Vol. 46 No. 7 情報処理学会論文誌 July 2005

# 軽量仮想計算機モニタを用いたOSデバッグ方式の提案

竹内 理†

近年，PC/AT 互換機上に高速 I/O を特徴とする独自 OS を開発するニーズが増大している．しかし，従来の PC/AT 互換機上で動作する OS 向けのデバッグ環境は，デバッグ環境の安定稼働が保証できる，様々な OS や I/O デバイスに大きな開発なく適用できる，デバッグ時にも高速動作（特に高速な I/O 実行）が可能である，の 3 条件を同時に充足することができなかった．本論文では，これらの 3 条件を同時に充足する独自 OS デバッグ方式として，軽量な仮想計算機モニタを用いた独自 OS デバッグ方式を提案する．本方式では，開発中の独自 OS をリモートデバッグ機能を備える軽量仮想計算機モニタ上で動作させる．軽量仮想計算機モニタは実ハードウェアと同様のインタフェースを独自 OS に提供するため，PC/AT 互換上で動作するいかなる独自 OS にも本方式は適用可能である．また，軽量仮想計算機モニタは，軽量メモリ領域保護機能を備え，独自 OS の異常動作時にも，仮想計算機モニタが保持するリモートデバッグ機能の安定稼働を保証する．さらに，軽量仮想計算機モニタは部分ハードウェアエミュレーションを行い，独自 OS が様々な I/O デバイス上で高い I/O 性能を達成することを可能にする．本デバッグ方式の定量的な性能評価を行い，提案した仮想計算機モニタ上で動作する独自 OS（HiTactix）は，Hosted Virtual Machine Monitor 上で動作するときと比して，I/O 性能が 5.4 倍程度向上すること等が明らかになった．

# OS Debugging Method Using a Lightweight Virtual Machine Monitor

Tadashi Takeuchi†

Recently, demands for implementing original operating systems which can achieve high I/O performance on PC/AT compatible hardware have been increasing. However, conventional operating system debugging environments have not been able to satisfy the following demands at the same time: 1) assuring the stability of the debugging environments 2) an easy customization of the debugging environment for new operating systems and new I/O devices, 3) efficient execution (especially I/O execution) of the operating system under the debugging environment. In this paper, we propose a novel operating system debugging method using the lightweight virtual machine monitor. This method can be applied to any operating systems on PC/AT compatible hardware, bacause the virtual machine monitor provides real hardware interfaces. The monitor also provides lightweight memory region protection mechanism, which enables the remote debugging function in the monitor to work stably even when the operating system under this debugging environment works inproperly owing to its bugs. Besides, the monitor provides partial emulation mechanism. Owing to this mechanism, the operating system on the monitor can achieve efficient I/O execution on various I/O devices. We evaluated this debugging method quantitavively, and confirmed that this method can provide about 5.4 times as high I/O performance as the existing hosted virtual machine monitor.

## 1. はじめに

近年，PC/AT 互換機上に自社開発の独自 OS を搭載し，I/O 性能において差別化をはかるアプライアンスサーバ製品が数多く市場に出回ってきている．Novell 社のキャッシュサーバ製品 Volera Media Excelarator[1] ☆はその典型例である．また，Kasenna Streaming Accelerator[2] ☆☆のように，独自改変を加えた Linux を用いている製品もある．日立グループにおいても，自社開発のストリーミング専用 OS HiTactix[3)~5)] を搭載した映像配信サーバ HEC21/VS[6] ☆☆☆を製品化している．

これにともない，高速 I/O 機能を特徴として持つ独自 OS を PC/AT 互換機上で開発するニーズが増大している．このような独自 OS は，より高速な I/O

† 株式会社日立製作所システム開発研究所
Hitachi Ltd., Systems Development Laboratory

☆ Volera Media Excelerator は米国 Novell 社の登録商標です．
☆☆ Kasenna Streaming Accelerator は米国 Kasenna 社の登録商標です．
☆☆☆ HEC21/VS は日立エンジニアリング株式会社の登録商標です．

デバイス（10ギガビットEthernet☆対応デバイス等）や高速I/O支援機能を搭載した高機能I/Oデバイス（I20対応デバイス等）が出現するたびに，デバイスドライバの開発や，I/Oデバイス搭載機能を利用した新規I/O機能の開発を行う必要がある．そのため，このような独自OSのデバッグ環境は，様々なI/Oデバイスに対応可能で，かつ，高負荷I/O実行時の機能デバッグが効率的に行えることが望ましい．ICE等のデバッグ用ハードウェアが提供されていないPC/AT互換機では，従来，OSのデバッグ環境として，

(1) 汎用OS上に構築されたハードウェアシミュレータ（または仮想計算機モニタ）および当該ハードウェアシミュレータと連動するソフトウェアデバッガ[7)~9)]

(2) ソフトウェアリモートデバッガ[10)]

(3) OS内部に組み込まれた当該OS専用のデバッガ[11)]

等を利用していた．しかし，これらはいずれも，A）デバッグ環境の安定稼働が保証できる，B）様々なOSやI/Oデバイスに追加開発なく適用できる，C）デバッグ時にも高速動作（特に高速なI/O実行）が可能である，の3条件を同時に充足できない．(1)は，I/Oデバイス（またはI/Oプロセッサ）や特権命令の厳密な動作シミュレーションをハードウェアシミュレータ（または仮想計算機モニタ）で行うので，新規I/Oデバイスのエミュレータに大きな開発が必要になるうえ，I/O性能も十分得られず，B），C）の条件を充足できない．(2)は，開発中のOSのバグに起因する異常動作によりリモートデバッグ動作（たとえばシリアル通信）を阻害する可能性があるためA）の条件を充足できない．また，(3)は，OSが変わると多大な開発が必要となるうえ，OSとデバッガ間のメモリ保護等も実現できていないため，A），B）の条件を充足できない．

本論文では，上記3条件を同時に充足する，PC/AT互換機上で動作する独自OSのデバッグ方式として，軽量仮想計算機モニタを用いる方式を提案する．本方式では，リモートデバッグ機能を備える軽量仮想計算機モニタ上で開発中の独自OSを動作させる．軽量仮想計算機モニタは，従来の仮想計算機モニタと同様に，実ハードウェアと同様のインタフェースを独自OSに提供する．そのため，PC/AT互換上で動作するいかなる独自OSにも，本方式は適用可能である．

本軽量仮想計算機モニタは，従来の仮想計算機モニタと同様に，ハードウェア資源の仮想化を行う．しかしこの仮想化は，従来のように，1つのハードウェア上に複数のOSを同時実行させることが目的ではない．仮想計算機モニタ上で動作するOSが実ハードウェア資源に直接アクセスすることを防ぐことで，たとえOSが異常動作を行っても実ハードウェア資源の状態を正常に保ち，仮想計算機モニタ内に保持するリモートデバッグ機能を安定稼働させることが目的である．

本軽量仮想計算機モニタは，仮想化対象となるハードウェア資源を，リモートデバッグ機能の安定稼働の保証のために必要最小限な資源のみに絞る部分ハードウェアエミュレーション機能を持つ．独自OSは上記仮想化対象以外のI/Oデバイスに直接アクセスできるため，様々なI/Oデバイスに対するI/O処理を高速に実行できる．さらに，従来の仮想計算機モニタよりも低オーバヘッドで仮想計算機モニタとOS間のメモリ保護を実現する軽量メモリ領域保護機能もあわせて提供し，開発中のOSが異常動作しても，仮想計算機モニタが保持するリモートデバッグ機能が安定稼働することを保証する．

以下，2章では，新規に提案する軽量仮想計算機モニタを用いた独自OSデバッグ方式の概要について述べる．そして，提供する軽量仮想計算機モニタに，部分ハードウェアエミュレーション機能と軽量メモリ領域保護機能を備えることで，上記A）~C）の3条件の同時充足が可能であることを示す．次に3章と4章において，軽量仮想計算機モニタの特徴機能である部分ハードウェアエミュレーション機能と軽量メモリ領域保護機能の実装方式の概要について示す．次に，5章において，部分ハードウェアエミュレーション機能と軽量メモリ領域保護機能を実装する際に直面した実装上の問題点について示す．6章では，本OSデバッグ方式の定量的な評価結果について述べる．最後に7章で関連研究との比較について述べ，8章でまとめを述べる．

## 2. 軽量計算機モニタを用いたOSデバッグ方式の概要

本章では，本論文で新規に提案する軽量仮想計算機モニタを用いた独自OSデバッグ方式の概要について述べる．まず，2.1節で，提案するOSデバッグ方式を適用可能なデバッグ対象，およびデバッグ範囲について述べる．次に，2.2節で，提案するデバッグ方式の方式概要について述べる．

### 2.1 デバッグ対象，範囲

提案するデバッグ方式は，独自OSをデバッグ対象

☆ Ethernetは米国Xerox社の登録商標です．

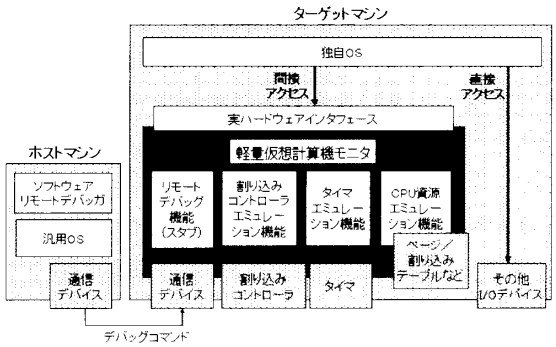


**図1** 提案デバッグ環境の構成
Fig. 1 Proposed debugging environment architecture.

とする．ここでいう「独自OS」とは，ソースコードを入手可能であるOSのことを指す．また，独自OSとしては，特に高速I/Oデバイスを介した高いI/O性能を達成することを指向したOSに着目する．

また，本デバッグ方式のデバッグ範囲はスーパバイザモードで動作する独自OSのデバッグに限る．一般にユーザモードで動作する独自OS上アプリケーションのデバッグについては考慮せず，これらのアプリケーションはもし動作するとしても安定稼働することを仮定する．

上記デバッグ対象，範囲を持つことで，たとえば，独自OSに新機能コードの追加をしており，その簡単なテストアプリケーションとともに新機能コードのデバッグを行う，あるいは，独自OS向けの新規デバイスドライバを追加開発しており，既開発の（安定稼働している）アプリケーションを用いて新規デバイスドライバのデバッグをする等の場合に提案デバッグ方式を利用可能になる．

### 2.2 方式概要

提案するデバッグ方式は，PC/AT互換機上で動作する独自OSの開発の際に，以下の条件を充足するデバッグ環境を提供することを目標としている．

(1) デバッグ環境の安定稼働が保証できる．
(2) 様々なOSやI/Oデバイスに追加開発なく適用できる．
(3) デバッグ時にも高いI/O性能で動作させることができる．

上記目的を達成するために，本デバッグ方式では，**図1**に示す構成を持つデバッグ環境を提供する．

本デバッグ環境は，従来のソフトウェアリモートデバッガを利用する際のデバッグ環境と類似した構成を持つ．デバッグ環境は，ホストマシンとターゲットマシンからなり，ホストマシン上ではリモートデバッグ機能を持つソフトウェアデバッガが動作する．ソフトウェアデバッガは，デバッグコマンド（ターゲットマシンのメモリ参照/更新，レジスタ参照/更新等）をユーザから受け取り，当該コマンドをターゲットマシンに転送する．

しかし，本方式は，独自OSとは独立に開発された軽量仮想計算機モニタがあらかじめターゲットマシンにデバッグ環境として組み込まれている点が従来のソフトウェアリモートデバッガと異なる．軽量仮想計算機モニタには，上記デバッグコマンドの受信，コマンドの実行，コマンド実行結果の返信を行うリモートデバッグ機能を保持する．さらに，上記リモートデバッグ機能が利用するハードウェア資源の仮想化（エミュレーション）機能も提供する．

軽量仮想計算機モニタは，リモートデバッグ機能が使用するハードウェア資源（割込みコントローラ/タイマ/ページテーブル/割込みハンドラテーブル等）のみをエミュレートする部分ハードウェアエミュレーション機能を持つ．独自OSは，当該資源に軽量仮想計算機モニタを介してアクセスする．そのため，独自OSがバグに起因する異常動作を行っても，これらのハードウェア資源の状態を正常に保てる．しかし，それ以外のハードウェア資源，特に高速I/Oデバイス（SCSIコントローラ，Ethernetコントローラ等）へのアクセスは，独自OSが直接行える．

仮想化されたハードウェア資源へのアクセスインタフェースは，実ハードウェア資源へのアクセスインタフェースと同様である．そのため，実ハードウェア上で動作する独自OSは，大きな改変を加えることなく，本軽量仮想計算機モニタ上でも動作する．ただし，実ハードウェア上で動作する独自OSを本軽量仮想計算機モニタ上で動作させる際に，バグの混入の恐れの少ない簡単なコードの改変（アセンブリ言語で記述された命令列の機械的な置き換え等）は行ってもよいことにした．PC/AT互換機上で動作する従来の仮想計算機モニタは，バイナリで提供される複数の商用OSを1つのマシン上で同時に利用することを目的としている．そのため，上記のようなコード改変は目的達成を不可能にし，あまり採用されていなかった．代わりに，仮想計算機モニタは，独自OS実行時にバイナリの動的な置き換えを行っていたが[12]，仮想計算機モニタ上で動作する独自OSの実行性能の低下を招いていた．本仮想計算機モニタの目的は独自OSのデバッグであるため，完全なバイナリ互換性の確保より，独自OSの実行性能の確保を優先することにした．

また，IA-32仕様CPUが提供するページテーブルでは，各ページのアクセス保護を定義する際に2段階の特権レベルのみ（スーパバイザレベルとユーザレベ

ル）が使用できる[13]☆．軽量仮想計算機モニタに含まれるリモートデバッグ機能を安定稼働させるためには，軽量仮想計算機モニタ，独自OS，独自OS上で動作するアプリケーションをそれぞれ別の特権レベルで動作させる必要がある．そのため，本軽量仮想計算機モニタでは，軽量仮想計算機モニタと独自OSはスーパバイザレベルで，独自OS上で動作するアプリケーションはユーザレベルで動作させている．さらに，軽量仮想計算機モニタの動作領域への独自OSからの不正アクセスを防止する軽量メモリ領域保護機能も提供し，たとえ独自OSが，バグに起因した異常動作を行っても，リモートデバッグ機能が安定して稼働することを保証している．本軽量メモリ領域保護機能も，独自OS側にバグ混入の恐れの少ない簡単なコード改変（ページテーブル使用領域の仮想計算機モニタへの通知コードの追加等）を行うことにより，バイナリの動的置き換えをすることなく，保護を低オーバヘッドで実現している．

上記デバッグ方式は，軽量メモリ保護機能の提供と，リモートデバッグ機能が利用するハードウェア資源の仮想化を行う部分エミュレーション機能により目標(1)を実現する．また，軽量仮想計算機モニタが実ハードウェアと同様のインタフェースを独自OSに提供し，さらに独自OSに様々な高速I/Oデバイスへの直接アクセスを許可することにより，目標(2)を実現する．さらに上記直接アクセスの許可は，目標(3)の実現にも寄与している．1章で示したとおり，デバッグ環境の安定稼働を保証する従来のデバッグ方式は，ハードウェアシミュレータ（または仮想計算機モニタ）を用いる方式だけであるが，提案方式はこの方式よりもデバッグ中のOSに提供するI/O性能を改善する．I/O性能改善の定量的な評価結果は6章で示す．

## 3. 部分ハードウェアエミュレーション機能の実装方式

本章では，前章で述べた部分ハードウェアシミュレーション機能の実装方式について述べる．まず，3.1節で，エミュレーションの対象とすべきハードウェア資源について明らかにする．次に，3.2節で，部分ハードウェアエミュレーション機能が上記ハードウェア資源のエミュレーションをどのように実装しているか，その概要を述べる．

☆ IA-32仕様CPU自体は，レベル0〜3の4段階の特権レベルを提供している．しかし，ページテーブルを用いて各ページのアクセス保護を設定する際には，レベル0〜2の特権レベルは同一レベルとして扱われる．

### 3.1 エミュレーション対象のハードウェア資源

部分ハードウェアエミュレーション機能では，軽量仮想計算機モニタに含まれるリモートデバッグ機能が使用するハードウェア資源のみをエミュレートする．

リモートデバッグ機能は，外部デバイスとして，割込みコントローラ（8259Aチップ），タイマ（8253チップ），ホストマシンとターゲットマシン間の通信を行う比較的低速な通信デバイス（現在の実装ではIEEE1394コントローラ）を使用する．現在の軽量仮想計算機モニタの実装では，割込みコントローラとタイマのエミュレートを行っている．しかし，独自OSからはホストマシンとターゲットマシン間の通信を行う通信デバイスへのアクセスを行わない（独自OSは当該通信デバイスのデバイスドライバを保持しない）ことを前提とし，当該通信デバイスのエミュレートは行っていない．現在までに軽量仮想計算機モニタ上での動作を確認したHiTactix，$\mu$ITRON仕様OS（TOPPERS/JSP）[14]，BSD/OSバージョン4.3[15]☆☆は，この条件を満たしている☆☆☆．

また，リモートデバッグ機能は独自OSと同じCPUを時分割で共有する．そのため，独自OSがCPUを占有し動作し始めた後でも，必要なときにリモートデバッグ機能に制御が戻ることを保証する必要がある．具体的には，独自OSが異常動作を行い例外が発生した場合にでも，確実に独自OS定義の例外ハンドラの起動前に軽量仮想計算機モニタ定義の例外ハンドラが起動し，リモートデバッグ機能を動作させる必要がある．リモートデバッグ機能の動作により，ホストマシン上で動作するソフトウェアデバッガに当該例外発生を通知可能になる．また，たとえ独自OSが割込み発生をマスクしつつ無限ループに陥った場合にでも，一定時間間隔に1回は軽量仮想計算機モニタ定義の割込みハンドラに制御が移ることも保証しなければならない．ホストマシン上で動作するソフトウェアデバッガから発行されるブレーク要求に，リモートデバッグ機能が応答する必要があるためである．

この実現のために，軽量仮想計算機モニタは，割込

☆☆ BSD/OSは米国Windriver社の登録商標です．

☆☆☆ 本条件を満たさない独自OSの場合は，従来の仮想計算機モニタと同様に当該通信デバイスのエミュレートを行う必要がある．しかし，そのエミュレーションオーバヘッドが問題になるほど，比較的低速な通信デバイスに高い負荷をかける独自OSは，本論文では対象としない．また，当該通信デバイスのエミュレーションを行わなくても，デバッグ環境の安定稼働は保証できる．4章で示す軽量メモリ領域保護機能を利用して，独自OSから当該デバイスのレジスタ群へのアクセスを仮想計算機モニタは防止している．

み/例外ハンドラ起動時に参照するCPUのハードウェア資源を仮想化する．具体的には，割込み/例外ハンドラテーブル（IDT），セグメンテーションテーブル（GDT/LDT），ページテーブル，割込み制御ビット（EFLAGSレジスタのIFビット），コンテキスト格納領域（TSS）の仮想化を行っている．

### 3.2 実装方式の概要

前節で示したハードウェア資源のエミュレーションを実現するために，本軽量仮想計算機モニタでは，仮想計算機モニタを特権レベル0で，独自OSを特権レベル1で，独自OS上のアプリケーションを特権レベル3で動作させる．本軽量仮想計算機モニタでは，実ハードウェア資源上で，独自OSが特権レベル0で，独自OS上のアプリケーションを特権レベル3で走行させることを仮定している．この仮定は，IA-32仕様CPU上で動作するOS/2以降のよく知られたOSすべてにおいて成立する[16]．

割込みコントローラとタイマ以外のデバイスを独自OSから直接アクセスさせるために，軽量仮想計算機モニタはコンテキスト格納領域内のI/O許可マップを設定する．割込みコントローラとタイマデバイスのI/Oポートは特権レベル0からのみアクセス可能にするが，それ以外のポートはどの特権レベルからもアクセス可能にする☆．独自OSは，割込みコントローラとタイマデバイスのI/Oポートにアクセスしようとすると，一般保護例外が発生する．例外発生を契機に，軽量仮想計算機モニタに制御が移り，割込みコントローラまたはタイマエミュレーションモジュールが走行する．しかし，それ以外のデバイスのI/Oポートには，独自OSは軽量仮想計算機モニタを介さずに直接アクセスする．また，PCIデバイス（ただし，ホストマシンとターゲットマシン間の通信を行う通信デバイスは除く）のレジスタ群をマップするメモリ領域についても，独自OSから例外を発生することなく当該メモリ領域にアクセス可能にすべく，軽量仮想計算機モニタはページテーブルの設定を行う．

また，割込み/例外発生時に確実に軽量仮想計算機モニタ定義の割込み/例外ハンドラが起動することを保証するために，CPUが提供するハードウェア資源は，**表1**に示す仮想化がなされる．

☆ 割込みコントローラ，タイマ以外のI/Oポートをどの特権レベルからもアクセス可能にすることにより，独自OS上のアプリケーションは割込みコントローラ，タイマ以外のデバイスに直接アクセスが可能になる．そのため，独自OS上のアプリケーションのバグが独自OSの異常動作を引き起こす可能性があるが，本論文では独自OSのデバッグ機能提供を目的としているため，問題ないと判断した．

**表1** CPU資源の仮想化方法の概要
Table 1 Overviews of CPU resource virtualization method.

| 資源名 | 仮想化方法 |
|---|---|
| IDT | IDTのシャドウテーブルをモニタは保持．<br>シャドウテーブルには，モニタ定義のハンドラを登録，<br>独自OS定義のハンドラを登録しない．<br>モニタ定義のハンドラは，モニタ動作用のコード/メモリセグメントで動作． |
| GDT<br>LDT | GDT/LDTのシャドウテーブルをモニタは保持．<br>独自OS動作用のコード/メモリセグメントディスクリプタは特権レベルを1に設定．<br>モニタ動作用のコード/メモリセグメントディスクリプタを追加登録，特権レベルは0に設定．<br>独自OS定義のゲート/TSSディスクリプタは無効化． |
| ページテーブル | 4章を参照．<br>シャドウページテーブルをモニタ保持．<br>モニタ定義の特権スタック，GDT/IDT/TSS，ハンドラコードの物理常駐を保証． |
| IFビット | ソフトウェアで代替ビットを保持．<br>代替ビットにかかわらず，独自OS実行中は割込み許可． |
| TSS | TSSのシャドウをモニタは保持．<br>モニタ定義の特権スタックセグメントセレクタ，スタックポインタをシャドウのSS0，ESP0に登録．<br>I/O許可ビットマップを設定． |

独自OS（または独自OS上で動作するアプリケーション）走行中に割込み/例外が発生した場合は，以下の手順で軽量仮想計算機モニタ定義の割込み/例外ハンドラに制御がわたり，さらに必要に応じて独自OS定義の割込み/例外ハンドラを起動する．

(1) 割込み/例外ハンドラテーブルのベースポインタ（IDTR）およびセグメントテーブルのベースポインタ（GDTR）はシャドウテーブルを指している．割込み/例外が発生すると，軽量仮想計算機モニタ定義のハンドラが起動．コードセグメントも，モニタ動作用のセグメントに更新．

(2) コンテキスト格納領域（TR）もシャドウを指しているため，スタックセグメントもシャドウのSS0に格納されているモニタ動作用のセグメントに更新．スタックポインタはESP0に格納されている軽量仮想計算機モニタ定義の特権スタックへのポインタに更新．

(3) 割込み制御ビットは独自OS（または独自OS上のアプリケーション）走行中はつねに割込み許可状態に設定．そのため，割込み発生時にも，ただちに軽量仮想計算機モニタに制御がわたる．また，軽量仮想計算機モニタ定義の特権

スタック領域，シャドウセグメントテーブル，シャドウ割込み/例外ハンドラテーブル，シャドウコンテキスト格納領域，軽量仮想計算機モニタ定義の割込み/例外ハンドラコードは4章に示す方法で物理常駐が保証されている．また，GDTR/IDTR/TR等の更新はスーパバイザレベルではあっても特権レベル1で動作する独自OSからは行えないため，これらのレジスタの格納値も正常であることが保証される．そのため，(1)，(2)の処理途中に別な例外は発生し得ず，確実に割込み/例外ハンドラが起動される．

(4) リモートデバッグ機能，デバイスエミュレーション処理等の軽量仮想計算機モニタ内のモジュールが動作．発生した例外が，リモートデバッグ機能起動のための例外（デバッグ例外）やデバイスエミュレーションのための例外であった場合には処理を完了．それ以外は(5)以降の独自OSへの割込み/例外通知処理を継続．

(5) 割込み発生時には，割込み制御ビットの代替ビットを検査．代替ビットが割込み禁止状態の場合には，独自OSへの割込み通知をペンディング．

(6) OS定義の割込み/例外テーブル，コンテキスト格納領域を参照．ソフトウェアにより，割込み/例外発生にともなうスタック切替え処理，命令ポインタの更新をエミュレート．独自OS定義の割込み/例外ハンドラが起動する．

## 4. 軽量メモリ領域保護機能の実装方式

2章で述べたとおり，軽量メモリ領域保護機能は，軽量仮想計算機モニタの動作領域への独自OSからの不正アクセスを防止する機能である．本章では，この機能の実装方式について説明する．まず，4.1節で，本機能の実装方針について述べ，さらに，4.2節と4.3節で，本機能の実装方式の概要について述べる．

### 4.1 実装方針

軽量メモリ領域保護機能では，軽量仮想計算機モニタの動作領域をダイナミックマッピング領域とスタティックマッピング領域の2つの領域に分割して管理している．

ダイナミックマッピング領域とは，軽量仮想計算機モニタ走行中ばかりでなく，独自OS走行中にも，独自OSが使用していない仮想空間領域にマッピングされるメモリ領域である．独自OSが使用していない仮想空間領域はたえず変化しうるので，独自OS走行中は，当該領域の仮想空間上のアドレスを動的に決定する．一方，スタティックマッピング領域は，独自OS走行中にはマッピングされず，軽量仮想計算機モニタ動作時にのみマッピングされる領域である．軽量仮想計算機モニタ走行中に使用する仮想空間の領域配置は独自OSとは独立に決定できるため，スタティックマッピング領域の仮想空間上のアドレスは固定でよい．このように軽量仮想計算機モニタの動作領域を2つの領域に分割することで，マッピングアドレスを動的に更新する際に必要となる処理オーバヘッドの低減，必要な空き仮想空間領域量の低減，独自OSからの不正な書き込みの防止対象とすべきメモリ領域の低減，を実現している．

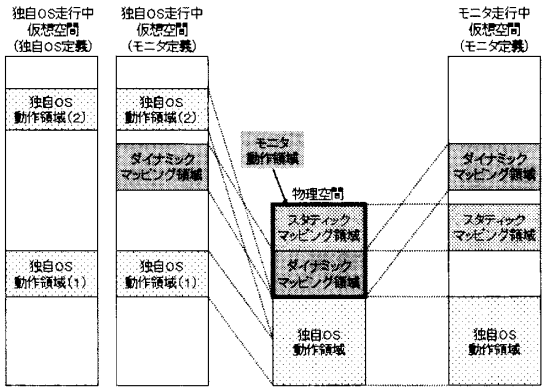


図2 仮想空間構成
Fig. 2 Virtual space organizations.

ダイナミックマッピング領域には，独自OSの通常走行時や，軽量仮想計算機モニタ定義の割込み/例外ハンドラの起動の際に参照する可能性がある軽量仮想計算機モニタのコードおよびデータ構造（シャドウ割込み/例外ハンドラテーブル等）が格納されている．一方，スタティックマッピング領域には，残りの軽量仮想計算機モニタのコードおよびデータ構造が格納されている☆．

図2に示すとおり，独自OS走行中は，独自OS定義の仮想空間に，ダイナミックマッピング領域が追加マッピングされている．3.2節で述べたとおり，軽量仮想計算機モニタは，独自OSが使用しているページテーブルのシャドウを保持する．シャドウページテーブルは，独自OSが使用しているページテーブルの単純なコピーと，ダイナミックマッピング領域の追加ページマッピング情報からなる．独自OS走行中に，割込み/例外が発生すると，ダイナミックマッピング領域内に格納されている軽量仮想計算機モニタ定義のコード（割込み/例外ハンドラの入口コード）が仮想

☆ ホストマシンとターゲットマシン間の通信を行う通信デバイスのレジスタ群がマッピングされているメモリ領域もスタティックマッピング領域に含まれる．

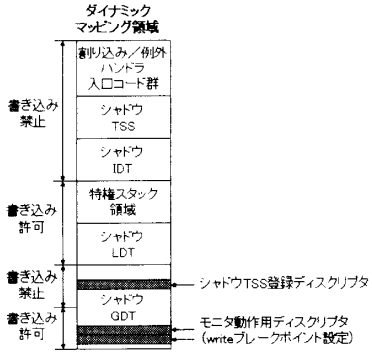


図3 ダイナミックマッピング領域の構成
Fig. 3 Dyanmic mapping region organizations.

空間の切替えを行い，スタティックマッピング領域に含まれるコードおよびデータ構造にもアクセス可能になる．

ダイナミックマッピング領域は，独自OS走行中に使用する仮想空間の空き領域にマッピングされる．どの領域にマッピングされても走行可能とするために，当該マッピングの実行前に，ダイナミックマッピング領域内のコードやデータ構造に含まれるアドレス情報の更新を行う．また，ダイナミックマッピング領域内のデータ構造に，軽量仮想計算機モニタ動作時にもアクセス可能とすべく，軽量仮想計算機モニタ走行中に使用する仮想空間におけるダイナミックマッピング領域のページマッピング情報も更新する．

軽量メモリ領域保護機能では，以下の2つを保証することにより，独自OSから軽量仮想計算機モニタのコードやデータ構造への不正アクセスを防止する．

- ダイナミックマッピング領域への不正書き込みを独自OSが行うことを防止する．ここでいう「不正書き込み」とは，独自OSの書き込みにより，軽量仮想計算機モニタ定義の割込み/例外ハンドラが起動不可能になることをいう．
- ダイナミックマッピング領域およびスタッティックマッピング領域のページマッピング情報が，独自OS定義のページテーブルに加えられることを防止する．

上記それぞれの実装方式の概要については，次節および次々節において説明する．

### 4.2 ダイナミックマッピング領域保護方式

ダイナミックマッピング領域の構成を**図3**に示す．ダイナミックマッピング領域には，独自OSの通常走行時や，軽量仮想計算機モニタ定義の割込み/例外ハンドラの起動時に参照する可能性がある軽量仮想計算機モニタのコードおよびデータ構造のすべて，具体的には，シャドウセグメントテーションテーブル（GDT/LDTのシャドウ），特権スタック領域（軽量仮想計算機モニタ定義の割込み/例外ハンドラ起動時に使用），シャドウ割込み/例外ハンドラテーブル（IDTのシャドウ），シャドウコンテキスト格納領域（TSSのシャドウ），軽量仮想計算機定義の割込み/例外ハンドラの入口コード（起動すると仮想空間の切替えを実行，その後にハンドラ本体を起動する）が格納されている．

独自OS走行時に使用する仮想空間では，ダイナミックマッピング領域は原則として書き込み禁止でマッピングされている．しかし，シャドウセグメンテーションテーブルと特権スタック領域に対しては，独自OSが正常動作を行っている際にも書き込みが発生する．セグメントレジスタへのセレクタロードの際に，CPUは自動的に，シャドウセグメントテーションテーブルのディスクリプタのビジービットをオンにする．特権スタック領域には，割込み/例外ハンドラ起動時に，現在のスタックセグメントセレクタ，スタックポインタ等の情報が格納される．

そのため，シャドウセグメンテーションテーブルの一部と特権スタック領域は，書き込みも許可してマッピングする代わりに，図3に示す方法によって，正常な軽量仮想計算機モニタ割込み/例外ハンドラの起動を保証している．シャドウセグメンテーションテーブルに含まれる軽量仮想計算機モニタ動作用のセグメントディスクリプタにwriteブレークポイントが設定されている☆．また，シャドウコンテキスト格納領域の登録ディスクリプタは，書き込み禁止ページにマッピングされている．それ以外（シャドウセグメントテーブル内の独自OS動作用のセグメントディスクリプタや特権スタック領域）は書き込み許可ページにマッピングされているが，これらの領域は破壊されても，軽量仮想計算機モニタ定義の割込み/例外ハンドラの正常起動を阻害することはない．

### 4.3 ページマッピングのチェック方式

軽量仮想計算機モニタは，ダイナミックマッピング領域およびスタッティックマッピング領域のページマッピング情報が，独自OS定義のページテーブルに加えられることを，以下の方法により防止する．

☆ 軽量仮想計算機モニタ定義の割込み/例外ハンドラ起動にともない，軽量仮想計算機モニタ動作用のディスクリプタのビジービットはオンに更新される．しかし，IA-32仕様CPUの仕様では，割込み/例外の発生とともにブレークポイントはすべて無効となるため，本ビジービットの更新にともない，デバッグ例外が発生することはない．また，IA-32の仕様により特権レベル1で動作する独自OSはブレークポイントを更新できず，独自OSのバグにより，本writeブレークポイントが無効化されることはない．

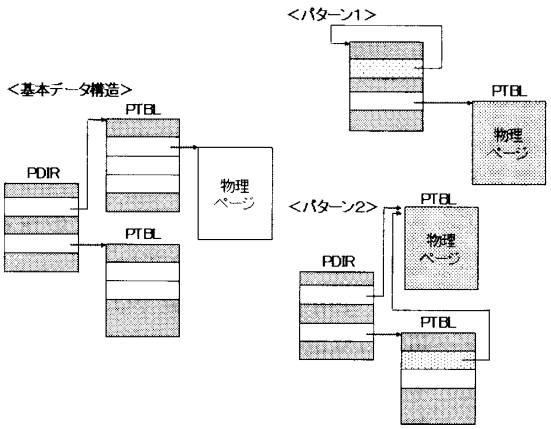


図 4 IA-32 のページテーブルの構成
Fig. 4 IA-32 page table structures.

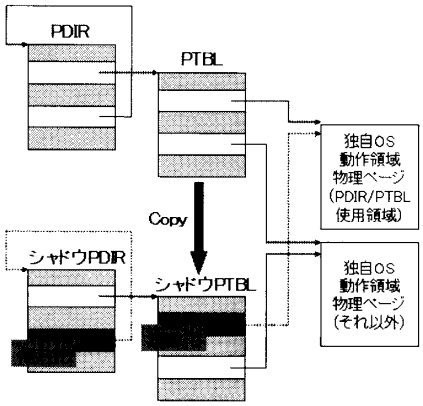


図 5 シャドウページテーブルのデータ構造
Fig. 5 Data structures for shadow page tables.

- 独自 OS 定義のページテーブルとして使用されている物理メモリ領域へのデータ書き込みをページ不在例外として検出する.
- 上記ページ不在例外を契機に軽量仮想計算機モニタが動作する. 軽量仮想計算機モニタは, 書き込みにより, ダイナミックマッピング領域やスタティックマッピング領域へのマッピング情報が生成されないことを確認後, 書き込み処理を実行する.

IA-32 仕様の CPU では, **図 4** で示すような 2 段構造のページテーブルを保持する[13]. 以下では, 1 段目のページテーブルを PDIR, 2 段目のページテーブルを PTBL と表記する.

IA-32 仕様の CPU が提供するページテーブを利用して PDIR/PTBL を仮想空間にマッピングする際には, 以下の 2 つのいずれかの方法により行う.

(1) PDIR のエントリの 1 つに自分自身の PDIR を指し示すエントリを追加する.
(2) PTBL のエントリの 1 つに他の PDIR/PTBL を指し示すエントリを追加する.

(1) を行うと, PDIR の当該エントリに対応する仮想空間領域 (4 MB 分) に, 現仮想空間で使用しているすべての PDIR/PTBL がマッピングされる. 一方, (2) を行うと, PTBL の当該エントリに対応する仮想空間領域 (4 KB 分) に, 当該エントリにより指し示されている PDIR/PTBL がマッピングされる. 実装したページマッピングのチェック方式では, これらの PDIR/PTBL エントリを検索し, かつ対応する PDIR/PTBL のシャドウのエントリを無効化することで, 独自 OS 定義のページテーブルとして使用されている物理メモリ領域へのデータ書き込みを検出している.

上記検出のため, 軽量仮想計算機モニタは, **図 5** に示すシャドウページテーブルを保持する. 軽量仮想計算機モニタが作成するシャドウページテーブルは, PDIR のシャドウと PTBL のシャドウからなる.

PDIR のシャドウの各エントリは, 独自 OS 定義の PTBL の物理アドレスを保持しない. 代わりに, 各 PTBL のシャドウの物理アドレスを保持する. ただし, 2 段目のページテーブルとして独自 OS 定義の PDIR を指し示しているエントリは上記 (1) に対応するエントリと判定し無効化する. 一方, PTBL のシャドウの各エントリには, 独自 OS 定義の PTBL のエントリのコピーを保持する. ただし, すぐ後で述べる方法により, 上記 (2) に対応するエントリを検出し, 当該エントリはコピーを持たず無効化する. ただし (2) でいっている「他の PDIR/PTBL」とは, 現プロセスで使用している PDIR, PTBL だけに限らない. 独自 OS で生成されている全プロセスのいずれかで使用されている独自 OS 定義の PDIR, PTBL すべてを指す. 使用する可能性のある独自 OS 定義の PDIR, PTBL へのいかなる書き込みも, その書き込み実行時に正当性をチェックすることで, 仮想空間の切替えのたびに, 新しい仮想空間のページマッピング情報の正当性をチェックすることを不要にしている.

使用する可能性のある独自 OS 定義の PDIR, PTBL のページマッピング情報は, **図 6** に示すデータ構造を用いて管理する. 本データ構造は, 独自 OS 動作領域内の物理ページごとに, シャドウ PDIR, シャドウ PTBL, PDIR 参照数, PTBL 参照数, マッピングリストを持つ. PDIR 参照数は, 当該物理ページを独自 OS 定義の PDIR として利用している仮想空間の数を示す. PTBL 参照数は, 当該物理ページを 2 段目のページテーブルとして設定している独自 OS 定義の PDIR のエントリの数を示す. マッピングリストは, 当該物理ページへのマッピング情報を持つシャドウ PTBL のエントリのアドレスを格納したリストで

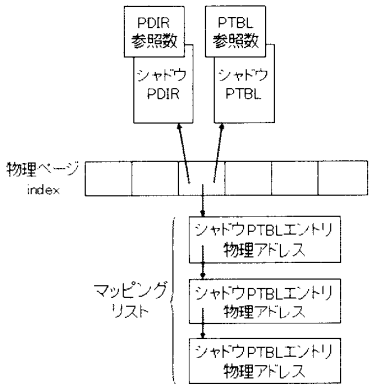


**図 6** ページマッピング情報管理データ構造
Fig. 6 Data structures for managing page mapping information.

ある．

本データ構造により，新しいプロセスが生成され，独自OS定義のPDIR，PTBLとして使用している物理ページが増えた場合に，当該ページをマップしており，上記(2)に対応するPTBLエントリのシャドウを高速に無効化できる．新しいプロセスが生成された場合におけるページマッピング情報の更新手順は以下のとおりである．

(1) 新規プロセスの独自OS定義のPDIRを参照し，各物理ページのPDIR，PTBL参照数を増やす．PDIR/PTBL参照数が0から1に更新された場合には，新たに当該物理ページに定義されている独自OS定義のPDIR/PTBLに対応するシャドウPDIR/PTBLの初期化，具体的には使用物理メモリ領域の確保と，独自OS定義のPDIR/PTBLに基づいた当該領域の初期化を実施する．

(2) 新規プロセスの独自OS定義の各PTBLを参照し，マッピングリストを追加する．

(3) (1)でPDIR/PTBL参照数が0から1に更新された場合には，当該PDIR/PTBLを指し示す独自OS定義のPTBLのエントリすべてを検出し，対応するシャドウを無効化する．具体的には，マッピングリストに登録されている各シャドウPTBLのエントリを無効化する．

独自OS定義のプロセスが削除された場合にも，ページマッピング情報の更新を行い，不要なシャドウPDIR，PTBL用の物理メモリ領域の解放や，シャドウPTBLエントリの有効化等を行う．

本ページマッピング情報の更新は，独自OS定義のプロセスの生成や削除を実行した際にのみ行われる．プロセス切替えの際には，更新は必要ない．独自OS定義のプロセスの生成，削除をどのように軽量仮想計算機モニタが認識するかについては，5.2節で述べる．

## 5. 実装上の問題点

本章では，まず5.1節と5.2節で，3章と4章で述べた部分ハードウェアエミュレーション機能，および軽量メモリ領域保護機能を実装するにあたり直面した問題点とその解決方法について述べる．

本章で述べる実装上の問題点の多くは，独自OSのソースコードの改変により解決している．2.1節で述べたとおり，提案しているOSデバッグ方式では，ソースコードを入手可能な独自OSをデバッグ対象としてるため，ソースコードの改変が必要であることは問題ない．しかし，改変にともない，独自OSにバグが混入する可能性が高まると問題になる．改変の複雑さや改変量に関する考察は5.3節で述べる．

### 5.1 部分ハードウェアエミュレーション機能の実装上の問題点

部分ハードウェアエミュレーション機能の実装を行うために，以下の3点の問題を解決する必要が生じた．

- 独自OSが使用するコードセグメント/スタックセグメントセレクタの変換
- 割込み制御ビットの参照/更新要求の捕捉
- 例外発生要因の特定

以下で，上記3点の問題の内容と解決策について述べる．

第1の問題点は，独自OSが使用するコードセグメント/スタックセグメントセレクタの変換に関する問題である．独自OSは，実ハードウェア上で動作する際には特権レベル0で走行するが，軽量仮想計算機モニタ上で動作する際には特権レベル1で走行する．そのため，軽量仮想計算機モニタ上で独自OSが走行する際に使用するコードセグメント/スタックセグメントセレクタは，実ハードウェア上で動作する際と異なる．独自OSにこの違いを隠蔽する必要がある．

上記違いを隠蔽するために，軽量仮想計算機モニタでは，以下を実行している．まず，シャドウのセグメントテーブル（GDT/LDT）では，独自OS定義のゲートディスクリプタ，TSSディスクリプタをすべて無効にしている．ゲート通過時（独自OS上のアプリケーションからのシステムコール発行時等）やタスクスイッチ時に例外が発生し，軽量仮想計算機モニタが変換動作を行う．実際に独自OSが特権レベル1で動作している場合でも，軽量仮想計算機モニタは特権レベル0のコードセグメント/スタックセグメントセレクタを独自OS定義の特権レベルスタックやコンテキスト格納領域にセーブする．また，ゲート通過後やコ

ンテキストスイッチ後に特権レベル0のコードセグメント/スタックセグメントセレクタを持つように独自OSが指示している場合でも，軽量仮想計算機モニタは特権レベル1のコードセグメント/スタックセグメントセレクタをセグメントレジスタに設定する．

さらに，スタックセグメントレジスタを汎用レジスタにロードする命令☆を独自OSが実行する際にも，軽量仮想計算機モニタは，レジスタ値の変換を行う必要がある．しかし，独自OSが上記命令を実行しても例外は発生せず，軽量仮想計算機モニタは動作できない．そのため，独自OSを軽量仮想計算機モニタ上で動作させる場合には，上記命令をトラップ命令☆☆に置き換え，この変換を行うことにした．

第2の問題は，独自OSから発行される割込み制御ビット（EFLAGSレジスタのIFビット）の参照/更新要求の捕捉に関する問題である．軽量仮想計算機モニタは，上記ビットをソフトウェアの代替ビットを用いてエミュレートする．独自OSからの上記ビット参照/更新要求が発行された場合には，実ハードウェアの状態を参照/更新する代わりに，代替ビットの参照/更新を行う必要がある．

割込み制御ビットの参照/更新は以下を契機に行われる．

(1) cli/sti命令の発行
(2) タスクスイッチ命令（ljmp命令）の発行
(3) 割込み/例外ベクタの起動（EFLAGSレジスタの退避）
(4) 割込み/例外ベクタからのリターン命令（iret命令）の発行（EFLAGSレジスタの回復）
(5) pushf/popf命令の発行

独自OSが(1)〜(3)を実行する際には，例外が発生，軽量仮想計算機モニタが動作して割込み制御ビットの参照/更新処理のエミュレートを行う．しかし，独自OSが(4)，(5)を実行しても例外は発生せず，軽量仮想計算機モニタによるエミュレートが行えない．そのため，(4)，(5)に関しても，スタックセグメントセレクタの汎用レジスタへのロード命令と同様，トラップ命令への置き換えを行うことにした．

第3の問題は，例外要因の特定に関する問題である．3.2節で述べたように，軽量仮想計算機モニタ定義の例外ハンドラは，例外発生要因がリモートデバッグ機能起動やデバイスエミュレーションに起因する例外であった場合には，独自OSに対して例外発生通知を行ってはならない．

デバイスエミュレーションに起因する例外か否かは，例外を起こした命令列を解析することにより判別できる．しかし，リモートデバッグ機能を起動すべき例外か否かは，発生した例外が独自OSのバグに起因しているか否かで判別すべきであり，厳密な判別は不可能である．現在の実装では，上記判別をデバッグ例外が発生したか否かにより行っている．独自OSのバグに起因した例外であることを軽量仮想計算機モニタに明示的に通知するため，独自OSには，カーネルパニック発生時にはデバッグ例外を発行するように命令列の追加が行われている．しかし，独自OSのバグに起因しているがカーネルパニックルーチンに到達せず，デバッグ例外以外の例外が発生し，かつ独自OS定義の例外ハンドラの起動が無限に繰り返されることがある．この場合には，ホストマシン上のソフトウェアデバッガからのブレーク命令の発行により，上記ハンドラの起動を止めることにした．また，デバッグ例外を使用する独自OS上で動作するアプリケーション（アプリケーションデバッガ等）は，デバッグ例外が独自OSに通知されないため使用不可能になるが，本軽量仮想計算機は独自OSのデバッグを目的としているため，問題ないと判断した．

### 5.2 軽量メモリ領域保護機能の実装上の問題点

軽量メモリ領域保護機能の実装を行う際には，以下の2点の問題を解決する必要が生じた．

- 独自OSが指示するDMA転送にともなう軽量仮想計算機モニタ動作領域の破壊の防止．
- 軽量仮想計算機モニタによる独自OS定義のプロセス生成，削除の認識方法

以下で，上記2点の問題の内容と解決策について述べる．

第1の問題点は，独自OSが指示するDMA転送にともなう軽量仮想計算機モニタ動作領域の破壊の防止に関する問題である．軽量メモリ領域保護機能は，独自OS定義のページテーブルの正当性をチェックすることにより，軽量仮想計算機モニタ動作領域が独自OSにより破壊されることを防いでいる．しかし，本方式はCPUからメモリへの不正な書き込みは防げるものの，独自OSが制御する高速I/OデバイスからのDMA転送に起因する不正な書き込みは防げない．現在の実装では，独自OSにDMA転送先領域の正当

☆ コードセグメントレジスタを汎用レジスタにロードすることはできない．

☆☆ 現在の実装では，使用するトラップ番号は軽量仮想計算機モニタが定義する固定値を使用している．もし，このトラップ番号を独自OSが使用している場合には，独自OS実装者が定義できる構成ファイル等でこのトラップ番号を指定する機能が必要となる．また，独自OS実装者は構成ファイルに従って，独自OSの改変を行わなければならない．

性をチェックするコードを挿入することにより，この問題を回避している．

第2の問題点は，軽量仮想計算機モニタによる独自OS定義のプロセス生成，削除の認識方法に関する問題である．独自OSは軽量仮想計算機モニタに対して実ハードウェアと同様なインタフェースを用いてアクセスする．そのため，軽量仮想計算機モニタは，独自OS定義のプロセス生成/削除を認識することは困難である．現在の実装では，以下によりこの問題を回避している．まず，プロセスの生成に関しては，仮想空間の切替え（プロセス切替え）が発生するたびに，切替え先プロセスがPDIRとして利用している物理ページのPDIR参照数をチェックする．PDIR参照数が0であれば，軽量仮想計算機モニタは，新規にプロセスが生成されたと判断し，4.3節で述べたページマッピング情報の更新を行う．一方，プロセス削除に関しては，独自OSのコードを改変することで，軽量仮想計算機モニタにその削除を認識させている．具体的には，独自OS定義のPDIR領域の解放を行うコードの直前に，トラップ命令を発行するコードを追加している．

### 5.3 考　　察

5.1節と5.2節で述べた解決策では，以下に示す独自OSの改変が必要になる．

(1) スタックセグメントレジスタを汎用レジスタにロードする命令のトラップ命令への置き換え
(2) iret/pushf/popf命令のトラップ命令への置き換え
(3) カーネルパニック時にデバッグ例外発行命令を追加
(4) DMA転送領域の正当性をチェックするコードの追加
(5) PDIR領域の解放時にトラップ命令を追加

本節では，これらの改変が独自OSのバグ混入を引き起こす可能性があるかを検証するため，その改変の複雑さや改変量について考察する．

まず，(1)，(2)については機械的なコードの置き換えで済むので，改変は容易である．また，(3)についても，たとえ正しい位置に命令追加ができなくても，デバッグ環境の安定稼働を阻害しない（正しくデバッグ例外が発生しなくても，ホストマシンからブレーク命令を発行すればデバッグ可能である）ため，大きな問題にならない．

(4)については，独自OSで扱っているすべてのPCIデバイスドライバのソースコード改変が必要となりやや改変量が多くなる．また，各デバイスドライバごとに，DMA転送アドレスを指定しているソースコードの部分を捜し出す必要があり，改変もやや複雑となる．しかし，DMA転送領域を指定するソースコードの部分には，各独自OSが準備しているV to Pのアドレス変換ルーチンが呼ばれることがほとんどであること，および，独自OSから認識不可能な高位アドレスに存在する軽量計算機モニタ動作領域をDMA転送先として指定するバグの発生確率が大きくないこと，からこの改変も大きな問題なく実行できると判断している．

(5)については，改変量はトラップ命令の追加のみで済むため大きくないが，独自OSのページテーブル関連ルーチンの解析が必要となり，改変がやや複雑になる．しかし，移植性の高さを保つことを考えて実装されている独自OSは，ページテーブル関連ルーチンはコンパクトに実装されていることがほとんどであるため，この改変も大きな問題がないと判断している．たとえば，BSD/OSバージョン4.3の場合，解析対象ルーチンが2,100行程度であり，かつ改変量は5行であった．

## 6. 性能評価

本章では提案したOSデバッグ方式で使用している軽量仮想計算機モニタの定量的な性能評価結果について説明する．性能評価は，軽量仮想計算機モニタ上で動作する独自OSのI/O性能と，プロセス生成/削除の性能について行った．以下では，上記のそれぞれについて，評価方法の概要と評価結果について説明する．

### 6.1 I/O性能の評価

軽量仮想計算機モニタが提供するI/O性能を評価するために用いた実験環境を**図7**に示す．

本実験では，PC/AT互換機（PentiumIII☆ 1.26 GHz搭載）上に，実ハードウェア上で動作するHiTactix，軽量仮想計算機モニタ上で動作するHiTactix，Linux版VMware Workstation 4[12),17),18)]☆☆上で動作するHiTactixを搭載した．VMware Workstation 4は，多様な高速I/Oデバイス上での動作をサポートするPC/AT互換機向け既存仮想計算機モニタの代表例である．そして，上記の各HiTactix上で，データ配信アプリケーションを動作させた．データ配信アプリケーションは，3つのUltra160 SCSIディスクから均等なレートで2MBずつデータを読み出し，当該データを1MBずつに分割してからギガビットEthernetに対してUDP送信する．上記データ配信処理の配信

☆ PentiumIIIは米国Intel社の登録商標です．
☆☆ VMware Workstation 4は米国VMware Inc.の登録商標です．

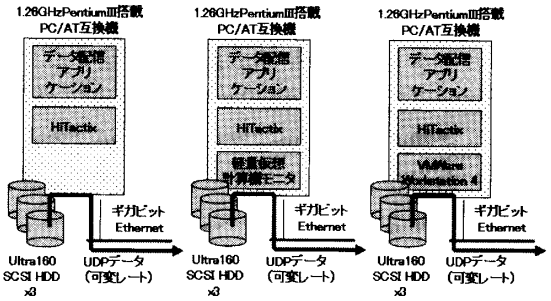


図 7 I/O 性能評価のための実験環境
Fig. 7 Experimental environments for I/O performance evaluations.

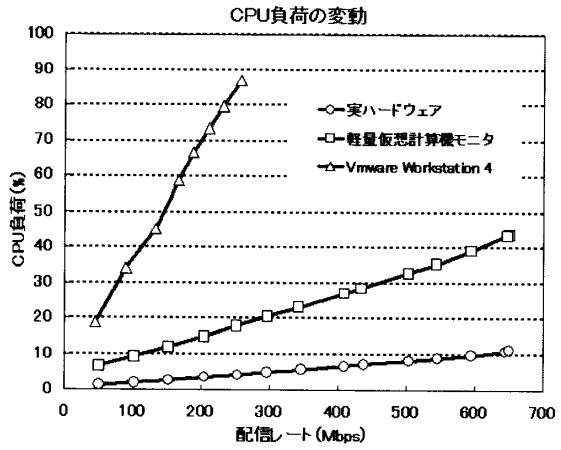


図 8 I/O 性能の評価結果
Fig. 8 I/O performance evaluation results.

レートを変動させた場合における各 PC/AT 互換機の CPU 負荷の変動を測定し，軽量仮想計算機モニタは VMware Workstation 4 と比べてどの程度高速な I/O 性能を提供できるのか，および，軽量仮想計算機モニタが提供する I/O 性能は実ハードウェアと比してどの程度低減するのか，について定量的に評価した．さらに，軽量仮想計算機モニタの仮想化オーバヘッド要因（I/O 性能低減要因）の解析もあわせて行い，軽量仮想計算機モニタと VMware Workstation 4 との間の提供 I/O 性能に差がつく要因を推定した．仮想化オーバヘッドの測定は，軽量仮想計算機モニタのソースコードに，CPU クロックを用いた実行時間測定コードを追加することにより行った．

CPU 負荷の変動の評価結果を**図 8** に示す．グラフの横軸はデータ配信レートを，縦軸は配信実行時の PC/AT 互換機の CPU 負荷を示している．各 PC/AT 互換機で消費している CPU 負荷の比率を算出するために，得られた結果から CPU 負荷が 100%に達するときの配信レートを予測し，その配信レートの大小を比較した．得られた配信レートは，実ハードウェアが 6,092 Mbps，軽量仮想計算機モニタが 1,587 Mbps，VMware が 294 Mbps となった．すなわち，軽量仮想計算機モニタは，Hosted Virtual Machine Monitor architecture を持つ VMware Workstation 4 と比して 5.4 倍の I/O 性能を達成していること，しかし，実ハードウェアと比べると，その I/O 性能は，1/3.8 に低減していることが分かる．

**表 2** 仮想化オーバヘッド要因の解析
Table 2 Virtuallization overhead analysis.

| 要因 | オーバヘッド比率 | 実行時間 (ms/s) | 実行単価 ($\mu$s) | 実行回数 (回/s) |
|---|---|---|---|---|
| モニタ定義例外/割込みハンドラ起動 | 32.19% | 105.2 | 0.234 | 448,167 |
| 割込みコントローラエミュレーション | 38.50% | 125.9 | 0.549 | 229,076 |
| 例外/割込み通知エミュレーション | 11.02% | 36.0 | 9.63 | 3,737 |
| hlt 命令エミュレーション | 8.96% | 29.3 | 0.137 | 213,527 |
| 空間生成/削除エミュレーション | 0.00% | – | – | – |
| ページテーブル更新エミュレーション | 0.00% | – | – | – |
| その他 | 9.34% | 30.5 | – | – |

また，650 Mbps のレートでデータ配信を実行している際における軽量仮想計算機モニタの仮想化オーバヘッド要因の解析結果を**表 2** に示す．この表は，各要因ごとに，オーバヘッド比率（独自 OS コード以外を実行している時間全体に対する当該要因処理を実行している時間の割合），実行時間（1 秒あたりの平均総実行時間），実行単価（1 回あたりの平均実行時間），実行回数（1 秒あたりの平均実行回数）を表記している．この解析結果から，軽量仮想計算機モニタ定義の例外/割込みハンドラの起動オーバヘッド（仮想空間切替えを実行し，エミュレーション処理を行うハンドラ本体を起動するオーバヘッド），割込みコントローラエミュレーション，独自 OS への例外/割込み通知エミュレーション，hlt 命令のエミュレーション☆がその主要オーバヘッドであることが分かる．独自 OS への例外/割込み通知エミュレーションはその実行単価が大きいために，それ以外は実行回数が多いために大きなオーバヘッドになっている．一方，データ配信アプリケーション実行中には仮想空間の生成/削除やページテーブル更新は行われず，4.3 節で述べた処理は実

☆ hlt 命令のエミュレーションでは，独自 OS のコードが hlt 命令を発行するたびに例外を発生させ仮想計算機モニタを起動する．しかし，仮想計算機モニタはただちにリターンするため，独自 OS は例外発生を繰り返す．割込みが発生した場合にのみ，独自 OS の実行コンテキスト格納領域に含まれるインストラクションポインタ（EIP）の値を更新し，独自 OS の実行を先に進める．そのため，独自 OS が消費する CPU 負荷が低いほど，hlt 命令エミュレーションの負荷は大きく計測される．

行不要である．

割込みコントローラエミュレーション，独自OSへの例外/割込み通知エミュレーション，hlt命令エミュレーションはVMware Workstation 4でも実行の必要があり，同等のオーバヘッドが必要であると考えられる．VMware Workstation 4が軽量仮想計算機モニタよりも低いI/O性能のしか提供できない理由として，以下が考えられる．

- Host OS動作中に割込みが発生すると，仮想計算機モニタ定義の割込みハンドラ起動のため，仮想空間だけでなくCPUの状態すべてを切り替えるWorld Switchを行う．上記を仮想空間の切替えだけで実行する軽量仮想計算機モニタと比して，オーバヘッドが大きくなる．
- I/O要求の発行は独自OSが行うが，I/O処理はHost OSのデバイスドライバが実行する．そのため，独自OSがI/O要求発行後にHost OSへのWorldSwitchが必要になる．また，独自OSからHost OSへのデータ受渡しのためのリマッピング処理等が必要になる．
- I/O処理実行はHost OSが行う．たとえ独自OSが低オーバヘッドでI/O処理が実行できたとしても，Host OSの高いI/O処理オーバヘッドが必ず余計にかかる．
- Ethernetコントローラ，ディスクコントローラのI/Oデバイスエミュレーションを行っている．
- 実ハードウェアとの完全なバイナリ互換性を保証するために，独自OS実行時にバイナリの動的な置き換えが必要であり，このための処理オーバヘッドが余計にかかっている．

### 6.2 プロセス生成/削除性能の評価

プロセスの生成/削除を頻繁に繰り返す処理を独自OS上で実行させると，4.3節で述べたページマッピング情報管理データ構造の更新処理のために，実行性能が低下する．上記更新処理がどの程度の実行性能の低下を招くかを定量的に評価するために，BSD/OSバージョン4.3上における，BSD/OSのカーネルソースコードのコンパイル処理を用いた性能評価を行った．カーネルのソースコードのコンパイルは，make，gcc，cpp，cc1，collect2，as，ld等の多数のアプリケーションの起動/終了にともない，頻繁にプロセスの生成/削除を繰り返す．

本評価実験では，PC/AT互換機（PentiumIII 1.26 GHz搭載）上に，実ハードウェア上で動作するBSD/OS，および軽量仮想計算機モニタ上で動作するBSD/OSを搭載する．そしてこのBSD/OS上で上記カーネルソースコードのコンパイル処理を実行し，両者の実行時間を比較，実行性能の低下の度合いを定量的に測定した．また，あわせて軽量仮想計算機モニタの仮想化オーバヘッドも測定し，実行性能低下要因の解析も行った．

**表3** コンパイル実行時間の比較
Table 3 Comparison of compilation execution time.

| 環境 | 経過時間 | 合計実行時間 | ユーザモード実行時間 | スーパバイザモード実行時間 |
|---|---|---|---|---|
| 実ハードウェア | 123.27秒 | 108.75秒 | 99.50秒 | 9.25秒 |
| 軽量モニタ | 185.27秒 | 171.42秒 | 104.03秒 | 67.39秒 |

実行時間の測定結果を**表3**に示す．この表では，実ハードウェアおよび軽量仮想計算機モニタ上で上記コンパイル処理を実行させた場合における，経過時間（実行完了までに要した時間），合計実行時間（コンパイル処理がCPUを占有した時間の総計），ユーザモード実行時間（合計実行時間のうち，ユーザモードでコンパイル処理が走行していた時間の総計），システムモード実行時間（合計実行時間のうち，スーパバイザモードでコンパイル処理が走行していた時間の総計）を示している．なお，このコンパイル処理中には，2,016回ずつプロセスの生成/削除が行われている．1秒あたりのプロセス生成/削除回数の平均は，実ハードウェアでは1秒あたり16.4回，軽量仮想計算機モニタでは1秒あたり10.9回である．この程度の頻度でプロセスの生成/削除を繰り返す処理を軽量仮想計算機モニタ上で実行させると，実ハードウェア上で実行させた場合と比して実行時間は約57.6%増大する．この実行時間の増大は，スーパバイザモードで動作するBSD/OSカーネルコードの実行時間の増大が主要要因となっている．

上記コンパイル処理の仮想化オーバヘッドの測定結果を**表4**に示す．この測定結果から以下が明らかになった．

- 仮想空間生成/削除にともなうページマッピング情報管理データ構造の更新処理オーバヘッドは，1回あたりの平均で183.5$\mu$秒程度である（表4に表記されている実行回数は，仮想空間切替え時に，切替え後の仮想空間が新規であるかチェックするための処理起動回数も含んでいる．実際に仮想空間の生成/削除を行っているのは，1秒あたりでは，284回のうち21.8回☆だけである）．こ

☆ 1秒あたり10.9回の生成と削除を行うため，総計で21.8回になる．

表 4 仮想化オーバヘッド要因の解析
Table 4 Virtuallization overhead analysis.

| 要因 | オーバヘッド比率 | 実行時間 (ms/s) | 実行単価 ($\mu$s) | 実行回数 (回/s) |
|---|---|---|---|---|
| モニタ定義例外/割込みハンドラ起動 | 53.39% | 180.6 | 0.900 | 200,707 |
| 割込みコントローラエミュレーション | 12.01% | 40.6 | 0.650 | 62,421 |
| 例外/割込み通知エミュレーション | 6.11% | 20.7 | 6.80 | 3,043 |
| hlt 命令エミュレーション | 2.28% | 7.8 | 0.189 | 40,810 |
| 空間生成/削除エミュレーション | 1.18% | 4.0 | 14.1 | 284 |
| ページテーブル更新エミュレーション | 18.77% | 63.5 | 0.774 | 81,996 |
| その他 | 6.26% | 21.2 | – | – |

のオーバヘッドが大きくないのは，オンデマンドページングを行う BSD/OS では，プロセス生成直後に有効な PTBL エントリが少ないためだと考えられる．

- 逆に，ページテーブルの更新にともなうページマッピング情報管理データ構造の更新オーバヘッドは，実行単価は小さいが，実行回数が多いため，全体として大きなオーバヘッドになっている．この実行回数が多い理由は，プロセス生成後にオンデマンドページングにともなうページテーブル更新処理が多発すること，および BSD/OS では，プロセス生成時に，新規プロセス用のページテーブルとして使用する物理メモリ領域を，一時的に現プロセスの仮想空間にマッピング後初期化する処理が多発するためだと考えられる．
- BSD/OS がページテーブルの更新，または仮想空間の生成/削除を行うと，軽量仮想計算機モニタ定義の例外ハンドラの起動と，エミュレーション処理が行われる．これらのオーバヘッドの総計が，ページマッピング情報管理データ構造の保持に必要なオーバヘッドの総計になる．このオーバヘッドの総計の全仮想化オーバヘッドに対する比率は，$53.39 \times (284 + 81996) \div 200707 + 1.18 + 18.77 = 41.83\%$に達する．すなわち，このデータ構造の保持のために，コンパイル処理の実行時間が $57.6 \times 41.83 \div 100 = 24.10\%$増大していると考えられる．
- 他の主要仮想化オーバヘッドは，割込みコントローラや hlt 命令エミュレーション，および上記エミュレーションに先立ち行われる計量仮想計算機モニタ定義の例外/割込みハンドラ起動である．これらは，ページマッピング情報管理データ構造を持たなくても必要なオーバヘッドである．

## 7. 関連研究

本論文が目的としている効率的な OS のデバッグを目的とした関連研究やソフトウェアは数多く存在する．

1 章で示した kgdb[10] 等のソフトウェアリモートデバッガ，kdb[11] 等の OS 内部に組み込まれた当該 OS 専用のデバッガ等はその例である．しかし，これらのデバッガはデバッグ環境の安定稼働を保証しておらず，OS のバグに起因する異常動作により，デバッグ継続が不可能になる可能性がある．本論文で提案した軽量仮想計算機モニタを使用した OS デバッグ方法では，上記デバッガと比して多少の OS の I/O 性能低減等のデメリットが発生するものの，代わりにデバイスエミュレーションや軽量メモリ領域保護機能等を利用してデバッグ環境の安定稼働の保証を提供している．

Temporal Debugger[7] は，汎用 OS 上に構築されたハードウェアシミュレータと連動するソフトウェアデバッガの一例である．通常のソースレベルのリモートデバッグだけでなく，汎用 OS の実行タイミング情報をユーザに提供している．そのため，デバッグ環境の安定稼働を保証するだけでなく，I/O 性能のエンハンス等も本デバッガを用いて行える．しかし，本デバッガの利用には完全なハードウェアシミュレータが必須であり，次々と新種の高速 I/O デバイスが利用可能になる PC/AT 互換機向け OS への適用が難しい．本論文で提案した OS デバッグ方式は，部分的なハードウェアエミュレーションを行うことで，新種の高速 I/O デバイスへの適用を容易にしている．

また，本論文で提案した OS デバッグ方式は，仮想計算機モニタを低オーバヘッドで動作させることにより，デバッグ対象となる OS に高い実行性能（特に高い I/O 性能）を提供可能にしている．従来から，仮想計算機モニタを低オーバヘッドで動作させる研究は数多く存在する．

1960 年代より，System/370 をはじめとするメインフレームにおいて仮想計算機モニタを高速に動作させる数多く研究[19)~22)] はその代表例である．これらの研究における仮想計算機モニタは，実ハードウェアと完全なバイナリ互換性を保証しながら，複数 OS を 1 つのハードウェア上でできるだけ高速に並行実行させ，かつ，OS 間のアクセス保護も完全に保証している．しかしこれらの研究では，実ハードウェアに仮想化に必要な機能要件[23)] があらかじめ備わっていることを前提としている．また，I/O デバイス等も多様な種類が存在することを仮定していないため，多様な I/O デ

バイスを少ない開発工数でサポートする必要性を想定していない．また，仮想計算機モニタの高速化のために，実ハードウェアに機能追加を行ってもよいとしている．本論文で提案した軽量仮想計算機モニタは，実ハードウェアとの完全なバイナリ互換性を保証する必要はない，複数OSを1つのハードウェア上で並行実行させる必要はない，と仮定する代わりに，仮想化に必要な機能を備えていないPC/AT互換機を実ハードウェアの仕様を変更せずに使用することを前提としている．また，多様なI/Oデバイスを少ない開発工数でサポートすることも実現している．

VMware Workstation[12),17)]は，PC/AT互換機上で動作する仮想計算機モニタである．Hosted Virtual Machine Architectureを持つことで，多様なI/Oデバイスをサポートできる．さらに，1つのハードウェア上での複数OSの並行実行や，実ハードウェアとの完全なバイナリ互換性の保証も実現している．本論文の軽量仮想計算機モニタは，複数OSの並行実行や完全なバイナリ互換性の保証をしない代わりに，6.1節で示したとおり，デバッグ中のOSの高いI/O性能を実現している．また，VMware Workstationは，I/Oデバイスのエミュレーションにより I/O動作の正当性保証を行う．そのため，決められた独自OSのデバイスドライバしか動作させることができない．それに対して，軽量仮想計算機モニタは，DMA転送先領域の正当性を独自OS自身に保証させる代わりに，独自OSから高速I/Oデバイスへの直接アクセスを許している．そのため，軽量仮想計算機モニタ上で独自OSが持つ様々なデバイスドライバを動作させることができる．すなわち，多様なデバイスのドライバのデバッグに利用できる．

VMware ESX Server[18)]もPC/AT互換機上で動作する仮想計算機モニタである．しかし，VMware ESX Serverはデバイスドライバを仮想計算機モニタ自身が持つため，多様なI/Oデバイスのサポートが困難である．軽量仮想計算機モニタは，多様なI/Oデバイスのサポートを容易に実現できる．

Denali[24)]やXen[16)]もPC/AT互換機上で動作する仮想計算機モニタである．しかし，これらは仮想計算機モニタを低オーバヘッドで動作させることを最優先課題とし，実ハードウェアと完全に一致する，あるいは実ハードウェアと近いインタフェースを提供することを目指していない．そのため，実ハードウェア上で動作するOSをこれらの仮想計算機モニタ上で動作させようとする際に，OSにバグが混入する可能性が高くなる．軽量仮想計算機モニタは，実ハードウェアとほぼ互換なインタフェースを提供することで，実ハードウェア上で動作するOSを当該モニタ上で動作させようとする際に必要となるOSの改変量をより少なくし，かつ改変の内容もより容易にしている．結果として，改変にともなうOSへのバグ混入の可能性が低くなる．

本論文で提案した軽量仮想計算機モニタは，従来の仮想計算機モニタと異なり，1つの実ハードウェア上で複数OSを並行実行させることを目的としていない．軽量仮想計算機モニタ内に含まれるリモートデバッグ機能をOSの異常動作から守り，デバッグ機能を安定稼働させることがその目的である．仮想計算機モニタに従来とは異なる利用用途を提唱する研究もいくつか存在する．

Disco[25)]は，高多重SMP向けのエンハンスがなされていない商用OSをccNUMA型高多重SMPマシンで高性能に動作させるために，仮想計算機モニタを用いる．高多重SMPマシンを複数の仮想計算機に見せ，かつ，各仮想計算機上で商用OSを動作させれば，仮想計算機モニタ内に実装されている高性能cc-NUMA向けメモリ管理機構やメモリ共有機能が，商用OSに特別なエンハンスを加えなくても利用されるようになり，システム全体として高いパフォーマンスが達成できる．仮想計算機モニタを従来と異なる用途で利用している点で軽量仮想計算機モニタと類似しているものの，本論文で課題としている多様なI/Oデバイスサポート等についてはDiscoは考慮しておらず，特に類似点はない．また，ターゲットハードウェアもPC/AT互換機ではないため，CPUのハードウェア資源の仮想化方法も大きく異なる．

Bressoudら[26)]は，仮想計算機モニタ内にFault-tolerant Computingを実現するためのプロトコルを実装している．1つのハードウェア上に1つのOSのみを動かすことを仮定し，仮想化の対象を必要最小限に限っている点で本研究と類似している．しかし，本論文における軽量メモリ領域保護機能のように，仮想計算機モニタとOS間のメモリ保護については考慮されていない．また，Disco同様，ターゲットハードウェアもPC/AT互換機ではないため，CPUのハードウェア資源の仮想化方法も大きく異なる．

## 8. まとめ

本研究では，デバッグ環境の安定稼働が保証できる，様々なOSやI/Oデバイスに大きな開発なく適用できる，デバッグ時にも高速動作（特に高いI/O性能）が可能である，の3条件を充足するPC/AT互換機上

の独自OSデバッグ方式として軽量仮想計算機モニタを用いる方式を新規に提案した．

提案したデバッグ方式は，従来のソフトウェアリモートデバッグ方式の改良である．従来と異なり，ターゲットマシン上でリモートデバッグ機能も内部に持つ軽量仮想計算機モニタが動作する．軽量仮想計算機モニタは，仮想化対象となるハードウェア資源を，リモートデバッグ機能の安定稼働の保証のために必要な最小限の資源に絞る部分ハードウェアエミュレーション機能を持つ．さらに，低オーバヘッドで仮想計算機モニタと独自OS間のメモリ保護を実現する軽量メモリ領域保護機能もあわせて提供する．この2機能により，上記3条件の充足が可能になった．

さらに，実装した軽量仮想計算機モニタの実行性能の評価を行った．その結果，Hosted Virtual Machine Monitorと比べて5.4倍のI/O性能は達成できること，および1秒間に16.4回のプロセスの生成/削除を繰り返す処理を，実行時間の増大を57.6%に抑えて実現できることが明らかになった．

## 参考文献


1) Novell社：Novel Media Excelerator. http://www.schweers.de/downloads/exceleratorxl_en.pdf
2) Kasenna社：Kasenna Streaming Accelerator. http://www.kasenna.com/newkasenna/products/Kasenna-Streaming-Accelerator-IP-Datasheet.08.15.031.pdf
3) Iwasaki, M., Takeuchi, T., Nakano, T. and Nakahara, M.: Isochronous Scheduling and its Application to Traffic Control, *19th IEEE Real-Time System Symposium*, pp.14–25 (1998).
4) 竹内　理ほか：HiTactix-BSD連動システムを応用した大規模双方向ストリームサーバの設計と実装，情報処理学会論文誌，Vol.43, No.1, pp.137–145 (2002).
5) 竹内　理ほか：外付けI/Oエンジン方式を用いたストリームサーバの実現，情報処理学会論文誌，Vol.44, No.7, pp.1680–1694 (2003).
6) 日立エンジニアリング（株）：HEC21/VS. http://www.hitachi-hec.co.jp/hec21-tw/hec21vs/hec21vs01.htm
7) Albertsson, L. and Magnusson, P.S.: Using Complete System Simulation for Temporal Debugging of General Purpose Operating Systems and Workloads, *IEEE 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecomunication Systems*, pp.191–198 (2000).
8) 安田絹子ほか：オペレーティングシステム開発環境の設計と実装（言語処理/OS支援アーキテクチャ，および一般），情報処理学会研究報告「システムソフトウェアとオペレーティング・システム」，Vol.072-011, pp.872–879 (1996).
9) 清水正明ほか：OSデバッグ環境の設計と実装，情報処理学会研究報告「オペレーティング・システム」，Vol.057-001, pp.1–8 (1992).
10) Kale, A.S.: kgdb: linux kernel source level debugger. http://kgdb.sourceforge.net
11) SGI社：KDB (Built-in Kernel Debugger). http://oss.sgi.com/projects/kdb/
12) Devine, S.W., Bugnion, E. and Rosenblum, M.: Virtualization System Including a Virtual Machine Monitor for a Computer with a Segmented Architecture, US Patent, No.US 6,397,242 B1 (1998).
13) Intel社：IA-32 Intel Architecture Software Developer's Manual Volumes I, II and III (2001).
14) TOPPERSプロジェクト：TOPPERSプロジェクト．http://www.toppers.jp/
15) マクニカネットワークス（株）：BSDI Internet Server Edition. http://www.networks.macnica.net/windriver/index.html
16) Barham, P., Dragovic, B., Fraser, K., Hand, S. and Harris, T.: Xen and the Art of Virtualization, *ACM Symposium on Operating Systems Principles*, pp.164–177 (2003).
17) Sugerman, J., Venkiachalam, G. and Lim, B.-H.: Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor, *USENIX Annual Technical Conference*, pp.1–14 (2001).
18) Waldspurger, C.A.: Memory Resource Management in VMware ESX Server, *USENIX 5th Symposium on Operating Systems Design and Implementation*, pp.181–194 (2002).
19) Goldberg, R.P.: Survey of Virtual Machine Reserach, *IEEE Computer*, pp.34–45 (1974).
20) Creasy, R.J.: The origin of the VM/370 time-sharing system, *IBM Journal of Reserach and Development*, Vol.25, No.5, pp.483–490 (1981).
21) Gum, P.H.: System/370 Extended Architecture: Facilities for Virtual Machines, *IBM Journal of Reserach and Development*, Vol.27, No.6, pp.530–544 (1983).
22) 梅野英典ほか：仮想計算機システムの高性能化方式，情報処理，Vol.31, No.12, pp.1665–1680 (1990).
23) Popek, G.J. and Goldberk, R.P.: Formal Requirements for Virtualizable Third Generation Architectures, *Comm. ACM*, Vol.17, No.7, pp.412–421 (1974).
24) Whitaker, A., Shaw, M. and Gribble, S.D.:

Denali: Lightweight Virtual Machines for Distributed and Network Applications, University of Washington Technical Report 02-02-01 (2002).
25) Bugnion, E., Devine, S. and Rosenblum, M.: Disco: Running Commodity Operating Systems on Scalable Multiprocessors, *ACM Symposium on Operating Systems Principles*, pp.143–156 (1997).
26) Bressoud, T.C. and Schneider, F.B.: Hypervisor-based Fault-Tolerance, *ACM Symposium on Operating Systems Principles*, pp.1–11 (1995).

(平成16年7月9日受付)
(平成17年5月9日採録)

**竹内 理**（正会員）

昭和44年生．平成4年東京大学理学部情報科学科卒業．平成6年同大学大学院理学系研究科情報科学専攻修士課程修了．同年（株）日立製作所システム開発研究所入社．連続メディア処理向きマイクロカーネルの研究，特にリアルタイムスケジューリング方式，リアルタイム通信方式，異種OS共存技術，ストリーミングサービスアーキテクチャ，OSデバッグ方式の研究に従事．