プロセス・ネットワークによる分散型オペレーティング・システムの性能に関する考察†

高野陽介†† 田胡和哉†††* 益田隆司††††

分散透明な利用環境を実現する分散型オペレーティング・システムの研究開発が盛んである．著者らはこれまでに，システムをプロセス間通信で結合されたプロセスの集合により実現するプロセス・ネットワーク方式を用いて分散型オペレーティング・システムを実現する方式を提案し，この方法に基づいて試作システムを構築した．現在，本方式を用いて，OS の機能分散や利用者プログラムの並列処理を行う実用的な分散処理システムの実現を進めている．この目的のためには，OS の内部動作に関する知見が必要である．そこで，試作システムを用いて性能測定や内部動作の解析を行い，動作特性の解明を試みた．その結果，OS を構成するプロセスのレベルの性能に関して，プロセス間通信が OS の負荷の主な原因になっていること，OS 内部の並列度が高く，プロセッサ間で OS が並列動作していること，同期型のプロセス間通信方式を用いても OS の性能に与える影響は小さいことなどの性質が明らかになった．また，利用者プログラムのレベルでは，利用者プログラムの負荷分散によりスループットが向上すること，およびパイプによる利用者プロセス間通信を用いて別々のプロセッサ上の利用者プログラムを結合し並列処理の効果が得られることを確認した．これらの結果から，プロセス間通信の高速化を図ることにより，効率的な OS の機能分散，利用者プログラムの並列処理が実現できると考えられる．

1. まえがき

マイクロプロセッサを活用した汎用の計算機システムとして，個人用計算機を複数台結合したネットワーク・システムやマルチ・マイクロプロセッサ・システムの利用が普及しつつある．これに伴い，分散透明な利用環境などの良好な使い勝手を実現する分散型オペレーティング・システム（以下，分散型 OS と略す）の研究開発が盛んである[2),6)]．

分散型 OS の果たす役割として，多数のプロセッサの処理能力を有効利用することにより良好な性能を実現すること，および周辺機器を有効利用するためにデータの効率的な共有を実現することがあげられる．この目的のためには，分散型 OS の設計において効率的な機能分散の実現を可能にする設計方式を導入する必要がある．

最近では，分散核（distributed kernel）を利用して分散型 OS を設計する例が増加している．分散核による方法では，分散透明なプロセス間通信，記憶管理等の基本的な機能を分散核によって実現し，それ以外の OS の機能や利用者プログラムは分散核の外部のプロセスとして実現する．このため，OS の機能や利用者プログラムの分散を柔軟に行うことができる．分散核を利用している例としては，スタンフォード大学のVシステム[1)]，カーネギー・メロン大学の Mach システム[3)]があげられる．また，Vシステムなどいくつかの分散型 OS では，利用者プロセスをプロセッサ間で移動（migrate）して負荷の分散を図る方法が試みられている[7)]．

われわれは，分散透明なプロセス間通信で結合されたプロセスの集合によって OS を実現するプロセス・ネットワーク方式[5)]により分散型 OS を設計する方式を提案し，この方法に基づいて試作システムを実現した[4)]．プロセス・ネットワーク方式では，利用者プロセスと別個のプロセスにより OS を実行する．そこで，OS を実行するプロセスをシステム・プロセスとよぶことにする．

本論文では，プロセス間通信という用語により OS 内部のシステム・プロセス間の通信を指し，利用者プロセス間の通信に対しては利用者プロセス間通信という用語を用いることにする．また，プロセス間通信には，同一プロセッサ上のシステム・プロセスの間で行われるプロセス間通信と，別々のプロセッサ上のシステム・プロセスの間で行われるプロセス間通信の２通りがある．そこで，前者をプロセッサ内のプロセス間通信，後者をプロセッサ間のプロセス間通信とよぶこ

† Performance of a Distributed Operating System by Process Network Method by YOUSUKE TAKANO (Doctorial Program in Engineering, University of Tsukuba), KAZUYA TAGO (Institute of Information Sciences and Electronics, University of Tsukuba) and MASUDA TAKASHI (Department of Information Science, Faculty of Science, University of Tokyo).

†† 筑波大学工学研究科
††† 筑波大学電子・情報工学系
†††† 東京大学理学部情報科学科
* 現在 東京大学工学部計数工学科

とにする.

プロセス・ネットワーク方式では，OS を静的に配置された軽量な (light-weight) システム・プロセスにより実現する．このため，OS 内部の並列度を高くできること，システム・プロセスの実現コストが小さいこと，およびシステム・プロセスの名前管理のオーバヘッドがないことを特徴とする．これは，多数のプロセッサを含むシステムの制御に適し，OS 機能の分散の実現に有効に利用できると考えられる．

現在，プロセス・ネットワーク方式による分散型 OS を備える実用的な分散処理システム Agora-1 の実現を進めている．Agora-1 は，68010 マイクロプロセッサを組み込んだ複数台の計算機を疎結合，密結合構成により結合した分散型のエンジニアリング・ワークステーションで，数人の利用者により使用することができる．

Agora-1 では，複数のプロセッサの処理能力を効率良く利用する目的から，プロセッサ間を密に結合して，LAN を用いた分散処理システムよりも細かい単位での処理の分散を試みる．分散型 OS の内部では，小規模なシステム・プロセスを単位とした OS の機能の分散を図るとともに，利用者プログラムのレベルでは，多重プログラミングの負荷の分散，および利用者プログラム内の並列処理を複数のプロセッサを用いて実現する．

プロセッサ間を密に結合するために，LAN の下位ネットワークとして，数台のプロセッサを応答性の良い通信方式で結合したネットワークを構築し，その上にシステムを構築する．さらに，プロセッサ内，プロセッサ間のプロセス間通信機構をハードウェアによって実現する．

このような形態の分散処理を実現するためには，分散型 OS の性能に関して解析を行い，その知見をもとに適切なシステム設計を行う必要がある．そこで，試作システムを用いて性能測定，内部動作の解析，および利用者プログラムの分散処理の実験を行った．本論文では，これらの実験方法と結果について述べ，プロセス・ネットワーク方式による分散型 OS の性能に関する性質について考察する．

2. 分散型 OS の性能に影響を与える要因

分散型 OS の性能を左右する要因のうち，分散型 OS の設計方式に依存する要因としては，OS を構成する並列動作可能な処理単位の規模と処理単位の間の通信に要する時間の2つが重要である．処理単位は，プロセス，オブジェクト等を指す．OS が小規模で多数の処理単位により構成されている場合，プロセッサ間の OS の処理の並列性，およびディスク入出力動作等と OS の処理との並列性が高くなる．たとえば，処理単位が他のプロセッサ上の処理の終了や入出力動作の終了を待って実行を中断している場合，それ以外の処理単位は実行可能状態にあるので別の処理を実行することができる．このように，処理単位の規模を小さく設定することにより，プロセッサの処理能力が有効に利用され良好なスループットを得ることができる．

一方，処理単位の規模を小さくすることにより，処理単位の間の通信の頻度が増加する．したがって，通信に要する時間が大きい場合，処理単位を小さくしても，逆に負荷が増加する可能性がある．規模の小さい処理単位の集合体により分散型 OS を構築する場合には，実際に得られる並列処理の効果と通信に要する時間の間のバランスを十分考慮する必要がある．

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，たとえば，バッファの1つ1つ，オープンされたファイルの1つ1つを別々のシステム・プロセスによって管理する．したがって，個々のシステム・プロセスの規模が小さく，OS 全体が多数のシステム・プロセスによって構成される．そこで，プロセス・ネットワーク方式を用いて適切なシステム設計を行うための知見として，プロセス間通信の性能が OS の性能に与える影響，および OS 内部で実際に得られる並列処理の効果について調べることにした．

並列処理の効果に影響を与えるもう1つの要因として，プロセッサ間のプロセス間通信の方式による影響があげられる．プロセス・ネットワーク方式では，ランデヴによる同期型の通信方式を用いている．通信方式としてランデヴを用いることには，いくつかの得失が考えられる．

ランデヴを用いることによる利点の1つは，プログラムの検証，特にデッドロックの危険性の解析が容易になることである．たとえば，非同期型の通信方式を用いた場合，同一システム・プロセスからの通信が複数個同時に別のシステム・プロセスに伝達される可能性が発生する．これは，デッドロックの可能性を拡大し，その発見を難しくするだけでなく，システム・プロセスのプログラミングを複雑化する．

もう1つの利点は，通信機構を簡略化でき通信の高速化を促進できることである．ランデヴは，非同期型の通信方式に比べ通信引数を格納する領域にあふれが発生する可能性が小さい．さらに，どのようにプロセス間通信が混雑しても，すべての時点において通信引数を格納する領域の総量は一定の量を越えない．したがって，通信引数を格納する領域としてあらかじめこの量のメモリを用意しておくことにより，十分な領域が確保できないためにプロセス間通信が失敗することがなくなる．これにより，通信引数の再送等のプロトコルを削除でき，プロセッサ間の通信機構が簡略化できる．

一方，ランデヴは非同期方式を用いた場合に比べてプロセッサ間の並列度が得られにくい傾向がある．すなわち，ランデヴ呼出しを発行したシステム・プロセスは，ランデヴが終了するまで実行が中断されているので，通信相手のプロセッサと並列に動作することができない．

試作システムにおける実験では，プロセス間通信の応答時間の影響，OS 内部の並列処理による影響，通信方式による影響の3つの要因に注目して解析を行う．その結果に基づき，並列処理の効果とプロセス間通信の負荷のバランスについて考察を行う．

3. 試作システムの構成

図1に，実験に用いた試作システムの構成を示す．ハードウェアは，16ビット幅の通信回線と回線の使用権をトークン・パッシング方式により制御する機構を備えた専用の通信ハードウェアを用いて2台の PC-9801 パーソナル・コンピュータを結合したものである．通信ハードウェアの転送速度は，約4 Mbps である．一方の PC-9801 には，ハード・ディスク装置が接続されている．ハードディスク装置が装備されている方の PC-9801 を以後 pc98a，ディスクレスの方を pc98b とよぶことにする．利用者プログラムを pc98a 上で実行する場合は，ディスク装置をローカル・ディスクとして使用し，利用者プログラムを pc98b 上で実行する場合は，ディスク装置をリモート・ディスクとして使用する．

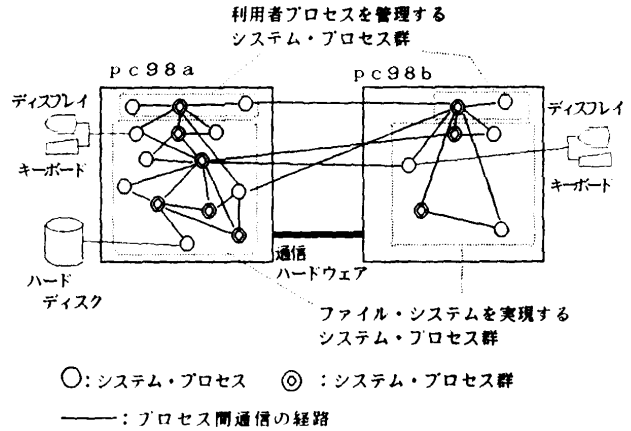

図1 試作システムの構成
Fig. 1 Configuration of the proto-type system.

試作システムの分散型 OS は，UNIXsystemⅢ* と互換性のあるシステム・コールのインタフェイスを提供し，PC-9801 用の UNIX システム (PC-UX) 上で使用しているユーティリティ・プログラムの大部分が実行可能である．試作システムは，2人の利用者が同時に使用することができる．また，一方の PC-9801 をバック・エンド・プロセッサとして用い，フロント・エンドのジョブと並行してジョブを実行することができる．OS は，利用者プロセスの分散実行を実現するため，利用者プロセスの遠隔生成 (remote fork) の機能を提供する．

分散型 OS は，プロセス間通信機能とプロセスの切換えを実現する分散 OS 核と，プロセス間通信で結合されたシステム・プロセスの集合から構成される．試作システムでは，分散 OS 核をソフトウェアによって実現している．

OS の機能分散，すなわちシステム・プロセスのプロセッサへの配置は次のように行った．まず，利用者プロセスの管理プロセス，および主記憶の管理プロセスを2つのプロセッサに重複して配置した．これらは，各プロセッサ上の利用者プロセスからの要求を独立に処理する．ファイル・システムを構成するシステム・プロセス群については，ファイル・ポインタの移動などの一部の機能を実現するプロセスを両方のプロセッサに重複して配置し，それ以外の inode 管理，バッファ管理，ハード・ディスク管理などの機能を実現するプロセス群は pc98a 上に集中して配置した．この分散型 OS は，pc98a 上に 88 個，pc98b 上に 60 個の合計 148 個のシステム・プロセスによって構成されている．

pc98a 上の利用者プロセスにより発行されたシ

* UNIX は AT&T のベル研究所で開発された OS の名称である．

ステム・コールは，pc98a 上のシステム・プロセスのみによって処理される．一方，pc98b 上の利用者プロセスにより発行されたシステム・コールは，2つのプロセッサを用いて並列処理される．その処理方法は次の3つに分類できる．

（1） システム・コールがディスク装置やバッファへのアクセスを含まない場合，たとえば，ファイル・ポインタの移動を行う lseek システム・コールやメモリ割付けを行う sbrk システム・コールなどは，pc98b 上のシステム・プロセスだけで処理を完了することができる．したがって，pc98a 上の利用者プロセスの場合とシステム・コールの応答時間は同じになる．

（2） 利用者プロセスがシステム・コールの処理の完了を待たずに実行を再開できる場合，たとえば，ファイルのクローズを行う close システム・コール，ファイル出力を行う write システム・コール，ファイルの抹消を行う unlink システム・コールの処理などでは，pc98b 上で一部の処理を行い，残りの処理を pc98a 上で実行する．この場合，システム・コールからの戻り値が確定した時点で pc98b 上の利用者プロセスを再開し，残りのシステム・コールの処理を pc98a で継続する．再開された利用者プロセスと OS の処理を並列に実行することができるので，1台のプロセッサでシステム・コールの処理を行った場合よりも応答時間が短縮される可能性がある．

（3） そのほかのシステム・コールでも OS の処理を pc98a，pc98b の間で並列に処理する．たとえば，プログラムの起動を行う execve システム・コールの処理では，pc98b 側で主記憶の割付け，主記憶の初期化，実行環境の設定を行い，これと並行して pc98a では実行プログラムの読み込みを行う．

4. OS の内部動作に関する考察

OS 内部の動作特性を調べるため，2章で述べた3つの点に注目して実験を行った．

試作システムにおける測定，解析は次のように行った．利用者プログラムの応答時間は，ストップウォッチで数回計測した測定値の平均を取ることにより求めた．測定は，試作システム上に測定対象以外の負荷が存在しない条件下で行った．また，測定ごとにハード・ディスクのデータを一定の内容に初期化した．これにより，利用者プログラムが生成するファイルのディスク・ブロックの構成を一定にし，参照に必要な時間を一定にすることができる．

システム・コールの処理の内訳などの OS 内部の動作解析に必要なデータの測定は，OS の実行プログラムをもとに機械語単位で所用時間を合計し，システム・プロセス，分散 OS 核ごとのタイムチャートを作成することにより求めた．

性能の比較対象として，PC-UX システムを用いた．試作システムの OS は，多くの部分を PC-UX システムをもとにして実現している．PC-UX システムの内部の手続きのうち，利用できるものはそのまま試作システムの OS に使用し，そのままではプロセス・ネットワーク方式には組み込めない部分でも，条件分岐や実行文など PC-UX システムのプログラムとできる限り同じになるようにプログラミングした．このため，試作システムは，PC-UX システムとシステム・コールのインタフェイスが共通であるだけでなく，バッファの個数，利用者プロセスの実現方式などのシステム内部のデータ構造や資源管理方式の点でも共通である．さらに，開発に用いたCコンパイラにより PC-UX システムの場合とほぼ同等の機械語プログラムが生成されることを確認している．以上の理由で，比較対象として PC-UX システムを用いることにより，システムの設計方式が性能に与える影響を調べることができる．

4.1 実験結果

（1） プロセス間通信の負荷

表1に，試作システム，PC-UX システムにおいて利用者プログラムの応答時間を測定した結果を示す．表1の②は，利用者プログラムを pc98a において実行し，システム・コールを1台のプロセッサを用いて処理した場合，③は利用者プログラムを pc98b にお

表 1 利用者プログラムの応答時間
Table 1 Response times of user programs.
(単位：sec)

OS / 利用者プログラム	PC-UX システム ①	試作システム	
		pc 98a で実行②	pc 98b で実行③
Cコンパイラ(1) (ソースコード30 bytes)	27.4	27.1	29.6
Cコンパイラ(2) (ソースコード1011 bytes)	44.7	42.7	45.8
Cコンパイラ(3) (ソースコード6876 bytes)	85.8	87.0	90.5
YACC (ソースコード1178 bytes)	21.4	21.1	21.9
文書清書(nroff) (ソースコード6612 bytes)	20.1	19.4	21.4

いて実行し，システム・コールを2台のプロセッサを用いて並列処理した場合の測定値である．1分以内の短いジョブでは，PC-UX システムよりも②が多少短い，あるいは同等である．これに対し，コンパイラ(3)のような長いジョブでは①<②<③の順である．

表1のコンパイラ(3)の実行では，18626 回のプロセス間通信の発生が測定された．表1の②の場合，18626 回すべてがプロセッサ内のプロセス間通信になる．プロセッサ内のプロセス間通信1回の処理にかかる時間は 158 μsec～219 μsec である．時間の幅は，プロセスの待合せ状態の違いによる．プロセッサ内，およびプロセッサ間のプロセス間通信の応答時間は，プロセス間通信のみを行う1対のシステム・プロセスを作成し，その間でプロセス間通信を数万回繰り返し実行するのに要した応答時間を測定して，その測定値を回数で割ることにより求めた．

一方，表1の③の場合には，18626 回のうちの 2625 回がプロセッサ間のプロセス間通信である．プロセッサ間のプロセス間通信1回の応答時間は 2.41 msec～3.96 msec である．応答時間の幅は，プロセスの待合せ状態とプロセッサ間で転送されるデータ量の違いによるものである．pc98b 上で利用者プログラムを実行した場合，プロセッサ間のプロセス間通信の発生回数は，この例と同様にプロセス間通信の総回数の 10% 前後であることが，多くの利用者プログラムの実行において測定されている．

OS の処理の一部はディスク入出力と並列に実行されるので，並列部分に含まれるプロセス間通信処理の時間は利用者プログラムの応答時間に影響を与えない．そこで，システム・コール個々の実行に注目し，その応答時間に影響を与えるプロセス間通信処理の時間を解析した．**表2**に，システム・コールの処理に含まれるプロセス間通信処理の時間とプロセッサ間の並列処理の時間の測定結果を示す．並列処理の時間については次節で述べる．

表2の⑤～⑦は，システム・コールの処理を1台のプロセッサを用いて実行した場合のシステム・コールの応答時間とそれに含まれるプロセッサ内のプロセス間通信の処理時間の合計を測定した結果である．⑤+⑥は，OS の実行に要した時間である．ディスク入力と並列に実行されているプロセス間通信処理の時間は⑥の時間に含まれている．したがって，⑦の時間が変更された場合，その変更分がそのまま⑥のシステム・コールの応答時間上に差引きとなって表れる．表2の⑤と⑦を比較すると，システム・コールの応答時間の 50% 前後をプロセス間通信が占めていることがわかる．

表2の①～④は，2台のプロセッサを用いてシステム・コールの処理を行った場合のシステム・コールの応答時間とそれに含まれるプロセッサ間のプロセス間

表2 システム・コールの処理に含まれる並列処理とプロセス間通信処理
Table 2 Time of parallel execution and inter-process communication in the execution of system calls.

(単位：msec)

システム・コール	システム・コールの処理を2つのプロセッサで行った場合				1つのプロセッサで行った場合		
	システム・コールの応答時間①	プロセッサ間のプロセス間通信の負荷②	ディスク入力と並列処理している時間③	プロセッサ間で並列処理している時間④	システム・コールの応答時間⑤	ディスク入力と並列動作している時間⑥	プロセッサ内のプロセス間通信の負荷⑦
read (512 バイト読み込み，バッファにデータがある場合)	10.1	6.4 (通信回数 2)	0	0.5	4.2	0	1.8 (通信回数 10)
read (512 バイト読み込み，バッファにデータがない場合)	8.2+d^*	5.4 (通信回数 2)	1.3	0.5	3.9+d^*	0.7	1.7 (通信回数 8)
write (512 バイト書き込み)	6.3	4.0 (通信回数 1)	0	2.0	4.3	0	1.8 (通信回数 10)
open (カレントディレクトリ上の既存ファイルをオープンした場合，バッファにデータがある場合)	25.4	20.9 (通信回数 10)	0	6.1	10.6	0	5.9 (通信回数 33)
close	5.8	4.8 (通信回数 2)	0	3.2	4.2	0	2.0 (通信回数 12)
exece (テキスト，データとも 512 バイト以内のプログラム実行，バッファにデータがない場合)	53.1 +Σd^*	32.9 (通信回数 13)	21.3	11.0	34.7 +Σd^*	17.8	14.2 (通信回数 90)

*d は，ディスク入力の応答時間

通信の応答時間の合計を測定した結果である．①+③+④は，2つの PC-9801 で OS の実行に要した時間の合計である．ディスク入力中に行われているプロセス間通信処理の時間は③に含まれている．表2の①と②を比較すると，システム・コールの応答時間の半分以上がプロセッサ間のプロセス間通信の負荷であることがわかる．1回のプロセッサ間のプロセス間通信処理の時間の内訳をみると，通信ハードウェアによる転送時間が全体の約3％であり，残りはデータの複写，記憶管理，プロセス間の待合せ処理等の通信プロトコルをソフトウェアが実行する時間である．

（2） システム・プロセスの並列処理

並列処理の効果を確認するため，システム・プロセスのプロセッサ配置を変更し，試作システムの OS に比べ，プロセッサ間のシステム・プロセスの並列度が小さい構成の OS を作成して性能を比較した．その比較を**表3**に示す．表3の①は，ハード・ディスクを管理するシステム・プロセスを1つだけ pc98a に配置し，残りのシステム・プロセスをすべて pc98b に配置した OS である．①，②とも利用者プログラムは pc98b 上で実行し，システム・コールの処理は2つのプロセッサを用いて実行する．システム・プロセスの配置が異なること，およびプロセス間通信にかかる総時間が異なることを除けば①と②の条件は同じである．

表3をみると，①は②に比べてプロセッサ間のプロセス間通信の回数が少ないのにもかかわらず，応答時間はより長くなっている．このことから，応答時間の差異は，②において処理がプロセッサ間で時間的に重複しているためであるとみなすことができる．

表3 システム・プロセスのプロセッサ配置の異なる2種類の OS の応答時間

Table 3 Response times on two operating systems which have the different system process allocations on multiple processors.

（単位：sec）

利用者プログラム ＼ OS	試作システムとはシステム・プロセスの配置の異なる OS における pc98b での実行①	試作システムの pc98b での実行②
Cコンパイラ(1)（ソースコード 6876 bytes）	92.2 [1326]	90.5 [3102]
Cコンパイラ(2)（ソースコード 96413 bytes）	858.9 [10496]	831.8 [21856]
パイプライン処理 (ls|sort)	4.7 [79]	3.8 [276]

[ ] の中はプロセッサにわたるプロセス間通信の回数

表1でも並列処理の効果に起因すると考えられる結果が得られている．たとえば，表1のコンパイル(3)の②と③の応答時間の差は3.5秒であるが，前節で述べたプロセッサ間のプロセス間通信の回数とその応答時間から単純に計算すると少なくとも6.3秒の差が生じていることになる．

実際には，表1のコンパイル(3)の③における2つのプロセッサ上の分散 OS 核の処理時間の合計は，コンパイル(3)の②における分散 OS 核の処理時間よりも 10.66 sec 大きいことが測定により判明した．この値は，プロセッサ間のプロセス間通信の応答時間 2.41～3.96 msec に通信回数 2625 回を乗じた値よりも大きい．これは，プロセッサ間のプロセス間通信処理の内部で並列処理が行われており，2つのプロセッサの処理時間の合計でみると並列処理の時間が加算されるためである．プロセッサ間のプロセス間通信1回の処理に2つのプロセッサが要する時間の合計は 2.97 msec～5.92 msec である．一方，コンパイル(3)の③の実行において並列処理が行われていた時間，すなわち pc98a と pc98b がともにアイドル状態でなかった時間は6.07秒であった．したがって，10.66－6.07＝4.59 が応答時間の増分になる．この値は，表1のコンパイル(3)の②と③の応答時間の相違にほぼ一致している．

表2の④は，システム・コールごとにその処理に含まれる並列処理の時間を測定した結果である．並列処理の大きさをみると，システム・コールの処理の終了を待たずに利用者プロセスを再開できる write や close では応答時間に占める並列処理の時間の比率が大きい．応答時間が長い open, exece でも内部の並列処理の効果が表れている．

（3） 同期通信方式を用いることによる得失

プロセッサ内のプロセス間通信の場合，システム・プロセス間でアドレス空間を共有しているので，同期方式を用いることにより，通信引数をポインタを用いて受け渡すことができる．非同期方式では必ず通信引数のコピーを作成する必要があるため，この場合は同期方式の方が効率が良いと考えられる．一方，プロセッサ間のプロセス間通信の場合，非同期型，同期型いずれの方式でも通信引数のコピーを作成する必要があるため，プロセッサ間の並列性を得やすい非同期方式の方が効率が良い可能性がある．

そこで，プロセッサ間のプロセス間通信機構の一部

表 4 非同期型の通信プロトコルを組み込んだ場合の応答時間
Table 4 Response times on a prototype system with non-blocking communication method.
(単位: sec)

利用者プログラム \ OS	非同期方式を用いた場合の pc98b での実行①	同期方式を用いた場合の pc98b での実行②
Cコンパイラ(1) (ソースコード 8012 bytes)	98.1	99.6
Cコンパイラ(2) (ソースコード 49797 bytes)	479.1	484.7
Cコンパイラ(3) (ソースコード 96413 bytes)	831.7	841.1

に非同期方式の通信プロトコルを実験的に組み込んで，同期方式を使用することによる性能の損失がどの程度であるかを調べてみた．ランデヴ時には，システム・プロセス間で双方向の通信引数の転送を行えるので，実際に非同期な通信にできるのは，ランデヴ呼出しを受け付けたシステム・プロセスからの通信引数の返信がない場合に限られる．一方，OS のプログラム上でプロセッサ間のプロセス間通信の大部分は送信データのみを含むプロセス間通信であることを確認した．この実験においても，実際に非同期に処理できたプロセッサ間のプロセス間通信の回数は，すべてのプロセッサ間のプロセス間通信の回数の約 91% であった．したがって，この実験ではプロセッサ間のプロセス間通信のすべてを非同期方式で処理する場合に近い性能を実現できると考えられる．

実験では，非同期に処理する方式として通信引数の返信がない場合はランデヴ解除を待たずにランデヴ呼出しを発行したプロセスの実行を再開させることにした．この方法を用いた場合，通信引数のコピーを格納する領域が不足する可能性が生じるが，この実験では領域不足が発生しないと仮定して行うことにした．

このような非同期型プロトコルを部分的に組み込んだ試作システムとこれまでの同期型プロトコルを組み込んだ試作システムの上で利用者プログラムの応答時間を測定した．その結果を**表 4** に示す．表 4 によれば，非同期型プロトコルを組み込んだ場合は，同期型プロトコルを組み込んだ場合に比べて応答時間が平均 1% 短縮されている．

4.2 考　　察

まず，プロセス間通信の負荷について考察する．表 2 によれば，OS の処理時間の半分以上がプロセス間通信の処理に占められており，重大な負荷になっていることがわかる．また，プロセッサ内，プロセッサ間の別では次のような比較ができる．表 1 のコンパイラ(3)の例をみると，発生しているプロセッサ内とプロセッサ間のプロセス間通信の回数の比は約 6：1 である．回数の点でみると，プロセッサ内のプロセス間通信の影響力が大きい．

一方，プロセッサ間のプロセス間通信の応答時間はプロセッサ内の応答時間の約 17 倍である．したがって，処理時間の合計でみると，プロセッサ内とプロセッサ間のプロセス間通信の比は約 6：17 になる．表 2 の②と⑦の値もほぼ同様の値になっている．

さらに，試作システムではプロセッサ内のプロセス間通信機構をアセンブリ言語で作成し，その応答時間を可能な限り短くした．これに対し，プロセッサ間のプロセス間通信機構はC言語のプログラムで，そこに含まれるソフトウェア以外の負荷，すなわちプロセッサ間の物理的なデータ転送の時間は全体の 3 %と小さい．以上から，プロセッサ間のプロセス間通信機構の改善の余地が大きく，また改善の効果も大きいと考えることができる．

次に，システム・プロセスの並列処理について考察する．同期方式であるランデヴによってプロセッサ間を結合する点について，表 4 の結果から性能面での損失は約 1 %であることが判明した．プロセスの規模が小さいため，プロセス間の同期によって失われる並列性が小さいこと，プロセッサ間のランデヴの解除の処理がディスク入力動作と並列に実行されている場合があり，その負荷が応答時間に反映しないことが損失が小さい理由としてあげられる．同期型の方式を用いることによる設計，デバックの容易性を考慮した場合，この損失は許容できる範囲であると考えられる．

表 2，表 3 の結果により，試作システムにおいて実際にシステム・プロセスの並列処理が行われていることが確認できる．特に，表 2 の④の並列処理は，システム・コールの応答時間を実際に短縮している．しかしながら，表 1 の②と③を比較すると，並列処理の効果は利用者プログラムの応答時間には反映していない．この原因は，表 2 の②と④の差異が示すように，プロセッサ間のプロセス間通信の負荷が並列処理の効果を上回っているためであると考えられる．したがって，プロセッサ間のプロセス間通信の処理時間を短くすることができれば，プロセッサ間のシステム・プロセスの並列性を性能向上に結びつけることができると

考えられる.

表2において②<④の関係が成り立てば，2台のプロセッサを用いて処理した場合のほうが1台のプロセッサで処理した場合よりもシステム・コールの応答時間が短くなる．したがって，たとえば close システム・コールの場合，プロセッサ間のプロセス間通信の負荷を現在の 66% 以下にすることができれば，1台のプロセッサで処理した場合よりも応答時間が短くなる．同様に，open システム・コールでは現在の 50% 以下，exece システム・コールでは現在の 44% 以下にできれば，1台のプロセッサで処理した場合よりも短くなる．

プロセス・ネットワーク方式による分散型 OS の性能を改善する上で，プロセス間通信の性能の改善が有効であることを確認した．特に，プロセッサ間のプロセス間通信を高速化することにより，一部のシステム・コールでは1台のプロセッサで処理した場合よりも応答時間を短くし，その他のシステム・コールでも1台のプロセッサで処理した場合に近くすることが可能である．このことから，プロセッサ間でディスク装置等の計算機資源を共有する疎結合のシステム構成において，利用者プログラムが配置されるプロセッサに依存せず同等の応答時間を実現できると考えられる．

5. 利用者プログラムの分散処理

プロセス・ネットワーク方式による分散型 OS の環境において，複数の利用者プログラムを別々のプロセッサで分散実行した場合の性能に関して調べた．本章では，そのための2つの実験の結果について述べる．

1つは，多重プログラミングの負荷分散に関する実験である．試作システムでは次のコマンド行により，2つのコンパイラを多重に実行させることができる．

```
cc sample1.c -o sample1 &
cc sample2.c -o sample2
wait
```

実験では，それぞれのコンパイラの実行を別々のプロセッサ上で行った場合の性能について調べる．

もう1つは，利用者プログラムの並列処理に関する実験である．試作システムでは，次のコマンド行のように，パイプを用いて利用者プロセスの間の通信を行うことができる．

```
sample1|sample2
```

実験では，sample1 と sample2 の実行を別々のプロセッサ上で行った場合の性能について調べる．

(1) 負荷分散

ファイル・システムを共有する条件のもとで，多重プログラミングの負荷を複数のプロセッサに分散させた場合の性能を調べ，単一プロセッサ上で多重プログラミングを行った場合と比較した．サンプルとして，ファイル・システムを頻繁に使用するコンパイラを選んだ．

実験では，同一の内容を含む2つのソース・ファイルのコンパイルを2つの PC-9801 でほぼ同時に開始し，両方の実行が完了した時点を応答時間として記録した．その応答時間を**表5**の④に示す．表5の①～③は④の比較対象として1台の PC-9801 の上で2つの利用者プログラムを多重に実行した結果である．

表5の④では，利用者プログラムは別々のプロセッサで実行されるのに対し，ファイル参照の際の OS の内部処理の一部，および物理入出力は pc98a 側で集中的に実行される．

2つのコンパイラの起動は，次のようなシェル・スクリプトを pc98a 上で実行することにより行った．

```
cc sample1.c -o sample1 &
remote cc sample2.c -o sample2
wait
```

remote コマンドは，子プロセスを pc98b 上に遠隔生成 (remote fork) して pc98b 上でコマンド行に指定されたプログラムを起動し，pc98a 上でその終了を待ち合わせるプログラムである．

表5によれば，コンパイラの実行では PC-UX システムと比較して 15～30% ほど応答時間が短くなっている．この結果から，ファイル・システムを共有する疎結合のシステム構成において，利用者プログラム

表 5 利用者プログラムの負荷分散
Table 5 Load balancing of user programs
(単位: sec)

OS / 利用者プログラム	PC-UX システム上で多重に実行した場合①	試作システム		
		pc98a 上で多重に実行した場合②	pc98b 上で多重に実行した場合③	pc98a, pc98b に分散した場合④
C コンパイラ(1)の並列実行 (ソースコード 30 bytes)	45.9	43.2	43.4	37.4
C コンパイラ(2)の並列実行 (ソースコード 1011 bytes)	70.5	70.9	74.1	61.1
C コンパイラ(3)の並列実行 (ソースコード 6876 bytes)	149.4	153.9	155.1	115.6

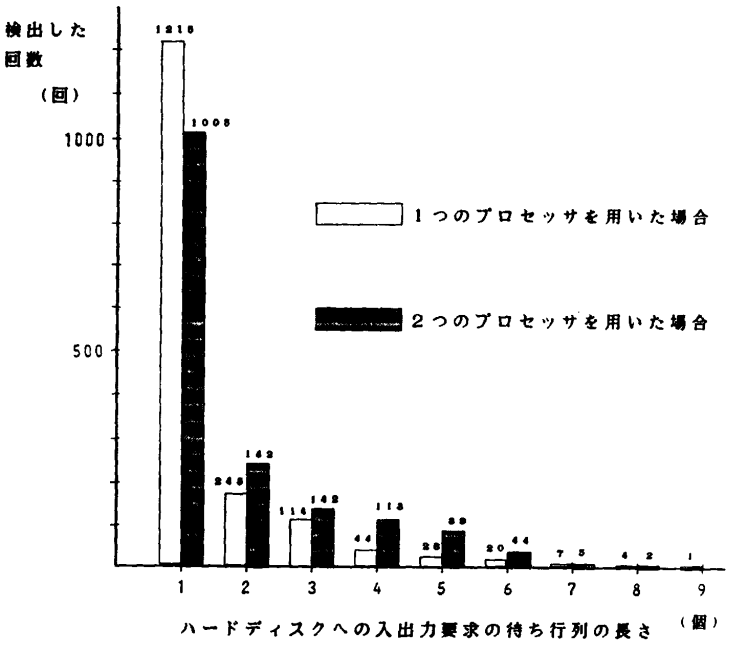

図 2 ハードディスクへの入出力要求の待ち行列の長さの分布
Fig. 2 A distribution of the length of the queue of I/O requests to the harddisk.

の負荷を分散することにより多重プログラミングのスループットが向上することを確認できた．

一方，この並列実行ではプロセッサ間でハードディスクを共有していることにより，入出力待ちの負荷が増している可能性がある．そこで，ハードディスク装置に対する入出力要求の待ち行列の長さの分布を測定してみた．**図 2** に，ハードディスク管理プロセスが入出力要求を受け付けた時点で，その時の待ち行列の長さを記録した結果を示す．サンプルは，表 5 の C コンパイラ (3) の並列実行である．図 2 では，比較対象として表 5 の②，すなわち単一プロセッサ上で多重プログラミングをした場合の結果を示した．図 2 によれば，2 つのプロセッサを用いて多重プログラミングを実現した場合の方が待ち行列の長さが長くなっており，入出力待ちの負荷が増していることがわかる．これは，利用者プログラムが 2 つのプロセッサによって実行されることにより，単一プロセッサ上で多重プログラミングを行う場合に比べて，入出力要求の発行される頻度が高くなっているためである．したがって，バッファリング方式の改善等の入出力待ちを解消する方法を導入することにより，スループットをより向上できると考えられる．

試作システムでは，1 つのファイル・システムを 2 つの利用者プログラムにより共有する構成における負荷分散の実験を行ったが，実際の分散処理システムでは，より多くのプロセッサ上の利用者プログラムによりファイル・システムが共有されると考えられる．プロセス・ネットワーク方式は，OS を構成するシステム・プロセスが軽量で小規模であることを特徴としている．このため，ファイル・システムに対する処理要求が密になり，プロセッサ中の動作可能なシステム・プロセスの数が多くなるほど，プロセッサの処理能力をより細かい単位で利用できるようになる．

この性質を確かめるために，PC-UX システムと試作システムにおいて，1 台のプロセッサ上に利用者プロセスを複数個生成して，そこから OS に対して利用者プログラムの起動を行う exece システム・コールをほぼ同時に発行する実験を行った．実験では，利用者プロセスを 1～8 個連続して生成し，そこで約 10 K バイトの利用者プログラムを起動する exece システム・コールを発行する．同一の利用者プログラムを起動することにより，ディスクのアクセス時間の影響を受けないようにした．利用者プログラムは，開始直後に終了するように変更しておき，すべての利用者プログラムが終了するまでの時間を応答時間として記録した．これにより，多くのプロセッサ上の利用者プログラムからファイル・システムを使用する場合に近い状況を実現できる．OS 内部では同時に複数個の exece システム・コールの処理が実行されており，プ

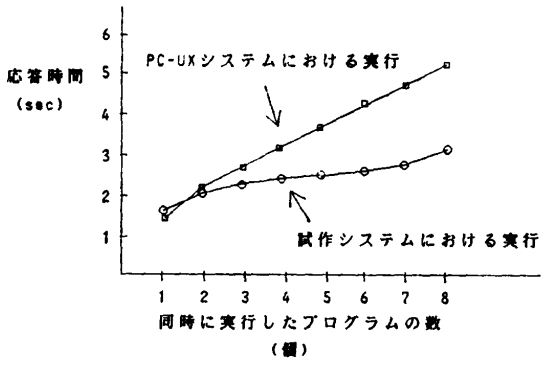

図 3 オペレーティング・システムのスループット
Fig. 3 Throughput of operating systems

ロセスの切換えが頻繁に起こっていると考えられる．したがって，OS 内部が軽量なプロセスで構成されている試作システムの応答時間の方が短くなることが予想される．実験の結果を図3に示す．図3によれば，試作システムの応答時間の増加の割合が減少している部分があり，それによって PC-UX システムよりも応答時間が短くなっている．

この結果から，より多くのプロセッサ上の利用者プログラムによりファイル・システムが共有される場合においても，プロセス・ネットワーク方式による分散型 OS を用いて良好な性能を実現できることが予想される．

（2） 利用者プログラムの並列処理

UNIX の利用者プロセス間通信機能であるパイプを用いて，互いに依存関係にある利用者プログラムを別々のプロセッサに配置した場合の効果について調べた．利用者プロセス間の通信の頻度と並列処理の効果の関係を調べるために，次のような実験を行った．まず，パイプを通して1バイトのデータを2,000回送受信する1対の利用者プログラムを作成した．この利用者プログラムは，外部の引数によって送受信から次の送受信までの時間間隔を設定できる．時間間隔が小さいことは，利用者プロセス間の通信の頻度が大きいことに相当する．この時間間隔を変化させることにより，処理の応答時間がどのように変化するかを測定した．その結果を図4に示す．

図4より，利用者プロセス間の通信の頻度がおよそ3 msec に1回より大きい場合，2台のプロセッサを用いた並列処理の効果は得られないが，それよりも頻度が小さくなるにしたがって並列処理の効果が得られるようになることがわかる．

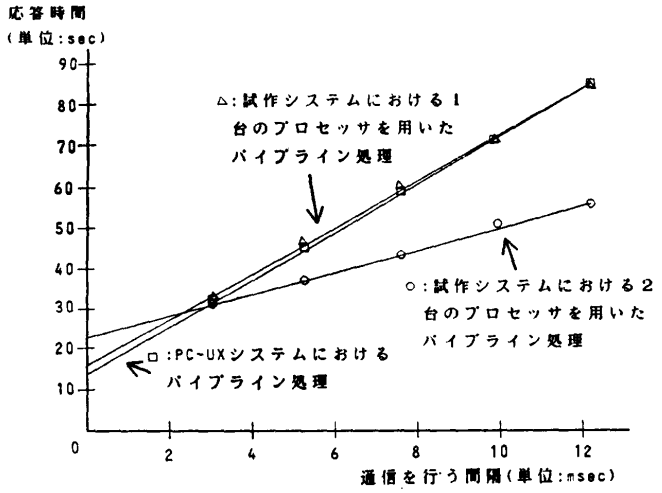

図4　パイプによる利用者プロセス間通信の性能

Fig. 4　Performances of inter-process communication by using pipe.

図4から各システムにおけるパイプの処理時間が計算できる．パイプの処理時間は，通信の間隔が0の場合に相当する応答時間を通信の回数である2,000回で割った値である．これによると，PC-UX システムでのパイプの処理時間は 7.4 msec, 試作システムの1台のプロセッサ内でのパイプの処理時間は 8.2 msec, 試作システムの2台のプロセッサ間のパイプの処理時間は 11.6 msec である．プロセッサにわたるパイプの処理時間が比較的大きいため，頻繁にパイプによる通信を行う場合は，並列処理の効果がパイプの負荷と相殺するのである．

6. むすび

本論文では，試作システムにおける測定，解析によって，プロセス・ネットワーク方式による分散型 OS の動作特性に関する知見を得ることを試みた．その結果，OS に関する知見を得ることを試みた．その結果，OS 内部ではプロセス間通信が OS の負荷の主な原因になっていること，OS 内部の並列度が高く，プロセッサの間で OS の処理の並列化が実際に行われていること，同期型のプロセス間通信方式を用いても OS の性能はそれほど損われないことなどの性質が明らかになった．

また，利用者プログラムの分散処理に関しては，プロセッサ間で単一のディスク装置を共有する疎結合の分散処理システムにおいて，負荷分散の効果により多重プログラミングのスループットが向上すること，パイプによる利用者プロセス間通信を用いた並列処理で負荷分散の効果が得られることを確認した．今回の実験で得られた結果から考慮して，Agora-1 におけるプロセス間通信の高速化により，有効な性能改善が期待できる．

今後の課題は，Agora-1 においてその有効性を評価すること，およびこの形態の分散処理システムに関してより一般的な考察を行うことである．

参考文献

1) Cheriton, D. R.: The V Kernel: A Software Base for Distributed Systems, *IEEE Softw.*, Vol. 1, No. 4, pp. 19-42 (1984).
2) Lazowska, E. D., Zahorjan, J., Cheriton, D. R. and Zwaenepoel, W.: File Access Performance of Diskless Workstations, *ACM Trans. Comput. Syst.*, Vol. 4, No. 3, pp. 238-268 (1986).
3) Rashid, R.: Designs for Parallel Architectures, *UNIX Review*, April, pp. 36-43 (1987).
4) 高野, 田胡, 益田：プロセス・ネットワークによる分散型オペレーティング・システムの設計, 情報処理学会論文誌, Vol. 29, No. 4, pp. 359-367 (1988).
5) 田胡, 益田：オペレーティング・システム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 (1983).
6) Tanenbaum, A. S. and Renesse, R. V.: Distributed Operating Systems, *Comput. Serv.*, Vol. 17, No. 4, pp. 417-470 (1985).
7) Theimer, M. M., Lantz, K. A. and Cheriton, D. R.: Preemptable Remote Execution Facilities for the V-System, *Proceedings of the Tenth Symposium on Operating Systems Principles*, pp. 2-12 (1985).

(昭和63年6月20日受付)
(平成元年1月17日採録)

高野 陽介（正会員）

昭和37年生. 昭和59年筑波大学第三学群情報学類卒業. 現在同大学院博士課程工学研究科に在学中. オペレーティング・システムの設計方式, 分散処理システムに興味を持つ.

田胡 和哉（正会員）

昭和31年生. 昭和56年筑波大学第三学群情報学類卒業. 昭和61年同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9月から東京大学工学部計数工学科助手. 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. ACM, 計測自動制御学会各会員.

益田 隆司（正会員）

昭和14年生. 昭和38年東京大学工学部応用物理学科卒業. 昭和40年同大学院修士課程修了. 同年(株)日立製作所入社. 中央研究所, システム開発研究所に勤務. 昭和52年筑波大学電子・情報工学系講師, 昭和53年助教授, 昭和59年教授. 昭和63年3月から東京大学理学部情報科学科教授. 工学博士. オペレーティング・システムの研究開発を経て, その方式論, 計算機システムの性能評価の研究に従事.