オペレーティング・システム 33－5
（1986. 12. 13）

分散型オペレーティング・システムの設計方式に関する一提案

高野 陽介[*]　田胡 和哉[**]　福田 宗弘[***]　益田 隆司[**]

* 筑波大学　工学研究科　** 筑波大学　電子・情報工学系　*** 筑波大学　理工学研究科

茨城県新治郡桜村天王台1-1-1

本報告では、分散型オペレーティング・システムの設計，記述の容易化を目的として、分散処理システムの計算機資源の管理機能の各々に分散透明な通信で結合されたプロセスを配置してシステムを設計する方法を提案する。この方式に基づいて試作機を実現し、方式の実現可能性を示した。

試作機で得られた知見をもとに、提案方式を適用した実用的なシステムとして、ジョブの静的な分散を行ない、複数のユーザに分散透明な資源利用が可能な共通の環境を提供する分散型のワークステーションの開発を進めている。

A PROPOSAL FOR DESIGN METHOD OF DISTRIBUTED OPERATING SYSTEMS

Yousuke TAKANO(Doctorial Program of Engineering, University of Tsukuba), Kazuya TAGO(Institute of Information Sciences and Electronics, University of Tsukuba), Munehiro FUKUDA(Master's Program in Scientific Technology, University of Tsukuba) and Takashi MASUDA(Institute of Information Sciences and Electronics, University of Tsukuba)

University of Tsukuba, Sakura-Mura, Ibaraki, 305 Japan

For the purpose of simplifying the design of distributed operating systems, we propose a design methodology of them. In our proposal, operating systems are composed of processes each of which manages one of computer resources accessed exclusively. Processes communicate one another by using network-transparent communication primitives. We built a prototype machine, and showed the feasiblity of our proposal.

As an application of our proposal, we are developing a distributed workstation, the operating system of which makes static job arrangement over processors, and provides users with a common, network-transparant environment for resource usage.

１．はじめに

マイクロプロセッサの高性能，低価格化により、分散処理システムの利用価値が高まっている。これに伴い、分散処理システムの持つ計算機資源の有効利用を図り、その統一的な制御を行なう分散型ＯＳの研究が進められている[1]。分散型ＯＳは、計算機資源の利用に関して分散透明な環境を提供する特徴を持つ。たとえば、システムに複数の二次記憶装置が付加されている場合でも、単一の階層構造として利用することができる。しかし、その一方、プロセッサ間が密に結合されることにより、設計がより複雑になることが予想される。

そこで我々は、分散型ＯＳの設計，記述の容易化を目的として、分散処理システムの計算機資源の管理機能の各々にプロセスを配置し、分散透明な通信で結合することにより分散型ＯＳを設計する方式を提案する。

以下では、提案方式の詳細，および，分散処理システムへの適用について述べる。また、提案方式の実証を目的として実現した試作機，および，その結果をもとに、現在、設計を進めている分散型のワークステーションＡｇｏｒａ－Ｉについて述べる。

２．分散型ＯＳの設計方式の提案

２．１　設計方式の提案

提案では、計算機の持つ排他アクセスされる資源の管理機能のおのおのに、分散透明な通信で結合されたプロセスを配置する。この構成を以後、プロセス・ネットワークとよぶことにする[6]。

プロセス・ネットワークによる分散型ＯＳは、図１のような構造をもつ。システムを構成するプロセスは通信のみで結合されており、周辺機器からの割込み等の非同期的な要因も通信に変換され、機器を管理するプロセスに伝達される。また、ＯＳを構成するプロセスと、ユーザ・プログラムの実行を行なうプロセスは別個に実現する。以後それぞれを区別してシステム・プロセス，および，ユーザ・プロセスとよぶことにする。ユーザ・プロセスの管理には、ユーザ・プロセスと１対１に対応するシステム・プロセスを割り付ける。ユーザ・プロセスで発生したシステム・コール等の割り出しは、通信に変換されて対応するシステム・プロセスに伝達される。システム・コールを受け付けたシステム・プロセスは、要求のあった資源の管理を行なうシステム・プロセスに通信し、その処理を行なう。システム・プロセスの記述は、専用言語ＰＮＬ／Ｄによって行なう。ＰＮＬ／Ｄでは、個々のシステム・プロセス単位の実行プログラムの記述と、プロセス間通信の参照関係の記述に分離して行なう。

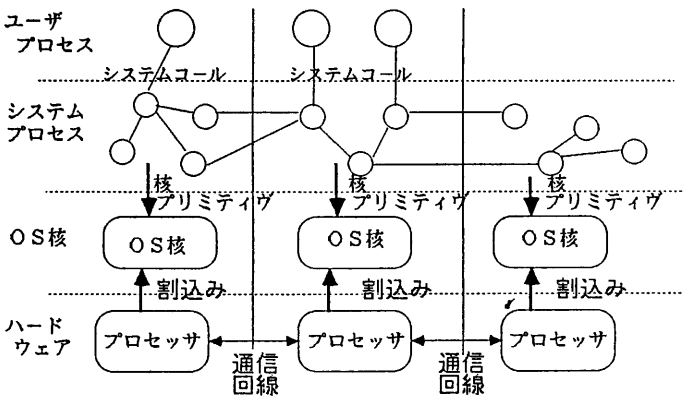

図１　プロセス・ネットワークによる分散型ＯＳ

プロセス・ネットワークの構造、すなわち、システム・プロセスの数、プロセス間通信の対応関係、システム・プロセスとその実行プログラムの対応等は、ＯＳのプログラミング時に決定される。システム・プロセスの動的な生成や、システム・プロセスに対する動的なメモリ割り当ては行なわない。

システム・プロセス間の通信機能とプロセスの切り換えは、各プロセッサ上のＯＳ核が実現する。システム・プロセスは、ＯＳ核の提供する通信命令を呼び出して、プロセス間通信を行なう。この通信命令を核プリミティヴと名付ける。ＯＳ核は、通信がプロセッサにわたる場合、プロセッサを結合する通信回線を制御してこれを実行することにより、分散透明な通信を実現する。

提案方式によれば、個々のシステム・プロセスの実行プログラムが通常の順序動作プログラムとして実現でき、制御構造の簡素化が図れる。これにより、システムの動作の理解が容易になる。また、システムの構造が静的であるため、システム・プロセス内部の記述とプロセス間の参照関係を別個に記述することができる。これにより、システム・プロセスのプロセッサへの配置，および，プロセス間の関係を集中的に記述でき、プロセス・ネットワークの変更やプロセッサへの配置の変更等が容易に行なえる。

提案方式では、システム・プロセスをユーザ・プロセスと分離して実現したことにより、システム・プロセスの環境保護機構，および，プロセス切り換えの際の環境切り換えが簡素化できる。また、システム・プロセスの動的な生成を行なわない。これらの特性により、ＯＳ核の機能は簡潔な構造で実現でき、プロセスの切り換えや通信の高速化が図れる。

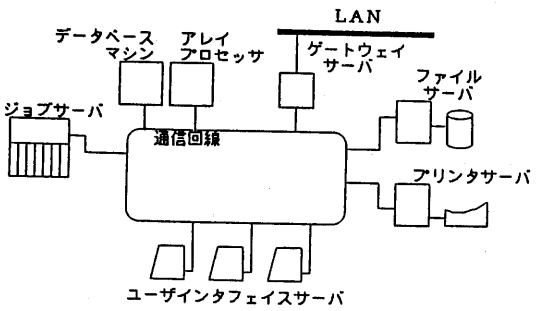

図２　分散型ワークステーションの構成

２．２　提案方式の実システムへの適用

我々は、提案方式を適用する対象として、図２に示すような分散型のワークステーションの実現を進めている。このシステムは、数人のユーザによりプログラム，文書，図面等の作成を行なう環境を想定し、このような目的で利用されている既存の計算機システムに比べ、より高い性能と使い勝手を実現することを目指す。

従来のワークステーションは単独で稼働するための全ての機能を備える構成をもつ。我々が実現を進めているシステムは、このようなワークステーション数台分の環境を集め、機能単位にプロセッサを配置して集中的に処理を行なう。これにより、ハードウェアの実現効率の向上，および，資源管理の容易化を図る。このような狙いから、システムは以下の構成をもつ。

1)ユーザ・インターフェースのプロセッサを複数台結合する。
2)ジョブ・サーバとして、密結合型のプロセッサ群で構成されたものを結合する。
3)データベース・マシンやアレイ・プロセッサのような特殊機を結合することができるようにする。
4)ゲートウェイ・サーバを介して外部のローカルエリアネットワークに接続する。すなわち、このシステムは、ローカルエリアネットワークの下位ネットワークとして運用される。

このようなシステムを単一のＯＳで制御することにより、ユーザに分散透明な利用環境を提供する。この環境は、ユーザ・インターフェースのプロセッサを通して、複数のユーザによる共有が可能である。

Ｖ－ｋｅｒｎｅｌ[2] やＡｃｃｅｎｔ[4] 等の既存の分散型ＯＳの設計と比較した場合の提案方式の特徴は次の点である。第一に、ユーザ・プロセスとは別のプロセスによりＯＳを実行する点が挙げられる。第二に、プロセッサ間を密に結合したシステム構成を狙う点が挙げられる。

３．　試作機の実現

提案方式では、プロセス間通信が高速化されている。その一方において、１回のシステム・コールの処理あたりに必要なプロセス間通信の回数の多いことが予想される。このため、通信の負荷，および，頻度の傾向と、それに対してどの程度の改善が必要かということを知る必要がある。これらの調査，分散透明なプロセス間通信機構の開発，および，提案方式の実証を目的として、その試作機を実現した[3][5]。以下では、試作機の設計，実現について述べる。

３．１　全体構成

ハードウェアは、プロセッサMAP16P、PC-9801Eを専用の通信回線で結合することにより構成した。

表１　核プリミティヴ

ｃａｌｌ，ｃａｌｌｓ	ランデヴ呼出しの発行
ａｃｃ	ランデヴ呼出しの受付および，ランデヴの開始
ｅｎｄｒ	ランデヴの終了

システム・プロセス間の通信方式は、ランデヴを用いる。ＯＳ核は、表１に示した核プリミティヴを提供する。ｃａｌｌプリミティヴは、ランデヴ呼出しの発行を行なう。その引数として、呼出しをかけるプロセスの識別子とその通信エントリ，および，通信引数を指定する。ａｃｃプリミティヴは、ランデヴ呼出しを受けつけて、ランデヴを開始する。その引数として、通信エントリを指定し、呼出しを選択的に受け付ける。ｅｎｄｒプリミティヴは、ランデヴを終了する。ｃａｌｌｓプリミティヴは、２つのシステム・プロセスが互いにランデヴ呼出しを行ないデッドロックに陥らないかどうかを検査し、デッドロックに陥る場合にはランデヴ呼出しを中止する点を除いて、ｃａｌｌプリミティヴと同様の動作を行なう。

システム・プロセス間のデータ転送は、参照呼び(call by reference)によって行なう。同一のプロセッサ上のランデヴ呼出しでは、相手のシステム・プロセスに通信データのアドレスを渡し、呼出しを受け付けたシステム・プロセスは、ランデヴの間、このアドレスを参照して通信データの受け渡しを行なう。プロセッサ間の通信では、呼出しを受け付けたシステム・プロセスの側のプロセッサ上に通信データのコピーを送ることにより、分散透明な通信を実現する。このコピーを、ＩＣＢ（Inter-processor Communication Buffer）と名付ける。

システム・プロセスの層はＵＮＩＸsystemⅢ* と互換性のあるシステム・コールを提供する。システム・プロセスの設計では、MAP16Pにはハードディスク装置と磁気テープ装置の管理プロセスを配置した。

* UNIX is a trademark of AT&T Bell Lab.

PC-9801Eには、ディスプレイ，キーボード，マウスの管理プロセスやファイル・システムを実現するプロセス群のほか、ジョブの実行を制御するスーパバイザ・プロセスを配置した。このように、ファイルの実体はMAP16P側に、それを利用するユーザはPC-9801E側に存在する。

システム・プロセスの記述は、ＰＮＬ／Ｄ言語を用い、システム・プロセス内部の記述と、プロセス間の参照関係の記述を分離して行なう。

システム・プロセス内部の記述は、Ｃ言語にモジュール管理を行なうための機構を付加することにより行なう。システム・プロセス内部におけるランデヴ呼出しの記述では、通信相手のエントリ名を直接参照するのではなく、モジュール内部で仮の名前を定義して用いる。これを送信エントリ名と名付ける。一方、受付に用いる実際のエントリ名を受信エントリ名と名付ける。

プロセス間の参照関係の記述は、プロセスの定義，および，通信エントリ名と受信エントリ名の対応付けにより行なう。参照関係記述の例を図３に示す。(1)-(12)は、プロセスの定義である。プロセスの定義は、プロセッサごとに行なう。(2)-(5) は、プロセスclientの定義である。(1) ではこのプロセスclientを配置するプロセッサの型名をPC98と定義し、(6) ではこのプロセッサの実体名をclient_processorと定義している。(3) は、プロセスclientの内部の記述を含むモジュール名がclient_module であることを定義する。(4) は、送信エントリio_requestを定義する。同様にして、(7)-(12)では、プロセスdriverをMAP 型のプロセッサio_processor上に配置することを定義している。たとえば、(10)では受信エントリioを定義している。

```
processor PC98 {                        (1)
      process {                         (2)
        text      client_module;        (3)
        output    io_request;           (4)
      } client;                         (5)
} client_processor;                     (6)

processor MAP {                         (7)
      process {                         (8)
        text      driver_module;        (9)
        input     io;                   (10)
      } driver;                         (11)
} io_processor;                         (12)

system {                                (13)
  client.io_request > driver.io;        (14)
}                                       (15)
```

図３　ＰＮＬ／Ｄプログラムの例

(13)-(15) は、プロセス間の参照関係の定義である。(14)では、送信エントリio_requestと送信エントリioを結合して、２つのプロセスの通信関係を定義している。

ＰＮＬ／Ｄの処理系は、現状で部分的にのみ動作しているので、試作機ではシステム・プロセスの記述の多くをＣ言語によって行なった。

以下の各節では、試作機のハードウェア，プロセス間通信，および，システム・プロセスのおのおのについて詳細に述べる。

３．２　ハードウェア構成

試作機では、プロセッサ間の通信における良好な応答性を得るため、専用の通信回線を実現した。すなわち、試作機は、プロセッサ間でデータを１６ビット並列転送するための専用バスと、プロセッサ間のハンドシェイク，バスの使用権制御を行なうためのハードウェアをもつ。バスのアービトレーションは、制御トークン・パッシング方式によって行なう。ＯＳ核は、この通信ハードウェアにデータの送信，受信，および，バスの解放の３命令を発行し、通信回線を利用する。

通信ハードウェアの設計は、ソフトウェアにかかる負担をできるだけ小さくすることを狙いとして行なった。第一に、転送するデータを格納する領域に２ポートＲＡＭを使用した。これにより、プロセッサ本体のバスには通信のデータ転送による負荷がかからない。第二に、シークエンサを用いて通信制御の自動化を図った。図５に通信の動作を示す。ＯＳ核は、あらかじめ通信ハードウェアに受信命令を発行し、送られてくるデータを格納するための領域を指定しておく（ａ）。他のプロセッサからの受信要求が到着すると、シークエンサはプロセッサに介入せずに指定されている領域へデータ転送を開始する（ｂ）。転送終了後、シークエンサは、プロセッサに割込みをかける（ｃ）。また、シークエンサには受信命令と送信命令を同時に発行できるようにした。これにより、回線制御のための割込みの回数を減らすことができる。

３．３　プロセス間通信の実現

システム・プロセスは、核プリミティヴを呼び出してプロセス間通信を実行する。核プリミティヴは、Ｃ言語上から関数として利用できる。

プロセス間通信機構の設計は以下のように行なった。プロセッサ間の通信では通信データの転送を必要とするので、この処理のために通信データの大きさを指定する必要がある。また、通信データが相手に渡されるだけのものであるか、または、指定したアドレスにデータが返却されるのかを区別する情報を指定する。これらの情報を表現するため、通信データは、アドレス、属性の対で指定する。この様子を図４のｃａｌｌプリミティヴ（①）の例で示す。ｐｒｏｃ，ｅｎｔはそれぞれ通信先のプロセスの識別子と通信エントリである。

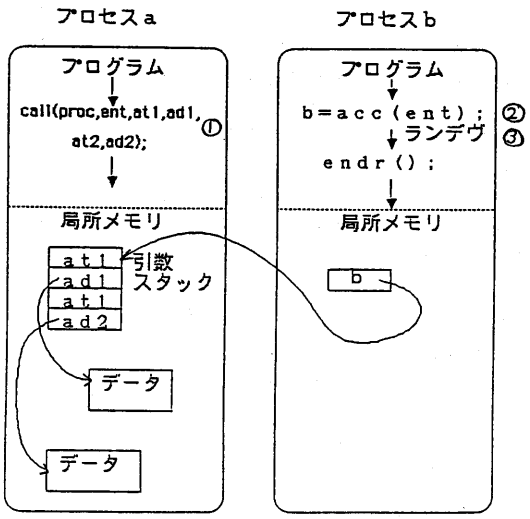

図４　プロセッサ内の通信におけるデータ転送

また、ａｔ１，ａｄ１，および，ａｔ２，ａｄ２は通信データを指定する属性、アドレスの対である。属性は、１６ビットの値で、上位２ビットが表２に示した情報、下位１４ビットがデータの大きさを指定する。

プロセッサ内での通信の実現手順を図４に示す。ランデヴ呼出しを行なうプロセスは、ｃａｌｌプリミティヴでデータのアドレスを引数として指定する（①）。呼出しを受け付けるプロセスは、ａｃｃプリミティヴの返却値としてその引数列を格納した領域のアドレスが渡される（②）。そして、引数列からデータを格納したアドレスを取り出し、このアドレスからデータを取り出すか、あるいは、書き込むかすることによってデータ交換を行なう（③）。同一プロセッサ上にあるプロセスの局所メモリは同一アドレス空間内に割り付けられており、プロセッサ内での通信では、受け付けたプロセスは、呼出しを行なったプロセスのデータを直接参照できる。

表２　通信データの属性情報

値	意味
０	引数はアドレスでなく即値として扱う。
１	引数は呼出し先に送付するデータのアドレスである。
２	引数は呼出し先から返されるデータを格納するアドレスである。
３	引数は呼出し先に送付されると同時にそこから返されるデータを格納するアドレスである。

プロセッサにわたる通信におけるデータ交換の実現手順を図５に示す。また、それに伴うデータ構造を図６に示す。図６の例では、属性値ａｔ１は呼び出すプロセスにデータを送付する指定であり、ａｔ２は呼び出すプロセスからデータを返却する指定であるものとする。ＯＳ核は、ｃａｌｌプリミティヴの発行を受け付けると、ｃａｌｌプリミティヴの引数列，および，引数のもつアドレスが指すデータの実体をパケット化し（①）、通信先のプロセスが存在するプロセッサ上のＯＳ核に送付する（②）。これをｃａｌｌパケットとよぶことにする。これを受けつけたＯＳ核はＩＣＢの領域を確保し（③）、２ポートＲＡＭ上に転送されているｃａｌｌパケットをもとにＩＣＢを作成する（④）。そこでは、データの配置に基づいて、送られてきた引数列上のアドレス値を付け替える必要がある。通信を受け付けたプロセスは、ａｃｃプリミティヴの返却値として渡されたアドレス値を参照してＩＣＢにアクセスし、データを交換する。ｅｎｄｒプリミティヴによってランデヴが終了すると、ＯＳ核はＩＣＢ中の返却されるデータをパケット化して返送する（⑤）。これをｅｎｄｒパケットとよぶことにする。これを受け付けたＯＳ核は、引数列中のアドレス値に従って、返送されてきたデータを呼出しを行なったプロセスの局所メモリに代入する（⑥）。

プロセッサ間通信のプロトコルは、通信の応答性を重視する目的から、通信が成功した時の性能が最も良くなるように設計した。すなわち、ランデヴ呼出しの通信に対するアクノレッジを省略して、ランデヴ終了の通信をこれに充てる。また、ランデヴ呼出し時にはプロセス間の待ち合わせ状況とは無関係に通信データを先送りする。これにより、ＩＣＢを用いた通信データのバッファリング効果が期待できる。

プロセッサ間のランデヴ呼出しでは、ＩＣＢに用いる領域の有無に従ってフロー制御を行なう。ＩＣＢの領域が確保できないのは、次の２つの場合である。１つは、ＩＣＢのための領域が不足している場合である。もう１つは、特定の通信エントリへの通信に通信バッファの大部分が占有されている場合である。この場合、他の通信の応答性が低下するのを防止するため、占有状態に至るようなランデヴ呼出しに対してはＩＣＢの割り当ては行なわない。通信データは先送りされているので、呼出しが不成功の場合、送られたデータは棄却される。そして、不成功に終わった呼出しの記録は受け付けた側に残され、のちに受けつけられる状態に至った際、記録をもとに通信の再送要求が発行される。これを受けて、呼出し側は最初と同じ通信動作を繰り返す。

３．３　ＯＳの実現

試作機の分散型ＯＳは、図７のように実現した。ま

ず、PC-9801E側に、ユーザ・プロセスと１対１に対応し、その管理を行なうプロセスを配置した。これをスーパバイザ・プロセスとよぶ。複数個のスーパバイザ・プロセスは、同一の実行プログラムを共有する。このようなプロセス群をプロセス・アレイと名付ける。スーパバイザ管理プロセスは、スーパバイザ・プロセスの割付け，解放を制御するプロセスである。次に、主記憶の管理に記憶管理プロセスを配置し、周辺装置に対し、個々が接続されているプロセッサ上にその管理プロセスを配置する。PC-9801E上には、タイマ，グラフィック・ディスプレイ・コントローラ（ＧＤＣ），ディスプレイとキーボード，マウス，フロッピーディスク装置の管理プロセスを、MAP16Pには、ハードディスク装置，磁気テープ装置の管理プロセスを配置する。また、スーパバイザ・プロセスとの間にファイル・システムを構成するプロセス群を配置する。

ユーザ・プロセスがハードディスク上のファイルを読む経過は、以下のようである。システム・コールを受け付けたスーパバイザ・プロセスは、ファイル・プロセスを呼び出す。ファイル・プロセスは、要求から

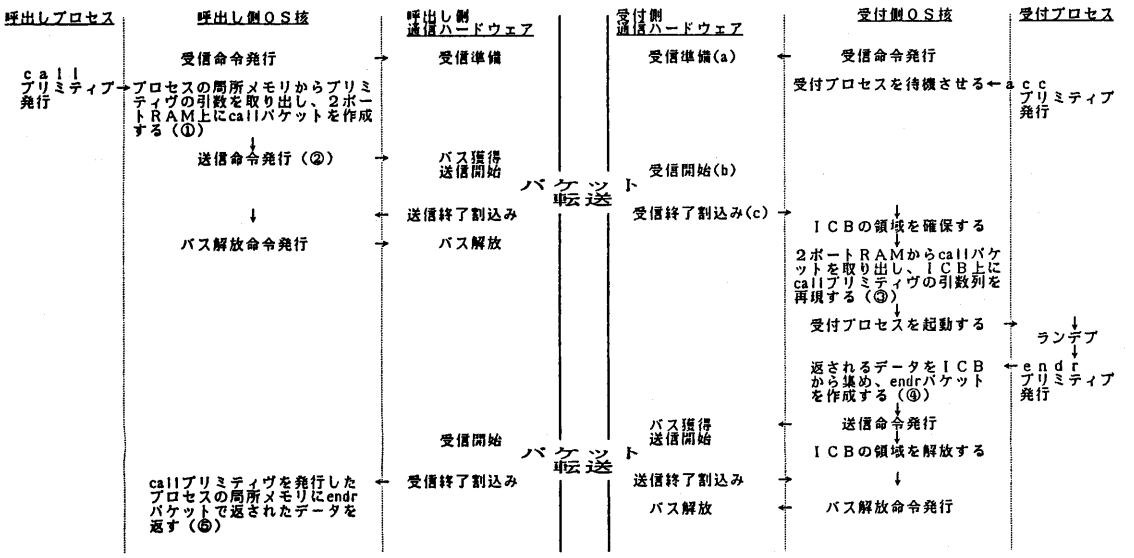

図５　　プロセッサ間の通信機構

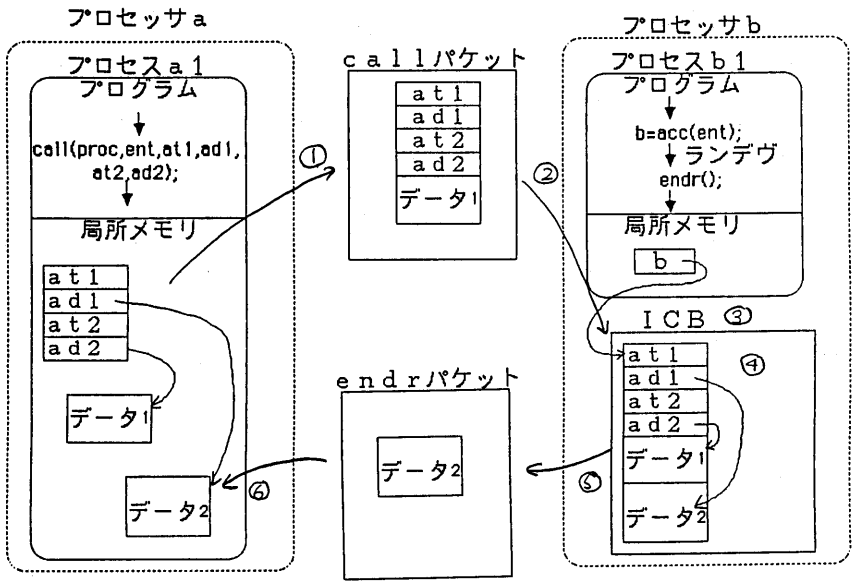

図６ プロセッサ間の通信におけるデータ転送

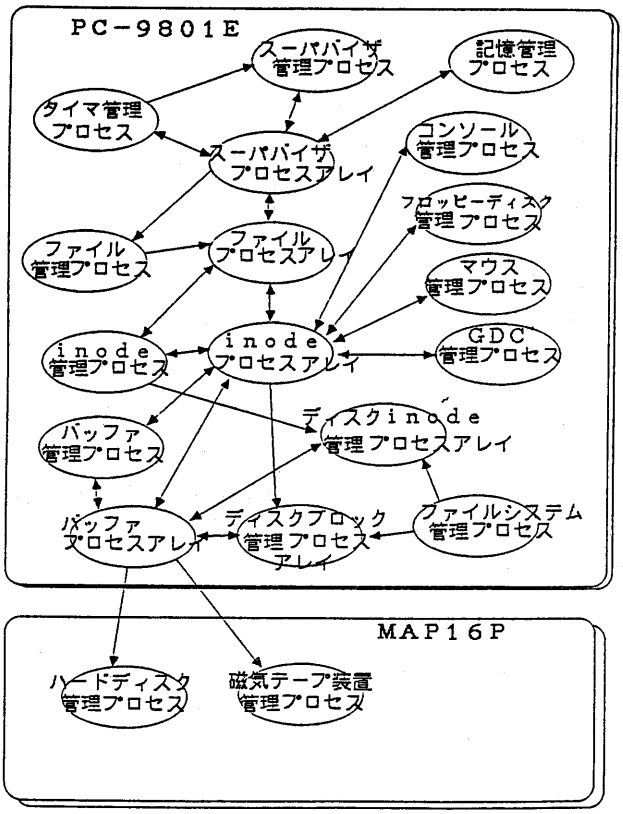

図7　試作機のＯＳ

アクセス権の確認とファイル上の論理ブロック番号の計算を行なって、ｉｎｏｄｅプロセスを呼び出す。ｉｎｏｄｅプロセスは、ハードディスク上の物理ブロック番号を計算する。そして、バッファ管理プロセスを呼び出して、当該物理ブロックを持っているバッファがあるかどうか調べる。ない場合は、新たなバッファが割り付けられる。バッファ管理プロセスの返答を受けて、ｉｎｏｄｅプロセスはバッファを管理しているバッファ・プロセスを呼び出し、物理ブロックを要求する。物理ブロックの内容を持っていない場合、バッファ・プロセスは、ハードディスク管理プロセスを呼び出して、物理ブロックを要求する。物理ブロックが獲得できると、逆方向に通信が行なわれ結果が伝えられる。

ファイル管理プロセス，ｉｎｏｄｅ管理プロセスはファイルのオープン，クローズ時にそれぞれファイル・プロセスとｉｎｏｄｅプロセスの割付け，解放を行なう。ディスク・ブロック管理プロセス・アレイとディスクｉｎｏｄｅ管理プロセス・アレイは、ディスク装置ごとのフリー・ブロックとフリーｉｎｏｄｅを管理する。ファイル・システム管理プロセスは、ファイル・システムのマウント等を制御する。

３．４　実現状況

現在、試作機は、MAP16P上のハードディスク装置からファイルを読み込んでブートし、エディットやコンパイル等の作業が行なえ、また、マウスやグラフィック・ディスプレイを用いて簡単な図形作成等を行なうことができる状態にある。

システム，および，プロセス間通信の性能の測定は今後の課題であるが、プロセス間通信にどの程度の負荷がかかるかについて、おおまかな目安を得るために、簡単な測定を行なった。

ユーザ・プロセスがハードディスクから入力を行なうのに要した時間を測定した。設計からこのシステム・コールの処理に１１回のプロセス間通信を要し、そのうち１回はプロセッサ間の通信であることがわかっている。測定の結果５１２バイト単位の入力では、システム・コールあたり約２９ｍｓのシステム時間を要した。試作機で使用したディスク装置の平均回転待ち時間は１２．５ｍｓ，平均シーク時間は４０ｍｓである。一方、ＰＣ－ＵＸ*での測定では、同様の処理に２４ｍｓを要した。この測定で使用したディスク装置の平均回転待ち時間は８．３ｍｓ、平均シーク時間は８５ｍｓである。

また、試作機のシステム時間には、１回のプロセッサ間通信に要する１１．８ｍｓと、１０回のプロセッサ内の通信に要する４．５ｍｓが含まれることがわかった。特に、プロセッサ間通信には２５６バイトの大きさのｃａｌｌパケットと７６８バイトの大きさのｅｎｄｒパケットの転送が含まれる。２５６バイトの物理的な転送には、転送に必要なソフトウェアの制御も含めて、２．３５ｍｓを要することが測定されている。これにより、プロセッサ間通信に要した１１．８ｍｓのうち、９．４ｍｓがデータ転送に使用され、残りの２．４ｍｓがプロセスの切り換え，バケットの作成，ＩＣＢの作成に使用されることがわかる。

測定の結果、提案方式によるシステムは通常のシステムとして運用できる見通しを得た。また、効率の良いシステムとするには、プロセス間通信の性能を改善する必要があることがわかった。

４．　Ａｇｏｒａ－Ｉの設計

試作機の実現の経験をふまえ、現在、分散型のワークステーションＡｇｏｒａ－Ｉを開発中である。

Ａｇｏｒａ－Ｉでは、ＯＳ核の動作論理が簡潔であることに着目し、ＯＳ核の機能の大部分をハードウェアによって実現し、通信の高速化を図る。これをハードウェア・カーネル（ＨＫ）と名付ける。ＨＫは、マイクロ・プログラムを実行する２つのシークエンサか

* PC-UX is a trademark of NEC Corporation.

らなる。一方は、プロセス切り換えとプロセッサ内の通信を制御し、もう一方は通信回線に結合され、プロセッサ間の通信を実現する。これらをそれぞれ、プロセス・コントローラ（ＰＣ）、および、コミュニケーション・コントローラ（ＣＣ）と名付ける。ＰＣは、核プリミティヴを受け付け、同期処理を実現し、次に実行すべきプロセスの識別子を出力する。従って、ソフトウェアは、ＰＣに核プリミティヴを発行すること，および，出力された識別子に従ってプロセスを切り換えることのみを行なう。プロセッサにわたる通信の場合、ＰＣは、ＣＣに命令を出す。このとき、ＣＣは、通信データをパケット化せずに、通信処理の進行に応じて、随時、送出を行なう。また、ＩＣＢを固定長にして管理の簡潔化することにより、その割付け，解放もＣＣが実現する。ＨＫでは、引数がない場合、プロセッサ間の通信に要する時間と、プロセッサ内の通信に要する時間とをほぼ同一にすることを目標としている。これは、ランデヴ呼出しに伴うプロセスの環境退避，および，切り換わるプロセスの環境回復に要する時間内に、プロセッサ間の通信を完了させることによって達成できる。

Ａｇｏｒａ－Ｉの各プロセッサのＣＰＵは、ＭＣ６８０１０／２０を用いる。Ａｇｏｒａ－Ｉの各プロセッサ，割り当てた機能，および，構成上の特徴を表３に示す。ディスプレイ，マウス等が付属するプロセッサを２台結合し、ユーザ・インターフェースを実現する。ハードディスク装置，レーザ・ビーム・プリンタイーサネット・インターフェースが付属するプロセッサは、ファイル・サーバ，プリンタ・サーバ，および，ゲートウェイ・サーバとして使用する。このプロセッサは、レーザ・ビーム・プリンタに出力する図形イメージを主記憶上で作成し、ビデオ信号を出力して、レーザ・ビーム・プリンタを制御する。密結合型のプロセッサ群で構成されたプロセッサは、ジョブ・サーバとして使用する。このプロセッサにもディスク・キャッシュとして大容量の主記憶を装備する。

５．　おわりに

分散型ＯＳを設計する１つの方式として、相互排除アクセスされる資源の各々に、分散透明な通信でのみ結合されたプロセスを配置する方法を提案し、この提案に基づく試作機の実現について述べた。今後の課題は、Ａｇｏｒａ－Ｉの実現とその評価である。

［参考文献］

[1] Andrew S.Tanenbaum and Robbert Van Renesse: Distributed Operating Syetems,Computing Surveys, Vol.17,No.4,pp.419-470(1985).

[2] David R.Cheriton: The V.Kernel:A Software Base for Distributed Systems,IEEE,software,Vol.1 ,No.2,pp.19-43(1984).

[3] 福田，田胡，高野，益田：分散システムAgora-proto のハードウェア，情報処理学会第３２回全国大会予稿集(1986).

[4] Rashid,R.F.,and Robertson,G.G: Accent:A communication oriented network operating system kernel,In Proceedings of the 8th Symposium on Operating Systems Principles.ACM,pp.64-75(1981).

[5] 高野，田胡，福田，益田：分散システムAgora-proto の実現，情報処理学会第３２回全国大会予稿集(1986).

[6] 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,pp524-534(1984).

表３　Ａｇｏｒａ－Ｉのプロセッサ構成

プロセッサ名	プロセッサの機能	構成
ユーザ・インターフェース プロセッサ	マルチ・ウィンドウ，エディタ等のインタラクティヴ・ジョブの実行	ビットマップ・ディスプレイ ＡＣＲＴＣ，マウス，キーボード
ジョブ・サーバ プロセッサ	コンパイラ，ｒｏｆｆ等のバッチ・ジョブの実行	密結合型のプロセッサ群 大容量の主記憶
機器管理 プロセッサ	ファイル管理，プリンタ制御，ゲートウェイ制御	ディスク装置 レーザ・ビーム・プリンタ・インターフェース イーサネット・インターフェース