Vol.2011-OS-118 No.13
2011/7/27

Tender オペレーティングシステムにおける実メモリ交換機能を用いたプロセス間通信機能

難 波 宏 則[†1]　山 内 利 宏[†1]　谷 口 秀 夫[†1]

プロセス間通信は，プロセス間協調処理で多く利用されている．このため，プロセス間通信を高速化することは，システム全体の性能向上に繋がる．***Tender*** におけるプロセス間通信は，コンテナボックスを経由してコンテナをやり取りする方式とコンテナボックスを経由せずプロセス間で直接コンテナをやり取りする方式がある．本稿では，***Tender*** におけるプロセス間通信を高速化する手法を提案する．具体的には，実メモリ交換機能を利用したプロセス間通信を実現する．また，提案手法によるプロセス間通信の送受信処理時間を評価する．

Interprocess communication function using the physical memory exchange function on *Tender* Operating System

Hironori Namba,[†1] Toshihiro Yamauchi[†1]
and Hideo Taniguchi[†1]

Interprocess communication (IPC) is often used to the cooperation processing between processes. Therefore, the performance enhancement of IPC affects the performance enhancement of the whole system. There is two IPC methods in ***Tender*** : one of the method is exchange container through containerbox, and the other method is exchange container between processes without containerbox. In this paper, we propose the technique for the speed-up of the interprocess communication in ***Tender***. Concretely, we archieves IPC using the physical memory exchange function. In addition, we evaluated the sending and receiving processing time of the IPC by the proposed technique.

†1 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

1. はじめに

プロセス間通信は，応用プログラム（AP）のプロセス間協調処理で多く利用されており，処理性能に与える影響は大きい．このため，通信処理を高速化することは，システム全体の性能向上に繋がる．

Psync[1] は，ネットワークの通信において，通信メッセージの順序をコード化するプロセス間通信メカニズムである．また，文献 2) では，マルチスレッド環境におけるメッセージベースのプロセス間通信のコストを省き，スレッド間での共有メモリ通信に置き換えるアーキテクチャの利点や欠点について述べている．文献 3) では，優先度に基づく処理を行うことにより，メッセージ送受信そのものの処理時間は 10–20 %増加したものの，優先度に応じた効率的な CPU の利用を可能にし，実時間性を向上させている．文献 4) は，単一の送信者と複数の受信者モデルにおける実行時間とメモリ消費のオーバヘッドを減少させるメカニズムを提案している．既存の 3 つの待ちなしアルゴリズムを改善することにより，14–70 %のメモリ消費量の減少と 17–66 %の実行時間の減少に成功している．また，文献 5) では，スケーラブルで遅延時間が短い Nemesis という新しいメッセージ伝送サブシステムについて評価している．Nemesis は，他の MPI 実現例と比較して，ゼロでないメッセージの伝送による遅延時間が短く，256KB より大きなメッセージを伝送するための帯域幅が良いと報告されている．

Tender オペレーティングシステム（以降，***Tender*** と呼ぶ）[6] では，資源「コンテナ」と資源「コンテナボックス」を利用してプロセス間通信を実現している[7]．***Tender*** におけるプロセス間通信は，コンテナボックスを経由してコンテナをやり取りする方式とコンテナボックスを経由せずプロセス間で直接コンテナをやり取りする方式がある．コンテナボックスを経由してコンテナをやり取りする方式は，複写，共有，および移動モードがある．一方，プロセス間で直接コンテナをやり取りする方式は，押し付け機能と奪い取り機能がある[8]．

本稿では，***Tender*** におけるプロセス間通信を高速化する手法として，実メモリ交換機能を用いた方式について述べる．また，評価結果を報告する．

2. *Tender* オペレーティングシステム

2.1 資源の分離と独立化

Tender では，OS が制御し管理する対象を資源として細分化し，資源の分離と独立化を行っている．資源の種類ごとに，資源を管理する管理表と，資源を操作するプログラムが

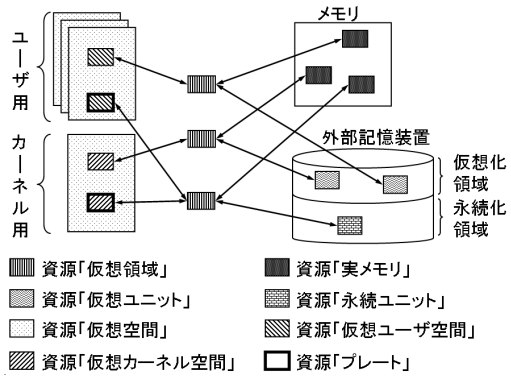

図 1 メモリ関連の資源の関係

存在する．資源は，資源名と資源識別子によって識別される．

2.2 メモリ関連資源

Tender におけるメモリ関連資源の関係について，**図 1** に示し，以下で説明する．

資源 「仮想領域」は，メモリ資源をイメージ化した資源である．メモリ上のデータは，資源「実メモリ」上，もしくは外部記憶装置上の領域に存在する．資源「実メモリ」は主記憶上の領域を表す資源である．資源「仮想ユニット」は，仮想化領域の管理単位である．仮想化領域とは，主記憶上のデータを一時的に保存するための領域であり，既存 OS のスワップ領域を利用して実現している．資源「永続ユニット」は，永続化領域の管理単位である．永続化領域とは，仮想記憶空間上のデータを永続化するために利用する領域であり，既存 OS のファイルシステム領域を利用して実現している．

資源「仮想空間」は，特定のアドレス領域を持つ仮想的な空間である．仮想アドレスから実アドレスへのアドレス変換表に相当する．資源「仮想ユーザ空間」は，ユーザ用の仮想空間に存在するデータを表す．資源「仮想ユーザ空間」は，OS や AP が資源「仮想領域」で管理されるデータにアクセスするために，資源「仮想領域」をユーザ用の資源「仮想空間」に貼り付けて生成する.「貼り付ける」とは，仮想アドレスを実アドレスに対応付けることである．一方，仮想アドレスと実アドレスの対応付け解除を「剥がし」と呼ぶ．資源「仮想カーネル空間」は，カーネル用の仮想空間に存在するデータを表し，資源「仮想領域」をカーネル用の資源「仮想空間」に貼り付けて生成する．

表 1 ***Tender*** におけるプロセス間通信機能

依頼元	依頼先	操作方法	処理モード
プロセス	コンテナボックス	送信	複写
			共有
			移動
		受信	複写
			共有
			移動
	プロセス	押し付け	複写
			共有
			移動
		奪い取り	複写
			共有
			移動

2.3 プロセス間通信機能

Tender では，資源「コンテナ」と資源「コンテナボックス」を利用してプロセス間通信を実現している[7)8)]．

資源「コンテナ」は，プロセス間でのコンテナのやり取りや共有により，プロセス間通信を実現する機能を持つ．プロセスは，この資源「コンテナ」をやり取りすることで，プロセス間通信を実現することが出来る．資源「コンテナボックス」は，プロセス間での資源「コンテナ」の受け渡しを仲介する．

プロセス間通信は，コンテナボックスを経由してコンテナをやり取りする方式とプロセス間で直接コンテナをやり取りする方式がある．操作方法と処理モードを含めたプロセス間通信の機能を**表 1** に示す．プロセスがコンテナボックスに対して行う操作方法として，送信と受信がある．また，プロセスがプロセスに対して行う操作方法として，押し付けと奪い取りがある．押し付けは，依頼元プロセスが指定したコンテナを強制的に依頼先プロセスの指定したメモリ空間上に貼り付ける．奪い取りは，依頼元プロセスが指定した依頼先プロセスのコンテナを強制的に依頼元プロセスのメモリ空間上に取り戻す．これらの機能は，***Tender*** 固有の機能である．

上記の各操作方法に対し，3 種類の処理モードがある．複写は，データが新しいコンテナに複写される．共有は，送受信プロセス間でデータを共有する．移動は，データを含むコンテナの所有権が移動する．

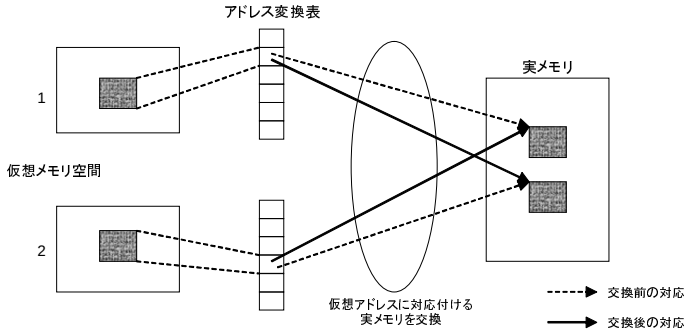

図 2 実メモリ交換機能

3. 実メモリ交換機能を用いたプロセス間通信

3.1 実メモリ交換機能

実メモリ交換機能[9] は，仮想空間上の領域に対応する実メモリを交換し，アドレス変換表を更新することにより，2 つの領域間で複写レスでのデータ授受を実現する機能である．実メモリ交換機能の様子を**図 2** に示す．仮想空間上の領域は，アドレス変換表によって仮想アドレスと実アドレスが対応付けられている．実メモリ交換機能では，仮想空間上の 2 つの領域について，それぞれの領域に対応するアドレス変換表を参照して各ページの仮想アドレスに対応する実アドレスを交換し，アドレス変換表を更新する．これにより，2 つの領域間でのメモリ間データ授受を実現している．このとき，複数ページ分の実メモリに対して交換を行うことが可能である．また，2 つの領域は異なる仮想空間上に存在していても良い．

3.2 設 計 方 針

文献 9) において，実メモリ交換機能は，メモリ間データ授受による送受信処理が高速であることが報告されている．そこで，実メモリ交換機能をプロセス間通信で使用することにより，高速なプロセス間通信を実現する．具体的には，プロセス間で直接コンテナをやり取りする操作方法に交換を実現し，コンテナボックスを経由してコンテナをやり取りする処理モードに交換モードを実現する．以降では，コンテナボックスを経由してコンテナの送受信を行う交換モードについて述べる．

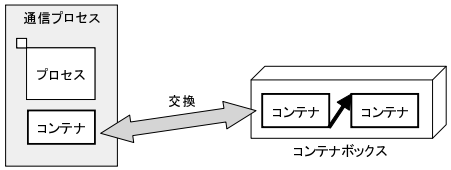

図 3 交換モードによるプロセス間通信

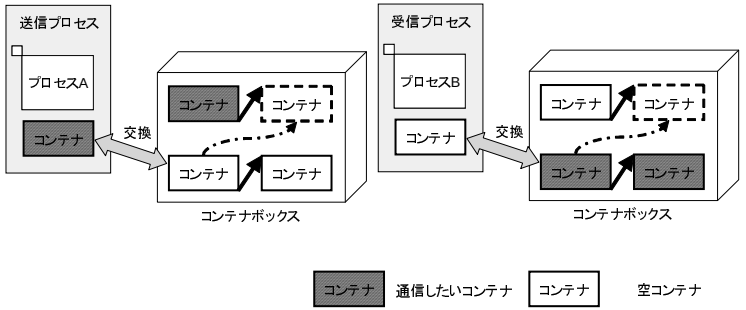

図 4 交換モードによる送受信

3.3 基 本 方 式

交換モードによるプロセス間通信の様子を**図 3** に示す．交換モードによる送受信は，プロセスが存在する仮想空間上のコンテナに割り当てられた実メモリ領域とコンテナボックス内に用意したコンテナに割り当てられた実メモリ領域を交換する．

交換モードによる送信と受信の様子を**図 4** に示す．コンテナボックスには，送信時の交換処理で使用する空コンテナを格納するキューと，受信時の交換処理で使用するコンテナを格納するキューがある．送信プロセスは，自仮想空間上のコンテナの実メモリとコンテナボックス内に用意された空コンテナの実メモリを交換する．交換処理後，実メモリ交換を行ったコンテナを受信時の交換処理で使用するキューの末尾に格納する．受信プロセスは，自仮想空間上の空コンテナの実メモリと，コンテナボックス内に格納されたコンテナの実メモリを交換することにより，コンテナを受信する．交換処理後，実メモリ交換を行った空コンテナを送信時の交換処理で使用するキューの末尾に格納する．

表 2 送信インタフェース

形式	int ctainersend(ctainerid, ctainerbxid, reqaddr, op, size)
引数	unsigned int ctainerid：送出するコンテナの資源識別子 unsigned int ctainerbxid：送出先のコンテナボックス資源識別子 unsigned int reqaddr：貼り付け希望アドレス unsigned int op：送信のモード指定 （1：COPY，2：SHARE，3：MOVE，4：SWAP） unsigned int *size：送出するコンテナのサイズ (単位：バイト)
戻り値	成功：コンテナ識別子，失敗：負の値
機能	コンテナをモード op で，コンテナボックスに貼り付け希望アドレスを 指定して送信する

3.4 課題と対処

交換モードを利用して通信を行う場合，空コンテナの管理方法が課題となる．具体的には，空コンテナの生成契機と削除契機である．空コンテナの生成契機として，コンテナボックス生成時と送信処理要求時がある．具体的な処理としては，送信時の交換処理により，空コンテナの数が減少した時に生成を行う．また，コンテナボックス生成時，空コンテナは初期値として設定した個数生成する．空コンテナの削除契機として，コンテナボックス削除時と受信処理要求時がある．具体的な処理としては，受信時の交換処理により，コンテナボックスに格納された空コンテナの数が設定した上限値より多くなった場合に削除を行う．また，コンテナボックス削除時に削除する．

なお，これらの処理は，コンテナの大きさ単位毎に行う必要があることは言うまでもない．

3.5 提供インタフェース

交換モード追加後の送信インタフェースを**表 2** に示す．ctainersend() の引数 op に交換モードを追加する．また，送信側プロセスが存在する仮想空間上のコンテナのサイズとコンテナボックス内に用意している空コンテナのサイズは一致している必要があるため，新たに引数として size を追加する．

交換モード追加後の受信インタフェースを**表 3** に示す．ctainerreceive() の引数 op に交換モードを追加する．交換による受信の場合，コンテナボックス内に格納されているコンテナのサイズに合わせたサイズのコンテナを受信側プロセスが存在する仮想空間上に用意しておく必要がある．このため，受信プロセスはあらかじめコンテナを生成しておき，コンテナ生成後，交換を実行する．

表 3 受信インタフェース

形式	int ctainerreceive(ctainerid, ctainerbxid, reqaddr, op, waittime, paddr, size)
引数	unsigned int ctainerid：受信するコンテナ識別子（0 の時は先頭のコンテナ） unsigned int ctainerbxid：受信元のコンテナボックス資源識別子 unsigned int reqaddr：コンテナの受信アドレスを指定 unsigned int op：受信のモード指定 （1：COPY，2：SHARE，3：MOVE，4：SWAP） unsigned int waittime：コンテナボックスにコンテナが無かった場合の タイムアウト時間を指定 unsigned int *paddr：実際に受信したコンテナが貼り付けられたアドレスを 格納する変数へのポインタ unsigned int *size：受信したコンテナの大きさを格納する変数へのポインタ
戻り値	成功：コンテナ識別子，失敗：負の値
機能	コンテナをモード op で，コンテナボックスから受信する

4. 評　　価

4.1 評 価 環 境

本機能を実現した ***Tender*** を Pentium4 2.4GHz の計算機上で走行させ，性能を評価する．以降の評価では，操作方法が送信，受信，押し付け，および交換の場合の通信処理の平均処理時間を用いる．また，交換の場合，空コンテナの生成と削除を伴う処理も評価する．

4.2 コンテナボックスを経由するプロセス間通信の評価

4.2.1 送信処理時間

コンテナボックスを経由するプロセス間通信において，各処理モードにおけるコンテナ送信処理時間を**図 5** に示す．図 5 には，各処理モードにおける処理時間とその内訳を示している．また，横軸の生成は，単にコンテナを生成する処理の処理時間を示している．各処理モードにおける処理時間は,「処理名 (全体)」で示し，各処理の内訳をそれぞれ示している．内訳において，コンテナ送出とは，コンテナをコンテナボックスに送出する処理を示しており，コンテナ生成とは，コンテナを生成する処理を示している．なお，図 5 において，空コンテナの生成を行う場合の交換の処理時間は，交換 (生成) と表記する．図 5 より，以下のことがわかる．

(1) 交換は，共有より遅い．交換の処理時間は，コンテナのサイズが 4KB のとき，共有の約 1.5 倍である．これは，交換では，交換対象となるコンテナを構成する仮想領域に対応する実メモリを交換しているのに対し，共有では，共有したいコンテナの情報

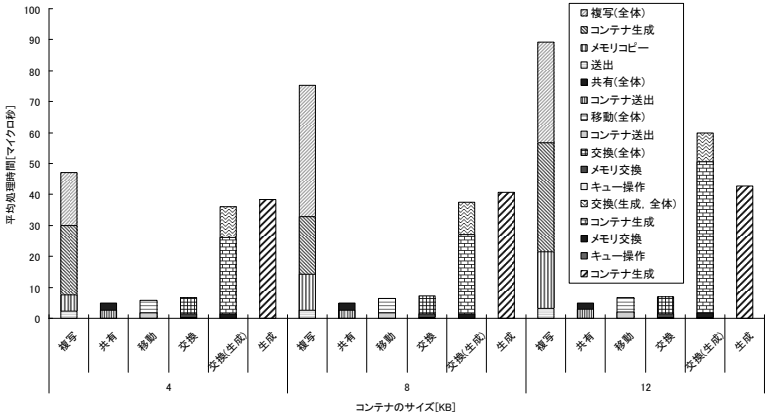

図 5 各処理モードにおけるコンテナ送信処理時間

だけをコンテナボックスに送信するためである．

(2) 交換は，移動より遅い．交換の処理時間は，コンテナのサイズが 4KB のとき，移動の約 1.2 倍である．これは，交換では，交換対象となるコンテナを構成する仮想領域に対応する実メモリを交換しているのに対し，移動では仮想空間からコンテナを剥がすだけでコンテナボックスに送信できるためである．

(3) 交換は，複写より速い．交換の処理時間は，コンテナのサイズが 4KB のとき，複写の約 1/7 である．

(4) 交換 (生成) は，複写より遅い．これは，コンテナ生成処理がデータ複写処理よりも多くの処理時間を要することを示している．

(5) 交換において，コンテナのサイズが 4KB のとき，メモリ交換の処理時間は交換処理全体の約 20 %を占めており，キュー操作の処理時間は交換処理全体の約 2 %を占めている．このことから，メモリ交換とキュー操作が交換処理の処理時間に与える影響は少ない．

(6) 交換 (生成) において，コンテナのサイズが 4KB のとき，コンテナ生成処理の処理時間は交換処理全体の約 70 %を占めており，メモリ交換の処理時間は交換処理全体の約 4 %を占めている．このことから，コンテナ生成が交換処理の処理時間に与える影響は非常に大きい．

(7) 共有，移動，および交換の処理時間は，コンテナサイズの大小に関わらずほとんど一定である．

(8) 生成の処理時間は，コンテナサイズの増加に伴い増加している．

(9) 交換 (生成) の処理時間は，コンテナサイズの増加に伴い増加している．これは，コンテナ生成の処理時間がコンテナサイズの影響を受けているためである．

図 5 より，交換は，共有や移動と同等の処理時間で送信処理を行える．しかし，交換処理中に空コンテナを生成する場合，非常に多くの処理時間を要する．具体的には，交換処理全体の約 70 %が空コンテナ生成処理の処理時間となる．また，交換におけるメモリ交換処理は，共有や移動におけるコンテナ送出処理と比べて高速である．これは，実メモリ交換処理がコンテナをコンテナボックスに送出する処理よりも高速であることを示している．さらに，交換は，コンテナサイズの増加による影響をほとんど受けていない．これは，実メモリ交換では，アドレス変換表の対応を変えるだけで目的のメモリ領域を参照可能になるため，メモリサイズの増加による影響が小さいためである．

4.2.2 受信処理時間

コンテナボックスを経由するプロセス間通信において，各処理モードにおけるコンテナ受信処理時間を**図 6** に示す．図 6 には，各処理モードにおける処理時間とその内訳を示している．また，横軸の削除は，単にコンテナを削除する処理の処理時間を示している．各処理モードにおける処理時間は,「処理名 (全体)」で示し，各処理の内訳をそれぞれ示している．内訳において，コンテナ情報取得とは，受信時にコンテナボックスからコンテナの情報を取得する処理であり，仮想ユーザ空間生成とは，受信するコンテナを貼り付けるための仮想ユーザ空間を生成する処理である．コンテナ生成は，複写モードの処理中で発生するコンテナ生成処理を示しており，コンテナ削除は，交換モードの処理中で発生する空コンテナ削除処理のを示している．なお，図 6 において，空コンテナの削除を行う場合の交換の処理時間は，交換 (削除) と表記する．図 6 より，以下のことがわかる．

(1) 交換の処理時間は，全ての処理モードの中で最も短く，コンテナのサイズが 4KB のとき，共有の約 3/5 であり，移動の約 3/5 である．送信処理の場合と異なり，交換が共有と移動より速いのは，受信時に仮想ユーザ空間を生成する処理を行わないためである．

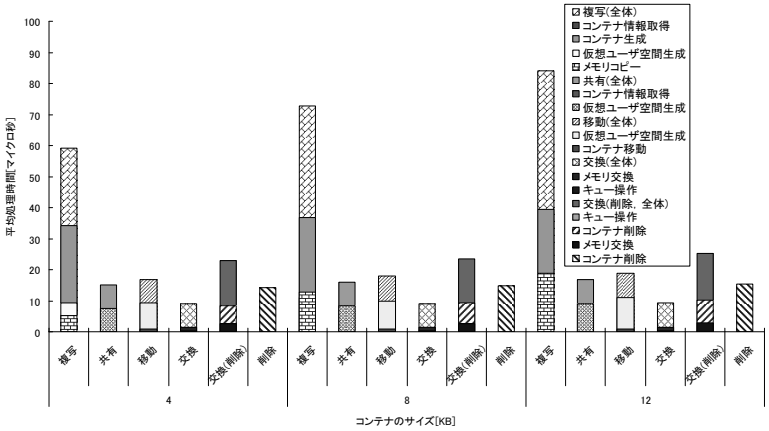

図 6 各処理モードにおけるコンテナ受信処理時間

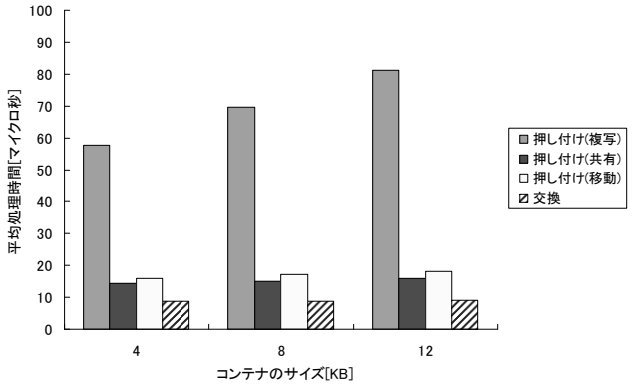

図 7 押し付け (各処理モード) と交換の通信処理時間

(2) 交換 (削除有) は，共有や移動より遅く，複写より速い．これは，コンテナ削除処理がデータ複写処理よりも少ない処理時間を要することを示している．

(3) 交換において，コンテナのサイズが 4KB のとき，メモリ交換処理の処理時間は交換処理全体の処理時間の約 13 %を占めており，キュー操作の処理時間は交換処理全体の約 2 %を占めている．このことから，メモリ交換とキュー操作が交換処理の処理時間に与える影響は少ない．

(4) 交換 (削除) において，コンテナのサイズが 4KB のとき，コンテナ削除処理の処理時間は交換処理全体の処理時間の約 36 %を占めており，メモリ交換の処理時間は交換処理全体の処理時間の約 10 %を占めている．このことから，コンテナ削除が交換処理の処理時間に与える影響は大きい．

(5) 共有，移動，および交換の処理時間は，コンテナサイズの大小に関わらずほとんど一定である．

(6) 削除の処理時間は，コンテナサイズの大小に関わらずほとんど一定である．

(7) 交換 (削除) の処理時間は，コンテナサイズの大小に関わらずほとんど一定である．これは，コンテナ削除の処理時間がコンテナサイズの増加による影響をほとんど受けていないためである．

図 6 より，交換は，全ての処理の中で最も短い処理時間で受信処理を行える．具体的には，交換は，コンテナ受信時に仮想ユーザ空間を生成しないため，共有や移動と比べて処理時間が短い．また，交換の処理時間は，コンテナサイズの増加による影響がほとんどない．これは，実メモリ交換では，アドレス変換表の対応を変えるだけで目的のメモリ領域を参照可能になるため，コンテナサイズの増加による影響が小さいためである．

4.3 コンテナボックスを経由しないプロセス間通信の評価

コンテナボックスを経由しないプロセス間通信において，操作方法が押し付け (処理モードは複写，共有，移動) の場合と交換の場合の通信処理時間を**図 7** に示す．図 7 より，以下のことがわかる．

(1) 交換の処理時間は，全ての処理モードによる押し付けと比べて最も短く，コンテナサイズが 4KB のとき，押し付け (共有) の約 4/7 であり，押し付け (移動) の約 1/2 である．

(2) 交換の処理時間は，コンテナサイズの大小に関わらずほとんど一定である．

図 7 より，交換は，全ての処理モードによる押し付けと比べて最も処理時間が短い．これ

は，押し付けでは，依頼先プロセスが存在する仮想空間上に仮想ユーザ空間を生成し，押し付け対象コンテナを貼り付ける．その際，複写ではメモリコピー，移動では仮想領域の剥がし処理が発生する．しかし，交換では，通信を行うプロセスはあらかじめコンテナを生成しているため，通信対象コンテナの貼り付けが発生しない．このため，コンテナボックスを経由しないプロセス間通信では，交換が最も高速となる．

次に，送信と受信を合わせて一つの通信処理として考える．図 5，図 6，および図 7 を比べると，交換は，依頼先に関わらず 10 マイクロ秒未満の処理時間である．共有と移動の処理時間は，図 5 では 10 マイクロ秒未満だが，図 6 では 15 マイクロ秒を超過している．また，図 7 において，共有と移動の処理時間はそれぞれ 10 マイクロ秒を超過している．以上より，送受信処理において，交換は，他の処理モードと比べて短時間でコンテナをやり取りすることができ，高速なプロセス間通信を実現している．

5. おわりに

Tender のプロセス間通信において，実メモリ交換機能を利用したプロセス間通信を実現した．具体的には，コンテナボックスを経由するプロセス間通信の処理モードに「交換モード」を実現し，コンテナボックスを経由しないプロセス間通信の操作方法に「交換」を実現した．実メモリ交換機能は，仮想空間上の領域に対応する実メモリを交換し，アドレス変換表を更新することにより，2 つの領域間で複写レスでのデータ授受を実現する．この機能により，コンテナの複写や移動を行う必要がなくなり，共有後のデータ書き換えといった問題を考慮する必要がなくなる．

交換モードを実現するためには，コンテナボックスに格納されてあるコンテナの数を制御する必要がある．具体的な課題として，空コンテナの生成契機と削除契機がある．空コンテナは，コンテナボックス生成時と送信処理要求時に設定した下限値を下回る場合に生成し，コンテナボックス削除時と受信処理要求時に設定した上限値を上回る場合に削除する．

送受信処理の処理時間の評価により，送信では共有が最も高速であることを示し，受信では交換が最も高速であることを示した．コンテナボックスを経由しないプロセス間通信では，交換が最も高速であることを示した．また，通信依頼先に関わらず，交換の処理時間は 10 マイクロ秒未満となることを示した．

謝辞 ***Tender*** のプロセス間通信機能の実装にご協力いただいた岡山大学 大学院自然科学研究科 (現在，株式会社 STNet) の川江純平氏に感謝します．

参 考 文 献

1) PETERSON, L.L., HOLZ, N.C. and SCHLICHTING, R.D.: Preserving and using context information in interprocess communication, pp.217–246 (1989).
2) Pham, T. Q., Garg, P. K. and Laboratories, H.-P.: On Migrating a Distributed Application to a Multi-Threaded Environment, *USENIX Summer 1992 Technical Conference* (1992).
3) kitayama, T., Tkuda, H. and Nakajima, T.: RT-IPC Extension for Real-Time Mach, *Proc. of the USENIX Symposium on Microkernels and Other kernel Architectures*, pp.91–104 (1993).
4) Huang, H., Pillai, P. and Shin, K.G.: Improving Wait-Free Algorithms for Interprocess Communication in Embedded Real-Time Systems, *Proc. of the USENIX annual Technical Conf.*, pp.303–316 (2002).
5) Buntinas, D., Mercier, G. and Gropp., W.: The Design and Evaluation of Nemesis, a Scalable Low-Latency Message-Passing Communication Subsystem, *In International Symposium on Cluster Computing and the Grid* (2006).
6) 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).
7) 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).
8) 福冨和弘，田端利宏，谷口秀夫：***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，情報処理学会研究報告，No.2003-OS-93, pp.25–32 (2003).
9) 門　直史，山内利宏，谷口秀夫：実メモリ交換機能によるゼロコピー通信処理の実現と評価，電子情報通信学会論文誌, Vol.J93–D, No.11, pp.2380–2389 (2010).