2003－OS－93 (4)
2003／5／8

*Tender*における即時同期を可能にするプロセス間通信機構の実現と評価

福冨 和弘† 田端 利宏†† 谷口 秀夫†††

†九州大学大学院システム情報科学府
††九州大学大学院システム情報科学研究院
†††岡山大学工学部

プロセス間通信は、多くの応用プログラムで利用されている。したがって、プロセス間通信機能は重要であるとともに、その利用形態はさまざまである。プロセス間通信において、データをやり取りする場合、メッセージ渡し方式が多く利用されている。しかし、メッセージ渡し方式によるプロセス間通信では、送信操作と受信操作の組み合わせで通信を完了するため、通信操作の即時性とデータ授受の同期を両立できない。そこで、本論文では、即時同期を可能にするプロセス間通信機構を提案する。また、提案機構の実現方式として、***Tender*** オペレーティングシステムの場合を述べ、基本性能評価の結果と応用プログラムにおける評価の結果を示す。

Implementation and Evaluation of Instant Synchronization InterProcess Communication on *Tender*

Kazuhiro FUKUTOMI†, Toshihiro TABATA†† and Hideo TANIGUCHI†††

†Graduate School of Information Science and Electrical Engineering, Kyushu University
††Faculty of Information Science and Electrical Engineering, Kyushu University
†††Faculty of Engineering, Okayama University

InterProcess Communication (IPC) is used in many application programs. IPC is important and used in various ways. Especially exchanging data, a common form of IPC is message passing. However, in the message passing, since communication is completed in the combination of the send and receive operation, IPC cannot contain two elements simultaneously — the instant communication operation and the synchronization of data transfer. In this paper, we propose the mechanism of Instant Synchronization InterProcess Communication. We describe the implementation of this mechanism on ***Tender*** operating system. Also, we show the basic performance of this mechanism and evaluate this mechanism using an application program.

1 はじめに

近年、ハードウェアの性能が飛躍的に向上し、多くの計算を伴う処理を行う応用プログラム (以降、AP と略す) を比較的短い処理時間で実行できるようになってきた。また、計算機の価格性能比の向上により、計算機が普及し、計算機を日常的に使用する機会も多くなってきた。これに伴い、計算機上で動作する AP や AP の実行を支援するオペレーティングシステム (以降、OS と略す) に対して、高機能化や多機能化が求められている。特に、プロセス間通信は、多くの AP で利用されている。したがって、プロセス間通信機能は重要であるとともに、その利用形態はさまざまである。

プロセス間通信において、データをやり取りする場合、メッセージ渡し方式 (以降、送受信方式と呼ぶ) が多く利用されている。送受信方式によるプロ

セス間通信では、送信操作と受信操作の組み合わせで通信を完了するため、通信処理の終了が通信相手の状況に依存したものになる。つまり、送受信の同期を求める場合、通信相手の送信、または受信処理が完了するまで待つ必要がある。しかし、これらの待ち処理の終了時刻を保証することは難しい。したがって、実時間処理のように、一定時間内に要求された処理を終える必要がある場合に好ましくない。また、通信操作の即時性を求める場合は、送受信の同期が保証されないため、操作要求を発行した時刻と、実際にデータをやり取りする時刻が異なり、好ましくない。この例として非同期通信がある。つまり、送受信方式を用いた場合、通信操作の即時性とデータ授受の同期を両立できない。

そこで、本論文では、即時同期を可能にするプロセス間通信機構を提案する。即時同期を用いることにより、通信操作の即時性とデータ授受の同期を両立できる。即時同期を可能にするプロセス間通信機構には、「押し付け」と「奪い取り」の 2 つの機能がある。このうち、奪い取りの機能については、文献 [1] で報告した。また、我々が開発している ***Tender*** (The ENduring operating system for Distributed EnviRonment) オペレーティングシステム [2] で実現されている。そこで、ここでは押し付けの機能を実現する際の課題と対処について述べ、***Tender*** における実装内容について述べる。さらに、即時同期を可能にするプロセス間通信機構の基本性能評価と AP における評価を述べる。基本性能評価については、押し付け、奪い取り、および送受信の処理時間を測定し、各方式の特徴を明らかにする。また、AP における評価については、即時同期を可能にするプロセス間通信機構の有効性を確かめるために、奪い取りと送受信を比較する。

2 即時同期を可能にするプロセス間通信機構

2.1 処理内容

即時同期とは、通信操作の即時性とデータ授受の同期を両立できることである。即時同期を可能にするプロセス間通信の処理内容を以下に述べる。

(1) 送信側プロセスが送信操作を行うと、通信相手プロセスの状況によらず、通信相手プロセスが存在する仮想記憶空間上にデータを届ける。

(2) 受信側プロセスが受信操作を行うと、通信相手プロセスの状況によらず、通信相手プロセスが所有するデータを、受信側プロセスが存在する仮想記憶空間上に持ってくる。

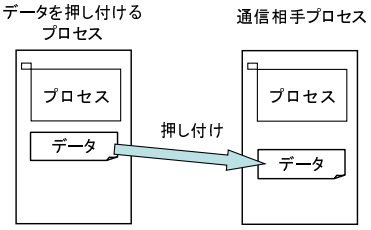


図 1 押し付け

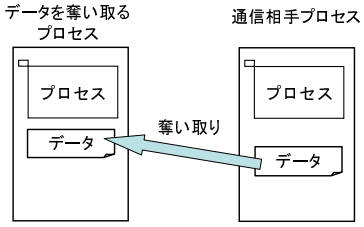


図 2 奪い取り

また、通信相手プロセスの状況に関係なく、直接データをやり取りするという特徴から、即時同期を可能にするプロセス間通信機構における送信操作を「押し付け」、受信操作を「奪い取り」と以降で記述する。なお、必要に応じて、押し付けプロセスに対する通信相手プロセスを「データを押し付けられるプロセス」、奪い取りプロセスに対する通信相手プロセスを「データを奪い取られるプロセス」と記述する。

押し付けと奪い取りの様子を図 1 と図 2 に示す。図 1 に示すように、押し付けはデータを通信相手プロセスに直接渡す操作である。また、図 2 に示すように、奪い取りはデータを通信相手プロセスから直接獲得する操作である。

2.2 特徴

押し付けと奪い取りの機能により、以下に述べるように即時性とデータ授受の同期を両立することができる。

(1) 押し付けプロセスから見た場合、押し付けの機能により、データを通信相手に押し付けた時点で、データの受け渡しの完了が保証される。したがって、操作を行った時点で、データ授受の同期を取ることができる。また、受信完了待ちを行う必要が無いため、即時性を実現できる。

(2) 奪い取りプロセスから見た場合、奪い取りの機能により、奪い取り操作を行った時点で、通信相手プロセスの状況によらず、データを奪い取ることができる。したがって、奪い取りプロセスが操作を

行った時点のデータを奪い取ることができるため、データ授受の同期を取ることができる。また、受信待ちを行う必要が無いため、即時性を実現できる。

実時間処理には、特定のイベントに対して、一定時間内に定められた処理を行うことが要求される。即時性とデータ授受の同期を両立したプロセス間通信機構を実現できれば、プロセス間通信処理に実時間性を持たせることができる。したがって、即時同期を可能にするプロセス間通信は、実時間処理を行うプログラムに有効であると考えられる。

なお、即時性を有するプロセス間通信として、共有メモリを利用する方法がある。しかし、即時同期を可能にするプロセス間通信機構は、共有メモリに比べて、以下に述べる利点がある。

即時同期を可能にするプロセス間通信機構では、送受信方式と同様に、データを一つのかたまり、つまりオブジェクトとしてやり取りする。したがって、即時同期を可能にするプロセス間通信機構では、アドレスを意識しなくてもよい。一方、共有メモリの場合、データ、つまり当該共有メモリにアクセスするためには、アドレスを指定する必要がある。

2.3 実現時の課題と対処

即時同期を可能にするプロセス間通信機構を実現する際、押し付けの機能においては以下の課題がある。

(1) 通信相手プロセスのメモリアクセス

(2) アドレス衝突

それぞれの課題と対処について以降に説明する。

2.3.1 通信相手プロセスのメモリアクセス

押し付け時の通信相手プロセス (データを押し付けられるプロセス) は通信操作を行っていないため、当該メモリ領域が仮想記憶空間上に存在しているか否かの情報を通知される契機を持たない。このため、当該メモリ領域が仮想記憶空間上に存在しない状態にもかかわらず、当該メモリ領域にアクセスしようとする可能性がある。そこで、計数型セマフォ(カウンタセマフォ) を用いて押し付け操作の同期を取ることにした。

なお、カウンタセマフォを利用する場合、通信相手プロセスが押し付け操作待ちの状態になる場合がある。しかし、押し付け操作を行ったプロセスから見た場合は、即時同期が実現されている。したがって、この場合は問題ないと考えられる。

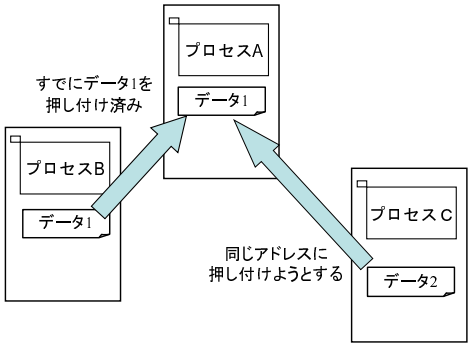


図 3 押し付け時におけるアドレス衝突

2.3.2 アドレス衝突

押し付け先のアドレスに既にデータが存在するにもかかわらず、さらにデータを押し付けようとした場合、アドレスが衝突することが考えられる。たとえば、図 3 のように、複数のプロセスが一つのプロセスに対してデータを押し付ける場合、個々の押し付けプロセスが、通信相手プロセスの存在する仮想記憶空間の同一のアドレスにデータを押し付ける可能性がある。このような問題への対処法として、アドレスが衝突する押し付け操作をエラーとしてユーザプログラムに復帰させる。これにより、たとえば、押し付け操作そのものを取り消すか、アドレスをずらして再び押し付け操作を行うか等、状況に応じて処理を選択できる。したがって、ユーザプログラム処理の自由度が高いという利点がある。

3 *Tender* オペレーティングシステム

3.1 特徴

我々は、プログラム構造に重点を置いた ***Tender*** オペレーティングシステム [2] を開発している。***Tender*** では、OS の操作する対象を資源として、分離し独立化している。資源には、資源名と資源識別子を付与し、資源操作のインタフェースを統一している。さらに、各資源を操作するプログラム部品を資源毎に分離し、各資源の管理表の間の参照関係を禁止している。

このように、資源の分離と独立化を行うことで、資源の事前生成や保留により、資源の生成や削除を伴う処理を高速化している。また、OS の動作や内部状態の理解や把握が容易になり、OS の理解を支援できる。さらに、プログラムを部品化できるため、機能の追加や変更が容易になっている。

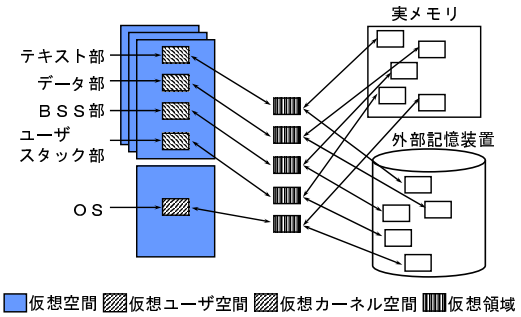


図 4　プロセスとメモリ関連資源の関係

3.2　プロセスとメモリ関連資源の関係

プロセスとメモリ関連資源の関係を図 4 に示す。資源「仮想領域」は、外部記憶装置あるいは実メモリのデータ格納域を仮想化した資源である。資源「仮想空間」は、一定の大きさを持つ仮想アドレスの空間であり、仮想アドレスを実アドレスに変換する変換表に相当する。さらに、資源「仮想領域」を資源「仮想空間」に「貼り付ける」ことにより、プロセッサが仮想アドレスでアクセス可能な資源「仮想カーネル空間」や資源「仮想ユーザ空間」を生成できる。貼り付けるとは、仮想アドレスと実アドレスを対応付けることである。また、仮想アドレスと実アドレスの対応付けを解除することを「剥がす」という。仮想領域を OS の存在する仮想空間に貼り付けた場合は、仮想カーネル空間を生成でき、仮想領域をユーザ用の仮想空間に貼り付けた場合には、仮想ユーザ空間を生成できる。資源「仮想カーネル空間」や、資源「仮想ユーザ空間」は、各々、カーネルモードのみ、カーネルモードとユーザモードの両方でアクセスできるメモリ空間である。

プロセスは、仮想空間上の仮想ユーザ空間を用いて生成される。プロセスのテキスト部、データ部、BSS 部、およびユーザスタック部はそれぞれ別の仮想ユーザ空間に読み込まれた形で存在する。メモリ関連の資源は、プロセスとは独立して存在可能である。このため、メモリ関連の資源を再利用してプロセスの生成と消滅を高速化することができる。

3.3　*Tender* のプロセス間通信機構

Tender では、プロセス間通信機能として、メモリ空間上のデータをプロセス間で移動、複写、共有する機能と、イベント発生時の処理やカウンタセマフォの機能を提供している [1]。***Tender*** では、資源「コンテナ」をやり取りすることでプロセス間通信を行う。***Tender*** におけるプロセス間通信には、資源「コンテナボックス」を介した送受信方式と即時同期を可能にするプロセス間通信におけるコンテナの奪い取りの機能がある。

3.3.1　資源「コンテナ」

資源「コンテナ」は、プロセスが利用可能なある大きさを持つメモリ空間で、仮想空間に存在する仮想ユーザ空間を用いて生成され、ユーザモードまたはカーネルモードでアクセス可能である。プロセスは、この資源「コンテナ」をやり取りすることで、プロセス間通信を行う。プロセス間での資源「コンテナ」を用いた通信の方法として、移動、共有、および複写の 3 つがある。移動は、コンテナを別のプロセスの仮想空間に移動することである。共有は、プロセス間でコンテナを共有することである。これにより、UNIX での共有メモリと同様の使い方ができる。複写は、対象コンテナの内容を新しいコンテナに複写して、新しいコンテナを別のプロセスの仮想空間に貼り付けることである。

また、即時同期を可能にするプロセス間通信の機能のうち、奪い取りの機能をすでに実現している。

3.3.2　資源「コンテナボックス」

資源「コンテナボックス」は、プロセス間での資源「コンテナ」の受け渡しの仲介をする。つまり、コンテナボックスを介したコンテナのやり取りによるプロセス間通信は、送受信方式のプロセス間通信と同等の機能である。具体的には、以下の処理を行う。コンテナボックス管理処理部は、コンテナの送出要求を受けると、送出時に指定されたコンテナボックスのキューの最後に、送出要求のあったコンテナを格納する。コンテナの受信要求があった場合には、コンテナの送出時の貼り付けアドレス要求と受信側の貼り付けアドレス要求を比較し、一致する場合にコンテナを受信プロセスに渡す。

プロセス間通信の様子を図 5 に示す。プロセス A とプロセス B は、コンテナボックスを介したプロセス間通信を行っている。プロセス A がコンテナボックスにコンテナを送出し、コンテナ B がコンテナボックスからコンテナを受信している。また、プロセス C の仮想空間のコンテナからプロセス B の仮想空間のコンテナへの矢印は、コンテナの奪い取りを表している。

3.3.3　資源「イベント」

資源「イベント」は、イベント発生時にそのイベントに割り当てられた処理を実行する。イベントに処理が割り当てられていないときには、カウンタセマフォの役割を果たす。その様子を図 6 に示す。イ

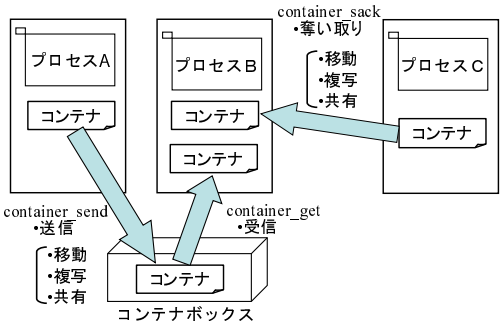


図 5 *Tender* におけるプロセス間通信

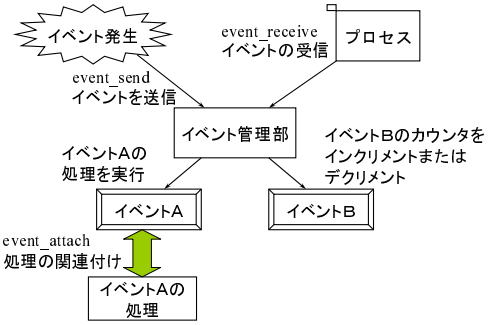


図 6 資源「イベント」

ベント A には、処理が割り当てられている。したがって、イベント管理部にイベント A を送信することでイベント A の発生を通知すると、イベント A に関連付けられた処理が実行される。イベント B には、処理は割り当てられていない。したがって、イベント管理部にイベント B を送信することでイベント B の発生を通知すると、イベント B のカウンタがインクリメントされる。インクリメントされる前の値を見て、イベント待ちのプロセスが存在した場合、そのプロセスを実行可能状態に復帰させる。イベント B の受信をイベント管理部に通知すると、イベント B のカウンタがデクリメントされる。その値を見て、イベント発生待ちか、処理を継続するかを判断する。

4 *Tender* における即時同期を可能にするプロセス間通信機構の実現

4.1 実装内容

Tender では、資源「コンテナ」をやり取りすることでプロセス間通信を行う機構が実現されており、資源「コンテナボックス」を介した送受信方式と即時同期を可能にするプロセス間通信におけるコンテナの奪い取りの機能がある。そこで、資源「コンテナ」に押し付けの機能を追加することで、即時同期を可能にするプロセス間通信の機能を実現した。

コンテナの押し付けの機能について、基本インタフェースを表 1 に示す。基本インタフェースでは、移動、複写、および共有のモードを指定できる。また、表 1 における「イベント」は、2.3.1 項で述べた通信相手プロセスのメモリアクセスに関する対処である。このことについては後述する。

4.2 課題への対処

4.2.1 押し付け時における通信相手プロセスのメモリアクセスへの対処

まだコンテナが押し付けられていないにもかかわらず、通信相手プロセス (コンテナを押し付けられるプロセス) がコンテナにアクセスしようとしてしまう問題への対処法として、2.3.1 項でカウンタセマフォを利用する方法を挙げた。この方法の実現のために、***Tender*** では、資源「イベント」を利用する。資源「イベント」をカウンタセマフォとして利用する場合、押し付け処理の同期を取ることができる。また、イベント発生時の処理実行機能として利用する場合は、プロセスによるソフトウェア割り込みとして利用できる。したがって、イベント発生時に実行される処理が当該コンテナにアクセスするようにプログラムを作成すると、コンテナ押し付けに同期して通信相手プロセスがデータの処理を行うことができる。

4.2.2 押し付け時におけるアドレス衝突への対処

コンテナ押し付け時におけるアドレス衝突の対処法として、2.3.2 項でアドレス衝突時にはエラーとしてユーザプログラムに復帰する方法を挙げた。***Tender*** では、すでに仮想領域が貼り付けられているアドレスに、さらに仮想領域を貼り付けようとするとエラーになる。コンテナは仮想領域で構成されているので、コンテナ押し付け時にアドレス衝突が起きると、当該仮想領域貼り付け時にエラーが返る。したがって、その時点でエラーとしてユーザプログラムに復帰させる。

5 評価

5.1 基本性能評価

5.1.1 測定環境と測定項目

測定には、プロセッサ PentiumIII 550MHz、メモリ 128MB の計算機を使用した。測定では、測定対象カーネルコールの発行前と発行後のハードウェアクロックの値を読み取り、その差分の値を計算した。

測定項目は以下の通りである。それぞれコンテナの大きさを変えて測定した。

表 1 コンテナの押し付けのインタフェース

形式	`int ctainerpush(ctainerid, op, reqaddr, vmid, eventid, paddr)`
引数	`unsigned int ctainerid`: 押し付けるコンテナ識別子 `unsigned int op`: 押し付けのモード (1 : 複写, 2 : 共有, 3 : 移動) `unsigned int reqaddr`: コンテナの押し付け要求アドレス `unsigned int vmid`: コンテナを押し付ける仮想空間識別子 `unsigned int eventid`: 送信するイベント識別子 `unsigned int *paddr`: 実際に押し付けたコンテナが貼り付けられたアドレスを格納する領域へのポインタ
戻り値	成功:コンテナ識別子、失敗:負の値
機能	`ctainerid` で指定したコンテナをモード `op` で仮想空間 `vmid` に押し付ける。 その後、イベント `eventid` にイベントを送信する。

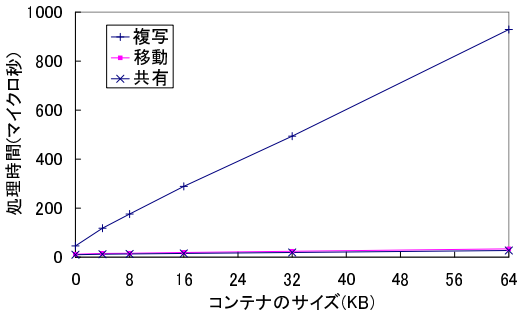


図 7 コンテナ押し付け処理時間

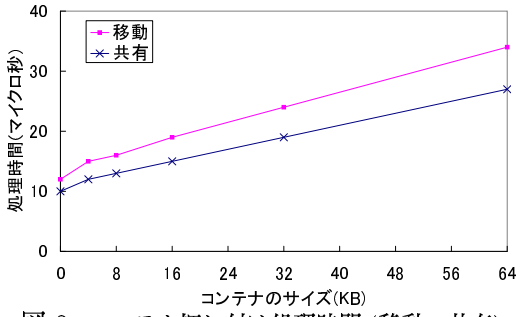


図 8 コンテナ押し付け処理時間 (移動、共有)

(1) コンテナの押し付けの処理時間

(2) コンテナの奪い取りの処理時間

(3) 送受信方式におけるコンテナの送信処理時間と受信処理時間を合算した処理時間

5.1.2 コンテナの押し付けの処理時間

コンテナの押し付けの処理時間を図 7 に示す。また、移動と共有について拡大したものを図 8 に示す。

図 7 と図 8 から、共有、移動、複写の順番で処理時間が短いことがわかる。複写は、新たにコンテナを生成し、そのコンテナに内容を複写する際にメモリコピーが発生するため、共有、移動に比べてかなり処理時間が長くなる。また、移動はコンテナを構成する仮想領域を剥がして貼り付ける必要があるのに対し、共有は仮想領域を貼り付けるだけでよい。したがって、移動に比べて共有のほうが処理時間が短い。

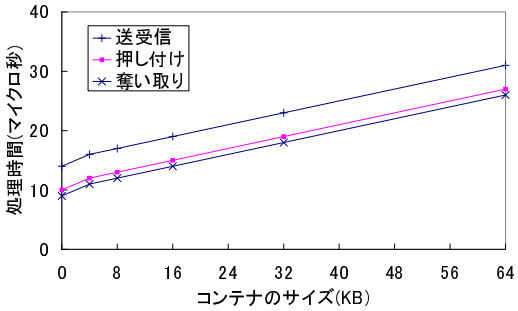


図 9 共有における各方式の処理時間の比較

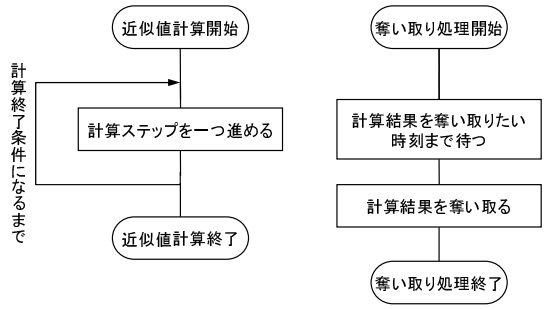
また、コンテナの奪い取りの処理時間、および送受信方式でコンテナの送信処理時間とコンテナの受信処理時間を合算した処理時間においても、同様の結果を得た。

5.1.3 移動、複写、および共有の 3 つのモードにおける各方式の比較

移動、複写、および共有の 3 つのモードそれぞれにおいて、送受信方式、押し付け方式、および奪い取り方式の 3 つの方式を比較した。ただし、送受信方式では、3 つのモードそれぞれについての送信処理時間と受信処理時間を合算した値とした。

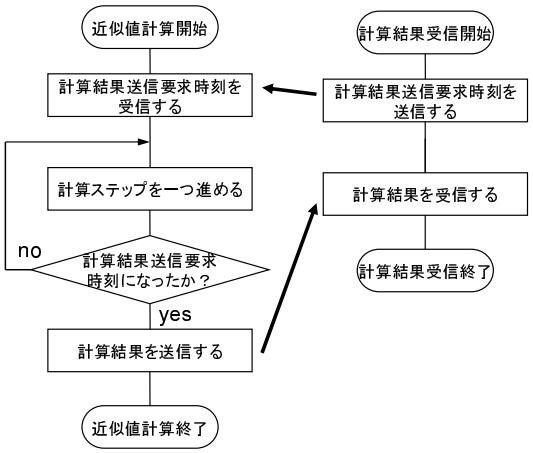
測定した結果、移動、複写、および共有の 3 つの各モードにおいて同様の結果を得た。このうち共有における処理時間の比較を図 9 に示す。

図 9 より、奪い取り方式、押し付け方式、送受信方式の順番で処理時間が短いことがわかる。これは、奪い取り方式を基準に考えた場合、押し付け方式はイベントを送信する処理が加わるためである。送受信方式は、送信、受信の 2 つのカーネルコール処理時間を合算しているため、1 回分のカーネルコール呼び出し、および復帰処理時間が加わるためである。



(a)計算プログラム　(b)計算結果取得プログラム

図 10　奪い取り方式を利用したプログラムのフローチャート



(a)計算プログラム　(b)計算結果取得プログラム

図 11 送受信方式を利用したプログラムのフローチャート

5.2　応用プログラムによる評価

5.2.1　評価プログラムの概要

奪い取りの機能を利用した AP と送受信の機能を利用した AP に関して、以下の点を比較するために評価プログラムを作成した。

(1) プログラム構造の比較

(2) プログラム実行速度の比較

評価プログラムとして、自然対数の底 e の近似解を求めるプログラムを用いた。プロセス間通信については、一方のプロセスが自然対数の底 e の値の計算を行い、もう一方のプロセスがその計算結果を受け取る。

奪い取り方式を用いたプログラムのフローチャートを図 10 に示す。計算プログラム (データを奪われるプログラム) は、通信操作を行う必要がないので、計算を行うだけでよい。また、計算結果取得プログラムは、計算結果が必要になる時間まで待ち、計算結果を奪い取る。

送受信方式を用いたプログラムのフローチャートを図 11 に示す。送受信方式では、まず、計算プログラムが、計算結果取得プログラムから計算結果送信要求時刻を受信する。計算結果取得プログラムは、計算結果送信要求時刻を送信した後、計算結果の受信待ちを行う。また、計算プログラムは、計算が 1 ステップ終了するごとに、計算結果送信要求時刻になったかを判断し、計算結果送信要求時刻になれば送信操作を行う。

奪い取り方式を用いたプログラムと送受信方式を用いたプログラムの実行速度の比較を行うために、それぞれのプログラムの実行時間と計算量の関係を比較した。具体的には、以下の処理を行った。計算結果取得プログラムが、必要な時刻に、計算ステップ数と計算結果を複写モードで繰り返し受け取る。受け取った計算ステップ数を比較した。測定には、5.1.1 項で述べた環境を使用した。

5.2.2　考察

送受信方式に比べ、奪い取り方式は大きく 3 つの利点を持つ。各利点について以下に述べる。

(利点 1) プログラム構造の簡潔化

図 10 に示すように、奪い取り方式の計算プログラム (計算結果を奪い取られるプログラム) は、通信操作を行う必要がないので、通信を意識することなく、自然対数の底 e の計算に専念できる。したがって、図 11 に示す送受信方式の計算プログラムに比べて、奪い取り方式の計算プログラムは、通信操作の分だけ、プログラムを記述する際のプログラムステップ数が減少することになる。これにより、アプリケーションプログラマが簡潔なプログラムを書くことを可能にする。

特に、評価プログラムのように複数回通信を行う場合、送受信方式では、計算プログラムと計算結果取得プログラム双方が、通信の回数を意識する必要がある。このため、たとえば、通信回数を変更する場合、双方のプログラムを修正する必要がある。一方、奪い取り方式の場合は、計算プログラム (計算結果を奪い取られるプログラム) は通信操作を行う必要がないので、プログラムの修正箇所はデータを奪い取るプログラムに限定される。したがって、プログラムの修正が容易になる。

また、一つの計算プログラムに対して、複数の計算結果取得プログラムが計算結果を要求するような場合、送受信方式の計算プログラムは、複数の計算結果取得プログラムに対応するように修正を加える必要がある。一方で、奪い取り方式の計算プログラ

表 2 奪い取り方式と送受信方式における実行時間と計算量 (単位:計算ステップ数)

経過時間 (秒)	10	20	40	100	200	400	800
奪い取り方式	357	713	1426	3579	7221	14735	30935
送受信方式	357	712	1423	3571	7203	14696	30845
計算ステップ数の差	0	1	3	8	18	39	90

ムは、修正を加えることなく、複数の計算結果取得プログラムに対応できる。

(利点 2) 状況に応じたプロセス間通信が可能

図 10 に示すように、奪い取り方式の計算結果取得プログラムは、必要な時刻になってデータを奪い取る。一方、図 11 に示すように、送受信方式は、一度送信要求時刻を計算プログラムに送ってしまうと変更ができなくなる。つまり、奪い取り方式では、奪い取りの時刻を状況に応じて動的に変更することができる。

また、(利点 1) で述べたような一つの計算プログラムに対して、複数の計算結果取得プログラムが計算結果を要求するような場合において、複数の計算結果取得プログラムのプロセス生成または削除を自由に行おうとする場合を考えると、奪い取り方式には次のような利点がある。たとえば、送受信方式において、計算結果取得プログラムが計算結果送信要求時刻を送信した後に、その計算結果取得プログラムのプロセスが削除された場合、計算結果送信要求時刻になったときに計算結果を受け取るプロセスが存在しないために、システムにより、エラーになるか、または不要な送受信データが残る。特に、不要な送受信データが残ってしまうようなシステムの場合は、計算プログラムに送信要求のキャンセル機能を実現しなければならない。しかし、奪い取り方式の場合、計算結果取得プログラムのプロセス生成または削除を自由に行っても、不要な送受信データは残らない。

このように、状況に応じたプロセス間通信を行う場合に、奪い取り方式は有効である。

(利点 3) プログラム実行速度の向上

奪い取り方式と送受信方式における評価プログラムの実行時間と計算量の関係を表 2 に示す。図 11 に示すように、送受信方式の計算プログラムは、1 計算ステップごとに計算結果送信要求時刻になったかを判断するため、その分、処理が遅くなることは避けられない。一方、図 10 に示すように、奪い取り方式では、計算結果取得プログラムが、必要な時刻にデータを奪い取るため、計算プログラム (計算結果を奪い取られるプログラム) が時刻を確認する必要がない。したがって、その分プログラムの速度が向上する。このことは、表 2 で、時間とともに計算ステップ数の差が開いていくという結果からわかる。また、送受信方式に対する奪い取り方式のプログラム実行速度比は時間とともに大きくなる。たとえば 100 秒後は 0.22%速いのに対し、800 秒後は 0.29%速い。

6 おわりに

即時同期を可能にするプロセス間通信機構について述べた。この機構により、通信相手の状況によらず、通信操作の即時性とデータ授受の同期を両立することができる。

即時同期を可能にするプロセス間通信機構を実現する際の課題として、押し付け時における通信相手プロセスのメモリアクセス問題と押し付け時におけるアドレス衝突問題があることを示した。また、***Tender*** 上での即時同期を可能にするプロセス間通信機構の実現法について述べた。さらに、即時同期を可能にするプロセス間通信機構の基本性能評価と AP を用いた評価を行った。AP を用いた評価の結果、送受信方式に比べ、奪い取り方式は、プログラムの構造が簡潔になることと状況に応じたプロセス間通信ができることを明らかにした。さらに、奪い取り方式は、プログラムの処理時間を短縮できることを明らかにした。

今後の課題は、資源「コンテナ」によるネットワークを介したプロセス間通信の実現がある。

参考文献

[1] 田端利宏 , 谷口秀夫: “***Tender*** オペレーティングシステムにおけるプロセス間通信機能の実現と評価” , 情報処理学会研究報告 , Vol.99 , No.32 , pp.95–100 , (1999).

[2] 谷口 秀夫 , 青木 義則 , 後藤 真孝 , 村上 大介 , 田端 利宏: “資源の独立化機構による ***Tender*** オペレーティングシステム” , 情報処理学会論文誌 , Vol.41 , No.12 , pp.3363–3374 (2000)