

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

田端 利宏 谷口 秀夫

九州大学大学院システム情報科学研究科

プログラム構造を重視して機能を実現した *Tender* (The ENduring operating system for Distributed EnviRonment) オペレーティングシステムのプロセス間通信機能について述べる。*Tender* では、単一仮想記憶と多重仮想記憶を融合したヘテロ仮想記憶を実現した。ヘテロ仮想記憶では、複数の仮想記憶空間が存在し、一つの仮想記憶空間に 0 個以上のプロセスが存在でき、プロセスが仮想記憶空間の間を移動できる。このため、プロセスの処理内容に合わせて、協調処理を行なうことが可能である。本論文では、*Tender* のメモリ関連資源とヘテロ仮想記憶の特徴を生かしたプロセス間通信機能について述べる。プロセス間通信に関わる資源として、資源「コンテナ」、「コンテナボックス」、「イベント」を導入し、実現した。また、実測と評価により、プロセス間通信の基本性能を示す。

Implementation and Evaluation of Inter Process Communication on *Tender*

Toshihiro TABATA and Hideo TANIGUCHI

Graduate School of Information Science and Electrical Engineering, Kyushu University

We describe inter process communication on *Tender* operating system. Heterogeneous Virtual Storage(HVS) of *Tender* is integrated both single virtual storage and multiple virtual storage. HVS has multiple virtual spaces. And there are over zero processes on a virtual space. And a process can migrate between virtual spaces. So that, processes can cooperate with other processes in processing. We create new resources "container," "container box" and "event" to realize efficient inter process communication. This paper describes mechanism of inter process communication and shows evaluation of inter process communication.

1 はじめに

我々は、プログラム構造を重視して機能を実現する *Tender*^[1]を開発している。*Tender*では、単一仮想記憶と多重仮想記憶を融合させたヘテロ仮想記憶^[2]を実現した。このため、プログラムの処理内容の疎密関係に合わせて、プロセスやデータの配置を変えることができ、プロセス間での効率的な協調作業が可能である。本論文では、*Tender*の基本構造とメモリ管理方式とヘテロ仮想記憶の特徴を生かしたプロセス間通信機能の実現方式と性能評価について述べる。

2 *Tender*

2.1 プログラム構造

プログラムは、大きく三つの部分に分けられる。それは、基盤部、表プログラム構造部、および拡張部である。

基盤部は、オペレーティングシステム (以降、OS と略す) の動作の最も基盤となる処理を行なう。特に、資源インタフェース制御部は、*Tender*特有の部分で、表プログラム構造と名付けたプログラム管理構造に基づき、資源を管理しているプログラム部分 (資源管理処理部) の呼出しを制御している。

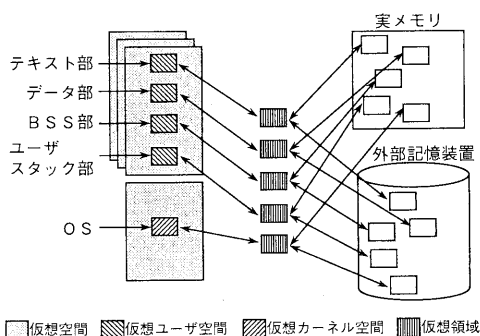


図1 プロセスとメモリ関連の資源

表プログラム構造部は、ソフトウェア部品化されたプログラム群（以降、プログラム部品と名付ける）とそのプログラム部品をポインタで管理しているプログラムポインタ表からなる。プログラムポインタ表は、配列形式であり、各配列の要素は、プログラム部品へのポインタを持つ。プログラムへの制御の移行は、プログラムポインタ表を介して行なわれる。このようなプログラム構造により、各プログラム部品の動作状況の把握が可能になる。

拡張部は、OS動作の拡張機能に位置づけられる処理を行う。

2.2 資源の分離と独立化

*Tender*は、OSの制御する対象を資源として管理する。資源には、識別のために、文字列による名前（以降、資源名と呼ぶ）や数字による識別子を付与し、かつ資源操作のインタフェースを統一している。また、プログラム構造上、資源の種類毎に管理表を個別に用意し、他資源の管理表へのポインタを禁止している。更に、資源の種類毎に管理するプログラムを個別に用意し、共通プログラムを排除している。

このように資源の分離と独立化を行うことで、資源の事前用意や保留により、資源の作成や削除を伴う処理を高速化している^[3]。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。更に、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。

2.3 メモリ管理機構

2.3.1 プロセスとメモリ関連の資源の関係

プロセスとメモリ関連の資源を図1に示す。「仮想空間」とは、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。「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。

仮想領域は、複数のページからなり、各ページは外部記憶装置か実メモリ上に存在する。さらに、仮想領域を仮想空間に「貼り付け」、「剥す」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「仮想カーネル空間」や「仮想ユーザ空間」を生成し、削除できる。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、剥すとは、仮想アドレスと実アドレスの対応付けを解除することである。仮想領域をOSの存在する仮想空間に貼り付けた場合には、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。仮想カーネル空間と仮想ユーザ空間は、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でアクセスできる。仮想空間の変換表は、2段階になっており、仮想空間の生成時には、1段階の変換表のみを生成することにより、仮想空間の生成を高速化している。仮想領域を仮想空間に貼り付ける時に、2段階の変換表を作成する。この時、必要最小限度の大きさの2段階の変換表を作成し、2段階の変換表の作成時間や利用するメモリの量を節約する。仮想領域を剥す時には、2段階の変換表をそのまま残し、内容のみを消去する。このため、次に同じアドレスに仮想領域を貼り付ける際には、2段階の変換表が存在するため、貼り付け処理を高速化できる。

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、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。プロセスの仮想空間の間の移動は、仮想ユーザ空間を貼り換えることで簡単に実現できる。メモリ関連の資源は、プロセスとは独立して存在可能なので、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]。

2.3.2 ヘテロ仮想記憶

*Tender*では、単一仮想記憶と多重仮想記憶を融合させたヘテロ仮想記憶を実現した。ヘテロ仮想記憶の様子を図2に示し以下に説明する。ヘテロ仮想記憶では、複数の仮想記憶空間が存在でき、一つの仮想記憶空間に0個以上のプロセスが存在できる。また、プロセスが仮想記憶空間の間を移動することが可能である。このため、データだけが存在する仮想記憶空間も存在でき、データの存在する仮想記憶空間にそのデータを利用するプロセスを作ることにより、データを中心としたプロセス間での協調処理が

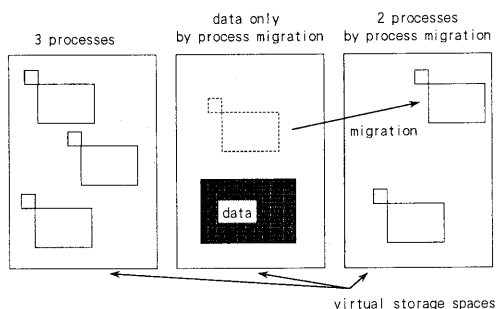


図2 ヘテロ仮想記憶

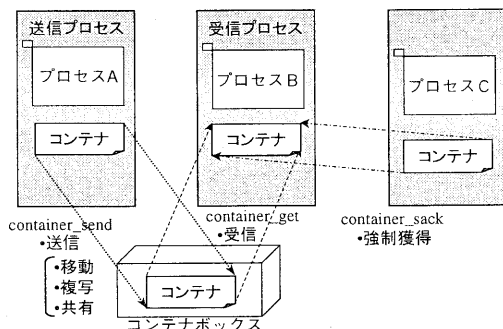


図3 プロセス間通信

可能である。また、処理内容に合わせて、プロセスが仮想記憶空間の間を移動できる。例えば、緊密な関係にあるプロセス同士では同じ仮想記憶空間に移動することで、仮想記憶空間の切替をなくすことが可能であり、データも共有できる。関係が緊密でないプロセス同士は、別の仮想記憶空間に配置することで、メモリ空間を保護し、広いメモリアドレスを利用できる。このため、効率的なプロセス間の協調処理が可能である。

3 プロセス間通信機能

3.1 基本機能

Tenderでは、プロセス間通信機能として、メモリ空間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する。

プロセス間通信機能を実現するために、資源「コンテナ」と資源「コンテナボックス」を導入した。

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」をやりとりすることで、プロセス間通信を実現することができる。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は、移動、共有、複写の三つのものがある。移動は、コンテナを別のプロセスの仮想空間に移動させることである。共有は、送出元のプロセスの仮想空間に存在するコンテナの基になっている仮想領域を受信プロセスの仮想空間にコンテナとして貼り付け、同じメモリ空間を共有するものであり、UNIXでの共有メモリと同じような使い方ができる。複写は、基となる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。

資源「コンテナボックス」は、プロセス間での資源「コンテナ」の受渡しの仲介をする。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3に示す。図3のプロセス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なっている。送出でコンテナがコンテナボックスに格納され、受信でコンテナが受信プロセスの仮想空間に貼り付けられる。コンテナボックスを介したプロセス間通信でメッセージの送信、メモリ空間の共有が可能となり、プロセス間での協調作業が可能となる。コンテナボックスを介せずコンテナを受け取ることも可能である。これをコンテナの強制獲得と呼ぶ。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、コンテナの強制獲得時のコンテナの受渡しを表している。強制獲得は、コンテナを奪い取りたいプロセスが、コンテナを直接指定して奪い取ることができる。このため、コンテナを利用しているプロセスの送出を待つことなくコンテナを奪い取ることが可能となる。

UNIXで良く利用されるプロセス間通信には、パイプやソケットがある。パイプやソケットでは、プロセス間で接続を確立し、通信を行なう。このため、事前に接続を確立する手順を踏まなければならない。それに対し、今回提案した手法では、コンテナボックスを生成するだけで、プロセス間通信を開始する

表 1 プロセス間通信インタフェース

通番	形式	機能
1-1	contbox_create(name)	資源名 name を持つコンテナボックスを生成し、コンテナボックス識別子 ctainerbxid を返す。
1-2	contbox_delete(ctainerbxid)	コンテナボックス ctainerbxid を削除する。
1-3	contbox_check(ctainerbxid, ctainerid)	コンテナボックス ctainerbxid の状態を確認する。
2-1	container_create(name, reqaddr, size, *paddr)	資源名 name を持つコンテナを、アドレス reqaddr に、大きさ size で生成し、コンテナ識別子 ctainerid を返す。
2-2	container_delete(ctainerid)	コンテナ ctainerid を削除する。
2-3	container_send(ctainerid, ctainerbxid, reqaddr, op)	コンテナ ctainerid をコンテナボックス ctainerbxid に、アドレス reqaddr に貼り付けるように、モード op で送出する。
2-4	container_get(ctainerid, ctainerbxid, reqaddr, waittime, *paddr, *size)	コンテナボックス ctainerbxid から、アドレス reqaddr にコンテナを受けとる。
2-5	container_sack(ctainerid, reqaddr, op, *paddr, *size)	コンテナ ctainerid をモード op で受信プロセスの存在する仮想空間のアドレス reqaddr に貼り付ける。
3-1	event_create(name)	資源名 name を持つイベントを生成し、イベント識別子 evtid を返す。
3-2	event_delete(evtid)	イベント evtid を削除する。
3-3	event_receive(evtid)	イベント evtid のカウンタをデクリメントし、カウンタの値が 0 より小さい時は、イベントが発生するまで待つ。
3-4	event_send(evtid)	イベント evtid を送信する。イベントに関数に関連付けられていた場合、その関数を実行する。そうでなければ、カウンタをインクリメントする。
3-5	event_attach(evtid, func)	イベント evtid に関数を関連付ける。

ことができる。また、一つのコンテナボックスを介して、1 対複数、複数対 1、複数対複数のプロセスでの通信を行なうことができる。さらに、現在のプロセス間通信は、絶対時間で同期を行っていない。これは、必ず通信相手のプロセスからの応答を待たなければならないからである。それに対し、強制獲得を利用した場合では、通信相手のプロセスの応答を待たずに処理結果を受けとることができる。例えば、ある時間以内に応答がない場合には、強制獲得でコンテナを受けとることで、何らかの結果を指定時間内に得ることができ、受信待ちを続ける必要がない。

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する資源として「イベント」を導入した。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関数を実行するか、イベントに関数が割り当てられていない時には、カウンタセマフォの役割を果たす。

3.2 提供インタフェース

プロセス間通信インタフェースを表 1 に示す。

コンテナボックスに関するインタフェースには、コ

ンテナボックスの生成と削除と状態の確認がある。状態の確認とは、指定されたコンテナが格納されているのか、またはいくつコンテナが格納されているのかを調べるインタフェースである。

コンテナに関するインタフェースには、コンテナの生成、削除、送出、受信、強制獲得がある。コンテナの送出のモードには、移動、複写、共有の 3 種類があり、送出情報も同時にコンテナボックスに対して送出される。送出情報には、貼り付けアドレスの情報が含まれており、今後は保護情報なども加える予定である。受信は、受信するプロセスの貼り付けアドレスの要求と送出時の情報とを比較し、一致する場合のみコンテナボックスからコンテナを受信させる。この時、コンテナボックスにコンテナがなかった場合には、指定された時間だけ、コンテナが送出されるのを待つ。受信コンテナは、コンテナボックスのキューの先頭のものか、指定した識別子を持つものかを選択できる。強制獲得は、コンテナボックスを介さずに直接指定したコンテナを受信ことができ、移動、複写、共有のモードを指定できる。

イベントに関するインタフェースは、生成、削除、受信、送信、関連付けがある。送信は、イベントが起こったことをイベント管理部に知らせ、イベント発生時の処理やカウンタのインクリメント処理を行なう。イベントの受信は、カウンタセマフォの値をデクリメントして、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。関数の関連付けは、イベント発生時に実行する関数をイベントに関連付けるインタフェースである。

3.3 実装内容

「コンテナ」、「コンテナボックス」、「イベント」は資源として実現され、管理部、管理表、プログラム部品からなり、各プログラム部品は、表プログラム構造部を介して呼び出される。

資源「コンテナ」は、仮想ユーザ空間を用いて生成される。コンテナが、複数のプロセスで共有され複数個の仮想空間に貼り付いている場合には、それぞれ異なる資源識別子を持った仮想ユーザ空間によりコンテナが構成されるが、実体である仮想領域は一つある。また、仮想領域の貼り付けと剥しにより、簡単に仮想空間の間での移動、複写、共有が実現できる。

コンテナボックス管理部は、各「コンテナボックス」ごとに送出されたコンテナの情報を管理表で管理しておき、プロセスがコンテナを受信する時に送出時の情報を渡す。コンテナボックスではキューで送出されたコンテナを管理しているが、指定されたコンテナを取り出すことができる。

資源「イベント」は、イベント管理部の持つ管理表により状態の管理が行なわれる。管理表に格納されている情報は、各イベントごとのカウンタの値と関連付けられている関数の情報である。資源「イベント」の様子を図4に示し、以下に説明する。イベントが発生するとイベントの発生をイベント管理部に送信する。関数が関連付けられたイベントAが発生した場合には、イベント管理部は、関連付けられた関数を実行する。関数の関連付けられていないイベントBが発生した場合には、イベント管理部は、イベントBのカウンタをインクリメントする。プロセスがイベントを受信する時には、イベント管理部がイベントBのカウンタをデクリメントし、その結果の値を0と比較し、プロセスに処理を継続させるか、イベントの発生を待たせるかを判断する。

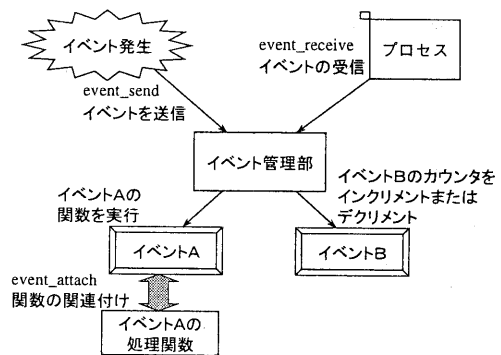


図4 資源「イベント」

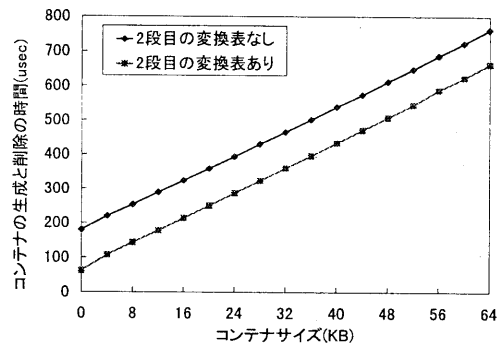


図5 コンテナの生成と削除の時間

4 性能評価

4.1 測定環境

測定は、プロセッサ PentiumII 450MHz の計算機を使用した。コンテナの生成と削除時間とコンテナボックスの生成削除時間とコンテナの送受信時間を測定した。コンテナの送受信は、別の仮想空間に存在するプロセスの間で行なった。ヘテロ仮想記憶には、仮想空間の生成の高速化のため、生成時には2段階目の変換表が作られない。このため、コンテナの生成やコンテナの送受信において、2段階目の変換表の有無による影響を評価した。

4.2 コンテナボックスとコンテナの生成と削除時間

資源「コンテナボックス」を生成し削除する操作を1000回繰り返し、一回あたりの平均時間を算出した。コンテナボックスの生成と削除の一回当たりの平均時間は、約5μsecであった。資源の登録と管理表のエントリの初期化だけなので、コンテナの生成と削除に比べ、高速である。

資源「コンテナ」の生成と削除を、コンテナサイズを0KBから、64KBまで4KBごとに大きさを変えて、測定し、一回当たりの処理時間を計測した。図5に測定結果を示す。図5より、コンテナの生成と削除時

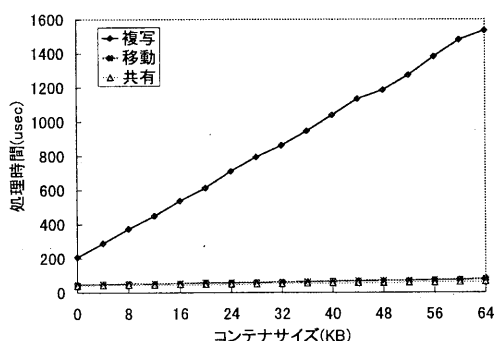


図6 コンテナ送受信時間

間は、コンテナサイズに比例して増加する。コンテナサイズに比例する主な原因は、コンテナの生成時に、実メモリの確保する処理を1ページずつ行なうためである。また、仮想領域を仮想空間に貼り付ける処理もコンテナサイズに比例するが、その影響はかなり小さい。このことは、図6の移動と共有の処理時間のグラフと比較するとわかる。これは、コンテナ送受信の移動と共有の処理時間の大部分は、仮想領域の貼り付けと剥しであるが、コンテナサイズに比例して増加する処理時間はわずかであることからわかる。また、2段目の変換表のある場合には、約100usec高速にコンテナを生成できる。この約100usecという時間は、2段目の変換表の生成に要する時間である。

4.3 コンテナの送受信の処理時間

コンテナ送受信時間の測定結果を図6に示す。これは、2段目の変換表のある場合の測定結果である。グラフより、共有、移動、複写の順番で処理が速いことがわかる。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、移動に比べかなり処理が遅い。共有と移動は、コンテナサイズに対して処理時間はほとんど一定である。共有は、移動に比べ高速である。これは、移動が送出元の空間から、メモリ空間を剥して、別の空間に貼り付けるのに対して、共有は、別の空間に貼り付けるだけで済むからである。

2段目の変換表がある場合とない時の影響について述べる。例として移動の場合の測定結果を図7に示す。2段目の変換表のある場合には、仮想領域を貼り付ける際に実アドレスと仮想アドレスの対応を書き込むだけなので処理は高速である。それに対し、2段目の変換表がない場合には、仮想領域を仮想空間に貼り付け時に2段目の変換表を作成する処理を伴うため、処理時間が長くなる。コンテナの生成と

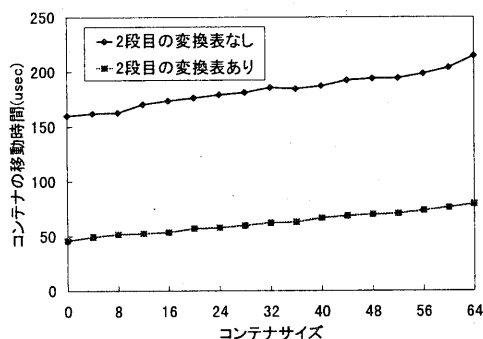


図7 2段目の変換表の効果

削除の時と同様に、2段目の変換表の有無により約100usecの処理時間差がある。同じアドレスへの一回目のコンテナの受渡し処理では処理時間が長くなるが、それ以降の処理では処理時間が短くなる。

5 まとめ

Tenderの特徴とメモリ管理機構とヘテロ仮想記憶の特徴について述べ、「コンテナ」、「コンテナボックス」、「イベント」を用いたプロセス間通信機能の実現方式と評価について述べた。評価した結果、コンテナボックスの生成は、5usecであり、コンテナの生成と削除には、コンテナサイズに比例した時間がかかり、2段目の変換表の存在する場合には約100usec処理時間を高速化できる。コンテナボックスを介したコンテナ送受信速度は、移動と共有の処理時間はほとんどコンテナサイズに対して一定であり、複写はコンテナサイズに比例して処理時間がかかる。

今後は、HVSでのプロセスの仮想空間の間の移動による協調処理とコンテナを利用したプロセス間通信による協調処理を比較し、両者の特徴と効果を明らかにする予定である。

参考文献

- [1] 谷口秀夫：“分散指向永続オペレーティングシステム **Tender**”，情報処理学会コンピュータシステムシンポジウム，シンポジウム論文集 Vol.95, No.7, pp.47-54(1995).
- [2] 谷口秀夫，長嶋直希，田端利宏：“単一仮想記憶と多重仮想記憶を共存させたヘテロ仮想記憶の実現”，情処研報，Vol.98, No.33, pp.87-94 (1998).
- [3] 田端利宏，谷口秀夫：“プロセス構成要素を再利用したプロセスの生成と消滅の高速化”，情処研報，Vol.98, No.33, pp.79-86 (1998).