社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2006－OS－102（5）
2006／5／12

ヘテロ仮想記憶（HVS）の評価

田端利宏†　谷口秀夫†

我々は，単一仮想記憶と多重仮想記憶を統合したヘテロ仮想記憶の機能を提案している．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る．また，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．本論文では，ヘテロ仮想記憶を利用した場合の有効事例を示すために，OS の基本機能の性能評価結果を報告する．また，Apache Web サーバで，ヘテロ仮想記憶の機能を利用するための実装方式とその評価結果を示し，有効性を明らかにする．

Evaluation of Heterogeneous Virtual Storage (HVS)

Toshihiro Tabata† and Hideo Taniguchi†

We have proposed Heterogeneous Virtual Storage (HVS) that is integrated both single virtual storage and multiple virtual storage. HVS has multiple virtual storage spaces. The number of processes on a virtual storage space is more than zero. Besides, a process can migrate between virtual storage spaces. Therefore, HVS has both single virtual storage's advantages and multiple virtual storage's advantages. In this paper, we report the evaluation of basic functions in OS in order to make clear the application of HVS functions. In addition, we implemented some functions in Apache Web server in order to use the functions of HVS, and describe the performance of Apache.

1. はじめに

多くの OS は，多重仮想記憶を採用しており，プロセス毎に独立した仮想記憶空間を提供している．多重仮想記憶では，プロセス毎に一つの仮想記憶空間を与えることで，プロセス間の保護を実現している．一方で，プロセス間の協調処理の際には，仮想記憶空間の切り替えの多発によるオーバヘッドを伴う．現在の多くサービスは複数プロセスで構成され，それらが緊密に連携を取りながら処理を行う．このようなサービスをさらに高速化するには，緊密な連携処理をするプロセス間処理のオーバヘッドを取り除くことが考えられる．

プロセス間連携処理のオーバヘッドを削減するために，スレッドが実現されている．スレッドを利用することにより，一つのプロセスの中で複数のスレッドがメモリ空間を共有でき，かつスレッド切り替えのオーバヘッドを削減できる．しかし，同一の仮想記憶空間で連携できるスレッドは，同一プロセス内に限られており，異なるプログラムから生成されたプロセス間での協調処理を高速化することはできない．

そこで，我々は，単一仮想記憶と多重仮想記憶を混在させたヘテロ仮想記憶（Heterogeneous Virtual Storage，以降，HVS と略す）を提案している [1)]．HVS は，複数の仮想記憶空間を提供し，一つの仮想記憶空間に 0 個以上のプロセスが存在する機能も提供する．これにより，複数のプロセスを同一の仮想記憶空間で実行することが可能となり，プロセス間の協調処理のオーバヘッドを減らすことができる．また，任意の仮想記憶空間にプロセスを配置する手段として，プロセス移動機能と任意の仮想記憶空間へのプロセス生成機能を提供する．これにより，協調処理の頻度に合わせて自由にプロセスが存在する仮想記憶空間を選ぶことができる．さらに，既存の仮想記憶空間にプロセスを生成することで，プロセス生成処理を高速化できる．このように，HVS を実現した OS では，一つの計算機に単一仮想記憶と多重仮想記憶を混在させることで，応用プログラム（以降，AP と略す）間の協調処理の頻度や処理形態に合わせて，仮想記憶空間を利用することが可能になる．

文献 1) では，HVS の実装と，プロセスの生成削除

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

と移動処理の評価のみ行っており，TLB フラッシュによる影響や AP による評価を行っていない．そこで，本論文では，HVS の特徴と利点について述べ，HVS の有効事例を明らかにするために行った OS 機能の評価結果を報告する．また，Apache Web サーバで HVS の機能を利用するための実装方式とその評価結果について述べる．

2. 関 連 研 究

既存の OS では，単一仮想記憶あるいは多重仮想記憶が実現されている．単一仮想記憶では，一つの仮想記憶空間を複数のプロセスで共有するため，プロセス毎のメモリ領域の保護が必要となる．Opal[2] は, 単一アドレス空間を提供する OS であり, プロセス間におけるデータの共有と保護をドメインと名づける概念により実現している．多重仮想記憶を実現している代表的な OS として，UNIX[3] がある．UNIX では, 共有メモリ機能により，プロセス間のデータ共有を可能にしている．

単一仮想記憶を実現する OS では，アドレスが連続した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．このため，プロセス間通信処理において仮想記憶空間の切り替えを必要とせず，処理の効率がよい．一方，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占有して利用する．多重仮想記憶では，プロセス毎に仮想記憶空間が異なるため，プロセス間通信においては，プロセス切り替えが必須である．既存 OS は，仮想記憶モデルは上記のいずれか一方を採用している．

一方，我々は，両方の仮想記憶モデルの混在を許す HVS を提案している．

また，プロセス間処理を効率よく行うために，スレッドが多くの OS で実現されている．スレッドは，一つのプロセス内での協調処理を効率よく行えるものの，仮想記憶空間を共有するため，スレッド間の保護が難しく，デバッグも難しいという問題がある．一方，HVS では，異なるプロセスを一つの仮想記憶空間に配置することができ，スレッドと同様にプロセス間でも処理の効率化を実現できる．スレッドとの大きな違いは，異なるプログラム間での協調処理を効率化できることと，状況に応じて仮想記憶空間を共有するかどうかを決定できることである．

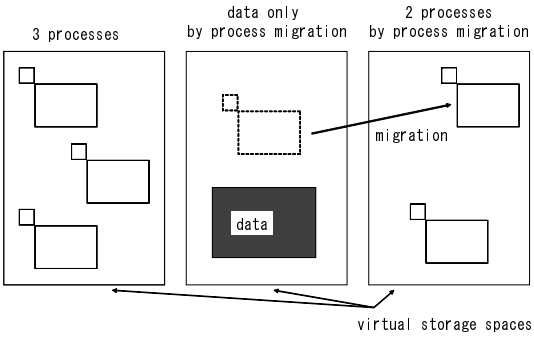

図 1 ヘテロ仮想記憶

3. ヘテロ仮想記憶

3.1 概　　要

HVS の概要を図 1 に示す．HVS とは，

（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，

（機能 4） プロセスを既存の仮想記憶空間に生成できる

機能を持つ仮想記憶モデルである．

（機能 1）に示すように，複数の仮想記憶空間を提供することで，多重仮想記憶と同様にプロセス毎に仮想記憶空間を提供することができる．（機能 2）に示すように，一つの仮想記憶空間に複数のプロセスを存在させることで，単一仮想記憶のように一つの仮想記憶空間を複数のプロセスで共有することもできる．また，（機能 3）に示すように，プロセス移動を実現することで，実行中のプロセスの仮想記憶モデルを単一仮想記憶と多重仮想記憶から選択できる．（機能 4）に示すように，既存の仮想記憶空間を利用してプロセスを生成することで，プロセスの生成処理を高速化できる．

つまり，他プロセスと緊密に協調処理を行う場合は，そのプロセスと同じ仮想記憶空間に移動することで，プロセス協調処理のオーバヘッドを削減できる．また，そのプロセスとの協調処理が終われば，別の仮想記憶空間に移動することで，自分の仮想記憶空間を自由に利用できる．

3.2 利　　点

プロセス間で通信を頻繁に行う場合，協調処理を行うプロセスを同一仮想記憶空間に配置することで，プロセス切り替え時の仮想記憶空間の切り替えを削減できる．これにより，走行プロセスを同一仮想記憶空間に存在するプロセスに切り替える場合，TLB をフラッシュすることなく走行プロセスを切り替えることができる．この結果，TLB ヒットミスを減少させることができる．また，スレッドモデルと同様に，プロセス

間で同じ仮想記憶空間に存在するデータを共有することができる．

緊密に処理を行うプロセスを同一仮想記憶空間の配置する方法として，別の仮想記憶空間に存在するプロセスを移動させる方法と，プロセスが存在する仮想記憶空間に新たにプロセスを生成する方法がある．これらの方法により，処理をより効率よく行うために，プロセスを配置する仮想記憶空間を選択できる．

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，データベースのように大きいデータを複数の小さいプロセスが共有し，時分割で排他して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．さらに，大きなデータの存在する仮想記憶空間を複数のプロセスで共有することで，プロセス毎にページ変換表を用意するのに比べて，使用するメモリ量を削減できる利点もある．文献 4) では，複数のプロセスでページ変換表を共有することで，メモリの消費を削減できることが示されている．

3.3 課題と対処

上記の特徴を生かした HVS を実現するには，以下の課題がある．

（課題 1） 同一仮想記憶空間への複数プロセス配置の実現

（課題 2） プロセスの存在しない仮想記憶空間の管理

（課題 3） 仮想記憶空間間のプロセス移動の実現

（課題 4） 既存仮想記憶空間へのプロセス生成の実現

（課題 5） プロセス間保護の実現

（課題 1）では，同一仮想記憶空間に存在するプロセスのプログラムアドレスが衝突することが問題となる．従来 OS では，仮想記憶空間は独立して管理されていなかったが,（課題 2）では，仮想記憶空間の管理方法が問題となる.（課題 3）と（課題 4）では，処理の結果として，同一の仮想記憶空間に複数のプロセスを配置できるため,（課題 1）と同様にプログラムアドレスの衝突が問題となる.（課題 5）では，同一仮想記憶空間に存在する各プロセスを安全に実行するために，保護機構をいかに実現するかという問題がある．

3.3.1 プログラムアドレス

プログラムを静的にリンクする場合，プログラムのアドレスはコンパイル時に静的に割り当てられる．このため，同一仮想記憶空間に複数のプロセスを配置した場合，アドレス位置が衝突する．これに対し，自由なアドレスに配置できる PIC（Position Independent Code）がある．実行時に自由にプロセス移動ができる点では，アドレスを自由に配置できる PIC の方がよい．

3.3.2 仮想記憶空間の管理

既存 OS では，多重仮想記憶を採用しているため，仮想記憶空間の情報はプロセス管理表に格納されている．このため，仮想記憶空間の存在は，プロセスの存在が前提となっている．HVS は，仮想記憶空間に存在するプロセスは 0 個以上なので，多重仮想記憶と同じ管理法では，仮想記憶空間にプロセスが存在しない場合に問題が生じる．このため，プロセスと仮想記憶空間を分離してそれぞれ管理する必要がある．つまり，仮想記憶空間にも ID，名前，所有者の概念を持たせ，プロセスとは独立して管理する．

3.3.3 同一仮想記憶空間に存在するプロセス間保護

同一仮想記憶空間に存在するプロセス間には，何らかの保護機構が必要である．同一の仮想記憶空間で実行するプロセスは，利用者が信頼できるプログラムであると仮定すれば，信頼できないプログラムを同一の仮想記憶空間に配置できないように制御できればよい．仮想記憶空間上にプロセスを配置する方法として，プロセス生成とプロセス移動がある．また，同一仮想記憶空間内でのメモリ保護を必要とする場合には，仮想記憶空間内で保護ドメインを実装する方法もある [5]．

3.4 機　　能

HVS の処理に関連する 5 つのカーネルコールを表 1 に示す．HVS に関連する仮想記憶空間を操作するカーネルコールと資源の操作権を制御するカーネルコールがある．

vmcreate は，新たに仮想記憶空間の生成するカーネルコールである．これにより，プロセスの存在しない仮想記憶空間を生成できる．

proccreate は，仮想記憶空間を指定してプロセスを生成するカーネルコールである．これにより，任意の仮想記憶空間にプロセスを配置できる．

procmove は，プロセスを任意の仮想記憶空間に移動させるカーネルコールである．これにより，プロセスは操作権の許す範囲で自由に仮想記憶空間の間を移動できる．

changeoid は，資源の所有者を変更するカーネルコールである．これにより，仮想記憶空間の所有者を設定できる．

表 1　HVS 関連のカーネルコール

No.	形式	機能
1	vmcreate(name, mod)	名前 name を持つ仮想記憶空間を新たに生成する．mod には，操作権を設定する．
2	procmove(pid, vmid)	プロセス pid を仮想記憶空間 vmid に移動させる．
3	proccreate(name, path, argv, vmid, mod)	パス名 path で指定されたプログラムを内容として持ち、データ部に引数 argv を読み込んだプロセスを提供する。プロセスには名前 name が付与される．vmid が 0 のときは，新たに仮想記憶空間を生成し，その空間にプロセスを生成する．vmid が指定されている場合は，指定された仮想記憶空間にプロセスを生成する．mod には，操作権を設定する．
4	changeoid(rid, oid, flag)	資源 rid の所有者を oid に変更する．資源 rid を構成するすべての資源の所有者も変更する場合は，0 以外を指定する．
5	changemod(rid, mod)	資源 rid の操作権を引数 mod に変更する．

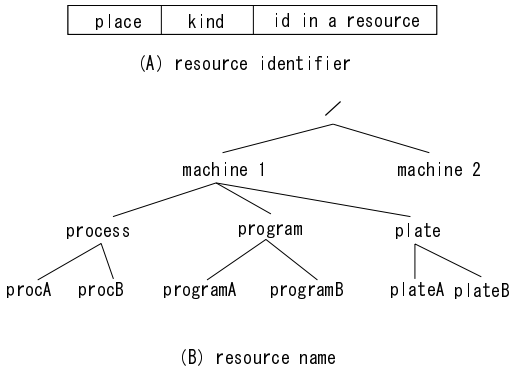

図 2　資源識別子と資源名

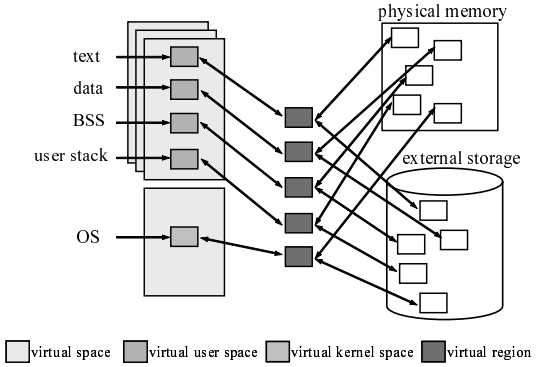

図 3　*Tender* のメモリ資源

changemod は，資源の操作権を変更するカーネルコールである．これにより，仮想記憶空間毎にプロセスの移動や生成の可否を設定できる

4. 実　　装

HVS を資源の分離と独立化を実現した ***Tender*** オペレーティングシステム [6] に実装した．本章では，***Tender*** の概要を説明し，HVS の実装について述べる．

4.1 *Tender* オペレーティングシステム

4.1.1 資源の分離と独立化

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．これらの資源には，図 2 に示すように，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．

4.1.2 メモリ管理とプロセス

メモリ管理とプロセスの関係について，資源の関連に注目した様子を図 3 に示す．図 3 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．

4.1.3 資源操作権制御 (RCC)

資源を保護するには，資源に対するアクセス制御機能が必要である．***Tender*** では，資源に対するアクセス制御機能（資源操作権制御 [7] と呼ぶ）を利用できる．資源には，5 つの操作があり，各操作の可否を制御できる．文献 7) では，所有者とそれ以外に対して操作権を設定できるが，新たにグループの概念を導入する．これにより，同一所有者間またはグループ間にだけ，仮想記憶空間の共有，プロセス移動，及びプロセス間通信を許可することができる．

4.2 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．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 HVS 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．

4.3　課題への対処

3.3 節で述べた課題への対処を述べる．

(1)　プログラムアドレスについては，HVS の各機能の評価と特定の AP での評価を行うことが目的であるため，実装に要する工数を考慮し，静的に異なるアドレスにリンクする方式を採用した．

(2)　***Tender*** では，仮想記憶空間を資源化し，仮想空間と名付け，独立化している．この仮想空間に所有者の概念を与え，管理することとした．

(3)　保護機能として，仮想記憶空間の所有者が，所有する仮想記憶空間へのプロセス生成とプロセス移動操作の可否を制御できる機能を用意した．これにより，プロセスの所有者が適切に仮想記憶空間のアクセス制御を行えば，信頼できないプロセスが同一の仮想記憶空間に存在することを防ぐことができる．

5.　評　　価

5.1　評 価 内 容

HVS の特徴と有効事例を明らかにするために，以下に示す OS の基本機能の評価結果を示す．

(1)　プロセス間通信

プロセス間処理において，配置された仮想記憶空間の違いがどの程度処理時間に影響を与えるのかを明らかにする．

(2)　プロセス移動

プロセスの再配置に要するオーバヘッドを明らかにする．

(3)　プロセス生成

既存の仮想記憶空間を利用することによるプロセス生成の高速化の程度を明らかにする．

(4)　プロセス切り替え

仮想記憶空間の切り替えが，どの程度プロセス切り替え処理に影響を与えるのかを明らかにする．

また，実 AP として，Apache Web サーバで HVS の機能を活用した場合の評価結果を報告する．なお，以降の基本機能の評価は，PentiumIII 750MHz の計算機で行った．

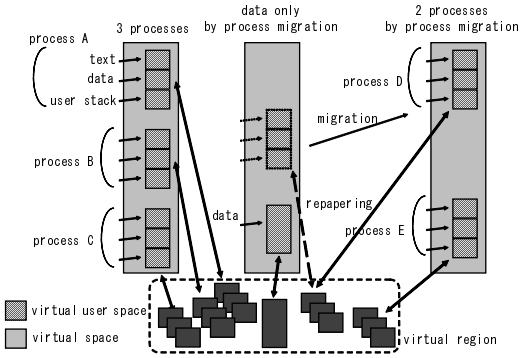

図 4　Resources used by processes.

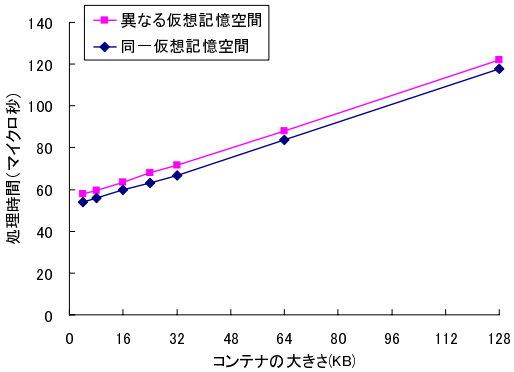

図 5　プロセス間通信の処理時間

5.2　基 本 機 能

5.2.1　プロセス間通信

Tender のメッセージ渡し方式のプロセス間通信機能 [8] を利用して，二つのプロセス間でのメッセージ（***Tender*** ではコンテナと呼ぶ）の送受信処理の時間を測定した．処理内容は，コンテナボックス（メッセージボックスに相当する）を生成し，一方のプロセスから送信したコンテナ（メッセージに相当する）を交互に送受信する．この処理を 100 往復繰り返したときの 1 往復当たりの処理時間を図 5 に示す．測定は，送受信プロセスが同一仮想記憶空間に存在する場合と，異なる仮想記憶空間に存在する場合で行った．

図 5 から，同一仮想記憶空間に存在する場合の方が，

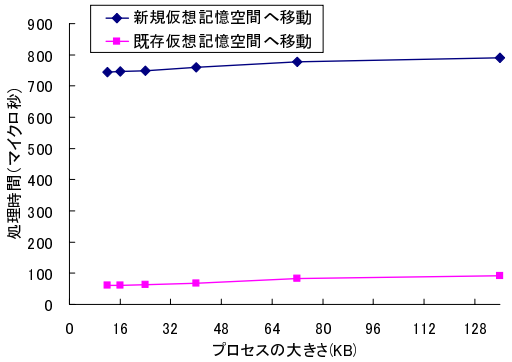

図 6　プロセス移動の処理時間

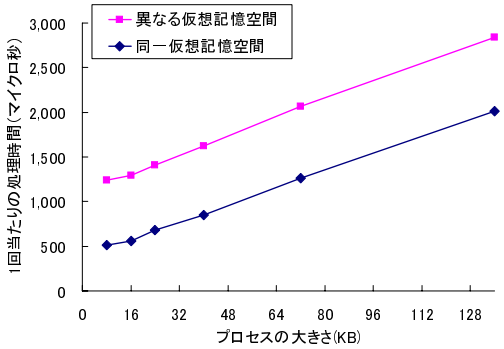

図 7　プロセス生成の処理時間

異なる仮想記憶空間に存在する場合に比べて，処理時間が約 4 マイクロ秒短いことがわかる．この処理時間の差は，コンテナのサイズに関係なくほぼ一定である．

この処理の差は，プロセス切り替え処理における仮想記憶空間の切り替え処理の有無である．つまり，同一仮想記憶空間内に存在するプロセスに走行プロセスを切り替える場合，TLB はフラッシュされない．このため，TLB のヒットミス回数を削減でき，処理を効率化できていると推察できる．

5.2.2　プロセス移動

HVS によって可能となるローカル計算機内でのプロセス移動について，その処理時間を測定した．プロセス移動対象の仮想記憶空間として，新たに仮想記憶空間を生成して移動する場合と，既存の仮想記憶空間に移動する場合を評価した．図 6 にその測定結果を示す．

既存仮想記憶空間への移動結果から，プロセス移動そのものの処理時間は，100 マイクロ秒未満である．この処理時間は，プロセスの大きさに比例する．また，新たに仮想記憶空間を生成して移動する場合は，その生成処理に 700 マイクロ秒程度を要することがわかる．

5.2.3　プロセス生成

HVS が提供する既存の仮想記憶空間へのプロセス生成を測定した結果を図 7 に示す．測定結果から，既存の仮想記憶空間を利用してプロセスを生成することにより，約 700〜800 マイクロ秒だけ高速化できていることがわかる．このことから，既存の仮想記憶空間を利用する効果は大きいといえる．なお，***Tender*** では，プロセスの生成時に実メモリを確保し，プロセスの内容をすべて読み込むため，プロセス生成処理時間はプロセスの大きさに比例している．

5.2.4　プロセス切り替え

同一仮想記憶空間間と異なる仮想記憶空間間でのプロセス切り替え処理において，処理時間差を測定した．

表 2　プロセス切り替え処理

プロセスの配置	処理時間（マイクロ秒）
同一仮想記憶空間	4.885
異なる仮想記憶空間	6.102

測定では，二つのプロセスが，プロセス切り替えカーネルコールを交互に発行し，連続してプロセス切り替え処理を行う．1000 回プロセス切り替え処理を連続して行った場合の 1 回当たりの処理時間を表 2 に示す．表 2 から，同一仮想記憶空間に存在するプロセス間でのプロセス切り替え処理は，約 20%処理時間が短くなっていることがわかる．これは，同一仮想記憶空間間でのプロセス切り替えの場合，TLB フラッシュが不要になり，その分だけ処理が効率化されていると推察できる．

5.2.5　有 効 事 例

以上の結果から，HVS が提供する各機能の有効事例を挙げる．

(1)　プロセス間通信の高速化の結果から，同一計算機内で頻繁にデータの送受信を行うようなサービスにおいて有効である．

(2)　既存の仮想記憶空間に移動する処理は，おおよそ 100 マイクロ秒未満である．このため，頻繁にデータの送受信を行うようなプロセスは，通信相手プロセスが存在する仮想記憶空間に移動することで，プロセス間通信のオーバヘッドを削減でき，サービス処理性能を向上させることが見込める．この場合，プロセス間通信の高速化の効果，通信頻度，及びプロセス移動のコストを考慮して，移動の可否を判断する必要がある．

(3)　プロセス生成処理の結果から，協調処理を行うプロセスを，最初から通信相手プロセスと同じ仮想記憶空間に生成することにより，プロセス生成処理とプロセス間通信処理を高速化できることがわかる．

(4)　プロセス切り替え処理の結果から，プロセス間

通信を行わない場合でも，同一仮想記憶空間に存在するプロセスに走行プロセスを切り替える処理を高速化できることがわかる．

5.3 Web サーバでの評価

5.3.1 処理内容

5.2.5 項で述べた有効事例に当てはまる処理として，Apache Web サーバにおける CGI プログラムの実行処理を用いて，HVS の機能を評価した．CGI プログラムを実行する際には，新たにプロセスを生成する必要があり，既存の仮想記憶空間に生成することで，処理を高速化できる．また，Apache のプロセスは，CGI プログラムの実行を生成後，その処理が終了するのを待つため，プロセス切り替えが発生する．この二つのプロセスを同一仮想記憶空間に配置することで，プロセス切り替え処理も高速化できる．

評価では，クライアント計算機（Pentium4 2GHz）で ApacheBench を実行し，Web サーバ（Apache 1.3.33，PentiumIII 750MHz の計算機で実行）にアクセスしたときの 1 回当たりの平均応答時間を求めた．Apache Bench の並列アクセス数を 4〜20 まで変更した．各計算機は，100Mbps のイーサネットで接続されている．ApacheBench にアクセスされた Web ページでは，アクセスがある毎に一つの perl スクリプトを実行する．このスクリプトは，アクセスカウンタのファイルをオープンして，数値を一つインクリメントする簡単なスクリプトである．

5.3.2 実装内容

Tender には，Apache を動作させるため BSD/OS 互換システムコールインタフェースが実装されており [9]，BSD/OS の実行ファイルをそのまま利用できる．しかし，fork と exec を利用したプロセス生成方式では，HVS の機能を利用できない．そこで，この測定では，新たに以降で述べる二つの CGI プログラムの実行方式を実装した．

BSD/OS における Apache の CGI プログラムの実行処理では，図 8 に示すように fork システムコールと execve システムコールが各 2 回実行される（以降，この方式を UNIX 互換方式と呼ぶ）．このため，生成されたプロセスは，親プロセスとは別の仮想記憶空間で実行される．したがって，このままでは HVS の利点である仮想記憶空間の共有による協調処理の効率化と，既存の仮想記憶空間を利用したプロセス生成の高速化が実現できない．

そこで，HVS の機能を活用するために，表 3 に示す bsd_proccreate システムコールを追加した．このシステムコールを利用すると，図 9 に示すように 2 回のシステムコールを実行するだけで CGI プログラムを実行できる．CGI プログラム実行するとき，最初に fork システムコールを実行する．この後で，親プロセスは，pipe を利用するためにファイルディスクリプタの処理を行う．子プロセス側でも，処理の継続に必要な処理を行い，UNIX 互換方式で httpd プロセスが execve システムコールを発行する処理を，bsd_proccreate システムコールを発行する処理に置き換える．この結果，bsd_proccreate システムコールは，内部では ***Tender*** 独自のプロセス生成の内部処理を呼び出して，プロセスを生成し，その後で BSD/OS

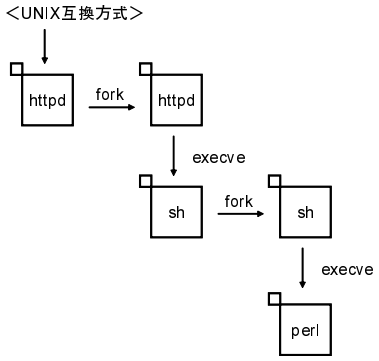

図 8 UNIX 互換方式でのプロセス生成処理

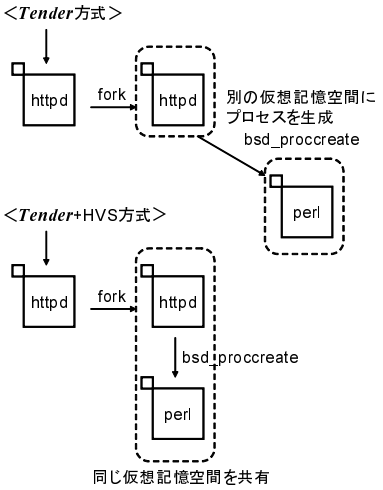

図 9 ***Tender*** でのプロセス生成処理

表 3 仮想空間指定でプロセスを生成するシステムコール

形式	機能
bsd_proccreate(path, argc, args, vmid)	プログラム path から，仮想空間 vmid にプロセスを生成する．プロセスには，親プロセスから必要なプロセス構造体の情報を複写する．また，引数の数 argc と引数へのポインタ args を設定する．

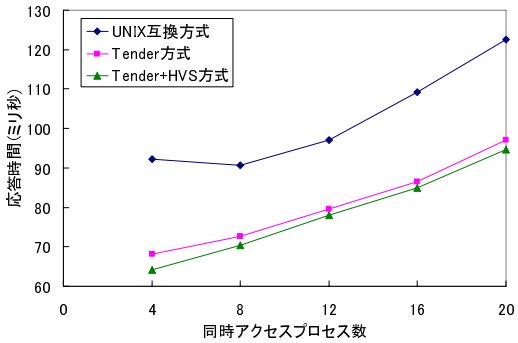

図 10 Web サーバの応答時間

プロセスをエミュレーションするために必要なプロセス構造体の情報（ファイルディスクリプタの情報など）を親プロセスから複写する処理を行う．このようにして，CGI プログラムを実行する処理を ***Tender*** 独自のインタフェースに合うように実装し直している．この結果，プロセス生成の内部関数を呼び出すときに，プロセスをどの仮想記憶空間に生成するのかを指定することができる．新たに仮想記憶空間を生成し，そこにプロセスを生成する場合を ***Tender*** 方式，既存の仮想記憶空間（bsd_proccreate システムコールを発行した httpd プロセスの仮想記憶空間）にプロセスを生成する場合を ***Tender***+HVS 方式と以降では呼ぶ．

5.3.3 測定結果

ApacheBench で測定した 1 回当たりの Web サーバの応答時間を図 10 に示す．

(1) ***Tender***+HVS 方式は，***Tender*** 方式よりも，2%〜6%ほど応答時間が短いことがわかる．これは，perl プロセス生成時に既存の仮想記憶空間を利用することと，同一仮想記憶空間に存在する httpd プロセスと perl プロセス間でのプロセス切り替え処理では，TLB フラッシュが起こらないためであると推察できる．

(2) いずれの場合も，同時アクセスプロセス数が多くなるほど，応答時間が長くなる傾向がある．

(3) UNIX 互換方式は，プロセス生成に関するシステムコールの実行回数が多い分だけ，応答時間が長くなっている．

以上の結果から，HVS の有効事例に当てはまる AP であれば，その機能を活用することにより，処理を効率化できるといえる．

6. おわりに

本論文では，単一仮想記憶と多重仮想記憶の特徴を合わせ持つヘテロ仮想記憶について述べ，その評価結果と有効事例を報告した．評価では，プロセス間通信，プロセス移動，プロセス生成，及びプロセス切り替え処理について評価した．この結果から，ヘテロ仮想記憶の利点として挙げたプロセス間での協調処理の効率化が可能であることを示し，有効事例を示した．また，有効事例の一つとして，Apache Web サーバにおける CGI プログラムの実行処理を評価した．評価の結果，HVS の機能を活用することで Web サーバの応答時間を 2%〜6%短縮できることを確認した．

謝辞 ***Tender*** のメモリ管理の実装と試験にご協力頂いた長嶋直希氏，Apache Web サーバの実装にご協力いただいた岡山大学工学部情報工学科の島津奈美氏に感謝します．

参考文献

1) 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，情報処理学会研究報告，1998-OS-078，Vol.98，No.33．pp.87–94 (1998)．
2) Jeffrey S. C., Henry M. L., Michael J. F., and Edward D. J. : Sharing and Protection in a Single-Address-Space Operating System, ACM Trans. on Computer Systems, Vol.12, No.4, pp.271-307 (1994).
3) M. K. MacKusick, K. Bostic, M. J. Karels, J. S. Quarterman, The Design and Implementation of the 4.4BSD Operating System, Addison-Wesley (2003).
4) Young-Woong KO, Chuck YOO, Shared Page Table: Sharing of Virtual Memory Resources, IEICE Transactions, Vol.E86-D, No.1, pp.45–55 (2003).
5) 品川高廣，河野健二，高橋雅彦，益田隆司：拡張コンポーネントのためのカーネルによる細粒度軽量保護ドメインの実現，情報処理学会論文誌，Vol.40，No.6，pp.2596–2606 (1999)．
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) 山本　淳，谷口秀夫：***Tender*** における統一的な資源操作権制御方式，” 情報処理学会 第 63 回全国大会講演論文集 (分冊 1)，pp.81–82 (2001)．
8) 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設計と実現，情報処理学会研究報告，1999-OS-081，Vol.99，No.32，pp.95–100 (1999)．
9) 田端利宏，野口直樹，中島耕太，谷口秀夫：***Tender*** における異種 OS インタフェースの共存手法，情報処理学会 第 64 回全国大会，3U-3 (2002)．