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

2007－OS－105 (7)
2007／4／5

Gandalf VMMにおけるShadow Pagingの設計と実装

伊藤 愛† 追川修一†

近年，マルチコアプロセッサを搭載したマシンは一般化しつつある．組み込みシステムにおいても，マルチコアプロセッサを搭載したシステムが増加している．このようなシステムでVMMを動作させることによって，資源の効率利用，安全性の向上，信頼性の向上を実現することができる．これまで，マルチコアプロセッサ指向の軽量VMMとして，Gandalfを設計，実装してきた．本論文では，ゲストOS間のメモリ保護を実現するシャドウページングについて述べる．シャドウページングを利用することで，VMMがゲストOSのメモリ利用を監視することができる．2方式のシャドウページングを設計し，実装を行った．それぞれの方式について評価実験を行い，シャドウページングの有無や方式の違いによるコスト差を比較し，考察を行う．

Design and Implementation of Shadow Paging on Gandalf VMM

Megumi Ito† and Shuichi Oikawa†

Recently, the use of multi-core processors is increasing. Many multi-core processors are employed by embedded systems. By using VMMs in embedded systems with multi-core processors, we can effectively utilize the resources, improve safety and reliability. We designed and implemented a multi-core processor-oriented lightweight VMM, Gandalf. This paper focuses on shadow paging, which enables memory protection among guest OSes. A VMM can monitor the use of memory by guest OSes through shadow paging. We designed and implemented the two models of shadow paging. We compare and discuss the costs between these models by the results from benchmark experiments.

1. はじめに

近年，マルチコアプロセッサを搭載したマシンは一般化しつつある．組み込みシステムにおいても，マルチコアプロセッサを搭載したシステムが増加している．このようなシステムでVMM（Virtual Machine Monitor）を動作させることによって，資源の効率利用，安全性の向上，信頼性の向上を実現することができる．VMMを用いて1つのマシンで複数のOS（Operating System）を同時に実行させることによって，計算機資源を有効に活用することができる．VMMによってOS同士は隔離されて実行されるので，アプリケーションの信頼度に応じて動作させるOSを分け，悪意のあるプログラムから情報を保護することで安全性を高めることができる．また，実行中のOSに障害が発生したときにすぐ切り替えられるように代替OSを実行させておくことによって，可用性の高いシステムを構築することもできる．

組み込みシステムで仮想化技術を利用する上で，重要な要求として大きく2つが挙げられる．まず，VMM自体の軽量化の要求である．VMMを通したOSの実行にコストがかかりすぎてしまうと，1つのマシンで複数のOSを同時に実行させることができるというメリットが失われてしまう．また，OSの改変をできるだけ行わないで動作させたいという要求がある．少ない移植コストでOSを実行できるということは，今まで開発してきたプログラム資源の有効活用につながり，また，OSのバージョンアップに対応するのも容易になる．

これらの要求を受け，本研究ではこれまで，マルチコアプロセッサ指向の軽量VMMとして，Gandalfを開発してきた．プロセッサとしてIA-32アーキテクチャ，Gandalf上で動作させるゲストOSとしてLinuxを対象として実装を行っている．現在，2つのプロセッサ上で2つのゲストOSが動作している．Gandalf上でゲストOSを動作させたときの性能はXen上で動作させたときよりも良いものとなっており，また，ゲストOSは最低限の書き換えを加えるだけで動作することができる．

本論文では，ゲストOS間のメモリ保護を実現するシャドウページングについて述べる．シャドウページングでは，ゲストOSが使用するページテーブルと

† 筑波大学大学院システム情報工学研究科
Department of Computer Science, University of Tsukuba

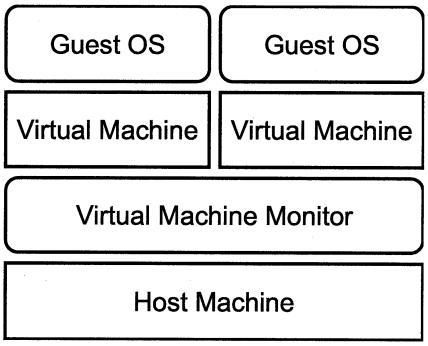

図 1 VMM を含むシステムの構造図
Fig. 1 Structure of a Virtual Machine Monitor based System

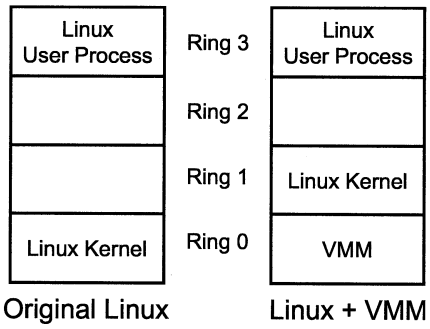

図 2 特権レベルの利用図
Fig. 2 Privilege Level Usage

は別に VMM 内にシャドウページテーブルを用意し，CPU は実際にはシャドウページテーブルを参照する．CPU が実際に参照するページテーブルは VMM 内に存在し，この内容を操作できるのは VMM のみであるので，シャドウページングを利用することで，VMM がゲスト OS のメモリ利用を監視することができる．また，ゲスト OS 内のページテーブルを仮想ページテーブルとして扱うことによって，メモリの仮想化を実現することもできる．

本研究では，Single Shadow Page Table 方式と Multiple Shadow Page Table 方式の 2 方式のシャドウページングを設計し，実装を行った．Multiple Shadow Page Table 方式では，ページテーブル用のメモリを再利用するための Reclamation Policy として FIFO 方式，Page Directory LRU 方式，Clock 方式の 3 方式を設計し，このうち FIFO 方式と Page Directory LRU 方式について実装した．評価実験を行い，シャドウページングの各方式とネイティブな Linux や No Shadow Page Table 方式との比較を行った．その結果，Single Shadow Page Table 方式よりも Multiple Shadow Page Table 方式の方がコストが少ないことが分かった．また，Multiple Shadow Page Table 方式の Reclamation Policy では，Page Directory LRU 方式より FIFO 方式の方が有効であることが分かった．

本論文の構成は以下のようになっている．まず，第 2 章で，本研究で開発しているマルチコアプロセッサ指向の軽量 VMM，Gandalf の概要について述べる．次に，第 3 章でシャドウページングの設計について述べ，続いて第 4 章でその実装について述べる．第 5 章では実装したシャドウページングについて実験と評価を行う．第 6 章で関連研究について述べ，最後に第 7 章でまとめと今後の課題について述べる．

2. Gandalf

マルチコアプロセッサ指向の軽量 VMM である Gandalf の概要について述べる．Gandalf は，プロセッサとして IA-32 アーキテクチャ，Gandalf 上で動作させるゲスト OS として Linux を対象としている．図 1 に，Gandalf を含むシステムの構造図を示す．ホストマシンとなる実機上で VMM である Gandalf が動作し，仮想的な計算機環境として仮想マシンを複数個生成する．これらの仮想マシンは独立しており，それぞれの仮想マシン上で個別にゲスト OS を動作させることが出来る．Gandalf では，各仮想マシンにはプロセッサをプロセッサ単位で割り当てる．従来の手法では 1 つのプロセッサを多重化して複数の仮想マシンで共有するモデルが多かったが，この手法では仮想マシンの管理と切り替えにコストがかかってしまう．プロセッサ単位で資源を割り当てることによって，より軽量な VMM を作ることができる．

IA-32 アーキテクチャには，0 から 3 までの 4 段階の特権レベルが存在する．数が小さいほうが特権レベルが高く，大きいほうが特権レベルは低い．重要ないくつかの命令は特権命令と呼ばれ，特権レベルが 0 の状態でないと実行できない．図 2 の左側に示すように，Linux は通常，カーネルをリング 0 で，ユーザプロセスをリング 3 で動作させている．このようにすることによって，カーネルは特権命令を使用してプロセッサを管理することができ，また，ユーザプロセスからカーネルを保護することができる．本研究では，VMM が Linux を管理するために VMM を Linux より高い特権レベルで動作させる必要がある．そこで，図 2 の右側に示すように，VMM をリング 0 で動作させ，Linux カーネルはひとつ特権レベルの低いリング 1 で動作させる．リング 1 に移動させたことによって，Linux カーネルは特権命令を使用できなくなってしまうが，これらの特権命令は VMM で適切にエミュレーションを行う．

Linux は通常，存在する物理メモリを全て使用して動作している．しかし，同時に VMM を動作させたり，複数の Linux を動作させたりするときには，物理メモリ領域を分け合いながら使用する必要がある．こ

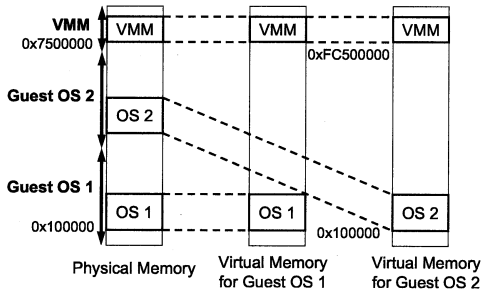

図 3 メモリマップ
Fig. 3 Memory Map

のため，図 3 の左側に示すように，VMM には物理メモリの上部を割り当て，その残りの部分を分割してそれぞれの Linux に割り当てている．また，仮想メモリについても同様に，Linux は通常仮想メモリ空間の全てを使用して動作しているが，VMM を同じ仮想メモリ空間で実行させる時，Linux が VMM の仮想メモリ領域をアクセスできてしまうと問題である．そのため，図 3 の右側に示すように，Linux の使用できる仮想メモリ領域から VMM の使用する部分を除外している．なお，使用可能メモリ領域の制限にはセグメント機構を利用しているが，簡単のため，VMM の使用メモリ領域は仮想メモリ空間の上端付近とした．

3. シャドウページングの設計

本章では，ゲスト OS 間のメモリ保護を実現するシャドウページングの設計について述べる．まず，シャドウページングの概要について述べ，続いて本研究で設計したシャドウページングの詳細について述べる．本研究では，シャドウページテーブルをひとつのみ使用する Single Shadow Page Table 方式と，複数のシャドウページテーブルを使用する Multiple Shadow Page Table 方式の 2 方式を設計した．また，シャドウページテーブルを使用しない No Shadow Page Table 方式についても述べる．

3.1 シャドウページングの概要

シャドウページングでは，ゲスト OS が使用するページテーブルであるゲストページテーブルとは別に，シャドウページテーブルと呼ばれるページテーブルを VMM 内に準備する．ゲスト OS の実行時には，CPU はゲスト OS 内のページテーブルではなく，シャドウページテーブルを参照する．シャドウページングの概要図を図 4 に示す．ゲスト OS が新しいページテーブルエントリを追加したことを VMM が検知すると，VMM はそのエントリを調べ，内容に問題がなければシャドウページテーブルにエントリをコピーする．

シャドウページテーブルにゲストページテーブルの内容を反映させる方法としては，ゲストページテーブルを read-only に設定しておいて，ゲスト OS がページテーブルエントリを更新するのを検知してシャドウページテーブルを更新する方法や，ページフォルトの処理中でシャドウページテーブルを更新する方法などがある．本研究では，ゲストページテーブルは read-only にせず，ページフォルトの処理中でシャドウページテーブルの内容を更新する方法を採っている．この方法では，ゲスト OS は自由にゲストページテーブルを書き換えることができるが，その内容はすぐにはシャドウページテーブルに反映されない．ゲストページテーブルに新しいエントリを追加しても，そのエントリはシャドウページテーブル内に存在しないので，そのページにアクセスするとページフォルトが発生する．VMM がページフォルトを処理する際に，新しいエントリがゲストページテーブルに追加されていることを検知したら，このエントリの内容を調べてシャドウページテーブルにコピーする．

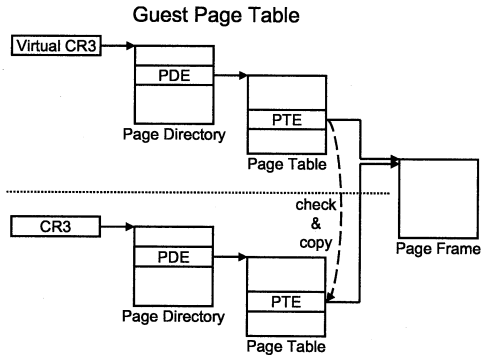

図 4 シャドウページテーブル概要図
Fig. 4 Overview of Shadow Page Table Structure

ゲストページテーブルでのエントリの更新は，invlpg 命令で検知する．IA-32 アーキテクチャでは，ページテーブルエントリを変更した際は必ず invlpg 命令で TLB をフラッシュする必要がある☆．invlpg 命令は特権命令であるため，ゲスト OS がこれを実行すると一般保護例外が発生し，VMM が呼び出される．VMM は invlpg 命令をエミュレーションする時に，invlpg 命令を実行すると同時に，対応するシャドウページテーブルのエントリを更新もしくはクリアする．エントリを更新した場合は，新しいエントリがシャドウページテーブルに反映される．エントリをクリアしただけの場合は，対応するアドレスにゲスト OS がアクセスした時にページフォルトが発生し，VMM でこれを処理する際に新しいエントリをコピーすることができる．

CPU が参照するページテーブルは VMM 内に存在し，この内容を操作できるのは VMM のみであるので，

☆ 従って，invlpg 命令を持たないプロセッサはサポートしない．

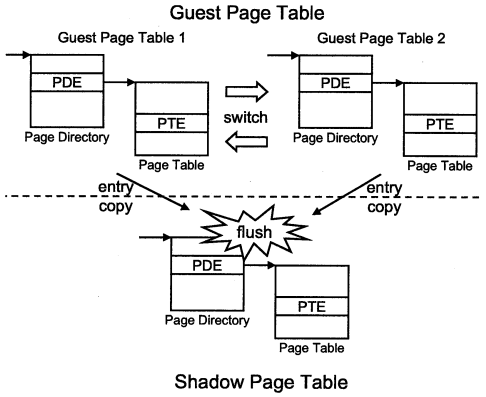

図 5 Single Shadow Page Table 方式
Fig. 5 Structure of Single Shadow Page Table

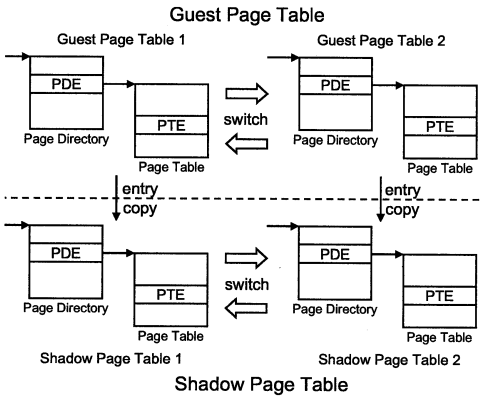

図 6 Multiple Shadow Page Table 方式
Fig. 6 Structure of Multiple Shadow Page Table

シャドウページングを利用することで，VMM がゲスト OS のメモリ利用を監視することができる．VMM は，ゲストページテーブルのエントリの内容に問題があれば，そのエントリを無効化したり，エントリの内容を書き換えてシャドウページテーブルに追加したりすることによって，メモリが正しく利用されていることを保証することができる．また，ゲスト OS 内のページテーブルを仮想ページテーブルとして扱うことによって，メモリの仮想化を実現することもできる．

3.2 Single Shadow Page Table 方式

Single Shadow Page Table 方式では，VMM は各ゲスト OS に対し 1 つのシャドウページテーブルを確保する．図 5 に Single Shadow Page Table 方式の概要図を示す．ゲスト OS は複数のページテーブルをプロセス毎に切り替えて使用しているが，VMM 内には 1 つのシャドウページテーブルしか確保されていないため，ゲストページテーブルの切り替えに合わせてシャドウページテーブルを切り替えることができない．このため，ゲストページテーブル切り替えの際には，シャドウページテーブルをフラッシュすることで，新しいシャドウページテーブルに切り替えを実現する．シャドウページテーブルをフラッシュすると，シャドウページテーブルにコピーしたエントリの内容が消えてしまうので，フラッシュ後には再びゲストページテーブルのエントリをコピーする必要がある．ページテーブルの切り替えが頻繁に発生する場合では，このシャドウページテーブルフラッシュとフラッシュ後のシャドウページテーブル更新のコストが重大な問題となる．しかし，Single Shadow Page Table 方式では，ゲスト OS 毎にシャドウページテーブルを 1 つしか用意する必要がないので，シャドウページテーブルの管理が簡単であり，また，実装が容易である．

3.3 Multiple Shadow Page Table 方式

Multiple Shadow Page Table 方式では，Single Shadow Page Table 方式とは異なり，各ゲストページテーブルにそれぞれシャドウページテーブルを割り当てる．ゲスト OS がページテーブルの切り替えを行う時には，同時にシャドウページテーブルの切り替えを行う．Multiple Shadow Page Table 方式の概要を図 6 に示す．

Multiple Shadow Page Table 方式では，ゲストページテーブル毎にシャドウページテーブルが存在するため，ゲストページテーブルの切り替えが起こってもシャドウページテーブルの切り替えを行うだけで，シャドウページテーブルのフラッシュは行わない．このため，ページテーブル切り替えのコストを抑えることができる．また，シャドウページテーブルの内容が VMM 内に保存されたままページテーブルが切り替えられるので，シャドウページテーブルをタグ付 TLB のように使用することができ，ゲストページテーブルのエントリをコピーするコストを削減することができる．

ただし，ゲスト OS 毎に複数のシャドウページテーブルが必要となるため，全てのゲストページテーブルにシャドウページテーブルを割り当てていると，VMM が使用するメモリが多くなりすぎてしまう．このため，本研究では，予めシャドウページテーブルに使用するメモリを静的に確保しておき，この中でメモリを使い回すという方法を取っている．メモリはページテーブルサイズの 4KB 単位で確保され，このメモリを順にページディレクトリとページテーブルの区別なくシャドウページテーブルに割り当てていく．使用されていないメモリがなくなったら，Reclamation Policy によって決定されるページテーブルを解放し，そのメモリを再利用する．ページディレクトリを解放してしまうとシャドウページテーブル自体が解放されてしまい，対応するゲストページテーブルへの切り替えが発生した時にシャドウページテーブルを再構築するコストが大きくなってしまうため，ページテーブルのみを再利用の対象としている．

また，各ゲスト OS に最大いくつのシャドウページ

テーブルを割り当てるかは，静的に与えられる変数によって決定される．この数を超えてシャドウページテーブルを割り当てたい時には，

シャドウページテーブル用に確保してあるメモリに対してあまりにもシャドウページテーブル数が多すぎると，シャドウページテーブル間でページテーブルを奪い合っているばかりになってしまう．逆に，使用されるゲストページテーブル数に対して割り当てることができる最大のシャドウページテーブル数が少なすぎると，シャドウページテーブルを割り当てるとすぐに他のゲストページテーブルに再割り当てされてしまうことになり，Multiple Shadow Page Table 方式の利点が失われてしまう．メモリ量に応じた適切なシャドウページテーブル数を指定することが重要である．

3.3.1 Reclamation Policy

Multiple Shadow Page Table 方式で，再利用するシャドウページテーブルを決定する Reclamation Policy として，FIFO，Page Directory LRU，Clock の 3 方式を設計した．

FIFO 方式では，ページディレクトリに使用されているメモリを除くものに関して，ページテーブル単位で FIFO キューを作成する．シャドウページテーブルに使用するメモリが不足したら，この FIFO キューの先頭にあるページテーブルをフラッシュして解放し，新しいシャドウページテーブルに割り当てる．

Page Directory LRU 方式では，ページディレクトリ単位で LRU キューを作成する．メモリの再利用が要求されたら，LRU キューの先頭にあるページディレクトリで使用されているページテーブルを全てフラッシュして解放する．

Clock 方式では，ページテーブル単位でキューを管理し，各ページテーブルのアクセスビットを利用して再利用するシャドウページテーブルを決定する．一定時間ごとにアクセスビットをクリアしておき，再利用するシャドウページテーブルが必要になった時点でアクセスビットがクリアされたままのページテーブルを選んで解放し，新しいシャドウページテーブルとして利用する．

3.4 No Shadow Page Table 方式

比較のため，シャドウページテーブルを使用しない No Shadow Page Table 方式を設計した．この方式では，ゲスト OS が使用するページテーブルを CPU が参照するページテーブルとして使用し，VMM 内ではシャドウページテーブルの管理は行わない．VMM のメモリ領域を確保するため，ページテーブル切り替えの際に VMM 用のページエントリを Linux のページテーブルにコピーして使用する．

No Shadow Page Table 方式では，CPU が参照するページテーブルをゲスト OS が直接変更できるため，ページテーブルの更新にかかるコストがかからない．また，VMM 内にシャドウページテーブル用のメモリを確保する必要がないので，VMM が使用するメモリ量を抑えることができる．しかし，ゲスト OS が不正なエントリをページテーブルに追加したとしても，VMM はそれを検知することができない．

4. シャドウページングの実装

第 3 章で述べたシャドウページングの設計に基づいた実装について述べる．

4.1 Single Shadow Page Table 方式

Single Shadow Page Table 方式の現在の実装では，シャドウページテーブル用にページディレクトリを 1 個，ページテーブルを 1024 個確保している．ページテーブルは 0 からのテーブル番号で管理されている．4KB のページマッピングを管理するためにページテーブルが必要な場合は，そのアドレスを 22 ビット右シフトした数に対応するテーブル番号のページテーブルを使用する．

ゲストページテーブルからエントリをコピーするためには，VMM がゲストページテーブルを参照できる必要がある．しかし，ゲスト OS から VMM に渡されるゲストページテーブルのアドレスは物理アドレスであり，そのままこのアドレスにアクセスすることはできない．ゲスト OS ではこのアドレスをカーネル内のどこかにマッピングして使用しているため，その仮想アドレスを参照すればゲストページテーブルの内容を得ることができるが，シャドウページテーブルを調べて対応する仮想アドレスを探し出すのはコストが大きい．そのため，VMM の使用する仮想メモリ領域内にゲストページテーブルをマップするための固定領域を設定し，使用中のゲストページテーブルのページディレクトリをマップすることでそのコストを削減した．また，4KB マッピングのためにページテーブルを使用している場合は，そのページテーブルもマップしている．

4.2 Multiple Shadow Page Table 方式

Multiple Shadow Page Table 方式では，シャドウページテーブル用のメモリに 4KB のページサイズ単位でテーブル番号を割り当て，この番号によってメモリを管理している．VMM ではこのメモリの使用状況を管理するテーブルを持っており，ページディレクトリとして使用中であるか，ページテーブルとして使用中であるか，あるいは使用されていないかを記録している．新しくページディレクトリやページテーブルとしてメモリを要求された時にはまずこのテーブルを調べ，使用されていないメモリを探す．全てのメモリが使用中であれば Reclamation Policy を呼び出し，再利用するページテーブルを探す．

また，それぞれのシャドウページテーブルを管理する構造体も存在し，シャドウページテーブルごとのシャドウ番号によって管理されている．この構造体には，

ページディレクトリへのポインタとテーブル番号，また，どのアドレスでどのテーブル番号のページテーブルが使用されているかが保存されている．ページテーブルの更新や解放の際はこの構造体が利用される．

各シャドウページテーブルに対応するゲストページテーブルは，シャドウ番号によるテーブルで管理されている．ゲストページテーブルはそのページディレクトリの開始アドレスが保存され，ゲストページテーブルの切り替え時に対応するシャドウページテーブルを検索するために使用される．使用中のシャドウページテーブルに対応するゲストページテーブルは，Single Shadow Page Table 方式と同様に，VMM 内の固定領域にマップされている．

5. 実　　験

実装したシャドウページングについて評価実験を行った．シャドウページングのコストについて評価するため，シングルゲスト OS で実験を行った．実験には Dell Precision 490 を用い，プロセッサは Intel Xeon 5130 2.0GHz を使用した．ゲスト OS として Linux 2.6.12.3 に必要な変更を加えたものを使用した．ベンチマークプログラムとして lmbench[7] を用い，ネイティブな Linux と No Shadow Page Table 方式，Single Shadow Page Table 方式，Multiple Shadow Page Table 方式との比較を行った．また，Multiple Shadow Page Table 方式において，Reclamation Policy の比較を行ったが，Clock 方式は未実装であるため，評価を行っていない．

5.1 シャドウページング方式の比較

シャドウページングの各方式のコストを比較するため，lmbench を用いてパイプのレイテンシとプロセス生成のコストを計測した．シャドウページングによるコストの増加を調べるため，No Shadow Page Table 方式についても計測した．比較のため，ネイティブな Linux についても計測した．Multiple Shadow Page Table 方式では，一番コストが低くなるように，最大のシャドウページテーブル数を 8，予め確保しておくシャドウページテーブル用のメモリをページテーブル 15 個分として計測した．Reclamation Policy として FIFO 方式を設定しているが，この状態では，Reclamation Policy によるメモリの再利用は起こっていない．評価結果を表 1 と図 7 に示す．

図 7 を見ると，Multiple Shadow Page Table 方式が全てのプログラムにおいて，Single Shadow Page Table 方式よりコストが低いことが分かる．これは，Single Shadow Page Table 方式ではゲストページテーブル切り替えのたびにシャドウページテーブルをフラッシュしているので，そのフラッシュとフラッシュ後のページテーブルエントリのコピーに大きなコストがかかっているためと考えられる．ただし，パイプレイテンシにおけるコスト差とプロセス生成におけるコスト差がほぼ同じであるため，これ以外の部分ではコストに影響するような差はないと考えられる．

表 1　lmbench の測定結果
Table 1　Results of lmbench

	Pipe latency	Process fork+exit	Process fork+exec
Linux	2.82 μsec	25.26 μsec	60.24 μsec
No Shadow	4.03 μsec	34.42 μsec	80.71 μsec
Multiple	4.07 μsec	44.93 μsec	106.92 μsec
Single	18.2 μsec	58.22 μsec	123.27 μsec

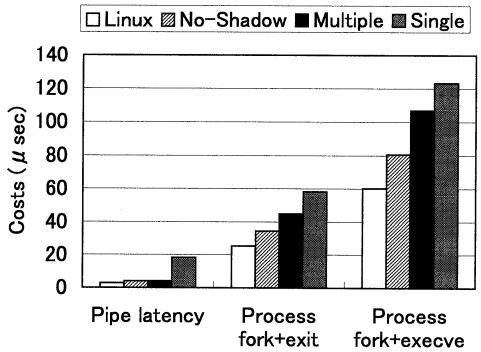

図 7　lmbench の測定結果の比較
Fig. 7　Comparison of Results of lmbench

No Shadow Page Table 方式と Multiple Shadow Page Table 方式のコスト差は，シャドウページングに関するコスト差であると考えられる．パイプレイテンシではさほど差がないものの，プロセスの fork+exit では 10μsec ほど，fork+exec では 26μsec ほどと差が広がってしまっている．これはページテーブルエントリをコピーするためのコストだと考えられ，fork+exit より fork+exec の方がコスト差が大きいのは，fork+exec ではゲスト OS が使用するページ数が多いからであると考えられる．

また，ネイティブな Linux と No Shadow Page Table 方式とのコスト差は，シャドウページングに関わらない部分の特権命令のエミュレーションとページフォルトの処理にかかるコストであると考えられる．この比較でも同様に，fork+exit より fork+exec の方がコスト差が大きいのは，その分発生する例外が多いからであると考えられる．

5.2 検索コスト評価

Multiple Shadow Page Table 方式で，最大のシャドウページテーブル数やシャドウページテーブル用のメモリ量の違いによるコストを比較するため，評価を行った．最大のシャドウページテーブル数をシャドウ数，シャドウページテーブル用のメモリ量をページテーブルサイズで何個分あるかというテーブル数で表す．シャドウ数とテーブル数を変化させて実験を行い，

表 2 検索コストの測定結果
Table 2 Costs of Shadow Page Table Search

	Process fork+exit	Process fork+exec
8 shadows, 15 tables	44.93 μsec	106.92 μsec
8 shadows, 1024 tables	46.70 μsec	109.82 μsec
100 shadows, 15 tables	46.32 μsec	108.41 μsec
100 shadows, 1024 tables	47.14 μsec	111.06 μsec

プロセスの fork+exit と fork+exec についてコストを計測した．シャドウ数はゲストページテーブルに合わせて切り替えるシャドウページテーブルの検索するとき，テーブル数は使用されていないメモリを検索するときのコストに影響すると考えられる．Reclamation Policy として FIFO 方式を設定しているが，Reclamation Policy によるメモリの再利用は起こっていない．結果を表 2 に示す．

シャドウ数が 8 の時のテーブル数の変化によるコストの変化を見ると，テーブル数が 1024 に増えると，fork+exit では 2μsec，fork+exec では 3μsec ほどのコスト増になっていることが分かる．使用するページ数が多いために，fork+exec の方がよりコスト増になっていると考えられる．

また，テーブル数が 15 の時のシャドウ数の変化によるコストの変化を見ると，こちらは fork+exit と fork+exec で増えたコストが同じ程度になっている．シャドウ数が影響するのはシャドウページテーブルの切り替え時であるので，fork+exit と fork+exec ではゲストページテーブルの切り替え数が変わらないためであると考えられる．

5.3 Reclamation Policy 比較

Multiple Shadow Page Table 方式において，Reclamation Policy の違いによるコストを比較するため，FIFO 方式と Page Directory LRU 方式について評価を行った．Clock 方式は未実装であるため，ここでの評価は行っていない．Reclamation Policy が呼び出されるように負荷をかけるため，8 プロセスでのコンテクストスイッチを各プロセスに割り当てられるメモリ量を変化させて実行した．シャドウ数は 20，テーブル数は 15 に設定した．結果を図 8 に示す．

各プロセスに割り当てられるメモリ量を変化させても，変わらずに FIFO 方式の方が Page Directory LRU 方式よりもコストが低いことが分かる．この原因として，LRU キューの方が FIFO キューよりも管理コストがかかると言う理由が考えられる．FIFO キューでは，新しいテーブルをキューの最後に足し，キューの最初のテーブルから順に再利用していけばよい．しかし，LRU キューではゲストページテーブルに合わせてシャドウページテーブルを切り替えるたびにキューを更新する必要があり，その際，キューを検索して切り替えるシャドウページテーブルを探さなければならないなどコストがかかる．

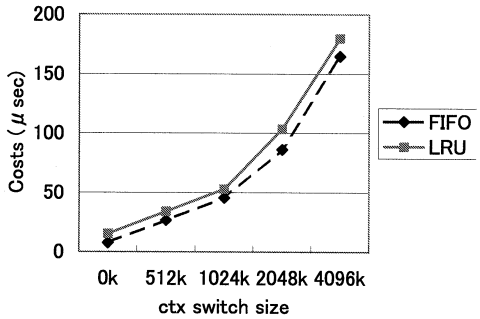

図 8 プロセスサイズの変化によるコンテクストスイッチのコスト
Fig. 8 Costs of Context Switches for Different Process Sizes

また，調べたところ，この実験では LRU 方式で Reclamation Policy が呼び出された際に，一度に解放されているページテーブルは高々3 個であり，一度に多くのページテーブルを再利用できるという Page Directory LRU 方式の利点が活かされていないことが分かった．また，ページテーブルの再利用を繰り返すうちに，LRU キューに存在していても再利用できるページテーブルがないシャドウページテーブルが増え，再利用できるページテーブルを見つけるまでにコストがかかっていることも分かった．全てのシャドウページテーブルが多くのページテーブルを使用している場合では Page Directory LRU 方式の方が有利かもしれないが，そのような場合は多くないので，通常は FIFO 方式の方が有効であると考えられる．

6. 関連研究

これまでの研究でもシャドウページングは利用されてきたが，その設計や実装については，詳しく述べられているものは少ない．また，シャドウページングに特化して評価が行われているものも少ない．

VMware[1] では，paravirtualization の実現のためにシャドウページングを使用している．しかし，VMware ではゲスト OS がページテーブルへの書き込みを行う時にトラップを発生させることによってシャドウページテーブルの更新を行っており，このために，ゲスト OS である Linux がページテーブルエントリを設定するのに使用しているマクロを全て VMI_SetPte() という関数に書き換える必要がある．VMI_SetPte() では，Linux のページテーブルエントリを書き換えると同時に VMware への通知を行い，シャドウページテーブルを書き換える．Gandalf ではページフォルトの処理中でシャドウページテーブルの更新を行っており，VMware とは異なる．

Xen 3.0[10] でも，Shadow Page Table が利用されている．Xen 3.0 では，ゲストページテーブルに対応

するページにゲスト OS が書き込めないように予め read-only に設定しておく．ゲスト OS がゲストページテーブルを書き換えたい時にはページフォルトが発生するので，その時点でのゲストページテーブルの状態を保存しておき，そのページを out of sync list に加えた上でゲスト OS が書き込めるように変更する．その後，ゲスト OS が TLB をフラッシュするような命令を実行したら，out of sync list のページの内容をシャドウページテーブルに反映し，再びゲストページテーブルを read-only に設定する．Gandalf ではゲストページテーブルを read-only にはせず，ページテーブルエントリの更新時ではなく対応するページへのアクセスが発生した時点でシャドウページテーブルの更新を行っている．

7. まとめと今後の課題

本論文では，本研究でこれまで開発してきたマルチコアプロセッサ指向の軽量 VMM の Gandalf において，ゲスト OS 間のメモリ保護を実現するシャドウページングについて述べた．

Single Shadow Page Table 方式と Multiple Shadow Page Table 方式の 2 方式のシャドウページングを設計し，実装を行った．Multiple Shadow Page Table 方式では，ページテーブル用のメモリを再利用するための Reclamation Policy として FIFO 方式，Page Directory LRU 方式，Clock 方式の 3 方式を設計し，このうち FIFO 方式と Page Directory LRU 方式について実装した．評価実験を行い，シャドウページングの各方式とネイティブな Linux や No Shadow Page Table 方式との比較を行った．その結果，Single Shadow Page Table 方式よりも Multiple Shadow Page Table 方式の方がコストが少ないことが分かった．また，Multiple Shadow Page Table 方式の Reclamation Policy では，directory LRU 方式より FIFO 方式の方が有効であることが分かった．

今後の課題としては，まず，未実装である Multiple Shadow Page Table 方式における Clock 方式による Reclamation Policy の実装が挙げられる．また，メモリ量に応じた適切なシャドウページテーブル数の設定方法についても，検討する必要がある．

参 考 文 献

1) Z. Amsden, D. Arai, D. Hecht, A. Holler, P.Subrahmanyam. VMI: An Interface for Paravirtualization. In *Proceedings of the 2006 Linux SYmposium*, Vol. 2, pp. 363–378, July 2006.
2) P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A.Warfield. Xen and the Art of Virtualization. In *Proceedings of the 19th ACM Symposium on Operating System Principles*, pp. 164–177, October 2003.
3) E. Bugnion, S. Devine, K. Govil, and M. Rosenblum. Disco: Running Commodity Operating Systems on Scalable Multiprocessors. In *Proceedings of the 16th ACM SIGOPS Symposium on Operating Systems Principles*, pp. 143–156, October 1997.
4) R.J.Creasy. The Origin of the VM/370 Time-Sharing System. *IBM Journal of Research and Development*, 25 (5), 1981.
5) R.P.Goldberg. Survey of Virtual Machine Research. *IEEE Computer*, pp. 34–45, June 1974.
6) Intel Corporation. IA-32 Intel Architecture Software Developer's Manual.
7) L. McVoy and C. Staelin. lmbench: Portable Tools for Performance Analysis. In *Proceedings of the USENIX Annual Technical Conference*, pp. 279–294, January 1996.
8) R.Meyer and L.Seawright. A Virtual Machine Time Sharing System. *IBM Systems Journal*, 9 (3), pp. 199–218, 1970.
9) G. Popek and R. Goldberg. Formal Requirements for Virtualizable 3rd Generation Architectures. *Communications of the A.C.M.*, 17(7):412–421, 1974.
10) I. Pratt, D. Magenheimer, H. Blanchard, J.Xenidis, J.Nakajima, A.Liguori. The Ongoing Evolution of Xen. In *Proceedings of the 2006 Linux Symposium*, Vol. 2, pp. 255–266, July 2006.
11) J.S.Robin and C.E.Irvine. Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor. In *Proceedings of the 9th USENIX Security Symposium*, pp. 129–144, August 2000.
12) M. Rosenblum and T. Garfinkel. Virtual Machine Monitors: Current Technology and Future Trends. *IEEE Computer*, pp. 39–47, May 2005.
13) J.Sugerman, G.Venkitachalam, and B.H.Lim. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In *Proceedings of 2001 USENIX Annual Technical Conference*, pp. 1–14, June 2001.
14) A.Whitaker, M.Shaw, and S.D.Gribble. Scale and Performance in the Denali Isolation Kernel. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, pp. 195–210, December 2002.