晃堂），「信号処理」（計測自動制御学会）など．IEEE，計測自動制御学会，電子情報通信学会，電気学会などの会員．