システムソフトウェアと
オペレーティング・システム 78－12
（１９９８．５．７）

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

谷口　秀夫　　長嶋　直希　　田端　利宏

九州大学 大学院システム情報科学研究科

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の機能を提案する。ヘテロ仮想記憶は、複数の仮想記憶空間を有し、一つの仮想記憶空間内には０個以上のプロセスが存在でき、プロセスには仮想記憶空間の間を移動できる機能を提供する。これにより、単一仮想記憶と多重仮想記憶の長所を最大限に引き出して利用することが可能になる。さらに、ヘテロ仮想記憶を***Tender*** オペレーティングシステムに実装し、評価した。既存の仮想記憶空間を利用することにより、プロセスの作成削除時間を削減できる。また、プロセスの仮想記憶空間移動時間は、移動するプロセスの大きさの影響をほとんど受けず、わずか0.7ミリ秒程度である。

Implementation of Heterogeneous Virtual Storage coexisted of Single Virtual Storage and Multiple Virtual Storage

Hideo TANIGUCHI, Naoki NAGASHIMA and Toshihiro TABATA

Graduate School of Information Science and Electrical Engineering, Kyushu University

We suggest function of Heterogeneous Virtual Storage(HVS) that is fused both single virtual storage and multiple virtual storage. HVS has multiple virtual storage spaces. And there are over zero processes on a space. And a process can migrate between virtual storage spaces. Therefore, HVS has both single virtual storage's advantages and multiple virtual storage's advantages. We implemented and evaluated HVS on Tender operating system. To use existed virtual storage space reduce the time of creating process and deleting process. And the time of process migration is not influence from the size of process, and is about 0.7 millisecond.

１．はじめに

オペレーティングシステム（以降、ＯＳと略す）の仮想記憶機能は、プロセスに広い記憶空間を与え、並行動作における保護を行う等、非常に重要な機能である。既存のＯＳでは、単一仮想記憶と多重仮想記憶の機能が代表的であり、それぞれの特徴を生かした利用が個別になされている。

本稿では、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提案し、***Tender*** オペレーティングシステム[1]での実装内容と評価結果について述べる。ヘテロ仮想記憶により、単一仮想記憶と多重仮想記憶の長所を最大限に引き出して利用することが可能になる。また、既存の仮想記憶空間を利用することでプロセスの作成削除時

間を短くでき、プロセスの仮想記憶空間移動時間は短いことを示す。

2．各仮想記憶の特徴

2．1　単一仮想記憶と多重仮想記憶

既存のＯＳが提供している仮想記憶は、単一仮想記憶と多重仮想記憶に分類できる。

単一仮想記憶を実現するＯＳでは、一つの仮想記憶空間を提供し、すべてのプロセスでこの仮想記憶空間を共有する。多重仮想記憶を実現するＯＳでは、複数の仮想記憶空間を提供し、各プロセスは一つの仮想記憶空間を共有して利用する。

単一仮想記憶と多重仮想記憶の比較を表１に示す。プロセス間の保護やプロセス空間の大きさは、各プロセス毎に仮想記憶空間を提供しているため、多重仮想記憶の方が優れている。逆に、プロセス間通信速度やプロセス切替速度は、仮想記憶空間の切替を必要としないため、単一仮想記憶が優れている。実行可能なプログラム形式（以降、ロードモジュールと呼ぶ）については、単一仮想記憶の場合、すべてのロードモジュールを一つの仮想記憶空間上に展開する必要があるため、自由なアドレスに配置できる形式（PIC:Position Independent Code）であることを要求する。これに対し、多重仮想記憶の場合、各ロードモジュールは各々別の仮想記憶空間上に展開するため、PIC形式である必要はなく、静的リンクにより固定のアドレス配置の形式でも良い。

こうのように、単一仮想記憶と多重仮想記憶は相反する特徴を持つ。したがって、従来から、この特徴を生かした利用がされており、例えば、共同利用の計算機ではプロセス間の保護を重視し多重仮想記憶を提供するＯＳを利用している。また、プログラムやファイルなどの資源利用単位であるプロセスとは別に、プロセッサの割当単位としてスレッドを考え、一つのプロセス内に複数のスレッドを実現する方法がある。この方法では、プロセスは多重仮想記憶、スレッドは単一仮想記憶の特徴を生かせる。

2．2　ヘテロ仮想記憶

前述したように、単一仮想記憶と多重仮想記憶は相反する特徴を持つ。したがって、両仮想記憶を共存制御できれば、これらの特徴を生かした利用が一つのＯＳ内で可能になる。そこで、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提案する。

ヘテロ仮想記憶の概要を図１に示す。ヘテロ仮想記憶は、

（機能１）複数の仮想記憶空間を提供し、

（機能２）一つの仮想記憶空間内には０個以上のプロセスが存在でき、

（機能３）プロセスは仮想記憶空間の間を移動できる、

ものである。

（機能１）と（機能２）により、単一仮想記憶

表１　単一仮想記憶と多重仮想記憶の比較

通番	比較項目	単一仮想記憶	多重仮想記憶
1	プロセス間の保護	弱い（×）	強い（○）
2	プロセス間通信速度	速い（○）	遅い（×）
3	プロセス切替速度	速い（○）	遅い（×）
4	プロセス空間の大きさ	狭い（×）	広い（○）
5	ロードモジュールへの要求	自由なアドレス配置を要求（×）	固定のアドレス配置でも良い（○）

と多重仮想記憶の長所のみを生かすことが可能になる。さらに、プロセスの仮想記憶空間移動機能（機能3）により、これらの長所を最大限に引き出すことが可能になる。例えば、処理が疎遠なプロセスは別の仮想記憶空間で走行し、処理が緊密なプロセスは同じ仮想記憶空間で走行する。さらに、処理の途中で疎遠さや緊密さが変化する場合は、プロセスがその程度に応じて仮想記憶空間を移動すれば良い。

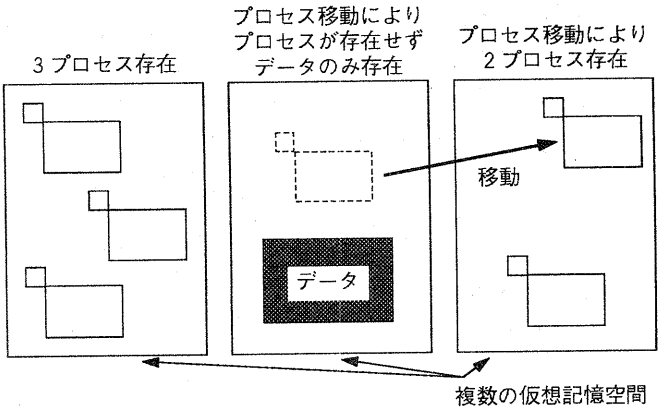

図1　ヘテロ仮想記憶

また、（機能2）により、データのみが存在する仮想記憶空間を作ることができる。これは、大きいデータを複数の小さいプロセスが共有し時分割で利用する場合に有効である。多重仮想記憶の場合のように各プロセスの仮想記憶空間に共有データを貼り付けて処理するのではなく、データが存在する仮想記憶空間にプロセスが移動して処理を行える。これは、データが大きくプロセスは小さい場合や、処理を行う時のみデータ操作させること（排他処理）をデータ中心に行いたい場合（データの所有権を重視する場合）に有効である。

上記の特徴を生かしたヘテロ仮想記憶を実現するには、以下の課題がある。

（1）プロセスの仮想記憶空間移動に伴うアドレス衝突の回避

（2）高速なプロセスの仮想記憶空間移動の実現

課題（1）への対処として、2案ある。一つは、プロセス作成時に各プロセスのアドレス位置をずらし仮想記憶空間移動後にアドレス衝突が起きないようにする方法である。もう一つは、仮想記憶空間移動後にアドレスが衝突する場合には、プロセスのアドレス位置をずらす方法である。前者は、実現が容易であるが、仮想記憶空間を利用する上での融通性に欠ける。一方、後者は、仮想記憶空間の特徴を最大限に利用できるものの、ロードモジュールはPICであることが必須になる。このため、実現機構が複雑化し、かつプログラム実行時の処理負荷も大きくなる。

3．*Tender* のメモリ管理機能とヘテロ仮想記憶

3．1　資源の分離と独立化

ＯＳは、扱う対象に識別子や名前を付与し、資源として管理している。この資源は、大きく二つに分類できる。一つは、ファイルや入出力装置のように電源断でも内容を保持できる資源である。もう一つは、プロセスやセマフォのように電源断で内容が消失する資源（以降、一時資源と名付ける）である。

既存のＯＳでは、一時資源の種類は少ない。一方、***Tender*** では、既存のＯＳの一時資源を細分割し、また新たな一時資源を導入して、多くの種類の一時資源を管理している。例えば、既存のＯＳのプロセスは、***Tender*** では「プロセス」「プログラム」「仮想ユーザ空間」「仮想空間」「仮想領域」「実メモリ」の八つの一時資源に分割されている。また、新たな一時資源として「演算」「データ」がある。以降、一時資源を単に資源と記述する。

さらに、***Tender*** では、資源を独立化している。この独立化の内容を機能とプログラム構造の観点から、以下に述べる。

（1）機能上、資源に資源識別子と資源名を付与し、かつ資源操作のインタフェースを統一している。

（2）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。また、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、***Tender*** では資源を分離し独立化しているため、以下のことが可能になっている。

（1）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できる。

（2）プログラムを部品化できる。

（3）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。

しかしながら、以下の新たな問題も抱えている。

（A）資源を扱う処理間の連携処理の負荷が増加する。

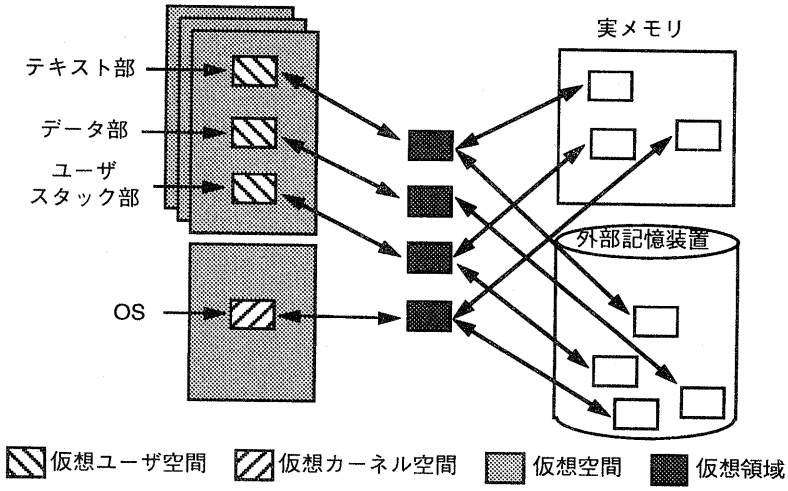

図２　メモリ管理関連の資源とプロセス

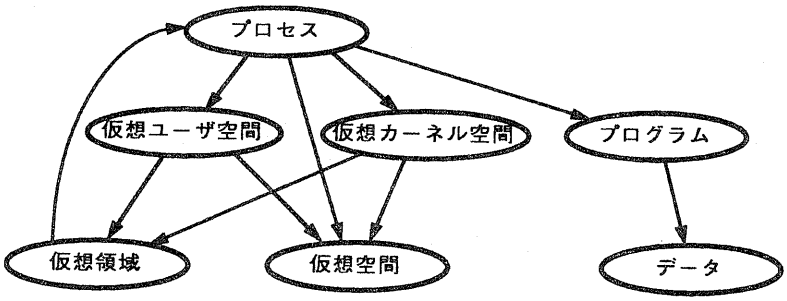

図３　プロセスが利用する資源

３．２　メモリ管理とプロセス

メモリ管理とプロセス[2]の関係について、資源の関連に注目した様子を図２に示す。図２において、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。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。さらに、仮想領域を仮想空間に「貼り付ける」ことにより、仮想カーネル空間や仮想ユーザ空間を資源として作成できる。仮想カーネル空間や仮想ユーザ空間は、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。仮想カーネル空間にはＯＳがあり、仮想ユーザ空間にはプロセスのテキスト部やデータ部やユーザスタック部がある。

次に、プロセスが利用する資源に注目した様子を図３に示す。プロセス資源は、直接的には、仮想ユーザ空間、仮想カーネル空間、プログラム、

仮想空間の資源を利用する。間接的には、仮想領域、仮想空間、データの資源を利用する。これらの資源は、独立に存在し得るので、例えば、プロセス作成時に、必ずしも新しい仮想ユーザ空間を作成する必要はなく、既に存在する仮想ユーザ空間を利用することができる。また、プロセス削除時に、必ずしも利用していた仮想ユーザ空間を削除する必要はない。

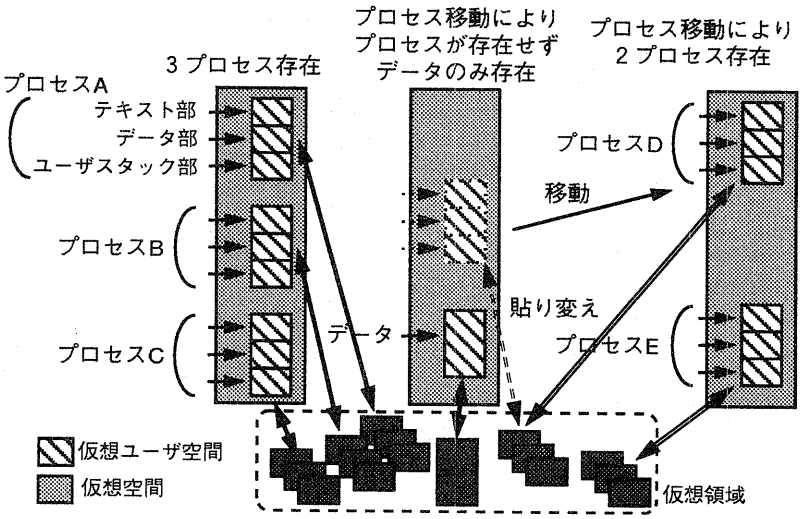

図4　ヘテロ仮想記憶とメモリ管理関連の資源

3．2　ヘテロ仮想記憶の実現

Tender では、プロセス資源とプロセスが利用するメモリ管理関連の資源は独立に存在する。また、それらの関係付けの変更は簡単である。このため、***Tender*** でのヘテロ仮想記憶の実現は、非常に容易である。その内容を以下に説明する。

（1）プロセスとは独立に、複数の仮想空間を作成でき、また必要に応じて仮想領域を仮想空間に貼り付け仮想カーネル空間や仮想ユーザ空間も資源として作成できる。

（2）一つの仮想空間上にある複数の仮想ユーザ空間を利用して、一つの仮想空間上で複数のプロセスを走行させることができる。

（3）プロセスは、利用する仮想ユーザ空間を、現在の仮想ユーザ空間から別の仮想空間上の仮想ユーザ空間に変更することにより、仮想記憶空間の間を移動できる。この変更は、仮想領域を別の仮想空間に貼り変えることだけでできる。

図1に示したヘテロ仮想記憶について、メモリ管理関連の資源も含めた様子を図4に示す。各プロセスのテキスト部とデータ部とユーザスタック部は仮想ユーザ空間に置かれている。仮想領域の貼り変えにより、プロセスDは仮想記憶空間を移動している。

プロセスの仮想記憶空間移動に伴うアドレス衝突を回避するため、プロセス作成時に各プロセスのテキスト部、データ部、ユーザスタック部のアドレス位置をずらし仮想記憶空間移動後にアドレス衝突が起きないようにした。同様に、カーネルスタックについても、アドレス位置をずらした。ただし、カーネルスタックについては、プロセス作成時に次の問題がある。各カーネルスタックが別空間にあり、そのアドレス位置が同一である場合、プロセス作成時に、子プロセスは親プロセスのカーネルスタックの内容をコピーして利用することができる。しかし、カーネルスタックの位置が異なる場合、カーネルスタック処理に関連したアドレスが格納されているため、別プロセスのカーネルスタックの内容をコピーして利用することができない。また、子プロセスを直接ユーザモードから走行させる方法もあるが、子プロセスの走行開始をＯＳ動作の可視化機能[3]から観察できない問題がある。そこで、ＯＳの開始処理におい

て最初に作成したプロセスのカーネルスタック内容を保存して利用するカーネルスタック借用法を考案し、実現した。この方法の様子を図５に示し、以下に説明する。

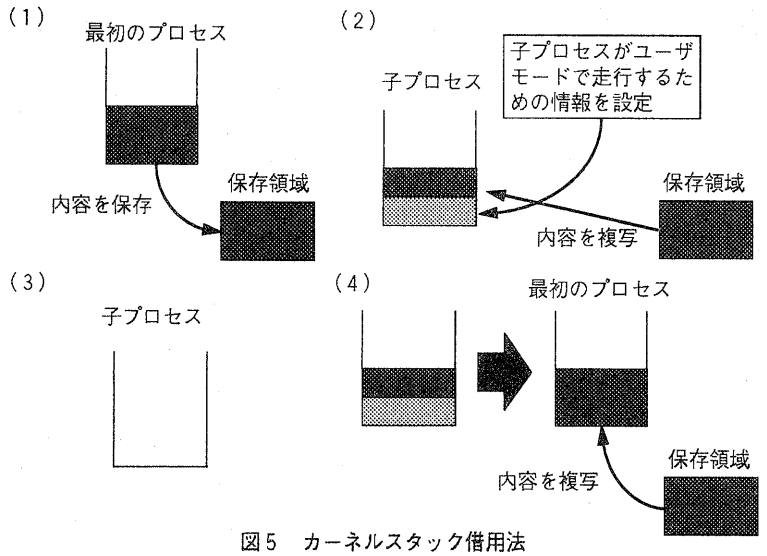

図５　カーネルスタック借用法

（１）開始処理では、ＯＳプログラムに内在する処理部分を最初のプロセスとして作成し、ユーザモードで起動させる。この最初のプロセスは、資源名"/tender/program/init"のプログラムをプロセスとして生成する。生成したプロセスへ切替時、最初のプロセスのカーネルスタックの内容を保存する。

（２）子プロセスは、最初のプロセスのカーネルスタックを借用し、保存された内容を複写し、ユーザモードで走行するための情報を設定する。

（３）子プロセスは、走行してユーザモードになり、再度カーネルモードになってカーネルスタックを必要とする時、空の新スタックを利用する。

（４）最初のプロセスが走行する時は、保存された内容を最初のプロセスのカーネルスタックに複写して、走行する。

上記（１）は、ＯＳの開始処理で行う処理である。上記（２）と（３）は、子プロセスが生成され走行開始する度に行う処理である。上記（４）は、最初のプロセスが走行開始する度に行う処理であり、一度目と二度目以降では、保存の内容や領域が異なる。この方法では、子プロセスの処理がユーザモードになる前に、新たな子プロセスの生成や最初のプロセスの走行が起こることを防ぐ必要がある。

先に述べたように、仮想領域の貼り変えだけでプロセスは仮想記憶空間を移動できるため、高速なプロセスの仮想記憶空間移動を実現した。さらに、仮想記憶空間内の同じアドレス位置へのプロセス移動をより高速化するため、仮想空間の実装を工夫した。この内容について説明する。仮想空間資源の実態は、仮想アドレスを実アドレスに変換するアドレス変換表に相当する。このアドレス変換表は２段階になっている。仮想空間の作成処理では、１段目の変換表のみを作成し、処理の高速化と利用メモリの節約を図った。仮想領域を仮想空間に貼り付ける際に、初めて、２段目の変換表を作成する。この時も、貼り付けようとする仮想領域に必要な最小限の２段目の変換表を作成し、処理の高速化と利用メモリの節約を図った。また、仮想領域を剥がす際には、２段目の変換表の内容は消去するが変換表自体は残すこととした。これにより、当該アドレス位置への仮想領域の再貼り付けが高速になり、その結果、仮想記憶空間内の同じアドレス位置へのプロセス移動が高速化される。

４．評価と考察

４．１　測定環境

Tender をプロセッサPentium90MHzの計算機上で走行させ、プロセスの作成削除時間と仮想記

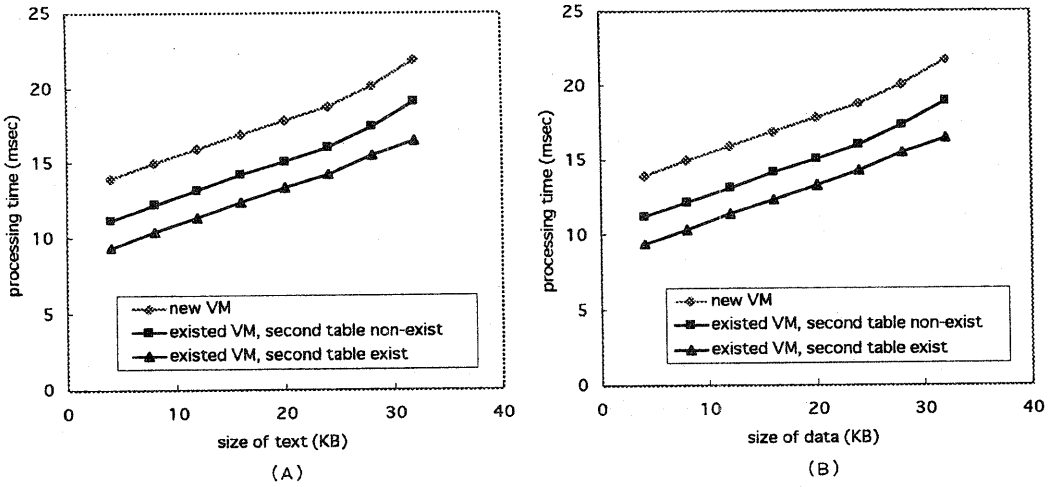

図6　プロセス作成削除時間

憶空間移動時間を測定した。ここで、プロセスの作成処理とは、プロセスの走行に必要な環境を作成する処理であり、最初のプロセッサ割り当てに関する処理を含まない。

測定は、***Tender*** の時間計測機能（単位１ミリ秒）を利用した。以降の実測値は、同じ処理を1000回行い、その平均時間である。

４．２　結果と考察

４．２．１　プロセス作成削除時間

外部記憶装置への入出力を発生させない環境において、プロセスの作成削除時間を測定した。結果を図６に示す。図６（A）は、プロセスのテキスト部の大きさと作成削除時間の関係を示しており、データ部の大きさは４ＫＢ固定である。図６（B）は、プロセスのデータ部の大きさと作成削除時間の関係を示しており、テキスト部の大きさは４ＫＢ固定である。図６から、以下のことがわかる。

（１）プロセスの作成削除時間は、テキスト部やデータ部の大きさに比例して増加する。ただし、（A）と（B）の実測値がほぼ同じであるので、その増加がテキスト部かデータ部かには依存しないといえる。

（２）新たに仮想空間を作成する場合に比べ、既存の仮想空間を利用する場合は、テキスト部やデータ部の大きさに関係なく約2.7ミリ秒速い。さらに、既存の仮想空間の２段目の変換表が存在する場合は、存在しない場合に比べ約1.8ミリ秒速い。

したがって、新たに仮想空間を作成する場合に比べ既存の仮想空間を利用することにより、プロセスの作成削除時間を削減でき、プロセスの大きさに関係なくプロセッサ処理を約4.5ミリ秒削減できるといえる。

４．２．２　仮想記憶空間移動時間

プロセスが仮想記憶空間の間を移動する時間を測定した。結果を図７に示す。図７（A）は、プロセスのテキスト部の大きさと仮想記憶空間移動時間の関係を示しており、データ部の大きさは４ＫＢ固定である。図７（B）は、プロセスのデータ部の大きさと仮想記憶空間移動時間の関係を示しており、テキスト部の大きさは４ＫＢ固定である。図７から、以下のことがわかる。

（１）プロセスの仮想記憶空間移動時間は、テキスト部やデータ部の大きさに比例して増加するものの、その増加の割合は非常に小さくほぼ一定である。新たに仮想空間を作成して移動する場合が最も大きく、約5.4ミリ秒である。また、（A）

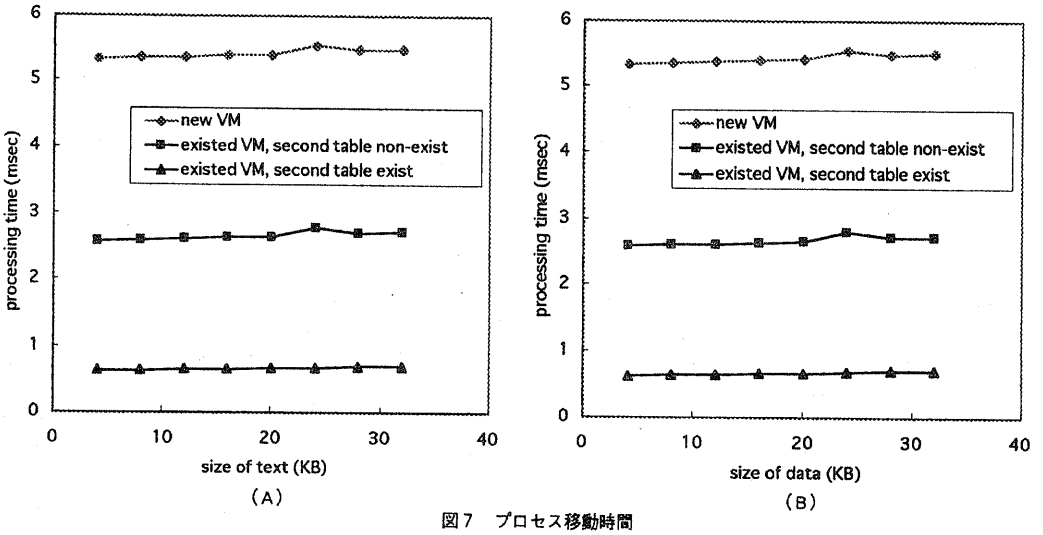

図7　プロセス移動時間

と（B）の実測値がほぼ同じであるので、その増加がテキスト部かデータ部かには依存しないといえる。

（2）新たに仮想空間を作成して移動する場合に比べ、既存の仮想空間に移動する場合は、約2.7ミリ秒速い。さらに、既存の仮想空間の2段目の変換表が存在する場合は、存在しない場合に比べ約2ミリ秒速い。

したがって、プロセスの仮想記憶空間移動時間は、移動するプロセスの大きさの影響をほとんど受けず、新たに仮想空間を作成して移動する場合でも約5.4ミリ秒である。さらに、最速な場合には、わずか0.7ミリ秒程度で仮想記憶空間の間を移動できることがわかる。

5．おわりに

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提案し、***Tender*** オペレーティングシステムでの実装内容と評価結果について述べた。ヘテロ仮想記憶は、複数の仮想記憶空間を有し、一つの仮想記憶空間内には0個以上のプロセスが存在でき、プロセスには仮想記憶空間の間を移動できる機能を提供する。これにより、単一仮想記憶と多重仮想記憶の長所を最大限に引き出して利用することが可能になる。***Tender*** オペレーティングシステムでは、資源を分離し独立化しているため、ヘテロ仮想記憶の実現が容易であり、かつ効率も良い。実測評価の結果、プロセスの作成削除時間は、既存の仮想空間を利用することにより、プロセスの大きさに関係なくプロセッサ処理を約4.5ミリ秒削減できた。また、プロセスの仮想記憶空間移動時間は、移動するプロセスの大きさの影響をほとんど受けず、新たに仮想空間を作成して移動する場合でも約5.4ミリ秒である。さらに、最速な場合には、わずか0.7ミリ秒程度で仮想記憶空間の間を移動できることを示した。

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，田中徳穂：“***Tender*** のプロセス管理構造”，情処研報，Vol.96, No.79, pp.109-114 (1996).

[3] 野口裕介，谷口秀夫，牛島和夫：“ＯＳ動作の可視化機能の設計”，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.96, No.7, pp.139-146(1996).