2008－OS－109 (10)
2008／8／7

準パススルー型仮想マシンモニタ BitVisor の設計と実装

品川高廣[†1] 榮樂英樹[†1] 谷本幸一[†1]
面和成[†1] 長谷川晶一[†1] 保理江高志[†1]
平野学[†2] 光来健一[†3] 大山恵弘[†4]
河合栄治[†5] 河野健二[†6] 千葉滋[†3]
新城靖[†1] 加藤和彦[†1]

近年，仮想マシン技術を用いてセキュリティ機能を実現する研究が増えつつある．しかし従来の仮想マシンモニタでは，ハードウェアの仮想化や複数 OS 間の保護・共有のために多数の機能が必要であり，仮想マシンモニタのサイズの肥大化が問題となる．本論文では，セキュリティ機能の実現を目的とした仮想マシンモニタ向けのアーキテクチャとして，準パススルー型と呼ぶ方式を提案する．準パススルー型仮想マシンモニタでは，ハードウェアへのアクセスの多くをパススルーすることにより，仮想化や共有機能を省略して仮想マシンモニタの肥大化を抑制する．一方，最低限必要なアクセスだけを仮想マシンモニタで捕捉することにより，確実にセキュリティ機能を実現することができる．実際に実装した結果，仮想マシンモニタのコア機能は 2 万行程度，ATA デバイスに対するドライバは 1200 行程度で実現できた．

BitVisor: Design and Implementation of the Para-Pass-Through Virtual Machine Monitor

Takahiro Shinagawa ,[†1] Hideki Eiraku ,[†1] Kouichi Tanimoto ,[†1]
Kazumasa Omote ,[†1] Syoichi Hasegawa ,[†1] Takashi Horie ,[†1]
Manabu Hirano ,[†2] Kenichi Kourai ,[†3] Yoshihiro Ohyama ,[†4]
Eiji Kawai ,[†5] Kenji Kono ,[†6] Shigeru Chiba ,[†3]
Yasushi Shinjo [†1] and Kazuhiko Kato [†1]

Recently, virtual machine technology is used by a number of researches to provide security services. Unfortunately, existing virtual machine monitors have various functionalities, such as hardware virtualization, protection and sharing of resources among guest operating systems, resulting in the growing size of virtual machine monitors. This paper presents the para-pass-through architecture, designed to minimize the code size of virtual machine monitores that provide security services. By allowing pass-through access to most of hardware devices from the guest operating system, a large part of virtualization and sharing functionalities can be eliminted from virtual machine monotors, while successfully enforcing necesarry security services by intercepting least necessary access by virtual machine monitors. The code sizes of our implementation are approximately 20 thousand lines for the core of the virtual machine monitor and 1.2 thousand lines for the driver of ATA devices.

†1 筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻
Department of Computer Science, Graduate School of Systems and Information Engineering, University of Tsukuba
†2 豊田工業高等専門学校情報工学科
Department of Information and Computer Engineering, Toyota National College of Technology
†3 東京工業大学大学院情報理工学研究科数理・計算科学専攻
Department of Mathematical and Computing Sciences, Graduate School of Information Science and Engineering, Tokyo Institute of Technology
†4 電気通信大学電気通信学部情報工学科
Department of Computer Science, Faculty of Electro-Communications, The University of Electro-Communications
†5 奈良先端科学技術大学院大学情報科学研究科
Graduate School of Information Science,, Nara Institute of Science and Technology
†6 慶應義塾大学理工学部情報工学科
Department of Information and Computer Science, Faculty of Science and Technology, Keio University

1. はじめに

近年，デスクトップ環境における情報漏洩が深刻な問題となっている．デスクトップ環境では，サーバ環境に比べて管理が不十分であったり，専門知識の乏しいユーザが使用することが多いため，セキュリティが十分に確保されていないことが多い．その結果，PCやUSBメモリの盗難・紛失，ウィルス感染などによって容易に個人情報・機密情報の漏洩が発生してしまう．

情報漏洩を防止するためには，暗号化や認証を用いることが有効である．多くのOSやアプリケーションで暗号化や認証機能が提供されているが，これらは設定が複雑であったり使い方が面倒であるなどの理由で，必ずしも広く普及してはいない．また，近年のOSやアプリケーションは肥大化・複雑化する傾向にあり，セキュリティ上の脆弱性を突いて暗号化や認証機能を無効化される危険性が増加しつつある．従って，OSやアプリケーションに頼らずに，暗号化や認証を容易にかつ確実に実現できる手法が必要となる．

我々は，暗号化や認証などのセキュリティ機能を仮想マシン技術を用いて実現する方式の研究をおこなっている．本研究では，デスクトップ環境からの情報漏洩防止を目的として，仮想マシンモニタを用いてストレージやネットワークを暗号化する．また，暗号鍵の管理やユーザ認証を安全におこなうために，仮想マシンモニタでICカードの管理をおこなう．仮想マシンモニタのレイヤで暗号化及び認証をおこなうことにより，ゲストOSには依存しない形で透過的かつ確実にセキュリティ機能を実現することを目指している．

仮想マシンモニタでセキュリティ機能を実現する場合，仮想マシンモニタ自身にセキュリティ上の脆弱性が存在しないことが重要である．仮想マシンモニタに脆弱性が存在すると，セキュリティ機能が回避されたり無効化されてしまう可能性がある．脆弱性を生じにくくするためには，仮想マシンモニタ全体を小さくシンプルに保つことが有効であるが，従来の仮想マシンモニタは多機能化が進んでおり，OSに匹敵するほど肥大化・複雑化してしまう傾向にある．

本論文では，仮想マシンモニタを小さく保ちつつセキュリティ機能を実現するためのアーキテクチャとして，**準パススルー型**と呼ぶ方式を提案する．準パススルー型とは，ゲストOSからハードウェアへのアクセスを可能な限りパススルー（通過）させつつ，セキュリティ機能の実現に最低限必要なアクセスのみを仮想マシンモニタで捕える方式である．ハードウェアを仮想化せずに直接ゲストOSに見せることにより，デバイス制御の大部分をゲストOSのデバイスドライバにおこなわせることが可能になり，仮想マシンモニタの構造を大幅に簡略化することができる．一方で，最低限必要なアクセスのみを仮想マシンモニタで捕えることにより，ストレージやネットワークの暗号化などのセキュリティ機能を確実に実現することができる．

準パススルー型アーキテクチャでは，仮想マシンモニタを小さくシンプルにするために，複数のゲストOSを同時に動作させる機能を省略している．本研究の目的はデスクトップ環境におけるセキュリティ機能向上であり，複数ゲストOSの同時動作は必ずしも必要ではない．また，近年ではセキュリティ機能の実現を目的として，複数ゲストOSをサポートしない仮想マシンモニタの研究も増えてきており[1),2)]，簡略化のために複数ゲストOSの同時動作を省略することは妥当と考えられる．これにより，仮想マシン間でのスケジューリングやメモリ管理などの資源共有機能，デバイスドライバやデバイス仮想化機能などを省略して，仮想マシンモニタのさらなる簡略化が可能になる．

本論文の構成は以下の通りである．2章では，本研究で想定する脅威モデル及び前提条件を説明する．3章では，準パススルー型仮想マシンモニタの設計，4章では実装について述べる．5章では，コードサイズ及びオーバーヘッド測定の結果を示す．6章で関連研究について述べ，7章で本論文をまとめる．

2. 脅威モデルと前提条件

本章では，本研究の脅威モデルと前提条件を述べる．

2.1 脅威モデル

本研究では，情報漏洩を防止するために，仮想マシンモニタでストレージ及びネットワーク通信を暗号化する．想定される攻撃には，ゲストOS内のデータを暗号化されていない状態で外部に取り出そうとする試みがある．このような攻撃は，主にソフトウェア的な攻撃とハードウェア的な攻撃の2つに分けられる．

ソフトウェア的な攻撃では，ウィルス等によりゲストOSの制御が完全に乗っ取られることを想定する．すなわち，攻撃者はユーザモード及びカーネルモードにおいて任意のプロセッサ命令を実行可能である．例えば，特権命令の実行やページテーブルの書き換え，DMAの制御を含む各種I/O命令の発行などが可能である．想定される攻撃としては，特殊なI/Oを発行することで仮想マシンモニタによる暗号化を回避する試みや，ページテーブルやDMAを操作して仮想マシンモニタのメモリ領域を書き換えて暗号化を無効化しようとする試みなどが考えられる．

ハードウェア的な攻撃では，PCやUSBメモリの盗難・紛失などにより，攻撃者がストレージに対して物理的にアクセスできることを想定する．すなわち，攻撃者は仮想マシンモニタを経由せずに，HDDやUSBメモリの内容を直接読み出すことが可能である．また，インターネット経由での通信などのように，攻撃者がネットワーク通信の途中でパケットの中身を盗聴したり改ざんしたりできることを想定する．

本研究では，画面や音声などを介した間接的な通信手段や隠れチャネルは扱わない．また，バスのプローブや不正なデバイスの接続など，稼働中のコンピュータへのハードウェア的な攻撃は想定しない．BIOS や各種ファームウェアなどは信頼できるものとする．

2.2 前提条件

本研究では，仮想化機能を安全に提供するために，Intel VT や AMD SVM などハードウェアによる仮想化支援機能があることを前提としている．ハードウェアの支援がない IA-32 プロセッサでは，仮想化機能を安全に実現することは難しいことが知られており[3]，本研究では対象としない．また，DMA のアクセス制御をおこなうために，IOMMU(Input/Output Memory Management Unit) 機能が搭載されていることを前提とする．プロセッサと IOMMU 以外は通常の PC アーキテクチャであり，仮想マシンモニタ及びゲスト OS は，32bit もしくは 64bit のシングルプロセッサ及びマルチプロセッサ上で動作するものとする．

3. 設　　計

本章では，準パススルー型仮想マシンモニタの設計について述べる．まず，設計方針と課題を説明し，次にアーキテクチャ概要，アクセス制御などを説明する．

3.1 方針と課題

本研究の目的は，仮想マシンモニタによりゲスト OS に依存しない形でセキュリティ機能を実現することである．仮想マシンモニタはゲスト OS より高い特権レベルで動作するため，ゲスト OS の制御が完全に乗っ取られてもセキュリティ機能を確実に実施できる．しかし，仮想マシンモニタ自身に脆弱性が存在すると，ゲスト OS からの攻撃によってセキュリティ機能が無効にされてしまう可能性がある．従って，セキュリティ機能を確実に実施するためには，仮想マシンモニタ自身が正しく動作することが必要である．

仮想マシンモニタを正しく動作させるためには，仮想マシンモニタを含む TCB(Trusted Computing Base) のサイズが重要になる．プログラムのバグの数はコードの行数に比例するとされており[4]，コードサイズが小さいほど脆弱性を少なくできる可能性が高い．また，コードサイズが小さいほど検証も容易になる．従って，セキュリティ機能を実現する仮想マシンモニタでは，仮想マシンモニタを含む TCB のコードサイズを可能な限り小さくできるアーキテクチャが望ましい．

仮想マシンモニタのアーキテクチャには，大きく分けて Type II と Type I の 2 種類がある[5]．Type II では，仮想マシンモニタがホスト OS 上で動作するため，ホスト OS も TCB に含まれることになる．従って，そのコードサイズも非常に大きくなる．例えば，Type II の仮想マシンモニタである QEMU を Linux 上でを動かした場合，ソースコードのサイズは QEMU が 31 万行，Linux カーネルが 5,600 万行となる．

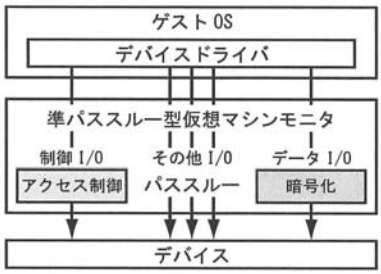

図 1 準パススルー型アーキテクチャ
Fig. 1 Para-passthrough Architecture

Type I では，Type II に比べてホスト OS がない分小さくできるが，ハードウェアを仮想化するために依然として多数の機能が必要である．例えば，実際のハードウェアを制御するデバイスドライバやゲスト OS に仮想的なデバイスを見せるためのデバイスモデル，複数ゲスト OS 間でハードウェアの共有及び保護をおこなう機能などが必要である．また，仮想化するハードウェアの種類も，CPU やメモリをはじめ，ハードディスク，ネットワーク，グラフィック，サウンド，タイマ，割り込みなど非常に多岐にわたる．

Xen[6] では，Domain0 と呼ばれる特定の仮想マシン上でデバイスドライバやデバイスモデルを動作させることで，仮想マシンモニタ本体のサイズを小さくしている．また，準仮想化によりデバイスモデルをシンプルにすることができる．しかし Domain0 の制御が乗っ取られると暗号化などのセキュリティ機能が無効化される可能性があるため，Domain0 も TCB の一部を構成していると考えられる．従って Domain0 を含めた TCB のサイズは依然として大きい．

仮想マシンモニタによるオーバーヘッドの削減を目的として，ゲスト OS からデバイスへの直接アクセスを可能にする方式が提案されている[7]．この方式では，仮想マシンモニタ内にデバイスドライバ及びデバイスモデルが不要となるため，仮想マシンモニタのサイズを小さくできる．しかしこの方式では，デバイスへ読み書きされるデータの内容を仮想マシンモニタで捕えることが出来なくなるため，このデバイスに対して暗号化などのセキュリティ機能を実現することは難しい．

本研究では，仮想マシンモニタでゲスト OS からデバイスへのアクセスを捕えつつ，仮想マシンモニタ内のデバイスドライバなどのコードサイズをいかに小さく抑えるかが課題となる．

3.2 アーキテクチャ概要

準パススルー型アーキテクチャとは，ゲスト OS からハードウェアへのアクセスを可能な限りパススルー（素通し）させつつ，暗号化などのセキュリティ機能の実現に最低限必要なアクセスだけを仮想マシンモニタで確実に捕える方式である．ゲスト OS からのアク

セスをパススルーさせることにより，デバイスの制御をゲストOSのデバイスドライバにおこなわせることができる．その結果，仮想マシンモニタのデバイスドライバやデバイスモデルを大幅に簡略化して，仮想マシンモニタのコードサイズを小さくできる．一方，完全なパススルー方式とは異なり，デバイスへの入出力に関わるI/Oアクセスは仮想マシンモニタで確実に捕えることにより，仮想マシンモニタのサイズを抑えつつも暗号化などのセキュリティ機能を実現できる．

仮想マシンモニタで捕える必要があるアクセスには，主に制御I/Oとデータ I/Oの2種類がある．制御I/Oとは，デバイスによるデータ転送を制御するためのI/Oで，転送するデータの場所やアクセス方法，データ転送の開始・終了などを指定するI/Oである．データ I/Oとは，実際にデータ転送をおこなうI/Oである．仮想マシンモニタで制御I/Oを捕えることにより，データ転送が開始されるタイミングやその内容を正確に把握できるため，ゲストOSとデバイス間のデータ転送を漏れなく確実に捕える事が出来る．制御I/Oの内容は，基本的には仮想マシンモニタによって記録されるだけであり，改変されることなくそのままデバイスに送られる．また，制御I/Oを発行するのは基本的にはゲストOSであり，仮想マシンモニタが能動的に制御I/Oを発行することは少ない．ただし，下記で述べるアクセス制御やデータの暗号化などのために，一部の制御I/Oの内容を修正することがある．

制御I/Oを仮想マシンモニタで捕えるもう1つの目的は，ゲストOSからの不正アクセスを防止することである．仮想マシンモニタが使用しているメモリやディスクなどの資源に対して，ゲストOSが不正に内容を書き換えることを防止するために，ゲストOSが発行するI/Oの内容を監視して，読み書きするデバイスやバッファのアドレスなどに対してアクセス制御をおこなう．仮想マシンモニタが使用するメモリやディスク領域は，ゲストOSからは予約領域として見えるように仮想化しており，基本的にはゲストOSがこれらの領域にアクセスすることはない．しかしゲストOSが乗っ取られた場合などに，悪意を持って意図的にアクセスを試みる可能性があるため，アクセス制御によりこれらの不正アクセスを防止する．

データI/Oを捕える目的は，暗号化などのセキュリティ機能を実施するためである．仮想マシンモニタで転送されるデータの内容をいったん受け取って，暗号化／復号化などの処理をおこなってからデバイスやゲストOSにデータを受け渡すことにより，ゲストOSからは透過的にセキュリティ機能を実施できる．

3.3 準パススルードライバ

準パススルー型アーキテクチャでは，制御I/Oやデータ I/Oの監視・変換をおこなうために，デバイスごとに小さなデバイスドライバが必要となる．本論文ではこれを**準パススルードライバ**と呼ぶ．

準パススルードライバは，データの監視・変換に必要最小限な処理だけをおこなうため，通常のデバイスドライバに比べると必要な機能は大幅に少ない．例えばIntel社のNICの場合，2万行のソースコードの約90%は，デバイスの初期化や終了処理，エラー処理などであり，データ送受信に関わる部分は約10%であることが報告されている[8]．従って，準パススルードライバは，完全なデバイスドライバとデバイスモデルを持つ場合に比べて，大幅にコード行数を削減できることが期待される．実際，我々の実装では，ATAデバイス向けの準パススルードライバは1200行程度である．

3.4 アクセス制御

準パススルー方式では，ゲストOSは基本的にハードウェアの名前空間をそのまま使用する．例えば，ゲストOSから見えるメモリの物理アドレス（ゲスト物理アドレス）は，ハードウェアの物理アドレス（マシン物理アドレス）と同じである．また，I/Oアドレス，ストレージの論理ブロックアドレス，各種バス（PCIバスやUSBバスなど）のアドレスなどもハードウェアのものをそのまま使用する．ただし，未知のPCIデバイスやUSBデバイスなどを接続されるとI/Oを正確に捉えられないため，ベンダーIDや製品IDなどを基に接続を許可するデバイスを限定する．

また，準パススルー方式では，グラフィックやサウンドなどセキュリティ機能を適用する必要がないデバイスは，制御I/Oもデータ I/Oも捕捉せずに完全なパススルーとすることができる．ただし，DMAを用いた仮想マシンモニタのメモリ領域に対する攻撃が考えられるため，ハードウェアによる保護機構を併用する．

4. 実　　装

本章では，準パススルー型仮想マシンモニタの実装について述べる．I/Oデバイス，メモリ，CPUなどのハードウェア毎に実装を説明する．

4.1 I/Oデバイス

ソフトウェアからI/Oデバイスにアクセスする方法は，I/O命令，メモリマップドI/O，DMA(Direct Memory Access)の3つがある．以下，それぞれの方式について，アクセスを捕える方式について述べる．

4.1.1 I/O命令

I/O命令は，I/Oアクセス専用のCPU命令である．I/O命令では，I/O空間のアドレスであるI/Oポートアドレスを指定して，1命令毎に1〜4バイトの単位でデバイスへのアクセスをおこなう．例えばIA-32では，IN命令やOUT命令などがあり，“IN DX, AL”という命令でDXレジスタに指定したI/OポートアドレスからALレジスタに値を読み出すことができる．

準パススルー型では，セキュリティ機能を実施するために，デバイスに対する制御I/O及びデータ I/Oを捕える必要がある．I/O命令は特権命令であるため，

I/O 命令を捕えること自体はプロセッサ・ハードウェアの仮想化支援機能によっておこなうことができる．例えば，Intel VT では，I/O ポートアドレス単位で I/O 命令発行時に仮想マシンモニタに制御が渡るかどうかを設定できる．I/O ポートアドレスは特定のデバイスに割り当てられているため，発行された I/O 命令の内容からアクセスしようとしているデバイスやアクセスの内容を識別できる．ただし，デバイスに割り当てられる I/O ポートアドレスの値は，PCI の設定などによって変更される可能性があるため，この設定をおこなう I/O 命令も監視して，デバイスと I/O アドレスの対応関係を正確に把握する．

制御 I/O の場合，仮想マシンモニタで I/O の内容を検査して，デバイスの状態把握やアクセス制御をおこなう．例えば，ディスクの場合，アクセスする論理ブロックアドレスやセクタサイズなどを把握して，仮想マシンモニタが使用する領域に不正にアクセスしないか確認する．その後，仮想マシンモニタ自身で同じ内容の I/O 命令を発行して，ゲスト OS からの I/O アクセスを監視しつつパススルーする．

データ I/O の場合は，暗号化などのために I/O 命令で読み書きされる値の内容を書き換える必要がある．しかし I/O 命令は 1 命令毎に 1〜4 バイトの単位での読み書きとなるのに対し，暗号化ではセクタ単位などある程度まとまった単位でおこなう必要がある場合が多い．そこで，複数の連続する I/O を仮想マシンモニタでバッファリングして，必要な量のデータが溜まった時点で暗号化／復号化などの変換をおこない，仮想マシンモニタで変換した I/O をまとめて発行する．

制御 I/O 及びデータ I/O 以外の I/O 命令の場合は，基本的に仮想マシンモニタに制御を渡すことなく直接ハードウェアへのアクセスを許可することができる．ただしアクセスを許可する I/O 命令は，セキュリティ上の問題が生じないことを十分に確認する必要がある．例えば，仮想マシンモニタが使用するメモリを書き換える可能性がある I/O 命令は，直接アクセスを許可せずに仮想マシンモニタで監視する必要がある．

マルチプロセッサの場合，複数のプロセッサから同時に I/O 命令を発行する攻撃が考えられる．シングルプロセッサでは，ゲスト OS がいったん I/O 命令を発行すると仮想マシンモニタに制御が渡り，処理が終わるまでゲスト OS に制御が戻ることはない（割り込みは禁止されている）．一方，マルチプロセッサの場合，1 つのプロセッサが発行した I/O を処理中に別のプロセッサで同じデバイスに対して I/O 命令をおこなうことで，TOCTTOU(Time Of Check To Time Of Use) 問題を利用した不正アクセスの可能性がある．従って，仮想マシンモニタでは，デバイスごとにロックを保持して，1 つのデバイスに対して同時に I/O 命令が発行されないようにする必要がある．

4.1.2 メモリマップド I/O

メモリマップド I/O は，専用の I/O 命令を使わずに，通常のメモリアクセス命令によって I/O をおこなう仕組みである．メモリマップド I/O は，後述するメモリ管理の仕組みを用いてアクセスを捕える．すなわちメモリマップド I/O に使われている物理アドレスに対しては，ページテーブルのエントリで読み込み及び書き込みのいずれか，または両方を禁止する．これにより，ゲスト OS がメモリマップド I/O にアクセスすると，仮想マシンモニタに制御が渡される．仮想マシンモニタはアクセスするアドレスと内容を命令エミュレーション等によって判別し，I/O 命令の場合と同様にアクセス制御や暗号化などの処理をおこなう．

メモリマップド I/O の捕捉はページ単位でおこなわれるため，必要のないアドレスへのアクセスも捕らえてしまう可能性がある．これは性能低下を引き起こす可能性はあるが，セキュリティ上の問題はない．

4.1.3 DMA

DMA(Direct Memory Access) は，CPU が介在することなく直接デバイスとメモリ間でデータ転送する仕組みである．DMA はデバイス及びメモリバスと接続されており，ソフトウェアから予め転送元と転送先の内容を設定して DMA の動作を開始すると，デバイスとメモリ間で自動的にデータ転送をおこなう．

最近の DMA 転送では，転送内容の制御にメモリ上の DMA ディスクリプタを利用するものが多い．デバイスを制御するホストコントローラは，DMA ディスクリプタの内容を逐次参照して，転送先/転送元のメモリアドレスやサイズ，転送方向を取得し，自動的にデータ転送をおこなう．ディスクリプタには複数のエントリがあり，1 つのエントリ分の転送が終わると自動的に次のエントリに進む．転送の終了は割り込みやポーリングによって OS に通知される．

準パススルー型仮想マシンモニタでは，暗号化などのために DMA で転送されるデータの中身を捕捉する必要がある．しかし，DMA 転送はハードウェアでおこなわれてしまうため，純粋なパススルー型では仮想マシンモニタで中身を捕えることが出来ない．

そこで本研究では，DMA により転送されるデータを捕捉するために，**シャドウ DMA ディスクリプタ**という仕組みを用いる．シャドウ DMA ディスクリプタとは，ゲスト OS が作成した DMA ディスクリプタ（ゲスト DMA ディスクリプタ）の代わりに，仮想マシンモニタが作成してホストコントローラに設定する DMA ディスクリプタである．シャドウ DMA ディスクリプタには，仮想マシンモニタ内部のバッファ（**シャドウバッファ**）領域がアドレスとして設定されており，デバイスとのデータ転送は，実際にはシャドウバッファ領域との間でおこなわれる．これにより，DMA 転送の制御の多くをゲスト OS におこなわせつつ，DMA 転送の内容を仮想マシンモニタで捕捉できる．

仮想マシンモニタはゲストDMAディスクリプタを適宜参照して，仮想的にホストコントローラの機能を実現する．すなわち，ゲストDMAディスクリプタの内容を解釈して，シャドウバッファ領域とゲストOSのバッファ領域との間でデータを転送する．このとき，データ転送はソフトウェアでおこなうため，暗号化・復号化などの処理をおこなうことが出来る．

仮想マシンモニタによるデータ転送のタイミングは読み込みと書き込みで異なる．読み込み時には，デバイスからシャドウバッファ領域への転送が先であるため，仮想マシンモニタでは，まずゲストOSによるDMA転送開始を指示する制御I/Oを捕える．このとき仮想マシンモニタは転送サイズを調べてシャドウDMAディスクリプタの内容を適切に設定した後，DMA転送開始のI/Oをハードウェアに送る．DMA転送が終わると，ゲストOSは割り込みやステータスレジスタの参照などの方法で転送が正常に終了したことを確認するため，このタイミングを仮想マシンモニタで捕えて，データを変換（復号化）しながら，シャドウバッファからゲストOSのバッファ領域に転送する．

書き込み時には，データの変換（暗号化）・転送が先であるため，DMA転送開始のI/Oを仮想マシンモニタで捕えたときに，データを変換（暗号化）しながら，ゲストOSのメモリ領域からシャドウバッファに転送する．同時に，転送サイズを調べてシャドウDMAディスクリプタの内容を適切に設定する．その後DMA転送開始のI/Oをハードウェアに送る．DMA転送が終了した時には特に処理は必要ない．

準パススルー型では，ゲストOSから仮想マシンモニタのメモリ領域への不正アクセスを防止するために，DMAディスクリプタに対するアクセス制御をおこなう．例えば，仮想マシンモニタがゲストDMAディスクリプタを参照する際には，バッファのアドレスが仮想マシンモニタの領域を指していないことを確認する．シャドウDMAディスクリプタ自体は，仮想マシンモニタのメモリ領域に存在するため，ゲストOSが不正に内容を書き換えることは出来ない．また，セキュリティ機能を実施しないデバイスに関しては，IOMMU[9]を用いてDMAのアクセス制御をおこなう．この場合，IOMMUはアクセス制御のためだけに用いており，全てのデバイスに対して同じ設定を適用可能であり，一度設定したら変更する必要もない．

4.2 メモリ

準パススルー型では，ゲストOSは基本的にはハードウェアの物理アドレスを直接使用する．従って，従来の仮想マシンモニタとは異なり，ゲスト物理アドレスをマシン物理アドレスに変換する作業は必要ない．そのため，アドレス変換に必要な機能を削減して，仮想マシンモニタのサイズを小さくできる．

仮想アドレスから物理アドレスへの変換は，ゲストOSのページテーブルの内容に沿っておこなう．しかしゲストOSは任意の物理アドレスをページテーブルに設定できるため，悪意をもったゲストOSが仮想マシンモニタの物理アドレスを設定して不正アクセスを試みる可能性がある．従って，ゲストOSのページテーブルをそのままは使用できない．

そこで準パススルー型仮想マシンモニタでは，従来の仮想マシンモニタと同様の手法でメモリの仮想化をおこなう．これは，ハードウェアによる多重ページテーブル機能（EPTやNPTなど）やシャドウページテーブル[6]などの手法で実現する．多重ページテーブル機能を用いる場合は，仮想マシンモニタが使用する領域以外を，ゲスト物理アドレス＝ホスト物理アドレスとなるように設定したページテーブルを使用する．この場合，アドレス変換が固定であるため，ページテーブルは一度設定すればよい．シャドウページテーブルの場合は，ゲストOSでページフォルトが発生するたびに，ゲストOSのページテーブルのエントリと同じ内容をシャドウページテーブルに反映する．物理アドレスが仮想マシンモニタの領域を指している場合は，未使用のメモリ領域を指すようにアドレス変換をおこなうことにより，不正アクセスを防止する．

4.3 CPU

準パススルー型では，CPUに関しては従来の仮想マシンモニタとほぼ同様の手法を用いる．すなわち，ゲストOSと仮想マシンモニタでCPUのコンテキストスイッチをおこなう．本研究では，CPUに仮想化支援機能があることを前提としており，この機能を用いることで，CPUのコンテキストスイッチは容易におこなうことができる．

4.4 そ の 他

現在の実装では，仮想マシンモニタでハードウェア割り込みの種類を判別していない．割り込みの種類を判別するためにはAPIC(Advanced Programmable Interrupt Controller)へのアクセスを監視する必要があるが，APICはタイマ関係で高頻度にアクセスされるため，メモリマップドI/Oの影響で頻繁に仮想マシンモニタへ制御が渡ってしまい，性能が著しく低下してしまう．そこで現在の実装では，ゲストOSの割り込みハンドラがデバイスの状態を確認するためのI/Oを発行した時点で必要な処理をおこなっている．

電源管理が有効な場合，ACPI(Advanced Configuration and Power Interface)によってハードウェアがスリープモードに入る場合がある．電源管理はゲストOSがおこなっているため，スリープモードから復帰したときに最初に制御が渡るアドレスはゲストOSによって設定されている．しかしスリープモードから復帰する際にはCPUの仮想化機能がOFFになっているため，仮想マシンモニタに制御が渡らなくなってしまう．現在の実装ではスリープモードに入ることを禁止しているが，スリープモードからの復帰処理を横取りすることで対応することは可能である．

表 1 BitVisor のコード行数
Table 1 Lines of code in BitVisor

機能	総数	実行時	デバッグ	初期化
コア	21,582	13,789	5,781	2,062
ATA ドライバ	1,287	1,279	0	8

表 2 プロセス及びメモリの lmbench 実行時間 (μs)
Table 2 Execution times of lmbench proc. & mem. (μs)

ホスト	null	fork	exec	prot	page	ctx
Linux	0.21	93.4	96.6	0.36	0.98	1.10
BitVisor	0.21	2859	3049	2.73	34.6	42.2

5. 評　価

本章では，準パススルー型仮想マシンモニタの実際の実装に基づくコードサイズ及びオーバーヘッドの測定結果について述べる．評価に使用したのは，BitVisor[10] のバージョン 0.3 である．BitVisor 0.3 では，Intel の VT 機能を搭載した CPU 上で動作する仮想マシンモニタのコア機能及び ATA デバイスに対する準パススルードライバが実装されている．ATA の準パススルードライバでは，XTS-AES[11] を用いた暗号化機能が実装されている．本実装上では，Windows や Linux など多くのゲスト OS が実際に動作している．

5.1 コードサイズ

まず，sloccount[12] を用いてソースコードの行数を測定した．sloccount は，コメントや空行などを除いた実質的な行数を測定するツールである．表 1 に測定結果を示す．仮想マシンモニタのコア機能は，21,582 行であったが，ランタイム時には不要な機能を除くと 13,789 行であった．この中には，シャドウページテーブルに基づくメモリ保護機能が 1,130 行，命令インタプリタが 2,239 行含まれている．ATA の準パススルードライバは 1,287 行（暗号化機能は除く）であった．

上記の結果から，準パススルー型仮想マシンモニタでは，Windows などが動作する実用的な機能を実装しても，比較的小さなコードサイズにできることがわかった．シャドウページテーブルは，ハードウェアによるページテーブル仮想化支援機能により不要になるため，更なるコードサイズ削減が期待できる．

5.2 オーバーヘッド

本節では，準パススルー型仮想マシンモニタによるオーバーヘッドの測定結果を示す．実験に使用したマシンは，CPU が Intel Core 2 Duo E6850(3.0GHz)，メモリが 2GB，ディスクが Western Digital Raptor WD740GD である．ゲスト OS は 32bit 版の Fedora 8 を使用した．カーネルバージョンは 2.6.25.9-40.fc8 である．BitVisor は 64bit 版でコンパイルしている．

まず最初に，lmbench[13] を使用して，OS の基本性能に与える影響を測定した．現在の実装ではシャドウページテーブルを用いており，ページフォルトが発生するたびに仮想マシンモニタに制御が移る．従って，OS のメモリ管理にオーバーヘッドが生じる．実験結果を表 2 に示す．`null` はヌルシステムコールの実行時間だが，ゲスト OS におけるユーザモードとカーネルモードの切り替え時は仮想マシンモニタに制御が渡らないため，オーバーヘッドはほとんどない．`fork`,`exec` はそれぞれのシステムコールの実行時間，`prot` はページの書き込み違反の処理時間，`page` はページフォルトの処理時間，`ctx` はコンテキストスイッチ（2 プロセス，64KB）の実行時間である．いずれも仮想マシンモニタに制御が渡るため，オーバヘッドが生じている．

表 3 ストレージアクセスのオーバーヘッド (MB/s)
Table 3 Overhead of storage access (MB/s)

ホスト	4KB	512KB	10MB
Linux	19.9	122.5	87.2
コアのみ	13.2	110.5	86.9
コア+ドライバ	10.6	99.8	86.1
コア+ドライバ+暗号化	6.73	51.9	42.8

次にストレージの暗号化機能の性能を測定した．測定は，Linux 単独，BitVisor のコア機能のみ，ATA ドライバあり（暗号化あり，暗号化なし）の 4 通りでおこなった．4KB, 64KB, 512KB, 10MB のそれぞれのサイズでブロックデバイスを読み込む時間を測定した．これはキャッシュの効果を含んでいる．実験結果を表 3 に示す．サイズが小さいときは，コア機能によるオーバーヘッドが大きいが，サイズが大きくなるにつれて相対的に影響は小さくなる．暗号化は CPU でおこなうため，オーバーヘッドは 49〜63%程度かかる．

6. 関 連 研 究

近年では，仮想マシンモニタでセキュリティ機能を実現する研究がいくつかおこなわれている．

Overshadow[2] では，アプリケーションからカーネルへの情報漏洩を防止するために仮想マシンモニタを用いる手法を提案している．アプリケーションが読み書きするページの内容をカーネルには暗号化して見せることにより，カーネルからのページアクセスは許可しつつも情報の読み取りを防止する方式を実現している．Overshadow では，アプリケーションとカーネル間の通信時に暗号化・復号化などの処理が入るため，OS 依存の方法でスタブルーチンを挿入する必要がある．本研究の手法は，ゲスト OS には依存しない．

SecVisor[1] では，カーネルコードの完全性を保つために仮想マシンモニタを用いる手法を提案している．メモリ管理や DMA のアクセス制御により，仮想マシンモニタが許可したコード以外はカーネルモードでの実行を禁止する．SecVisor では，セキュリティに機能を絞ることで仮想マシンモニタを非常に小さく出来るという点で思想が似ているが，I/O アクセスは完全な

パススルーとなっており，デバイスI/Oに対して暗号化などのセキュリティ機能を実現することは難しい．

LVMM[14]では，通常のOSが動作するプライマリVMに加えて，リモート管理をおこなうサービスOSが動作するセカンダリVMの2つのVMを同時に動作させる．セカンダリVMでネットワーク通信をおこなうために，ネットワークデバイスのみを仮想化・多重化しており，その他のデバイスはプライマリVMのOSが管理する．LVMMでは，ネットワークデバイスは完全な仮想化，その他のデバイスは完全なパススルーとなっており，準パススルー方式とは異なる．

Shaferら[15]は，ソフトウェアのみでIOMMUと同等の機能を実現する方式を提案している．この方式は，本論文のシャドウDMAディスクリプタと方式が似ている．しかし，Shaferらの方式ではアクセス制御のみをおこない，転送データの内容は変換しない．シャドウDMAディスクリプタでは，アクセス制御に加えて転送されるデータの内容を変換することができる．

7. まとめ

本論文では，準パススルー型仮想マシンモニタを提案した．仮想マシンモニタのコードサイズを最小化するために，ゲストOSからデバイスへのアクセスを可能な限りパススルーとしつつ，セキュリティ機能の実現に最低限必要なアクセスだけを捕える方式とした．DMA転送の内容を捕捉するために，シャドウDMAディスクリプタという仕組みを提案し，仮想マシンモニタ内に完全なデバイスドライバを持たずにデータの内容のみ暗号化などの変換をおこなえるようにした．実際に実装をおこなった結果，仮想マシンモニタ本体のサイズは2万行程度，ATAデバイスに対する準パススルードライバは1200行程度であった．

謝辞

本研究は，平成18年度文部科学省科学技術振興調整費の支援によりおこなわれた．

参考文献

1) Seshadri, A., Luk, M., Qu, N. and Perrig, A.: SecVisor: A Tiny Hypervisor to Provide Lifetime Kernel Code Integrity for Commodity OSes, *Proc. 21st ACM Symposium on Operating Systems Principles*, pp. 335–350 (2007).
2) Chen, X., Garfinkel, T., Lewis, E. C., Subrahmanyam, P., Waldspurger, C. A., Boneh, D., Dwoskin, J. and Ports, D. R.: Overshadow: A Virtualization-Based Approach to Retrofitting Protection in Commodity Operating Systems, *Proc. 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 2–13 (2008).
3) Robin, J. S.: Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monitor, *Proc. 9th USENIX Security Symposium* (2000).
4) Lipow, M.: Number of Faults per Line of Code, *IEEE Transactions on Software Engineering*, Vol. SE-8, No. 4, pp. 437–439 (1982).
5) Goldberg, R.: *Architectural Principles for Virtual Computer Systems*, PhD Thesis, Harvard University (1973).
6) Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the Art of Virtualization, *Proc. 19th ACM Symposium on Operating Systems Principles*, New York, NY, USA, ACM, pp. 164–177 (2003).
7) Willmann, P., Rixner, S. and Cox, A. L.: Protection Strategies for Direct Access to Virtualized I/O Devices, *Proc. 2008 USENIX Annual Technical Conference*, pp. 15–28 (2008).
8) Ganapathy, V., Renzelmann, M. J., Balakrishnan, A., Swift, M. M. and Jha, S.: The Design and Implementation of Microdrivers, *Proc. 13th International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 168–178 (2008).
9) Ben-Yehuda, M., Xenidis, J., Ostrowski, M., Rister, K., Bruemmer, A. and Doorn, L. V.: The Price of Safety: Evaluating IOMMU Performance, *Proc.Linux Symposium*, pp. 9–20 (2007).
10) 筑波大学: セキュアVMプロジェクト. `http://www.securevm.org/`.
11) IEEE: IEEE Standard for Cryptographic Protection of Data on Block-Oriented Storage Devices (2008). IEEE Std 1619-2007.
12) Wheeler, D.A.: Counting Source Lines of Code (SLOC). `http://www.dwheeler.com/sloc/`.
13) McVoy, L. and Staelin, C.: lmbench: Portable tools for performance analysis, *Proc. 1996 USENIX Annual Technical Conference* (1996).
14) Ramachandran, M., Smith, N., Wood, M., Garg, S., Stanley, J., Eduri, E., Rappoport, R., Chobotaro, A., Klotz, C. and Janz, L.: New Client Virtualization Usage Models Using Intel Virtualization Technology, *Intel Technology Journal*, Vol. 10, No. 03, pp. 205–216 (2006).
15) Shafer, J., Carr, D., Menon, A., Rixner, S., Cox, A. L., Zwaenepoel, W. and Willmann, P.: Concurrent Direct Network Access for Virtual Machine Monitors, *Proc.IEEE 13th International Symposium on High Performance Computer Architecture*, pp. 306–317 (2007).