
# 27. 分散プロセスの通信方法†

徳田　英幸††

## 1. はじめに

プロセス間の通信は，単なる事象の通知，待合せを行うための同期の役目だけでなく，実際のデータの転送機能を提供し，協同作業を行う上での重要な役割を持っている．**プロセス間通信**（以下，**IPC** と略す）の発展は，多重プログラミングシステムの同期操作の拡張の一方式として出現し，以後コンピュータネットワーク，複数台のミニコンピュータやマイクロプロセッサによる分散システム，高機能ローカルエリアネットワークなどの分散処理システムとともに発展してきている．

初期の IPC の概念は，主として単一プロセッサ内での多重プログラミングシステムにおける並行プロセス間での通信（ローカル IPC）を対象としていた．例えば，MULTICS システムの notify-wait 操作[45]，RC 4000 システムの send-wait 操作[9]，ISPL システムの port モデル[6]などがあげられる．一方，70年代初期のコンピュータネットワークの出現は，物理的に独立した複数のプロセッサ上で実行されているプロセス間での通信（リモート IPC）の必要性を認識させ，ネットワークオペレーティングシステムを構成する上での重要な基本機能としての位置づけをし，システム内でのローカルおよびリモート IPC の統一した取扱いをめざした[37),3)]．しかし，ネットワークにおけるプロトコルの階層化は，異機種プロセッサ間での接続およびネットワーク資源の共有を可能としたが，実効転送スピードの低下は，統一的な効率の良い IPC に基づく種々の分散型ソフトウェアの可能性を拡大するまでには至らなかった．

今日のような分散システム，分散型ソフトウェアを構成する上での重要な役割を果たす IPC としての発展は，Farber らの DCS システム[17]以降に見られる70年代後半からの分散システムにおける分散型オペレーティングシステム，分散処理言語，分散処理に適した計算のモデルなどの研究成果によるところが大きいといえよう[15]．

本稿では，分散処理システムにおける共有変数が存在しない環境下でのプロセス間通信法について述べる．まず，IPC の基本的な通信方式を分類し，2つのプロセス間通信問題を取り上げ，代表的な方式による解および問題点を示し，最後に，IPC に関する今後の動向について述べることにする．

## 2. IPC の基本方式

ここでは，特定の分散処理システム，分散処理言語，計算モデルなどに用いられるメッセージ通信の解説をさけ，単純なモデルを用いて，IPC の基本方式を分類することにする．

プロセス間のメッセージ通信における最も基本的な操作として

**send** (to-object, message)

**receive** (to-object, message)

があげられる．ここで，“to-object” は，交信する相手プロセスまたは，相手プロセスへの論理的な通信路をしめす対象物 (object) をしめし，“message” は通信のためのメッセージを表わしている．この通信を指定する対象物を基に，今日まで提案および実現されてきた IPC の方式は，大きく以下の2つのモデルに分類することができる．(**図-1** 参照)

1）**直接通信モデル** (direct IPC model)：相手の対象物として直接にプロセス名，アドレス，識別子 (process identifier)，プロセス内の手続き名などを指定する．手続き名を指定するモデルを，ここでは**手続**

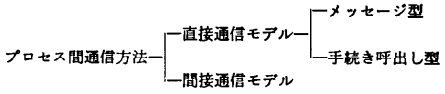


**図-1**　プロセス間通信方法の基本的な分類

† Interprocess Communication Facilities for Distributed Processes by Hideyuki TOKUDA (Department of Computer Science and C.C.N.G., University of Waterloo).

†† ウォータールー大学計算機科学科

**き呼出し型**と呼び，その他を**メッセージ型**と呼ぶことにする．

2）**間接通信モデル** (indirect IPC model)：相手の対象物が論理的な通信路をしめすポート (port) やリンク (link) などの第3の対象物を指定する．

メッセージ型の直接通信モデルとしては，多重プログラミングシステムの RC 4000[9]，Thoth[11]，分散システムの DCS[17]，MINET[28]，KOCOS[2]，shoshin[46]，計算モデルの Actor モデル[20]，分散処理向きの言語としての CSP[21]，PLITS[18]，ACT-1[26] などがあげられる．手続き呼出し型のモデルとしては，分散処理向きの言語の DP[10]，SR[4]，Ada[39]，オブジェクト指向型言語の smalltalk-80[50]，OPL[34]，などがあげられる．一方，間接通信モデルの例としては，多重プログラミングシステムの DEMOS[7]，マルチプロセッサシステムの HYDRA[49]，star OS[24]，分散システムの RIG[5]，HXDP[23]，Arache (旧名 Roscoe)[43]，Medusa[35]，そして言語関係においては，PCL[25]，*MOD[12]，Argus[27]，DPL-82[16]，CMU-IPC[38]，Virtual call[32]，CMUPORT[40]，communicating port[30]，Algebra モデル[31]などがあげられる．

一般に，これら send/receive 操作に対しては，**通信の機能を大きく決定する2つの要因，すなわち同期モードと交信パターン**が与えられる[29]．

同期モードは，プロセスがメッセージを交信する際に，相手プロセスとの待合せを必要とするかどうかを決定し，待合せする場合を**同期型**，そうでない場合を**非同期型**と呼ぶ．一般に，同期型の send 操作は，送信プロセスを受信プロセスがメッセージを受信するまで待ち状態にし，同期型の receive 操作は，受信プロセスをメッセージの受信まで待ち状態とする．非同期型の send 操作は受信プロセスとの待合せをせず，メッセージがバッファに格納された後，送信プロセスを再開させる．非同期の receive 操作には，割込み駆動型とポーリング型があり，前者はメッセージの到着後，割込みと同様に指定された受信処理ルーチンが呼出される方式で，後者は，操作の実行時点でメッセージの到着を調べ，メッセージが存在する場合は，それを受信し，そうでない場合は，メッセージの存在しない旨を伝える方式をとっている．

交信パターンは，送信側と受信側の対象物間での論理的な通信のパターンをしめし，以下の4つに大別される．

1）単一送信：特定のプロセスへの送信．

2）単一受信：特定のプロセスからの受信．

3）複数送信：特定のプロセスのグループまたは，不特定多数のプロセスへの送信．(一斉同報 (broadcast) は，同時に不特定多数のプロセスにメッセージを送信する場合をさす．)

4）複数受信：特定のプロセスのグループまたは，不特定多数のプロセスからの受信．

一般に，直接通信モデルでは，メッセージの send/receive 操作の実行時に，対象のプロセスの数に応じてその交信パターンが限定される* のに対し，間接通信モデルでは，ポートやリンクなどの属性として他プロセスへの通信路ごとに，これらの交信パターンを限定させ，単一送信路，単一受信路，単一送受信路，複数送信路，複数受信路，複数送受信路などを実現することができる．

### 2.1 直接通信モデルの基本操作

直接通信モデルにおける基本的な操作としては，前述の send/receive 操作も含めて以下の操作があげられる．

send (to_process, message, option)
receive (from_process, message, option)
request (to_process, message, result, option)
reply (to_process, result, option)

ここで，“to_process” と “from_process” は，相手プロセスの識別子 (process identier)，“message” と “result” は，転送または，受信すべきメッセージを表わしている．プロセスの識別子は，そのプロセスが生成された時点にシステムが与えるそのプロセスに固有な値を持ち，またすでに生成されているサーバプロセスなどの識別子は特定の識別子を持つプロセス (ネームサーバなどのプロセス) にサーバの提供するサービス名をしめして，その識別子が見つけ出されるものとする．

これら4つの基本操作には，同期型，非同期型としての意味づけを与えることができるが，このモデルでは，“*b*” と “*n*”** をプリフィックスとして与え，“bsend”，“breceive” は同期型，“nsend”，“nreceive” は非同期型を表わすことにする．また，request 操作は，相手プロセスへメッセージを送り，相手プロセスからの返信 (reply) が到着するまで待機する同期

* 手続き呼出し型モデルでは，手続き呼出しの性質上，送信プロセスは特定のプロセスの手続きしか呼出せないし，受信プロセスは，特定の送信プロセスを指定することはできない．したがって，2) と 3) のパターンは存在しない．

** “b” は blocking, “n” は non-blocking の略．

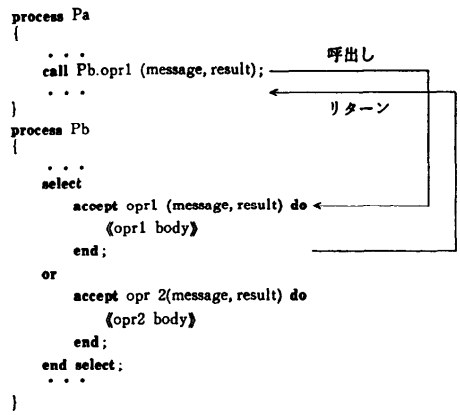


図-2 call-accept の機構

型とし，reply 操作は，相手プロセスへメッセージを返信し，直ちに実行を再開する非同期型を表わすことにする．そして特定のプロセスのグループへの複数送受信には複数のプロセス識別子を指定でき，不特定多数からの受信は次の receiveany 操作で行うことができ，送信プロセスの識別子がその値として返される．

Process=breceive_any (message, option)

各操作における“option”は，メッセージのタイムアウトなどのエラー処理，トランザクション処理などのための選択受信 (selective receive) などのオプションを指定できるものとする．

手続き呼出し型の直接通信モデルにおいては，相手プロセスの提供する手続きをリモート手続き呼出し (remote procedure call) することによって，メッセージの交信を行う．一般的な操作としては，図-2 にしめす call/accept の機構があげられる．

この例では，プロセス $p_a$ がプロセス $p_b$ の手続き $op_1$ を message でしめされたデータで呼出し，結果を result としてリターンする関係を表わしている．一方，受信側のプロセス $p_b$ は，Ada[39] における accept, select と同様な機能を用いて，送信側からの opr 1 または opr 2 の手続きの呼出しを同等に待合せしていることを表わしている．

### 2.2 間接通信モデルの基本操作

間接通信モデルにおける基本操作は，通信のためにどのような第3の対象物を介在させるかによっていろいろな形式をとっている．例えば，第3の対象物としては，システム内で独立した対象物としてのグローバルなポート[38]，プロセスに付属する対象物としてのローカルなポート[30]，論理的な通信路を対象物とするリンク[7),43)] などがあげられる．ここでは，基本的な機能を直接通信モデルと比較するために，以下の6つの操作によりメッセージの交信を行うローカルなポートのモデルをとりあげることにする．

createport (port, type)
destroyport (port)
connect (myport, otherport)
disconnect (myport, otherport)
send (port, message)
receive (port, message)

ここで，create 操作は，プロセス内に“type”で指定された型のポートを生成し，destroy 操作は，ポートを消滅させる機能があり，ポートの型としては，同期，非同期型の単一送信 ([b|n] single-out)*，単一受信 ([b|n] single_in)，複数送信 ([b|n] multi_out)**，複数受信 ([b|n] multi_in) などが指定できることとする．connect 操作は，2つの入力，出力用の指定されたポートを接続し，メッセージ通信のための通信路を確立する．また，接続は，入力側および，出力側のいずれか一方のプロセスから行うものとし，送信—送信，受信—受信などの接続は成立しない．ポート間の接続は，disconnect 操作で解消される．send, receive 操作は，指定したローカルなポートに対しての送信または，受信を行うものとする．

## 3. 基本的な例題と問題点

ここでは，前節で定義した直接通信および，間接通信モデルの基本操作を用い，2つの例題を通じて各通信方式における問題点について考えてみることにする．例題としては，プロセス間での種々の協同作業形態を作業を要求するリクエスタプロセス (requestor) と作業を行うサーバプロセス (server) との関係で分類されている **IPC 標準問題群 (IPC canonical problems)**[47] の中から基本的な $N$ 個のリクエスタと1個のサーバ問題とパイプライン方式で結合された $N$ 個のサーバ問題をとりあげることにする．

### 3.1 N・リクエスタ・1・サーバ (nR-1S) 問題

問題は，次の条件を満たすリクエスタとサーバの協同作業を IPC 基本操作を用いて記述することである．リクエスタは，必要とするサービスを提供するサーバを見つけ出し，そのサーバに要求メッセージを送り，

* [b|n] single-out は，bsingle-out および nsingle-out を表わし，“b” は同期型，“n” は非同期型を示している．

** 同期型の複数送信 (bmulti-out) は，すべての接続されている入力用ポートからメッセージが受信されるまで，送信プロセスを待ち状態とする．

```
process requestor
{
    processid server;
    ...
    server=find ("service");
    set_up (message);
    request (server, message, result);
    ...
}
```

```
process server
{
    processid pid;
    ...
    while (TRUE) {
        pid=breceive_any (message);
        do_service (message, result);
        reply (pid, result);
    }
    ...
}
```

〈アルゴリズム 3.1〉

**図-3** 直接通信モデルによる nR-1S 問題の解法

```
process requestor
{
    port reqport, replyport;
    createport (reqport, nsingle_out);
    createport (replyport, bsingle_in);
    ...
    connect (reqport, server.inport);
    send (reqport, message);
    receive (replyport, result);
    disconnect (reqport, serve.inport);
    ...
}
```

```
process server
{
    port inport, outport;
    createport (inport, bmulti_in);
    createport (outport, nsingle_out);
    ...
    while (TRUE) {
        receive (inport, message);
        do_service (message);
        connect (outport, message.reply);
        send (outport, result);
        disconnect (outport, message.reply);
    }
    ...
}
```

〈アルゴリズム 3.2〉

**図-4** 間接通信モデルによる nR-1S 問題の解法

結果が返送するまで待機する．一方，サーバは，任意の $N$ 個のリクエスタからの要求メッセージを順次受付け，サービスを実行して結果を返信し，次の要求メッセージの受信まで待機しなければならない．

直接通信モデルでは，同期型の request, breceive_any, 非同期型の reply 操作を用いることになり，リクエスタとサーバの協同作業は，**図-3** のアルゴリズム **3.1** のように記述することができる．ここで，リクエスタは find 操作によりサーバの識別子を見つけ出せるものとする．

一方，ローカルなポートを用いる間接通信モデルでは，入力，出力用の2つのポートをそれぞれのリクエスタ，サーバに生成することにより，協同作業に必要な通信路を確立することができる．リクエスタとサーバの関係は，**図-4** のアルゴリズム **3.2** で表わすことができる．このアルゴリズムでは，直接通信モデルとの相違を明確にするため，サーバのポートは，リクエスタのプロセス生成時に与えられていると仮定している．一般的には，connect-send-receive-disconnect という手順を合成し，

call (standard_request port, message, result)

という形式をとっているモデルもある[7]．

以下では，この例題を介しての通信モデル，アルゴリズム上の問題点について考察することにする．

- プロセスの結合 (process binding)

プロセス間の結合は，相手プロセスの持つグローバルな名前（例えば，サービス名）から，メッセージ通信に必要な固有の識別子，アドレスなどに変換し，送信または受信のためのアクセス権を得ることである．一般に，結合する時点としては，プログラムコンパイル時，リンク時，プロセスの生成時，任意の実行時などがあげられる．“動的な結合”を行うプロセスをネームサーバ (name server) または結合代理人 (binding agent)[48] などと呼び，特定の自明なアドレス，識別子，リンクなどが与えられている．したがって任意の時点でプロセスはネームサーバへの問合せができ，必要とするサービスを提供するプロセスの識別子を得ることができる．

CSP, PLITS などの分散処理言語における直接通信モデルでは，“静的な結合”（コンパイル時およびリンク時）が行われている．特に CSP のように不特定多数からの受信操作が可能でない場合，サーバプロセスのような汎用のプロセス* を記述できない．一方，直

* プログラムライブラリなどにおける汎用のプロセス群をも含む．

接通信モデルを使用している多くの分散システムにおいては，実行時にプロセスの動的な結合が行われている．

間接通信モデルにおいても，静的および動的な結合が行われる．CMU-IPC などに見られるような独立した対象物としてのグローバルなポートの場合，ポート名をサービス名と同等に扱うことにより，直接通信モデルで使用した find 操作を用いることが可能である．一方，ローカルなポートの場合，動的に結合するには，サービス名からプロセスとポートの対となった識別子，またはアドレスが必要となるのでプロセスの識別子だけと動的に結合させ，ポート名は，概知の値（例えば，standard_in, standard_out 等）で静的に結合させる方法もある．また，DEMOS や Arache に見られるリンクを用いるモデルでは，ネームサーバへの問合せを省くために，プロセスの生成時に特定のリンクを重要なシステムのサーバへ接続させてしまう方法がとられている．プロセスの結合のためのプロセス（サービス）の名前付け，ネームサーバの生成，追加，削除，更新に関する問題は，文献 1), 15), 48) に述べられている．

- 異常通信およびエラー処理

アルゴリズム 3.1, 3.2 においては，基本的なリクエスタ・サーバ間の協同作業の記述が中心で，実際のアルゴリズムとしては，プロセッサの故障，ネットワークの異常などによる異常状態に対するエラー処理を考慮する必要がある．一般に，同期型の操作の場合には，相手プロセスの消滅，プロセッサの停止，デットロックなどの異常状態が検出された際は，エラー情報をその通信に関与しているプロセスに通知し，待機中のプロセスを再開される必要がある．エラー情報の通知は，IPC 操作の“結果の値”としてプロセスに知らせる方法[46]や，プロセスの実行環境内の補助変数を介して知らせる方法[5]などがある．非同期型の操作では，通信を起動したプロセスは一般に待機中ではないので，同期型と別に例外処理 (exception handling) の機能が必要となる．基本的な機能としては，エラーを検出した後エラー情報とともに指定された例外処理ルーチンへ制御の流れを移動させることであり，ルーチンを起動する方法には，割込み処理のように任意の時点で呼出す方法[28]や，次の任意の IPC 操作実行時に呼出す方法[5]などがある．しかし，割込み方式は1つのプロセス内で任意の時点においてプロセス本体から例外処理ルーチンへ制御が移るので，本体が参照しているデータを例外処理ルーチンが変更してしまう可能性がある．したがって，1つのプロセス内のルーチン間での相互排除を配慮する必要がある．

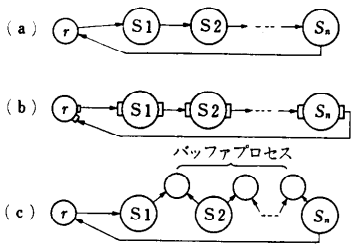


図-5 pipelined nS 問題におけるサーバプロセスの交信パターン

### 3.2 パイプライン式N・サーバ (pipelined nS) 問題

この問題は，一列に並んだ $N$ 個のサーバプロセスを用いて1つのサービスを提供し，パイプライン方式を使って効率よく仕事を行うための協同作業を記述することである．

まず，サーバ間の関係としては，直接，間接の両通信モデルにおいて非同期型の送信が与えられているので，図-5 (a), (b) に見られるような交信パターンをとる．一方，CSP などに見られる同期型だけの通信モデルにおいては，図-5(c) に示されたような $N$-1 個のバッファプロセスが効率上必要となってしまう．図-5 (a),(b) における両端でない $S_*$ から $S_{n-1}$ までのサーバは，左側のサーバからメッセージを同期型で受信し，担当の作業をそのメッセージに対して行い，右側のサーバへ非同期型の送信を行っている．また，サーバ $S_1$ はリクエスタからの要求メッセージを受信し，サーバ $S_n$ はその結果を返信している．ここでは，両端のサーバを除き，中間に位置するサーバの動作を概略すると，図-6 におけるアルゴリズム 3.3, 3.4 のように表わすことができる．アルゴリズム 3.3 にしめされる直接通信モデルでは，サーバ $S_i$ が left, right の関数により，サーバ $S_{i-1}$, $S_{i+1}$ と結合していることを表わしている．一方，アルゴリズム 3.4 の間接通信モデルでは，サーバ $S_i$ が非同期型の単一送信路をサーバ $S_{i+1}$ における同期型の単一受信路と接続させることにより，サーバ間の結合を確立している．

以下では，この例題に見られるような非同期型の送信におけるメッセージのバッファリング，順序の問題，そしてサーバプロセスの構成上の問題について考えることにする．

- 非同期型の送信

非同期型の送信においては，送り出されたメッセー

```
process server(i)
{
    ・・・
    while (TRUE) {
        breceive (left(i), message);
        do_service (message.field[i]);
        nsend (right(i), message);
    }
    ・・・
}
```

〈アルゴリズム 3.3〉

```
process server(i)
{
    port inport, outport;
    createport (inport, bsingle_in);
    createport (outport, nsingle_out);
    connect (inport, left(i). outport);
    ・・・
    while (TRUE) {
        receive (inport, message);
        do_service (message.field[i]);
        send (outport, message);
    }
    ・・・
}
```

〈アルゴリズム 3.4〉

**図-6** pipelined nS 問題におけるサーバ $S_i(1<i<N)$ のアルゴリズム

ジを相手プロセスが受信するまでシステム内に確保しなければならないバッファリングの問題と，次々と同一プロセスより送り出されるメッセージの受信順序の問題がある．このメッセージバッファに対して，PLITS, ACT-1 などの分散処理言語では，概存の Algol 系言語に見られるスタックやヒープエリアと同様に，無限にあるものと仮定してしまっているものがある．一方，実際のシステムで用いられている操作では，空のバッファがなくメッセージを確保できない場合の処置として，メッセージがバッファに確保されるまで送信プロセスを待機させる方法[38),46)]，待機せず，相手プロセスへの最後のメッセージバッファの上に重ねて確保してしまう方法[23),40)]，待機せず，メッセージをシステムの2次的なバッファに確保し，通常のバッファが空き次第そこへ移し，何らかの通知を送信プロセスに伝える方法[5),38)] などがとられている．また，これらの方法に加えて，間接通信モデルでは，ポートやリンクの生成時にその通信路の持てる最大メッセージ数，バッファなどを指定することが可能である．いずれの場合も，プロセス間の協同作業のセマンティクスに即した方法の選択が重要である．

メッセージの受信順序に関しては，送信プロセスにおける送信順序が受信プロセスにおいても同一に保存されているモデルと，されていないモデルがある．同一

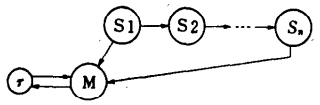


**図-7** マネージャプロセス M とサーバ $S_1$〜$S_n$ の関係

プロセスからの送信順序が保存されない場合には，プロセスレベルで各メッセージに番号を付け，メッセージの順序づけをしなければならない．したがって，多くのモデルでは下位のプロトコルで順序づけをし，プロセスレベルでは送信順序の保存を保証している．

- サーバプロセスの構成

ここでの例題のように，リクエスタからの要求メッセージが複数のサーバプロセスによって処理され，メッセージを受信したサーバでない他のプロセスから結果が返信されるような場合，直接通信モデルにおける request, call 操作などを用いることはできない．したがって，1つの方法としては，図-7 に見られるようにサービスを代表し，要求メッセージの管理を行うプロセス）マネージャ[46)]，マドミニストレイタ[19)] などと呼ばれる）を生成し，このプロセスに対して request, call 操作を行うようサーバの構成を変更することである．このようなマネージャプロセスによる $S_1$ から $S_n$ までのサーバの“カプセル化”は，サーバ内部の変更を外部より遮蔽する効果がある．一方，サーバの構成を変更しない方法としては，PLITS で使われているようなトランザクションに固有な番号 (transaction_id) をメッセージに付加してサーバへ送り，リクエスタは特定のプロセスからの返信を待つのではなく，この番号の付いているメッセージを待つようにする方法がある．また，ポートやリンクなどの間接通信モデルにおいてもサーバからの返信は，特定のポートまたは，リンクに対する受信操作で行われるので，特定のサーバに依存せず結果のメッセージを送り返すことが可能である．

## 4. 今後の動向

分散システムにおけるプロセスの通信法は，今後も複数マイクロコンピュータによる分散処理システム，高性能のローカルエリアネットワーク，オブジェクト指向型およびデータフロー型アーキテクチャなどの発展と共に進化を続け，対象とするシステムに最も適した形での通信法に対する系統的な設計法がソフトウェアおよびハードウェアの両面で重要となるであろう．具体的な動向については，以下の3点についてふれ，まとめとする．

1）高い信頼性を必要とする分散プログラムに関しては，Liskov らの Argus[27] に見られるようなリモート手続き呼出しで要求されるプロセスの動作を“アトミック動作 (atomic action)”として保証していこうというアプローチや，さらに細かな属性を与えようとする Ellis らの提案[14] などが拡張されていくであろう．

2）ローカルエリアの分野では，Nelson によるローカルな手続き呼出しと同一のセマンティクスを持ったリモート手続き呼出しモデルの実現[33]，Shrivastava らの提案[42]などは，ゼロックスネットワークシステムにおけるリモート手続き呼出しプロトコルの採用[13]と相まって今後，リモート手続き呼出しモデルが1つの位置を占めるであろう．また，spector のリモート命令のモデル[44] は，今後のローカルエリアネットワーク上での高度な分散処理を実現していく上での一方向を示している．

3）IBM の 925 ワークステーション[41]，ベル研の UNIX 用ビットマップ端末 blit[36]，ウォータールー大学のペイントプログラム[8] などの報告に見られるように，単一のシステム内においてもメッセージ通信により協同作業を行う複数プロセスに基づくシステム構成が広く用いられてきており，Intel 432[22] などに見られるようなアーキテクチャレベルでの IPC 機能のサポートがますます重要となっていくであろう．

## 参 考 文 献


1) Abraham, S. M. and Dalal, Y. K.: Techniques for Decentralized Management of Distributed Systems, Proc. COMPCON 80 Spring, IEEE, pp. 430-437 (1980).
2) Aiso, H., Kamibayashi, N., Tokuda, H. et al.: A Minicomputer Complex-KOCOS, Proc. IEEE/ACM 4th. Data Communications Symposium, pp. 5-7 to 5-12 (Oct. 1975).
3) Akkoyunlu, E., Bernstein, A. and Schantz, R.: Interprocess Communication Facilities for Network Operating System, Computer, Vol. 7, No. 6, pp. 46-55 (Jun. 1974).
4) Andrews, G. R.: Synchronizing Resources, ACM Trans. Prog. Lang. Syst., Vol. 3, No. 4, pp. 405-420 (Oct. 1981).
5) Ball, J. E., Feldman, J. R., Low, J. R. et al.: RIG, Rochester's Intelligent Gateway: System Overview, IEEE Trans. Soft. Eng., Vol. SE-2, No. 4, pp. 321-328 (Dec. 1976).
6) Balzer, R. M.: PORTS —a Method for Dynamic Interprogram Communication and Job Control, Proc. 1971 SJCC, pp. 485-489 (1971).
7) Baskett, F., Howard, J. H. and Montague, J. T.: Task Communication in DEMOS, Proc. 6th Symp. of Operating Systems Principles, ACM, pp. 23-32 (1977).
8) Beach, R. J., Beatty, J. C., Booth, K. S. and et al.: The Message is the Medium: Multiprocess Structuring of an Interactive Paint Program, Proc. of SIGGRAPH '82, ACM (1982).
9) Brinch Hansen, P.: The Nucleus of a Multiprogramming System, CACM, Vol. 13, No. 4, pp. 238-250 (Apr. 1970).
10) Brinch Hansen, P.: Distributed Processes: A Concurrent Programming Concept, CACM, Vol. 21, No. 11, pp. 934-941 (Nov. 1978).
11) Cheriton, D. R., Malcolm, M. A. et al.: Thoth, a Portable Real-Time Operating System, CACM, Vol. 22, No. 2, pp. 105-115 (Feb. 1979).
12) Cook, R. P.: *MOD—A Language for Distributed Programming, IEEE Trans. Soft. Eng., Vol. SE-6, No. 6, pp. 563-1980 (Nov. 1980).
13) Courier: The Remote Procedure Call Protocol, XSIS 038112, Xerox Corp. (1981).
14) Ellis, C. S., Feldman, J. A. and Heliotis, J. E.: Language Constructs and Support Systems for Distributed Computing, Proc. to Principles of Distributed Computing, ACM, pp. 1-9 (1982).
15) Enslow, P. and Gordon, R. L. (eds.): Interprocess Communication in Highly Distributed Systems Report GIT-ICS-79/09, Georgia Institute of Technology, (Dec. 1979).
16) Ericson, L. W.: DPL-82: a Language for Distributed Processing, Proc. of 3rd. Distributed Computing Systems Conf, IEEE, pp. 526-531 (1982).
17) Farber, D. J., Feldman, J. et al.: The Distributed Computing System, Proc. COMPCON 73, IEEE, pp. 31-34 (Mar. 1973).
18) Feldman, J. A.: High Level Programming for Distributed Computing, CACM, Vol. 22, No. 6, pp. 353-368 (Jun. 1979).
19) Gentleman, W. M.: Message Passing Between Sequential Processes: the Reply Primitive and the Administrator Concept, Software-Practice and Experience, Vol. 11, pp. 435-466 (1981).
20) Hewitt, C.: Viewing Control Structure as Pattern of Passing Messages, J. Artificial Intelligence, Vol. 8, No. 3, pp. 323-364 (Jun. 1977).
21) Hoare, C. A. R.: Communicating Sequential Processes, CACM, Vol. 21, No. 8, pp. 666-677

(Aug. 1978).
22) iAPX 432 General Data Processor Architecture Reference Manual, Intel 171860-001, Intel Corp. (1981).
23) Jensen, E. D.: The Honeywell Experimental Distributed Processor—An Overview, Computer, Vol. 11, No. 1, pp. 28-38 (Jan. 1978).
24) Jones, A. K., Chansler, R. J., Durham, I. et al.: StarOS, a multiprocessor Operating System for The Support of Task Forces, Proc. 7th Symp. of Operating Systems Principles, ACM, pp. 117-127 (Dec. 1979).
25) Lesser, V., Serrian D. and Bonar, J.: PCL: A Process-Oriented Job Control Language, Proc. of 1st. Distributed Computing Systems Conf, IEEE, pp. 315-329 (1979).
26) Lieberman, H.: A Preview of Act-I, AI Memo 625, MIT AI Laboratory (Apr. 1981).
27) Liskov, B.: On Linguistic Support for Distributed Programs, IEEE Trans. Soft. Eng., Vol. SE-8, No. 3, pp. 203-210 (May 1982).
28) Manning, E. G. and Peebles, R. W.: A Homogeneous Network for Data Sharing—Communications, Computer Networks, Vol. 1, No. 4, pp 211-224 (1977).
29) Manning, E. G., Tokuda, H. and Livesey, N. J.: Inter-Process Communication in Distributed Systems: One View, Proc. IFIP Congress 80, pp. 513-520 (1980).
30) Mao, T. W. and Yeh, R. T.: Communication Port —A Language Concept for Concurrent Programming, IEEE Trans. Soft. Eng., Vol. SE-6, No. 2, pp. 194-204 (Mar. 1980).
31) Miller, R.: A Calculus of Communicating Systems, Lect. Notes in Comp. Sci., Vol. 92, Springer-Verlag (1980).
32) Nabert, W. L.: Indirect Interprocess Communication: Virtual Calls as a Message Passing Discipline for Message-Switched Operating Systems, M. Math. Thesis, Dept. of Computer Science, University of Waterloo (1982).
33) Nelson, B. J.: Remote Procedure Call, Ph. D. Thesis, Computer Science Dept., Carnegie-Mellon University (1981).
34) Object Programming Language User's Manual, Intel 171823-001, Intel Corp. (1981).
35) Ousterhout, J. K., Scelza, D. A. and Sindhu, P. S.: Medusa: An Experiment in Distributed Operating System Structure, CACM, Vol. 23, No. 2, pp. 92-105 (Feb. 1980).
36) Pike, R.: The Blit Programmer's Manual, Tech. Memo, Bell. Lab. (Apr. 1982).
37) Proc. of ACM SIGCOMM/SIGOPS Interprocess Communications Workshop, Operating Systems Review, Vol. 9, No. 3 (Jul. 1975).
38) Rachid, R. F.: An Interprocess Communication Facility for UNIX, Tech. Report, Computer Science Dept., Carnegie-Mellon University (Feb. 1980).
39) Reference Manual for ADA Programming Language, United States Department of Defense (July 1980 and July 1982).
40) Reid, L. G.: Control and Communication in Programmed Systems, Ph. D. Thesis, Computer Science Dept., Carnegie-Mellon University (1980).
41) Selinger, R. D., Patlach, A. M. and Carlson, E. D.: The 925 Family of Office Workstations, Research Report RJ 3406, IBM Research Lab., San Jose (Feb. 1982).
42) Shrivastava, S. K. and Panzieri, F.: The Design of a Reliable Remote Procedure Call, Tech. Report 171, Computing Lab., Univ. of Newcastle upon Tyne (Oct. 1981).
43) Solomon, M. H. and Finkel, R. A.: The Roscoe Distributed Operating System, Proc. 7th Symp. of Operating Systems Principles, ACM, pp. 108-114 (Dec. 1979).
44) Spector, Z. A.: Multiprocessing Architecture for Local Computer Networks, Ph. D. Thesis, Dept. of Computer Science, Stanford University (1981).
45) Spier, M. and Organick, E.: The Multics Interprocess Communication Facility, Proc. 2nd Symp. of Operating Systems Principles, ACM, pp. 83-91 (1969).
46) Tokuda, H., Manning, E. G. and Radia, S. R.: Shoshin OS: a Message-based Operating System for a Distributed Software Testbed, Tech. Report, CCNG, University of Waterloo, (Apr. 1982). also in Proc. of 16th Hawaii Int. Conf. on System Science, Vol. 1, pp. 329-338 (Jan. 1983).
47) Tokuda, H.: An IPC Model for a Distributed Software Testbed, Tech. Report, CCNG, University of Waterloo (Aug. 1982). (to appear in Proc. of ACM SIGCOMM '83)
48) Watson, R. W.: Identifiers (Naming) in Distributed Systems, in Lect. Notes in Comp. Sci., Vol. 105, pp. 191-210, Springer-Verlag (1981).
49) Wulf, W., Cohen, E., Corwin, A. et al.: HYDRA: The kernel of a Multiprocessor Operating System, CACM, Vol. 17, No. 6, pp. 337-345 (Jun. 1974).
50) XEROX Learning Research Group: The Smalltalk-80 System, Byte, Vol. 6, No. 8 (Aug. 1981).

(昭和57年12月14日受付)