Vol.2014-OS-130 No.22
2014/7/29

Tender におけるプロセス間通信データ域に特化したプロセス間通信の設計

川野 直樹[1]　山内 利宏[1]　谷口 秀夫[1]

概要：協調処理では，プロセス間通信の性能が処理性能に大きな影響を与える．このため，プロセス間通信の高速化が必要である．本稿では，***Tender*** オペレーティングシステムにおいて，プロセス間通信に特化した領域（プロセス間通信データ域）を実現し，この領域を利用したプロセス間通信の設計と実現方式について述べる．プロセス間通信データ域とは，プロセス間の複写レスでのデータ授受機能を支援する領域である．プロセスは，この領域を利用して通信することにより，複写レスなデータ授受と仮想アドレスから実アドレスへの変換の高速化を実現し，プロセス間通信を高速化できる．また，評価では，***Tender*** オペレーティングシステムの既存のプロセス間通信との処理時間の比較結果を報告する．

1. はじめに

ソフトウェアで実現する機能が多くなり規模が増大している．このため，大規模なソフトウェアの開発では，機能のモジュール化と分離化が行なわれている．これにともない，応用プログラム（以降，AP）間でのデータの共有や交換が多発するようになっている [1]．このため，協調処理では，プロセス間通信性能が AP の処理性能に大きな影響を与える．したがって，プロセス間通信の高速化が必要である．

現在のオペレーティングシステム（以降，OS）のプロセス間通信は，通信するデータサイズが増加するにつれて処理時間が増加する [2]．また，データコピーが発生するプロセス間通信では，処理時間の増加率が大きい [3]．このため，データコピーが発生しない共有メモリを利用し，プロセス間通信によるオーバヘッドを削減する研究が行われている [4]．

マイクロカーネル OS では，プロセス間通信が頻繁に発生し，プロセス間通信によるオーバヘッドがシステムの性能を低下させている [5] [6]．この問題に対し，マイクロカーネル OS である ***AnT*** オペレーティングシステムでは，メモリ空間上にコア間通信データ域（ICA : Inter-core Communication Area）を配置し，対処している．ICA とは，プロセス間の複写レスでのデータ授受機能を支援する領域である．プロセスは，この領域を利用して通信することにより，複写レスなデータ授受と論実アドレス変換の高速化を実現し，プロセス間通信を高速化している [7]．そこで，モノリシックカーネルの OS でも，ICA を実現しプロセス間通信に利用することで，プロセス間通信を高速化できると考えられる．

本稿では，モノリシックカーネルの OS である ***Tender*** オペレーティングシステム [8]（以降，***Tender***）において，ICA と同等なプロセス間通信データ域（IPCA : Inter-Process Communication Area）を実現し，この領域を利用したプロセス間通信の設計と実現方式について述べる．評価では，プロセス間通信データ域を利用したプロセス間通信と ***Tender*** の既存のプロセス間通信との処理時間を比較結果を報告する．

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

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 は，OS が制御し，管理する対象を資源と呼び，資源の分離と独立化を行っている．資源は，資源の種類ごとの操作インタフェース（以降，資源操作のインタフェース）と個々の資源に関するデータを管理するテーブル（以降，資源管理表）を提供している．また，生成した個々の資源に対して，**図 1** に示すように資源名と資源識別子を付与し，識別している．資源名は，場所名，種類名，および固有名からなる文字列であり，資源識別子は，資源の場所，種類，および同一種類内の通番を情報として有する数値である．資源名は，AP が指定し，資源識別子は，OS が資源の生成時に決定する．

また，***Tender*** は，資源を操作するプログラムを必ず資源

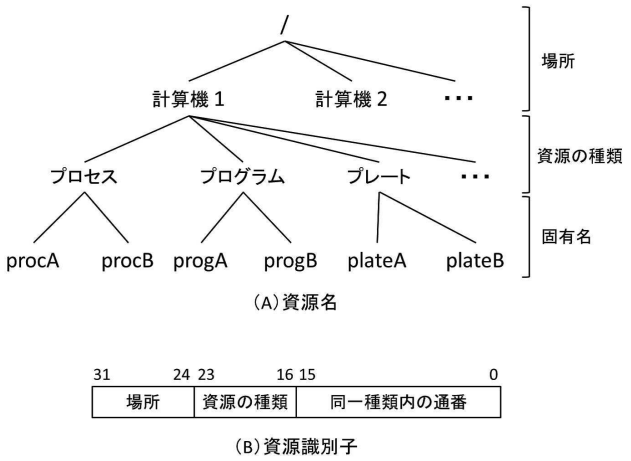

図 1 資源名と資源識別子

インタフェース制御（RIC : Resource Interface Controller）を介して呼び出す．このため，既存 OS のように直接 OS 機能を呼び出す場合に比べ，資源操作のオーバヘッドが増加する．この問題に対し，***Tender*** では，資源の分離と独立化の特徴を活かした資源の再利用機能により，資源操作の高速化を図っている．

2.2 メモリ関連資源

図 2 に ***Tender*** のメモリ関連資源の関係を示す．資源「仮想領域」は，外部記憶装置あるいは外部記憶装置と実メモリのデータ格納域を仮想化した資源である．資源「実メモリ」は，主記憶上の領域を表す資源である．資源「仮想空間」は，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（ページテーブルなど）に相当する．資源「仮想カーネル空間」は，資源「仮想領域」をカーネル用の資源「仮想空間」に貼り付けることで生成される．「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「仮想ユーザ空間」は，資源「仮想領域」をユーザ用の資源「仮想空間」に貼り付けることで生成される．資源「仮想カーネル空間」と資源「仮想ユーザ空間」は，プロセッサが仮想アドレスでアクセスできる空間であり，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．資源「仮想カーネル空間」には，OS があり，資源「仮想ユーザ空間」には，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がある．

2.3 既存のプロセス間通信手法

Tender は，資源「コンテナ」を利用してプロセス間通信を実現している [9]．資源「コンテナ」とは，プロセスが利用可能なある大きさのメモリ空間を保持し，プロセス間通信を実現する機能を持つ．プロセスは，プロセス間で資

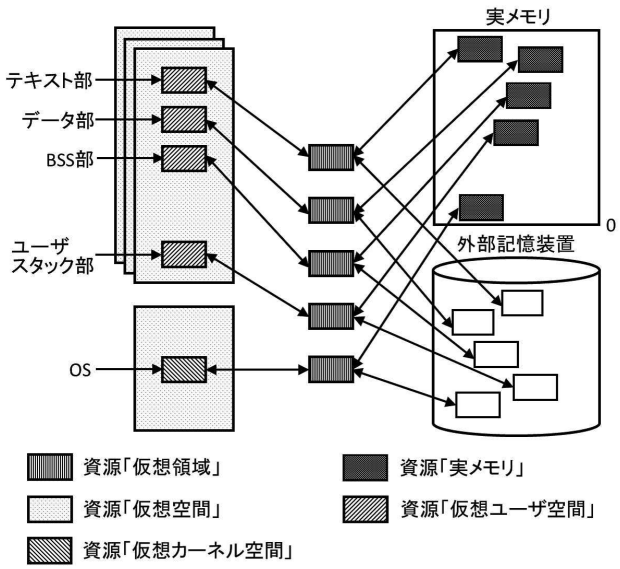

図 2 ***Tender*** のメモリ関連資源の関係

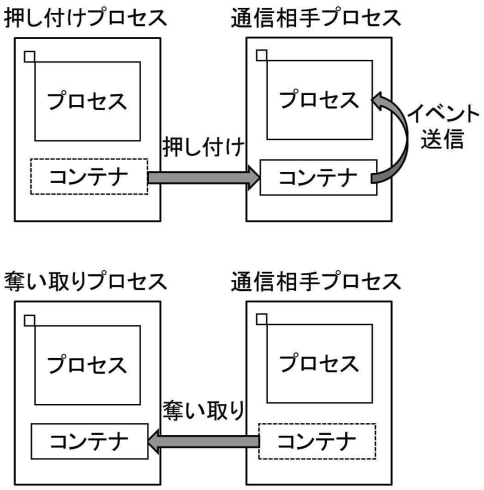

図 3 コンテナの押し付けと奪い取り

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できる．また，資源「コンテナ」は，資源「仮想領域」，資源「実メモリ」，および資源「仮想ユーザ空間」（または，資源「仮想カーネル空間」）の三つの資源から構成されている．

Tender では，資源「コンテナ」を利用して，即時同期が可能なプロセス間通信である押し付けと奪い取りを実現している [10]．即時同期とは，通信の即時性とデータ授受の同期を両立できることである．**図 3** にコンテナの押し付けと奪い取りの様子を示す．コンテナの押し付けは，押し付けプロセスがコンテナの送信操作を行うと，通信相手プロセスの状態によらず，通信相手プロセスが存在する仮想空間にコンテナを送信することである．このときに，資源「イベント」を利用して，通信相手プロセスにコンテナを受信した契機を知らせる．コンテナの奪い取りは，奪い取りプロセスがコンテナの受信操作を行うと，通信相手プロセ

スの状態によらず，通信相手プロセスが存在する仮想空間からコンテナを受信することである．また，コンテナの押し付けと奪い取りでは，移動，共有，および複写の三つの処理モードがある．移動は，送信プロセスの仮想空間に存在するコンテナを受信プロセスの仮想空間へ移動させることである．共有は，送信プロセスの仮想空間に存在するコンテナを受信プロセスの仮想空間に貼り付け，同じメモリ空間を共有することである．複写は，送信要求のあったコンテナのデータを新しいコンテナに複写し，受信プロセスの仮想空間に貼り付けることである．

3. プロセス間通信データ域

プロセス間通信を高速化するために，メモリ空間上にプロセス間の複写レスでのデータ授受機能を支援する領域を用意する．この領域をプロセス間通信データ域 (IPCA : Inter-Process Communication Area) と呼ぶ．IPCA には，以下の三つの特徴がある．

（特徴 1） ページ（4 KB）を単位とし，n ページ分の領域の確保と解放

（特徴 2） 確保した領域（n ページ）の実メモリ連続の保証

（特徴 3） 2 仮想空間の間での領域の貼り替え

（特徴 2）より，仮想アドレスから実アドレスへの変換の高速化を実現する．IPCA から確保したメモリは，実メモリ連続を保証している．また，IPCA からメモリを確保する際，実メモリ空間上の IPCA の開始アドレスからのオフセットと仮想空間上の IPCA の開始アドレスからのオフセットが一致するように確保する．これにより，仮想アドレスに対応する実アドレスを一意に決定できる．したがって，仮想アドレスに対応する実アドレスをアドレス変換表の参照なしに計算のみで求めることができる．計算式は，以下の通りである．

実アドレス = ((仮想アドレス
−仮想空間上の IPCA の開始アドレス)
+実メモリ空間上の IPCA の開始アドレス)

（特徴 3）より，複写レスなデータ授受を実現する．IPCA を利用したプロセス間でのデータ授受は，授受するデータを格納した IPCA のメモリをデータ授受元プロセスの仮想空間から剥がし，データ授受先プロセスの仮想空間に貼り付けることで行われる．これらの操作をまとめて IPCA の貼り替えと呼ぶ．

4. プロセス間通信データ域に特化したプロセス間通信の設計

4.1 設計方針

Tender のメモリ空間上に IPCA を配置し，IPCA を利用した複写レスなプロセス間通信を実現する．また，メモリ空間上に配置した IPCA を制御し，管理するために，IPCA を資源化する．これを資源「仮想 IPCA 空間」と呼ぶ．資源「仮想 IPCA 空間」の資源操作のインタフェースと処理を IPCA の特徴を利用して，資源操作の呼び出しによるオーバヘッドを削減するように設計する．これにより，高速なプロセス間通信の実現を目指す．

4.2 課題

Tender で IPCA に特化したプロセス間通信を実現するための課題について述べる．***Tender*** のメモリ空間上に IPCA を実現するためには，以下の課題がある．

（課題 1） メモリ空間上における IPCA の配置

メモリ空間上に IPCA を配置するためには，既存のメモリ空間の構成を変更する必要がある．IPCA をメモリ空間上に配置する際，メモリ空間上のどの位置に配置するのか検討する必要がある．

メモリ空間上に配置した IPCA を制御し，管理する機構が必要である．このため，***Tender*** では，資源「仮想 IPCA 空間」を実現する．資源「仮想 IPCA 空間」を実現するためには，以下の四つの課題がある．

（課題 2） IPCA の管理方法

メモリ空間上に配置した IPCA を適切に利用するためには，IPCA の使用状況やどの仮想空間に IPCA の領域が対応付けられているかを管理する必要がある．

（課題 3） IPCA のメモリの確保と解放

プロセス間通信を高速化するためには，プロセス間通信に利用する領域の確保と解放を高速化する必要がある．このため，IPCA の特徴をどのように利用してプロセス間通信に利用する領域の確保と解放を高速化するか検討する．

（課題 4） IPCA の貼り付けと剥がし

IPCA を利用することで仮想アドレスに対応する実アドレスを計算により求めることができる．この特徴を利用すると IPCA の貼り付けと剥がしを高速化できる．このため，既存の貼り付けと剥がしを IPCA に対応できるようにする必要がある．

（課題 5） 既存のメモリ管理の変更

資源「仮想 IPCA 空間」がメモリ空間上に配置した IPCA を管理する．このため，既存のメモリ管理が IPCA を操作できないようにするために，IPCA を管理の対象範囲から外すように変更する必要がある．

資源「仮想 IPCA 空間」が提供する資源操作のインタフェースを利用して，即時同期が可能であるプロセス間通信の押し付けと奪い取りを実現する．これをを実現するためには，以下の課題がある．

（課題 6） IPCA に特化したプロセス間通信の作成

資源「仮想 IPCA 空間」が提供する資源操作のインタフェースを利用して高速なプロセス間通信を設計する．こ

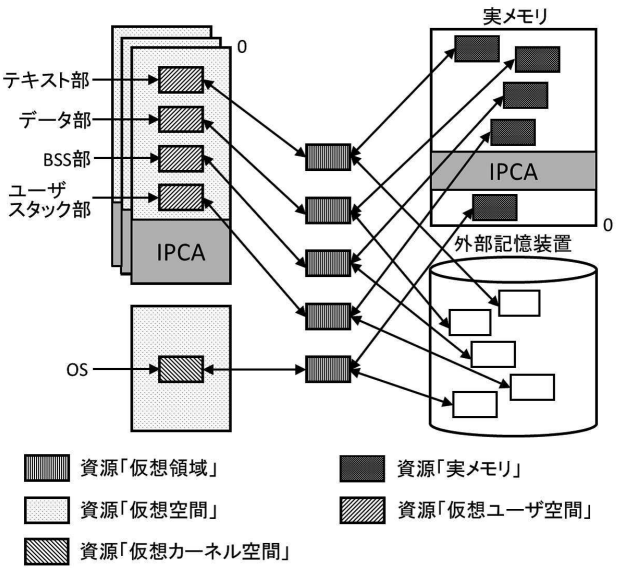

図 4 IPCA 導入後の ***Tender*** のメモリ空間の構成

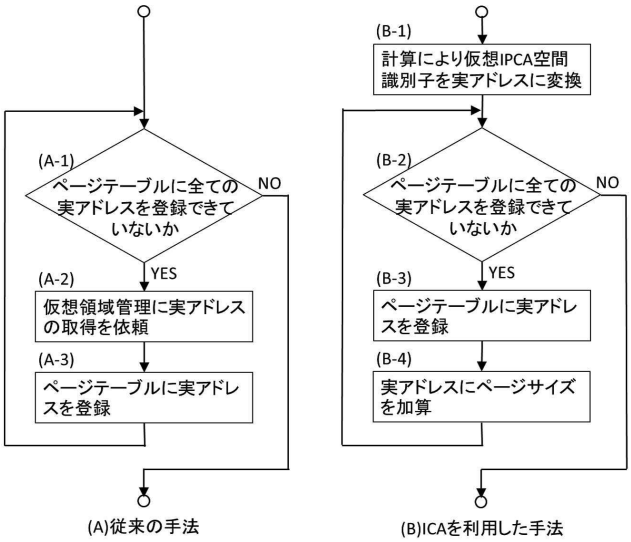

図 5 ページテーブルに実アドレスを登録する手法

のとき，資源「仮想 IPCA 空間」の資源操作のインタフェースをどのように利用して実現するのか検討する．

4.3 課題への対処

4.3.1 メモリ空間上における IPCA の配置

図 4 に IPCA 導入後の ***Tender*** のメモリ空間の構成を示す．仮想空間上では，IPCA をユーザ空間のユーザスタック部の後の空間に配置する．ユーザ空間に IPCA を配置することにより，ユーザモード，またはカーネルモードで走行するプロセスは，IPCA にアクセスできる．また，実メモリ空間上では，IPCA をカーネルが使用する領域の後の未使用な空間に配置する．

4.3.2 IPCA の管理方法

メモリ空間上に配置した IPCA を管理するために，資源「仮想 IPCA 空間」は，仮想 IPCA 空間の資源管理表（以降，仮想 IPCA 空間管理表）を用意する．仮想 IPCA 空間管理表は，資源「仮想 IPCA 空間」に関するデータを管理する．仮想 IPCA 空間管理表は，実メモリの資源識別子（以降，実メモリ識別子），仮想空間の資源識別子（以降，仮想空間識別子），仮想 IPCA 空間が貼り付いている仮想空間の数，および仮想 IPCA 空間の大きさの 4 種類のデータを保有する．実メモリ識別子は，仮想 IPCA 空間が実メモリのどの領域に対応しているかを示す．仮想空間識別子は，仮想 IPCA 空間がどの仮想空間に貼り付いているかを示す．

また，仮想 IPCA 空間管理表には，実メモリ空間上に存在する IPCA の使用状況を管理する IPCA ビットマップ管理表を用意する．IPCA ビットマップ管理表の 1 エントリ（1bit）は，IPCA の 1 ページの領域に対応している．また，IPCA ビットマップ管理表を使用中に設定する際，使用中にするエントリは，IPCA ビットマップ管理表の先頭エントリからオフセットとメモリ空間上の IPCA の先頭ページからのオフセットを一致させる．

4.3.3 IPCA のメモリの確保と解放

従来の方式では，プロセス間通信に利用するメモリの確保は，資源「仮想領域」の生成で行っている．資源「仮想領域」の生成では，生成する資源「仮想領域」のサイズに応じてページ単位で資源「実メモリ」を生成して割り当てている．このため，資源「仮想領域」と資源「実メモリ」の二つの資源の資源操作の呼び出しが発生し，オーバヘッドとなる．

IPCA を利用した方式では，***Tender*** の初期化時に，実メモリ空間上に IPCA として利用する領域を一括して確保する．次に，初期化処理時に確保した IPCA の使用状況を IPCA ビットマップ管理表で管理する．これにより，IPCA からメモリの確保と解放を行う際，IPCA ビットマップ管理表の操作のみとなり，資源操作の呼び出し回数を削減できる．したがって，高速なメモリの確保と解放を実現できる．また，IPCA の確保に成功したときは，図 1 に示す資源識別子の同一種類内の通番に IPCA のオフセット情報を付与する．これにより，仮想 IPCA 空間識別子から確保した IPCA のオフセット情報を取得でき，資源「仮想 IPCA 空間」を操作するときに，仮想 IPCA 空間資源識別子から実アドレスを求めることができる．

4.3.4 IPCA の貼り付けと剥がし

図 5 に ***Tender*** のページテーブルに実アドレスを登録する手法を示す．***Tender*** の従来の手法では，ページテーブルに登録する実アドレスを取得するために，資源「仮想領域」の資源管理表（以降，仮想領域管理表）を参照し，取得していた．また，複数ページの仮想アドレスを実アドレスに変換するためには，1 ページごとに仮想領域管理表を参照する．これにより，ページテーブルに実アドレスを登

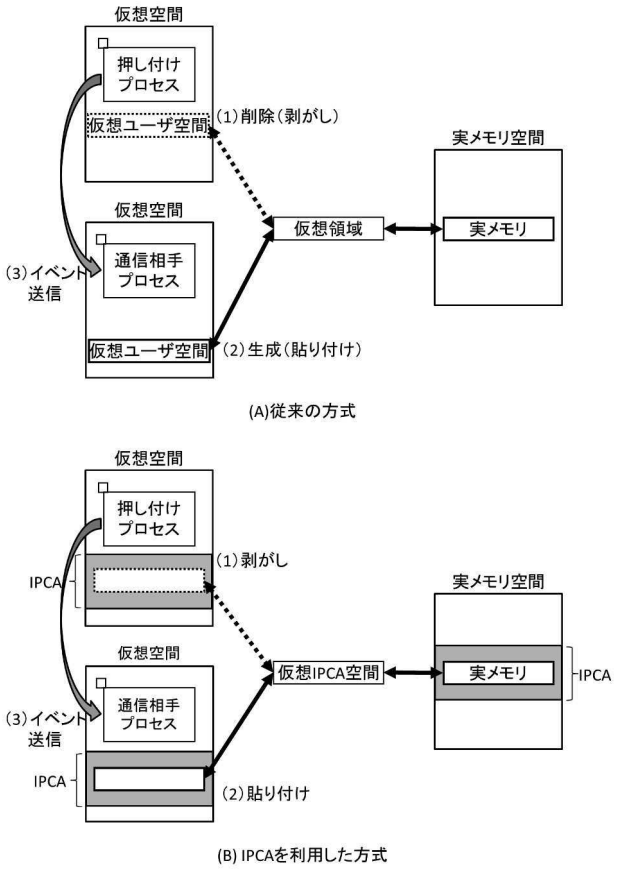

図 6 従来の押し付けと IPCA を利用した押し付け

録する際，他の資源の操作によるオーバヘッドが発生する．

これに対し，IPCA を利用した方式では，ページテーブルに登録する実アドレスをアドレス変換表なしに計算により求めることで対処する．

IPCA を利用した方式では，仮想アドレスから実アドレスへの変換を計算により行う．複数ページの仮想アドレスを実アドレスに変換するときは，1 ページ目の実アドレスにページサイズ（4 KB）を加算する．これにより，ページテーブルに実アドレスを登録する際，仮想アドレスから実アドレスへの変換を行うときに他の資源の管理表を参照する必要がなくなり，高速化できる．

4.3.5 既存のメモリ管理の変更

既存のメモリ管理の機能が IPCA を利用できないようにするために，ユーザ空間の仮想アドレスと実アドレスの対応付けを行う機能を変更する．***Tender*** では，資源「仮想ユーザ空間」がこの機能を保有する．資源「仮想ユーザ空間」が仮想アドレスと実アドレスの対応付けを行うときに，引数として仮想アドレスと貼り付ける空間のサイズが与えられる．仮想アドレスが IPCA 内の領域を指す場合，対応付けができないようにエラーを返却する．また，仮想アドレスが IPCA 内を指さないが，貼り付ける空間の一部が IPCA に重なる場合もエラーを返却する．

4.3.6 IPCA に特化したプロセス間通信の作成

図 6 に従来の押し付けの移動と IPCA を利用した押し付けの移動の様子を示す．従来の押し付けの移動では，押し付けプロセスの仮想空間から資源「仮想ユーザ空間」を削除し，通信相手プロセスの仮想空間に資源「仮想ユーザ空間」を生成することで，2 仮想空間の間での領域の貼り替えを実現している．しかし，この方法では，資源「仮想ユーザ空間」の生成と削除の資源操作の呼び出しがオーバヘッドとなっている．

これに対処するために，IPCA を利用した押し付けの移動では，資源「仮想 IPCA 空間」が提供する貼り付けと剥がしを利用する．これにより，IPCA を利用した押し付けの移動では，資源の生成と削除の資源操作の呼び出しが発生しなくなる．

4.4 資源「仮想 IPCA 空間」を利用したカーネルコール

資源「仮想 IPCA 空間」を利用した五つのカーネルコールを**表 1** に示す．これらのカーネルコールを利用することで，IPCA に特化したプロセス間通信を行うことができる．

vimcreate は，新たに仮想 IPCA 空間を生成するカーネルコールである．これにより，実メモリ空間上の IPCA からメモリを確保し，仮想空間に貼り付けることができる．引数の flag は，メモリの確保を IPCA の先頭から確保するのか，末尾から確保するのかを判定するフラグである．これは，IPCA から確保したメモリをすぐに解放する処理を行うときは，IPCA の先頭から確保し，すぐに解放しない処理を行うときは，IPCA の末尾から確保するように使い分けることで，IPCA の外部断片化を防ぐことができる．

vimdelete は，指定した仮想 IPCA 空間を削除するカーネルコールである．これにより，仮想空間から確保した IPCA のメモリを剥がし，メモリを解放できる．

vimpush は，指定したプロセスの仮想空間に仮想 IPCA 空間を押し付けるカーネルコールである．これにより，通信相手プロセスの状態によらず，通信相手プロセスにデータを送信できる．仮想 IPCA 空間を押し付ける処理モードには，移動と共有がある．

vimsack は，指定したプロセスの仮想空間から仮想 IPCA 空間を奪い取るカーネルコールである．これにより，通信相手プロセスの状態によらず，通信相手プロセスからデータを受信できる．仮想 IPCA 空間を奪い取る処理モードには，移動と共有がある．

vimunshare は，指定したプロセスの仮想空間から仮想 IPCA 空間を剥がすカーネルコールである．これにより，複数のプロセスで共有してる仮想 IPCA 空間から指定したプロセスを共有解除できる．

表 1 資源「仮想 IPCA 空間」を利用したカーネルコール

No.	形式	機能
1	vimcreate(name, pid, size, access, flag)	資源名 name を持つ仮想 IPCA 空間を新たに生成する．具体的な処理は，size 分の連続した領域をフラグ flag で指定した方法で確保し，プロセス pid の仮想空間に保護情報 access を設定して貼り付ける．
2	vimdelete(vimid)	仮想 IPCA 空間 vimid を削除する．
3	vimpush(vimid, op, pid, eventid)	処理モード op で指定したモードで，仮想 IPCA 空間 vimid を押し付け先プロセス pid の仮想空間に押し付ける．また，押し付け先プロセスにイベント eventid を送信し，仮想 IPCA 空間を押し付けたことを通知する．
4	vimsack(vimid, op, pid)	処理モード op で指定したモードで，仮想 IPCA 空間 vimid を奪い取り先プロセス pid の仮想空間から奪い取る．
5	vimunshare(vimid, pid)	仮想 IPCA 空間 vimid をプロセス pid の仮想空間から剥がす．

表 2 評価環境

OS	***Tender***
CPU	Intel Core i7-2600 3.4 GHz（4 コア）
RAM	8192 MB

5. 評価

5.1 評価内容

Tender に IPCA を実現したことによる有用性を示すために，プロセス間通信処理において，必要な機能を評価した．評価対象とする機能は，メモリの確保，メモリの解放，およびプロセス間通信である．以下に各機能の評価方法について述べる．

(1) メモリの確保

仮想 IPCA 空間の生成とコンテナの生成のカーネルコールにより，プロセス間通信に利用するメモリを確保した．評価では，メモリの確保とメモリ解放を交互に 1,000 回ずつ実行した．この際，メモリの確保のカーネルコールの処理時間を測定した．

(2) メモリの解放

仮想 IPCA 空間の削除とコンテナの削除のカーネルコールにより，プロセス間通信に利用するメモリを解放した．評価では，メモリの確保とメモリ解放を交互に 1,000 回ずつ実行した．この際，メモリの解放のカーネルコールの処理時間を測定した．

(3) プロセス間通信

仮想 IPCA 空間の押し付けと奪い取り，およびコンテナの押し付けと奪い取りのカーネルコールにより，即時同期が可能なプロセス間通信を行った．評価では，二つのプロセス間で同期処理を実行しつつ，交互に同じ種類のプロセス間通信処理を 1,000 回ずつ実行した．この際，片方のプロセスが呼び出すプロセス間通信のカーネルコールの処理時間を測定した．

なお，これらの評価は，**表 2** に示す評価環境で評価した．また，基本性能を評価するために，評価プログラムをシングルコアで動作させた．

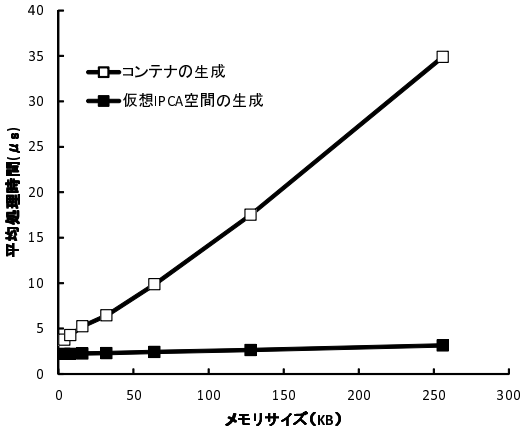

図 7 メモリ確保処理の評価結果

5.2 評価結果

5.2.1 メモリ確保処理

評価結果を**図 7** に示す．図 7 より，以下のことがわかる．

(1) 仮想 IPCA 空間の生成は，コンテナの生成より処理時間が短い．これは，仮想 IPCA 空間の生成において，メモリを確保する処理を高速化したためである．仮想 IPCA 空間の生成では，IPCA ビットマップ管理表の未使用エントリを使用中に設定することでメモリを確保できる．このため，資源操作の呼び出しが発生せず，オーバヘッドが小さくなった．一方，コンテナの生成では，仮想領域の生成でメモリを確保できる．仮想領域の生成では，資源「実メモリ」を生成して仮想領域にメモリを割り当てている．このため，資源「仮想領域」と資源「実メモリ」の二つの資源の資源操作の呼び出しが発生し，オーバヘッドが大きくなった．

(2) 仮想 IPCA 空間の生成は，コンテナの生成よりメモリサイズの増加に対する処理時間の増加率が小さい．これは，仮想 IPCA 空間の生成において，確保したメモリを仮想空間に貼り付ける処理を高速化したためである．仮想 IPCA 空間の生成では，貼り付け処理において，ページテーブルに登録する実アドレスをすべて計算により求めている．このため，資源操作の呼び出しが発生せず，処理時

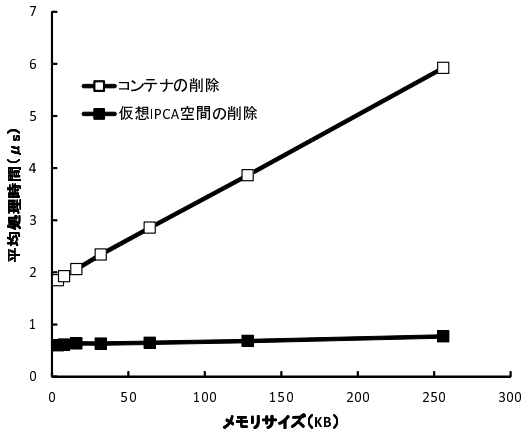

図 8 メモリ解放処理の評価結果

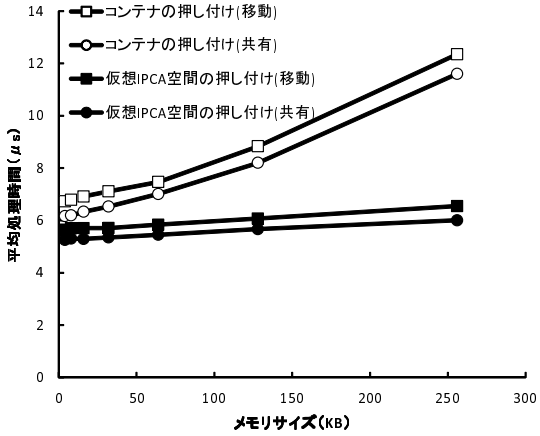

図 9 押し付けの評価結果

間の増加率が小さくなった．一方，コンテナの生成では，貼り付け処理において，1 ページごとに仮想領域管理表から実アドレスを取得している．このため，メモリサイズが増加するにつれて，資源操作の呼び出し回数が増加し，処理時間の増加率が大きくなった．

5.2.2 メモリ解放処理

評価結果を**図 8** に示す．図 8 より，以下のことがわかる．

（1） 仮想 IPCA 空間の削除は，コンテナの削除より処理時間が短い．これは，仮想 IPCA 空間の削除において，メモリを解放する処理を高速化したためである．仮想 IPCA 空間の削除では，IPCA ビットマップ管理表の該当エントリを未使用に設定することでメモリを解放できる．このため，資源操作の呼び出しが発生せず，オーバヘッドが小さくなった．一方，コンテナの削除では，仮想領域の削除でメモリを解放できる．仮想領域の削除では，仮想領域に割り当てている資源「実メモリ」を削除をしている．このため，資源「仮想領域」と資源「実メモリ」の二つの資源の資源操作の呼び出しが発生し，オーバヘッドが大きくなった．

（2） 仮想 IPCA 空間の削除は，コンテナの削除よりメモリサイズの増加に対する処理時間の増加率が小さい．これは，仮想 IPCA 空間の削除において，削除する資源の数がないためである．資源「仮想 IPCA 空間」の削除では，メモリの解放を IPCA ビットマップ管理表の該当エントリを未使用に設定する操作のみであり，資源の削除が発生しない．一方，コンテナの削除では，資源「仮想領域」，資源「実メモリ」，および資源「仮想ユーザ空間」の三つの資源を削除している．

5.2.3 プロセス間通信処理

評価結果を**図 9** と**図 10** に示す．各図より，以下のことがわかる．

（1） 仮想 IPCA 空間を利用したプロセス間通信は，コンテナを利用したプロセス間通信より処理時間が短い．これは，仮想 IPCA 空間を利用したプロセス間通信において，2 仮想空間の間での領域の貼り替えを高速化したためである．仮想 IPCA 空間を利用したプロセス間通信では，仮想 IPCA 空間の貼り付けと剥がしで 2 仮想空間での領域の貼り替えを行っている．一方，コンテナを利用したプロセス間通信では，仮想ユーザ空間の生成と削除で 2 仮想空間での領域の貼り替えを行っている．このため，資源「仮想ユーザ空間」の生成と削除の資源操作の呼び出しが発生し，オーバヘッドが大きくなった．

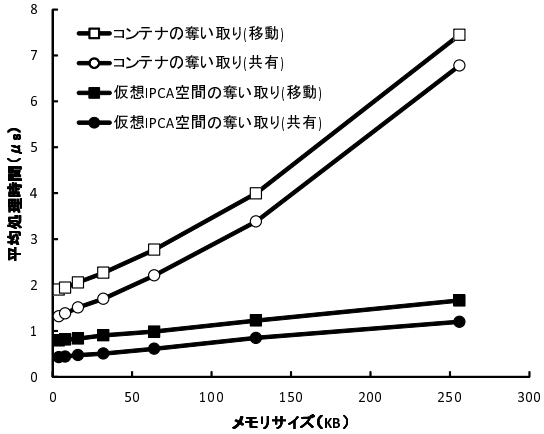

図 10 奪い取りの評価結果

（2） 仮想 IPCA 空間を利用したプロセス間通信は，コンテナを利用したプロセス間通信よりメモリサイズの増加に対する処理時間の増加率が小さい．これは，仮想 IPCA 空間を利用したプロセス間通信において，仮想空間にメモリを貼り付ける処理を高速化したためである．仮想 IPCA 空間を利用したプロセス間通信では，貼り付け処理において，ページテーブルに登録する実アドレスをすべて計算により求めている．このため，資源操作の呼び出しが発生せず，処理時間の増加率が小さくなった．一方，コンテナを利用したプロセス間通信では，貼り付け処理において，1 ページごとに仮想領域管理表から実アドレスを取得している．このため，メモリサイズが増加するにつれて，資源操作の呼び出し回数が増加し，処理時間の増加率が大きくなった．

（3） 奪い取りは，押し付けより処理時間が短い．これは，奪い取りでは，通信相手プロセスにイベントを送信しないのに対し，押し付けでは，通信相手プロセスにイベントを送信するためである．このため，押し付けでは，処理時間に通信相手プロセスとの同期処理が含まれるため，処理時間が長くなった．

（4） 共有は，移動より処理時間が短い．これは，共有では，貼り付け処理のみ行うのに対し，移動では，貼り付け処理と剥がし処理の二つの処理を行うためである．

6. おわりに

Tender のメモリ空間上に IPCA を実現し，IPCA に特化したプロセス間通信の設計と実現方式を述べた．プロセス間通信を高速化するために，***Tender*** のメモリ空間上に IPCA を配置し，その領域の制御と管理を行う資源「仮想 IPCA 空間」を実現した．また，資源「仮想 IPCA 空間」が提供する資源操作のインタフェースを利用して IPCA に特化したプロセス間通信を実現した．

プロセス間通信に利用するメモリの確保と解放では，***Tender*** の初期化処理時に IPCA として利用する領域を一括して確保し，IPCA ビットマップ管理表で使用状況を管理する．これにより，IPCA のメモリの確保と解放を IPCA ビットマップ管理表の操作のみで実現し，プロセス間通信におけるメモリの確保と解放の高速化を実現した．プロセス間通信では，ページテーブルに登録する実メモリを計算により求めることで，ページテーブルの更新処理を高速化し，プロセス間通信の高速化を実現した．

評価では，メモリの確保，メモリの解放，および，プロセス間通信の処理時間を測定した．評価結果より，IPCA を利用した機能の方が ***Tender*** の既存の機能より処理時間が短いことを示した．

残された課題として，資源「仮想 IPCA 空間」のマルチコア対応の実現がある．

謝辞 本研究の一部は，科学研究費補助金基盤研究（B）（課題番号：24300008），および科学研究費補助金若手研究（B）（課題番号：25730046）による．

参考文献

[1] Hong, Y. and Ping-ping, G.: Application of Windows Inter-process Communication in Software System Integration, *2010 International Conference on Intelligent System Design and Engineering Application*, Vol. 1, pp. 375–378 (2010).

[2] Immich, P., Bhagavatula, R. and Pendse, R.: Performance analysis of five interprocess communication mechanisms across UNIX operating systems, *Journal of Systems and Software*, Vol. 68, No. 1, pp. 27–43 (2003).

[3] Shapiro, J., Farber, D. and Smith, J.: The Measured Performance of a Fast Local IPC, *Proceedings of the 5th International Workshop on Object-Orientation in Operating Systems*, pp. 89–94 (1996).

[4] 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*, pp. 45–53 (1992).

[5] Hsieh, W., Kaashoek, M. and Weihl, W.: The Persistent Relevance of IPC Performance: New Techniques for Reducing the IPC Penalty, *Proceedings of the 4th Workshop on Workstation Operating Systems*, pp. 186–190 (1993).

[6] Liedtke, J.: Improving IPC by kernel design, *Proceedings of the 14th ACM symposium on Operating systems principles*, pp. 175–188 (1993).

[7] 岡本幸大，谷口秀夫：***AnT*** オペレーティングシステムにおける高速なサーバプログラム間通信機構の実現と評価，電子情報通信学会論文誌，Vol. J93-D, No. 10, pp. 1977–1989 (2010).

[8] 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).

[9] 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).

[10] 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).