情報処理学会論文誌　コンピューティングシステム　Vol.5　No.4　87–100 (Aug. 2012)

仮想計算機モニタXenにおけるRTOS向け割込み通知方式

渡邉 和樹[1,†1]　片山 吉章[2]　松本 利夫[2]　瀧本 栄二[3]　樫山 武浩[4]　毛利 公一[3,a)]

受付日 2012年1月17日, 採録日 2012年4月23日

概要：現在，仮想計算機（VM）上のゲストOSとして，リアルタイムOS（RTOS）と高機能OSを同時に動作させることを目的に，Xenを拡張している．VMではハードウェア割込みが仮想化されるため，RTOSのリアルタイム性が損なわれるという問題がある．具体的には，既存の割込み通知機構では，同時に動作する他のVMにおける割込みの負荷の影響を受けるため，割込みの通知に遅延や処理時間の揺らぎが発生する．この問題を解決するために，MSI（Message Signaled Interrupts）を用いるとともに，RTOS向けの割込み通知機構の設計と実装を行った．その結果，Xen元来の割込み通知機構と比較して，最大処理時間で約58%，平均処理時間で約81%の削減を達成した．また，処理時間の揺らぎも約22%に抑えることができ，リアルタイム性の保証に有効であることを確認した．

キーワード：仮想化技術，リアルタイム，割込み処理，オペレーティングシステム

Management Method of Interrupts for RTOS in Xen Hypervisor

Kazuki Watanabe[1,†1]　Yoshiaki Katayama[2]　Toshio Matsumoto[2]
Eiji Takimoto[3]　Takehiro Kashiyama[4]　Koichi Mouri[3,a)]

Received: January 17, 2012, Accepted: April 23, 2012

Abstract: We have been extending Xen hypervisor's capability to execute real-time operating systems (RTOS) and operating systems with high functionality (RichOS) concurrently on it. On virtual machines, interrupts triggered by devices are also virtualized. Therefore performance and response time of RTOS are degraded. Specifically, interrupt handler of Xen has the problem causing delay and jitter of the interrupt response time by interrupt load for other VMs. To solve this problem, we improved interrupt management mechanism of Xen using MSI (Message Signaled Interrupts) and a lightweight interrupt handler. As a result, we succeed in reducing about 81% of the interrupt response time average and 58% of the maximum interrupt response time. In addition, we succeed in reducing response time fluctuations to 22%.

Keywords: virtualization, real-time, interrupt processing, operating system

1. はじめに

近年，FA（Factory Automation）に代表されるシステムでは，ロボットの制御などリアルタイム性が要求される処理を行う制御用端末と，制御用端末のモニタリングなど高い機能性を提供する情報用端末の統合によるダウンサイジングが求められている．また，スマートフォンやPDAといった情報端末では，製品の多機能化にともない，リアルタイム性と高い機能性の両立が求められている．従来から用いられてきたリアルタイムOS（以下，RTOS）には，μITRON [1] やVxWorks [2]，QNX [3] がある．RTOSは，

[1] 立命館大学大学院理工学研究科
Graduate School of Science and Engineering, Ritsumeikan University, Kusatsu, Shiga 525–8577, Japan
[2] 三菱電機株式会社情報技術総合研究所
Information Technology R&D Center, Mitsubishi Electric Corporation, Kamakura, Kanagawa 247–8501, Japan
[3] 立命館大学情報理工学部
College of Information Science and Engineering, Ritsumeikan University, Kusatsu, Shiga 525–8577, Japan
[4] 立命館大学立命館グローバル・イノベーション研究機構
Ritsumeikan Global Innovation Research Organization, Ritsumeikan University, Kusatsu, Shiga 525–8577, Japan
[†1] 現在，三菱電機株式会社情報技術総合研究所
Presently with Information Technology R&D Center, Mitsubishi Electric Corporation
[a)] mouri@cs.ritsumei.ac.jp

機器の制御を主目的とした最小限の機能提供により，処理時間の予測を可能にし，リアルタイム性を保証している．一方，高い機能性を提供するOS（以下，高機能OS）には，WindowsやLinuxがある．高機能OSは，高い機能性を実現するために複雑なシステム構成になっているため，処理時間の予測は困難である．このように，リアルタイム性と高機能性は相反する性質であり，その両立は容易ではない．

リアルタイム性と高機能性を両立させる手法として，2通りの既存の手法がある．1つは，PikeOS [4]，Darma [5]，RTX [6] に代表される，RTOSと高機能OS両方の特性をあわせ持ったハイブリッドOSを用いる手法である．しかし，ハイブリッドOSはリアルタイム性と高機能性という相反する性質を持つため，その開発は容易ではなく，ソフトウェア開発コストの増加を招く．もう1つは，SH-Mobile [7] やOMAP [8] に代表される，単一のシステムに複数の計算機を搭載し，RTOSと高機能OSを個別に動作させる手法である．この手法は，結果的にシステムに搭載する計算機の数が増加し，さらに計算機間の通信機構が必要となるため，部品数が増加し小型化が達成できない．

これらの問題を解決する新たな手法として，仮想化技術により複数のOSを同時に動作させる方法が考えられる．単一の計算機上でRTOSと高機能OSを同時に動作させることで，リアルタイム性と高機能性の両立を実現しつつ，小型化を図ることができる．また，組込み機器向けのプロセッサ市場において，SH4A-MULTI [9] やCortex-A9 MPCore [10] のようにマルチコア化されたものや，Cortex-A15 [11] のように仮想化支援機構を搭載した製品の開発も進んでいる．このように，組込みシステム上に仮想計算機モニタ（以下，VMM）を搭載し，複数の仮想計算機（以下，VM）を実現することも現実的になっている．これにともない，RTS-Hypervisor [12] やSPUMONE [13] などのリアルタイム性を意識したVMMの開発が行われている．しかし，これらのVMMは，全ゲストOSに対して，すべての計算機資源を排他的に割り当てる必要がある点や，ゲストOS間における保護が実現できていないという問題がある．

以上の背景から，既存のVMMであるXen [14] を拡張し，リアルタイム性の保証を必要としない資源についての共有を可能とし，ゲストOS間の保護を実現しつつ，RTOSと高機能OSの同時動作を可能とするVMM「Natsume-Xen（以下，Natsume）」を開発している．

VMでRTOSを動作させる場合の課題として，リアルタイム性を意識した資源管理があげられる．VMでは，計算機資源が仮想化・共有されるため，資源の性能が同時に動作する他のVMやVMMの負荷に影響される．これは，RTOSの処理時間の予測を困難にし，リアルタイム性の保証を困難にする．そこで，リアルタイム性の保証に必要な資源についてのみRTOSに専有させることで，この解決を試みる．Xenには，PCI Pass-through [15] と呼ばれる，PCIデバイスをゲストOSに対して排他的に割り当てる機構が存在する．しかし，PCI Pass-throughは，ゲストOSがデバイスに対してI/O命令を直接実行することを可能にするが，デバイスから通知される割込みは，すべてのデバイスで共通に処理されている．そこで，デバイスごとの割込み通知を実現し，デバイスの完全な専有を可能にするRTOS向け割込み通知機構を設計・実装し，性能評価実験を行った．

以下，本論文では，2章で関連研究について述べ，3章でNatsumeの基となる仮想計算機モニタXenについて述べる．次に4章でXenの問題点をふまえ，仮想計算機モニタNatsumeの設計と実装を述べる．そして，5章で性能評価の手順とその結果に基づく考察について述べる．最後に6章で本論文をまとめる．

2. 関連研究

RTOSがゲストOSとして動作することを想定したVMMやプラットフォームの開発は，既存の研究においても行われている．単一の計算機上で，RTOSと高機能OSを共存させる研究として，RTS-Hypervisor [12]，SPUMONE [13]，MobiVMM [16], [17] があげられ，高機能OSにリアルタイム性を付加したハイブリッドOSとして，RTX [6] があげられる．また，ハードウェアレベルで複数OSを共存させるプラットフォームとしてLogical Partitioning [18] があげられる．

RTS-Hypervisor [12] は，Intel VT [19] が利用可能な環境を対象とした，仮想計算機環境である．RTS-Hypervisorでは，すべての資源をゲストOSに対して排他的，かつ直接割り当てることで，デバイスドライバからの資源への直接操作を可能にしている．また，RTS-Hypervisorが提供するVMMはシステムの初期化に必要な機能のみを有し，ゲストOSの起動後は，VM間の仮想ネットワークの提供のみを行う．これにより，処理の遅延をほぼゼロに抑え，リアルタイム性を保証している．しかし，RTS-Hypervisorはすべての計算機資源を排他的に割り当てる必要があるため，SH-Mobile [7] やOMAP [8] などの複数の計算機を搭載する手法と同様に，小型化が困難になるという問題もまた有する．

SPUMONE [13] は，準仮想化型の軽量なVMMである．SPUMONEは，計算機資源のうち，物理CPUのみを仮想化し，複数のゲストOSを動作させる．さらに，割込み番号とゲストOSを固定的に対応付けることで，割込みを特定のゲストOSが専有することを可能にする．しかし，ゲストOS間における計算機資源の共有をサポートしないため，ゲストOSごとに資源を用意する必要がある．すなわち，計算機資源の管理については，RTS-Hypervisorと同様の問題を有する．また，SPUMONEはCPU資源以外の

仮想化を行わないため，ゲスト OS のメモリ保護を実現できないという問題を有する．

MobiVMM [16], [17] は，携帯電話に特化した準仮想化型 VMM である．MobiVMM は，RTOS が動作する RTVM と，高機能 OS が動作する Best-effort VM を提供し，効率的で柔軟な CPU スケジューリングを実現する．また，割込み通知処理には擬似ポーリング I/O 方式を用いている．この方式では，割込みが発生した際，ゲスト OS へ割込みを転送せず，VMM 内のフラグをセットする．そして，ゲスト OS からフラグの変化をポーリングすることで，割込みの検知を実現する．これにより，他のゲスト OS に対する割込み負荷の影響の抑制を図っている．しかし，この方式は割込みをポーリングする間隔が，CPU スケジューリングの間隔に影響される．MobiVMM の CPU スケジューラは，20 msec のタイムカンタムを持つラウンドロビンアルゴリズムを採用しており，リアルタイム性が十分であるとは考えにくい．

RTX [6] は，Windows NT・2000・XP に対して，リアルタイム性を付加するカーネルモジュールである．RTX は，Windows のサブセットとして実装されており，Win32 環境と互換性を持ったリアルタイムタスク管理機能を提供する．割込みについては，1 度すべての割込みを RTX がハンドルしたうえで処理するため，リアルタイム性は高い．ただし，デバイスの共有は RTX より上層で実現される．また，RTX ではリアルタイムタスクに優先度を付加し，割込み処理もリアルタイムタスクとして実行する．これにより，割込み処理とリアルタイムタスク間において優先度を指定でき，リアルタイム性の保証を柔軟に実現する．しかし，RTX はシステムの開発者が割込みとタスクの優先度を指定し，チューニングを行う必要がある．また，Windows 以外の高機能 OS を利用できない．さらに，PikeOS [4] や Darma [5] と同様に，コスト増大というハイブリッド OS における問題を有する．

Logical Partitioning [18] は，ハードウェアサポートにより，複数のデバイスをパーティション単位で分割し，複数の OS を共存させるプラットフォームである．Logical Partitioning は，ファームウェアにより実現され，VMM などのソフトウェアを介さずに，各ゲスト OS へ資源を排他的に割り当てる．これにより，実環境に近い性能を実現できる．しかし，Logical Partitioning は，専用のハードウェアサポートが必要になる．また，各資源を排他的に割り当てるため，資源の柔軟な共有を実現することは困難である．

他にも，仮想化のオーバヘッドの削減を図る技術は広く研究されている．たとえば，Dong らの研究 [20], [21], [22] や Ahmad らの研究 [23] のように複数の割込みをまとめてゲスト OS へ通知するもの，Willmann らの研究 [24] のようにハードウェアサポートにより複数のゲスト OS からデバイスへ直接アクセスを可能にするもの，Landau らの研究 [25] のようにマルチコア CPU を活用しゲスト OS と VMM を同時に動作させるものがあげられる．しかし，これらの技術は，スループットの向上を目的としており，リアルタイム処理のように応答性能が求められる目的には不向きである．

以上のように，既存研究では，計算機資源の共有が実現できない点，ゲスト OS 間の保護が実現できない点，既存資産の流用が難しくソフトウェアの開発コストが増大する点といった問題点が存在する．そこで，Natsume では既存の VMM である Xen にリアルタイム性を付加するアプローチをとる．Xen を基にすることで，計算機資源の柔軟な割当てやゲスト OS 間の保護の実現，さらに豊富なソフトウェア資産やノウハウの流用による効率的なシステム開発が可能になり，既存研究における問題点を解決できる．

3. 仮想計算機モニタ Xen

3.1 概要と優位性

Xen は，オープンソースソフトウェアの VMM である．Xen は，CPU や I/O デバイスなどの計算機資源を抽象化し，VM 環境を実現する方式として，完全仮想化と準仮想化を提供している．

Xen は 2 章で述べたような問題を解決するための基盤として，次の理由で優れている．

- **計算機資源の柔軟な割当て**
  Xen が提供する VMM 主体の資源管理機構により，ゲスト OS 間で柔軟な資源管理を実現できる．これにより，高機能 OS に割り当てる資源や，リアルタイム性の保証に影響しない資源を各ゲスト OS 間で共有することが可能になる．たとえば，仮想ネットワークや共有メモリにより，ゲスト OS 間の通信を容易に実現できる．これにより，FA のように RTOS の状況を高機能 OS がモニタリングするシステムにおいて，専用の回線や NIC を個別に用意する必要がなくなる．その結果，システムの小型化に寄与できる．
- **ゲスト OS 間の保護の実現**
  Xen は，各ゲスト OS に対して Domain と呼ばれる仮想環境を提供する．各 Domain におけるゲスト OS に提供されるメモリ空間は独立しており，各ゲスト OS 間においてメモリやタスクの保護が実現されている．これにより，高機能 OS やアプリケーションの不具合から，RTOS を保護することができる．
- **豊富なソフトウェア資産やノウハウの存在**
  Xen は，高機能 OS 向けの既存の VMM として実績を持ち，Xen に対応済みの高機能 OS や管理用アプリケーションなど豊富なソフトウェア資産が存在する．また，Xen はオープンソースであり，既存の OS をゲスト OS として動作させるノウハウも公開されている．これにより，既存のソフトウェア資産の流用が可能で

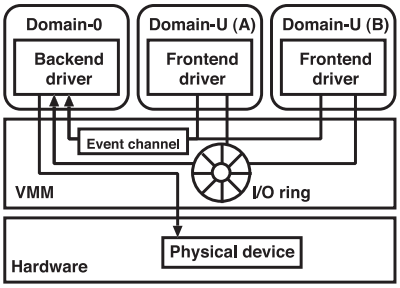

図 1　Xen の仮想デバイスドライバによるアクセス
Fig. 1　Access to I/O device via virtual device driver.

あり，かつ新たにゲスト OS とそのアプリケーションを作成することが容易になる．その結果，システムを開発する際におけるソフトウェアの開発コストの削減が可能になる．

以上から，Xen を拡張し，RTOS 向けの割込み通知機構の設計と実装を行った．なお，以下，仮想化のオーバヘッドが小さい準仮想化について述べる．Xen の準仮想化では，**図 1** に示すように，Domain-0 と呼ばれる，Xen 自身の制御や VM の管理を行うための Domain と，ゲスト OS が動作する Domain-U の 2 種類の VM からなる．

3.2　RTOS と資源管理

VM 環境で RTOS を動作させる場合，次の資源におけるリアルタイム性を確保する必要がある．以下，リアルタイム処理に必要な資源については VM へ専有させ，そうでない資源については各 VM 間で共有することとして，Xen の資源管理について検討する．

- CPU 資源（物理 CPU や CPU コア）
- 主記憶資源
- 入出力資源（I/O や割込み）

CPU 資源は，Xen が提供する XenTools [26] を用いることで，割り当てる VM を固定し，特定のゲスト OS による専有を実現できる．ただし，Xen ではゲスト OS がアイドル状態になった場合，XenTools による設定内容にかかわらず，CPU 資源が切り離され，IdleDomain と呼ばれる仮想 VM へ割り当てられる．これにより，RTOS がアイドル状態になった場合，割込み発生時に物理 CPU の再割当てや再スケジューリングが必要になり，割込み通知におけるオーバヘッドが発生する．これは，RTOS におけるリアルタイム性の保証を困難にする．これについては，4.3.5 項で述べる．

主記憶資源は，CPU 資源と同様に XenTools によってゲスト OS へ割り当てる資源量や割り当てる VM を固定することができる．また，Xen は direct-mode と呼ばれる主記憶管理機構を持つ．direct-mode では，MMU に対するアクセスを管理しつつ，MMU によるゲスト OS の保護を行うことで，ゲスト OS から直接主記憶にアクセスすることを可能にする．そのため，実環境と比較してメモリアクセスにおける遅延に差は生じない．

入出力資源について，まず Xen の準仮想化において標準的に採用されている方式である仮想デバイスドライバを用いた方式について述べる．Domain-U で動作するゲスト OS は，物理資源を抽象化した仮想資源を利用する際，ネイティブなデバイスドライバの代替として，フロントエンドドライバと呼ばれる仮想デバイスドライバを用いる（図 1 参照）．また，Domain-0 では，バックエンドドライバと呼ばれる仮想デバイスドライバを用いる．フロントエンドドライバは，VMM の機構であるイベントチャネルや I/O リングを通じて，処理要求と処理結果をバックエンドドライバとの間で授受するもので，物理資源に直接アクセスすることはない．一方，バックエンドドライバはフロントエンドドライバからの処理要求を受け，物理資源にアクセスし，その結果を返す．このように，1 つのバックエンドドライバに複数のフロントエンドドライバを対応付けることによって，ゲスト OS 間で物理資源の共有を実現している．しかし，この仕組みでは VMM や Domain-0 に負荷がかかった場合，Domain-0 による処理やその処理結果の授受に遅延が生じ，要求の完了が遅延する可能性がある．

以上のような問題を解決するために，Xen では，PCI Pass-through [15] と呼ばれるもう 1 つの入出力の手法が提供されている．PCI Pass-through は，特定の VM に PCI デバイスを排他的に割当て可能とする機能である．PCI Pass-through の対象となるデバイスは，他の VM から隠蔽され，仮想デバイスドライバ経由ではなく，ゲスト OS のデバイスドライバによる直接アクセスが可能になる．すなわち，ゲスト OS による I/O 処理の専有が実現できる．しかし，デバイスから発生する割込みについては，仮想デバイスドライバを用いた場合と同様に，イベントチャネルを経由してゲスト OS へ通知される．イベントチャネルは，割込み発生元デバイスの種類にかかわらず，すべての割込みを平等に扱い，割込み通知を直列化する．その結果，RTOS に優先して通知すべき割込みが発生した場合においても，処理を横取りし即座に割込みを RTOS へ通知することができない．また，PCI デバイスでは，その仕様上，複数のデバイスで IRQ を共有することが多い．RTOS と高機能 OS で排他的にデバイスが割り当てられていても，IRQ が共有されてしまうと，共有された IRQ の割込みはすべての OS へ通知される．さらに，両 OS は，通知された割込みが自身への割込みか否かを確認するためにデバイスにアクセスしなければならない．以上から，PCI Pass-through を利用した場合でも，割込み通知において他の VM や VMM における負荷の影響を受ける．これは，割込み通知の遅延や割込みの損失の原因となり，RTOS によるリアルタイム性の保証を困難にする．

また，リアルタイム処理ではタイマ割込みも重要となる．

Xen はタイマデバイスを抽象化し，仮想タイマと呼ばれる機能を提供している．仮想タイマは VM ごとに提供され，次に割込みを通知する時刻を記録している．タイマデバイスからの割込みが発生すると，VMM でカウントしている実時刻と，各 VM の仮想タイマを比較し，実時刻が超過していれば，仮想タイマ割込みを通知する．その後，実時刻から次に割込みを通知する時刻を算出し，再び仮想タイマにセットする．すなわち，Xen における仮想タイマ割込みは，タイマデバイスの割込みと直接対応しておらず，その精度が VMM における負荷に影響される．また，割込みの通知にはイベントチャネルが用いられるため，割込みの直列化の影響も受ける．

以上のように，Xen の資源管理機構は，ゲスト OS がアイドル状態になった場合に CPU 資源が切り離されるという問題，入出力資源の割込みや仮想タイマ割込みが直列化されて通知されるという問題，仮想タイマ割込みがタイマデバイスからの割込みと対応しておらず，精度が VMM の負荷に影響されるという問題を有する．

これらの問題を解決するためには，RTOS に対してつねに CPU 資源を割り当てつつ，割込み通知機構を専有させることで，割込みを即座に通知することを可能にし，割込みの直列化の影響を受けない割込み通知機構が必要になる．そこで，Xen に RTOS 向けの割込み通知機構を追加することで，Natsume の実現を目指す．

4. 仮想計算機モニタ Natsume-Xen

4.1 概要

Natsume は，VM を用いて RTOS と高機能 OS を同時動作させることを目的とした VMM である．リアルタイム性の保証が必要な処理を RTOS で実行し，同時に高度な情報処理を高機能 OS で実行することで，リアルタイム性と高機能性の両立を可能にする．また，単一の計算機上で複数の OS を動作させることで，小型化に寄与する．加えて，既存の VMM である Xen の準仮想化を基にすることで，高いパフォーマンスや OS 間の保護を実現しつつ，ソフトウェア資産の有効活用によるソフトウェアの開発コストの削減にも寄与する．

Natsume は，管理用インタフェース特権を持つ Domain-0，RTOS を動作させる RT-Domain，高機能 OS を動作させる Domain-U，計算機資源を提供する RT-Hypervisor で構成される．Domain-U では，Xen と同様の仮想化を行い，Xen の特徴である柔軟な資源割当てを行う．RT-Domain では，RTOS によるリアルタイム性の保証を実現するために，各資源の専有を可能にする資源管理を行う．

しかし，3 章で述べたように，Xen の割込み通知機構は，ゲスト OS における割込みの専有を実現できない．そこで，ゲスト OS による割込みの専有を実現する RTOS 向け割込み通知モデルを提案する．

4.2 RTOS 向け割込み通知モデル

提案モデルは，I/O 命令と割込み双方の専有を実現するために，デバイスごとに独立した割込み通知機構を利用可能にするものである．割込み通知機構を分離することで，割込み負荷による影響を抑制し，RTOS によるリアルタイム性の保証を可能にする．提案モデルの特徴を以下に示す．

(1) 共有デバイスと専有デバイスにおける割込み通知機構を分離するために，専有デバイスを割込みレベルで明確に識別する．
(2) 専有デバイスから通知される割込みを，既存の割込みハンドラとは独立した専用の割込みハンドラ（以下，Natsume ハンドラ）で受信する．Natsume ハンドラは，専有デバイスと 1 対 1 で対応付けられており，専有デバイス間で割込み通知経路が共有されることはない．
(3) 受信した割込みをゲスト OS へ即座に転送する．Natsume ハンドラは，既存の割込みハンドラより高い優先度で実行され，他デバイスから通知される割込みによる負荷の影響を受けることなく，即座に割込みをゲスト OS へ転送することができる．
(4) ゲスト OS に，CPU 資源をつねに割り当てることで，Natsume ハンドラから転送された割込みを即座にゲスト OS 内の割込みハンドラで処理することを可能にする．

提案モデルを実装するにあたって，割込み通知機構の設計を行った．その概要を**図 2** と以下に示す．

(1) システム起動時に，専有デバイスの情報を取得し，専有デバイスを識別する．
(2) 専有デバイスに専用の割込みベクタ（以下，ベクタ）を割り当て，共有デバイスと割込みレベルで区別する．専用のベクタは，既存の割込みハンドラに利用されるベクタより高い優先度を持つものを割り当てる．
(3) 専有デバイスからの割込みの通知先を RTOS が専有する CPU に固定する．
(4) 専有デバイスからの割込みを，Natsume ハンドラで処理する．
(5) イベントチャネルから制御を横取りし，RTOS の割込みハンドラに即座に制御を移す．

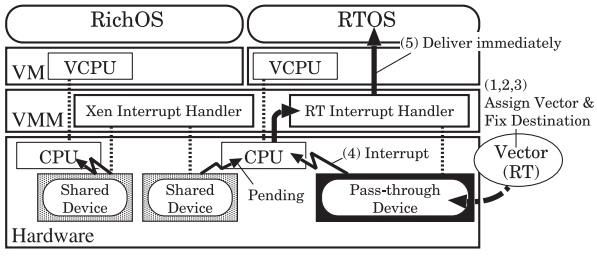

図 2　RTOS 向け割込み通知モデル
Fig. 2　Interrupt delivery model for RTOS.

4.3　実装

提案モデルを実現するために，以下に示す機構を Xen に実装した．

4.3.1　デバイス識別機能

Xen の起動オプションから専有デバイスに関する情報を取得し，専有デバイスを識別する機構を実装した．具体的には，Xen のブートローダである GRUB [27] に設定されているオプション文字列を取得し，PCI Pass-through の対象となっているデバイスの PCI バス・PCI デバイス・PCI 機能番号を取得する．

4.3.2　専用の割込みベクタを割り当てる機構

専有デバイスからの割込みを Natsume ハンドラで処理するために，専有デバイスに専用のベクタを割り当てる機構を実装した．専用のベクタと Natsume ハンドラを関連付けることで，専有デバイス・割込みハンドラ・ゲスト OS の対応付けを実現する．なお，専用のベクタと Natsume ハンドラは専有デバイスごとに提供される．これにより，専有デバイス間で割込み通知経路が共有されることを防止する．

通常，PCI デバイスのベクタは BIOS により割り当てられる IRQ によって決定される．しかし，PCI バスの仕様上，割当て可能な IRQ はすべての PCI デバイスで最大 4 つに制限される [28]．すなわち，デバイスごとに専用のベクタを割り当てるためには，IRQ に依存しないベクタ割当て機構が必要になる．具体的には，PCI が提供するメッセージベースの割込み通知機構である MSI [28]（Message Signaled Interrupts）や，ARM アーキテクチャにおいて提供される，自由なベクタの操作が可能な割込みコントローラである VIC [29]（Vectored Interrupt Controller）などを利用する必要がある．今回の実装では，MSI を用いたベクタの割当てを採用する．ゲスト OS が専有デバイスの初期化を行う際，VMM から自動的に MSI を有効にし，Natsume ハンドラ専用のベクタを割り当てる．

4.3.3　割込み通知先 CPU を固定する機構

専有デバイスからの割込みの通知先を，つねに RTOS が専有する CPU に固定する機構を実装した．Xen では，すべての割込みがその時点で最も負荷の低い CPU に通知されるように設定される．この設定は，すべてのゲスト OS で CPU を共有する場合にスループットの向上が期待できる．しかし，割込みを受信した CPU が，割込み通知先のゲスト OS に割り当てられていない場合，プロセッサ間割込み（IPI）によりゲスト OS が利用している CPU へ割込みが転送される．これは，RTOS が CPU を専有する提案モデルでは，割込み通知におけるオーバヘッドの原因となる．そこで，割込み通知先 CPU を RTOS が専有するものに固定することで，CPU 間における割込みの転送にともなうオーバヘッドの削減を図る．

4.3.4　Natsume ハンドラ

既存の割込みハンドラより高い優先度で動作する軽量な割込みハンドラを実装した．具体的には，既存の割込みハンドラから，デバイス割込み以外の割込みに関する処理や，割込みの共有に関する処理を削除し，軽量化・最適化を施した．たとえば，NMI（Non-Maskable Interrupt）やソフトウェア例外は，専用ベクタを用いて通知されることはないため，それらの割込みの判別や通知に関する処理を削除した．また，専用のベクタと専有デバイスは 1 対 1 で対応付けられているため，デバイスによるベクタの共有やゲスト OS によるデバイスの共有に関わる処理を削除した．

4.3.5　CPU の再割当てを防止する機構

CPU 資源が IdleDomain へ遷移することを抑制し，CPU 資源の専有をつねに実現する目的で，CPU の再割当てを防止する機構を簡易的に実装した．具体的には，ゲスト OS 上において最低優先度で動作する CPU バウンドなプログラムを実装した．

4.3.6　仮想タイマに依存しないタイマ割込み通知機構

仮想タイマから独立したタイマ割込み通知を実現するために，Natsume ハンドラを介して割込みを通知する，専用のタイマデバイスを確保する機構（本機構を Natsume タイマと記す）を実装した．専用タイマには周期割込みを生成する機能を持つタイマデバイスが必要になる．今回の実装では，PIT [30] を専用タイマとして用いる．

5.　評価

5.1　評価の目的

同時に動作する Domain-U のゲスト OS や VMM における割込み負荷にかかわらず，RT-Domain での割込みの通知における遅延の発生や，処理時間の揺らぎの増大が抑制可能であれば，提案モデルがリアルタイム性の保証に有効であると考えられる．具体的には，Domain-U のゲスト OS に対して外部から割込み負荷をかけつつ，RT-Domain のゲスト OS に対する割込み通知時間を計測し，最大処理時間や処理時間の揺らぎに影響がないことを確認できればよい．これを示すために以下の 4 種類の実験（**表 1** 参照）について，それぞれ Domain-U の数を 0 から 2 に変化させた，3 つの異なるパターンを用いて計測した．

(1) Xen：Xen の基本的な割込み処理性能を明確にするための実験をする．
(2) Natsume-CPU：Xen に，提案手法のうち CPU の再割当てを防止する機構を加えたものについて，その効果を明確にするための実験をする．
(3) Natsume-Vector：Xen に，提案手法のうち専用のベクタを割り当てる機構と Natsume ハンドラを加えたものについて，その効果を明確にするための実験をする．
(4) Natsume-ALL：提案手法のうち，上記の機構をすべて適用した Natsume-Xen において，割込み処理性能

表 1　比較実験のための環境
Table 1　Experimental environment for comparison.

	VMM	割込みベクタ	Idle 時の CPU 割当て	Timer
実験 (1)	Xen	共有	IdleDomain	-
実験 (2)	Natsume-CPU	共有	RT-Domain	-
実験 (3)	Natsume-Vector	専有	IdleDomain	-
実験 (4)	Natsume-ALL	専有	RT-Domain	-
実験 (5)	Xen-Timer	専有	RT-Domain	仮想タイマ
実験 (6)	Natsume-Timer	専有	RT-Domain	Natsume タイマ

表 2　評価環境
Table 2　Evaluation environment.

CPU	Intel Core i7 920（コア数 4 つ）
マザーボード	Intel DX58SO
評価用・負荷用デバイス	Intel 82567LM-2
負荷用デバイス	Intel 82541GI
評価用ゲスト OS	評価用 mini-os 1 環境
	（CPU1 個/RAM512 MB 割当て）
負荷用ゲスト OS	linux-2.6.18-xen 0,1,2 環境
	（各環境 CPU1 個/RAM512 MB 割当て）

を明確にするための実験をする．

以上の (1) から (4) のそれぞれにおいて，Domain-U の数を変化させつつ，処理性能や外部からの割込み負荷の変化が RT-Domain へ与える影響について計測する．

一方，タイマ割込みでは，周期的な割込みに揺らぎがないことが重要になる．そこで，割込み負荷が，タイマ割込みに与える影響について計測した．具体的には，以下の 2 種類の実験（表 1）について，負荷のないパターン，Domain-U へ割込み負荷をかけたパターン，RT-Domain へ割込み負荷をかけたパターンの 3 つの異なるパターンで計測を行った．

(5) Xen-Timer：Natsume-ALL と同様の環境に，従来の仮想タイマを適用した環境において，その性能を明確にするための実験をする．

(6) Natsume-Timer：Natsume-ALL と同様の環境に，Natsume タイマを適用した環境において，その性能を明確にするための実験をする．

以上の (5)，(6) において，割込み負荷をかける対象を変化させつつ，仮想タイマへ与える影響について計測した．

5.2　実験手法

実験 (1) から (4) における性能評価環境を**表 2** と**図 3** に示し，具体的な手法を以下に示す．

(1) 割込み負荷の生成を目的としたゲスト OS（以下，負荷用ゲスト OS）を用意する．負荷用ゲスト OS の数を増減させることで，割込み負荷の有無による影響を観測する．負荷用ゲスト OS は，Natsume の適用場面において RTOS と同時に高機能 OS が利用されている状況を想定し，Xen のゲスト OS として実績のある高機能 OS である Linux を用いる．

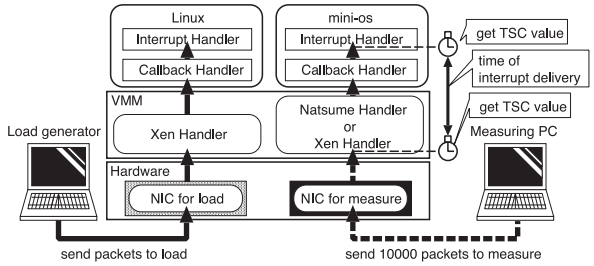

図 3　実験 (1)–(4) の性能評価実験の概要
Fig. 3　Overview of performance evaluation (Experiment (1)–(4)).

(2) 割込み通知時間の計測を目的としたゲスト OS（以下，評価用ゲスト OS）を 1 つ用意する．評価用ゲスト OS は，VMM における処理時間を明確に計測する目的で，Xen に付属するサンプル用のカーネルであり，ゲスト OS として最小限の機能しか持たない mini-os を RTOS として用いる．

(3) 負荷用ゲスト OS に仮想デバイスドライバを用いて，負荷用デバイスを割り当てる．また，評価用ゲスト OS に PCI Pass-through を用いて評価用デバイスを割り当てる．これらのデバイスは，外部から割込みを生成することが容易な NIC を用いる．

(4) 割込みが発生した時刻を計測するため，評価用デバイスに評価実験用に用意した専用のベクタ（以下，評価用ベクタ）を割り当てる．評価用ベクタは，割込みを受信した瞬間の TSC（Time Stamp Counter）の値を記録する機能を追加した評価用割込みハンドラに関連

付ける．評価用割込みハンドラは，Natsume ハンドラ，もしくは既存の割込みハンドラを基にして作成する．なお，5.3.2 項で後述するように，実験 (1)，(2) では専用のベクタを割り当てた影響を最小限にするために，オリジナルの Xen と同様の割込み処理を再現する機構を評価用割込みハンドラに追加する．

(5) 負荷用デバイスに対して，外部から大量のパケットを送信し，割込みによる負荷をかける．

(6) 評価用デバイスに対して，合計 10,000 個のパケットを送信し，評価用ゲスト OS に通知される評価用の割込みを生成する．

(7) 評価用デバイスに送信した 10,000 個のパケットについて，評価用デバイスから割込みが通知されたとき，VMM の評価用割込みハンドラと，評価用ゲスト OS の割込みハンドラの先頭で TSC の値を記録し，その差分から割込み通知に要した時間を算出する．

(8) 計測した割込み通知時間を基に，平均処理時間と最大処理時間を算出する．また，標準偏差を算出することで処理時間の揺らぎを求め，Xen と Natsume における性能の比較を行う．

また，実験 (5) と (6) における性能評価環境を表 2 と**図 4** に示し，具体的な手法を以下に示す．

(1) 負荷用ゲスト OS と評価用ゲスト OS を 1 つずつ用意する．実験 (1)–(4) と同様に，負荷用ゲスト OS として Linux，評価用ゲスト OS として mini-os を用いる．

(2) 負荷用ゲスト OS に，仮想デバイスドライバを用いて負荷用デバイスを割り当てる．また，評価用ゲスト OS に，PCI Pass-through を用いて別の負荷用デバイスを割り当てる．これらのデバイスは，実験 (1)–(4) と同様に NIC を用いる．

(3) 負荷用デバイスへ，外部から大量のパケットを送信し，割込みによる負荷をかける．

(4) 仮想タイマから，合計 5,001 個のタイマ割込みを発生させる．このとき，ゲスト OS のタイマ割込みハンドラの先頭で TSC の値を記録する．

(5) 各々の値において，1 つ前の割込みから経過した時刻を算出し，割込み間隔とする．すなわち，合計 5,000 個の結果が得られる．

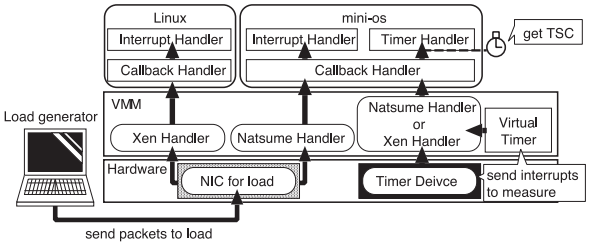

図 4　実験 (5)，(6) の性能評価実験の概要

Fig. 4　Overview of performance evaluation (Experiment (5), (6)).

5.3　評価用に作成したプログラム

5.3.1　評価用ゲスト OS

VMM における処理時間を正確に計測するために，ゲスト OS として動作可能な最小限の機能を持つ mini-os を評価用ゲスト OS とした．mini-os は，Xen に付属するサンプル用のカーネルであり，OS と VMM 間のインタフェースや初期化ルーチンをはじめとした最小限の機能しか持たない．このような特徴を持つ mini-os に以下の機構を追加し，評価用ゲスト OS を作成した．

- **評価用デバイスのデバイスドライバ**
  PCI Pass-through により割り当てられた評価用デバイスに対して，実環境と同様のインタフェース [31] を経由して，デバイスの初期化を行う．
- **評価用デバイスに対応した割込みハンドラ**
  評価用デバイスから割込みが発生したとき，割込み発生時刻を記録し，評価用デバイスからの割込みを再度受付可能にする．
- **記録した時刻を通知するインタフェース**
  10,000 個のパケットによる割込み発生時刻を計測した後，ハイパーコールを用いて VMM に計測結果を通知する．

5.3.2　評価用 Xen

評価用割込みハンドラを用いて，評価用デバイスから通知される割込みが発生した時刻を取得するためには，負荷用デバイスの割込み通知機構から処理を分離する必要がある．また，割込みを受信した CPU（CPU コア）が割込み通知先のゲスト OS が利用する CPU と異なる場合，TSC のソースが異なるため，VMM 内で取得した値とゲスト OS 内で取得した値から，割込み通知時間を算出できなくなる．すなわち，オリジナルの Xen を用いた評価を行う場合，正確な時刻に外部から割込みを発生させる手段が必要になるため，実験は容易ではない．そこで，提案モデルの一部である専用の割込みベクタを割り当てる機構と割込み通知先 CPU を固定する機構をオリジナルの Xen に適用した．さらに，これらの機構を適用しつつ，オリジナルの Xen を用いた場合と同等の評価を行う目的で，評価用デバイスと負荷用デバイスとの間で IRQ を共有している状態を再現する機構を追加した評価用 Xen を作成した．具体的には，**図 5** と**図 6** に示す機構を追加した．また，従来の Xen の挙動を**図 7** に示す．

- **割込みハンドラのロックを共有する機構（図 5）**
  イベントチャネルでは，同一の IRQ に対応する割込みハンドラは，割込みの直列化のために排他的に実行する．この現象を再現する目的で，評価用割込みハンドラと負荷用デバイスの割込みハンドラ間でロックを共有する機構を実装した．
- **割込みをすべてのゲスト OS に通知する機構（図 6）**
  複数のデバイスで IRQ が共有されている場合，割込

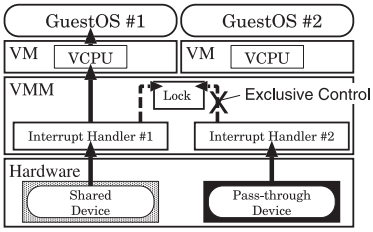

図 5　ロックを共有した割込みハンドラ
Fig. 5　Interrupt handlers that share locks.

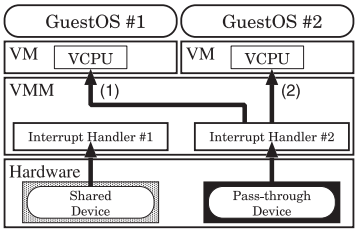

図 6　全ゲスト OS への割込み通知
Fig. 6　Delivery interrupts to all guest OSes.

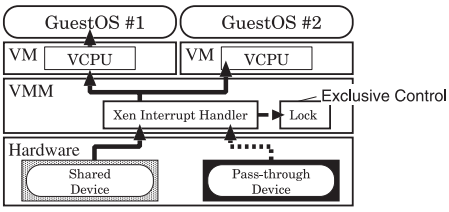

図 7　Xen における割込み通知
Fig. 7　Notification of interrupts in Xen.

みを受信した時点では，割込み発生元のデバイスを特定することができない．そのため，複数のゲスト OS に，IRQ を共有したデバイスを割り当てた場合，IRQ を共有しているデバイスを持つすべてのゲスト OS に対して，割込みが通知される．この現象を再現する目的で，評価用デバイスから割込みが発生した場合，負荷用ゲスト OS に対しても，割込みを通知する機構を実装した．なお，負荷用デバイスから発生した割込みを評価用ゲスト OS に転送する機構は，評価用ゲスト OS における割込み発生時刻の計測が困難になるため，実装していない．

これらの機構は，余分な条件分岐やデータ構造の探索などが発生しないように実装している．また，評価用 Xen は，割込み通知先 CPU を固定する機構が適用されている点，負荷用デバイスから発生した割込みが評価用ゲスト OS へ転送されない点において，オリジナルの Xen と比較して性能が向上していると考えられる．すなわち，評価用 Xen と比較して性能の改善が確認できる場合，オリジナルの Xen と比較しても，同様に性能の改善が確認できる．

5.4　実験の内容

5.4.1　デバイス割込みに関する実験

■実験 (1)（**Xen**）この実験では，Xen 元来の割込み処理性能を明確にするために，VMM に評価用 Xen を用いる．評価用 Xen は評価用割込みハンドラに，評価用デバイスと負荷用デバイスとの間で IRQ を共有している状態を再現する機構（図 5，図 6 参照）を適用している．この環境において，負荷用ゲスト OS の数を 0 から 2 個まで変化させつつ，評価用ゲスト OS に割込みが通知されるまでの時間を計測した．

■実験 (2)（**Natsume-CPU**）この実験では，提案手法のうち CPU の再割当てを防止する機構の効果を明確にするために，VMM に評価用 Xen を用いつつ，評価用ゲスト OS に当該機構を適用する．この環境において，実験 (1) と同様の実験を行った．

■実験 (3)（**Natsume-Vector**）この実験では，提案手法のうち専用のベクタを割り当てる機構と Natsume ハンドラの効果を明確にするために，評価用 Xen に当該機構を追加し，IRQ を共有している状態を再現する機構を無効にしたものを VMM として用いる．この環境において，他の環境と同様の実験を行った．

■実験 (4)（**Natsume-ALL**）この実験では，提案手法のすべての機構を適用した Natsume-Xen の割込み処理性能を明確にするために，評価用 Xen と評価用ゲスト OS ともに，提案手法の全機構を適用したものを用いる．この環境において，他の環境と同様の実験を行った．

5.4.2　タイマ割込みに関する実験

■実験 (5)（**Xen-Timer**）この実験では，Xen 元来の仮想タイマ割込みの精度を明確にするために，Natsume-ALL の環境に従来の仮想タイマを適用したものを用いる．この環境において，仮想タイマの周期を 1 msec に設定し，負荷用ゲスト OS や評価用ゲスト OS へ割込み負荷をかけつつ，評価用ゲスト OS が検知するタイマ割込みの間隔を計測した．

■実験 (6)（**Natsume-Timer**）この実験では，Natsume タイマの精度と効果を明確にするために，Natsume-ALL の環境に Natsume タイマを適用したものを用いる．この環境において Xen-Timer 環境と同様の実験を行った．ただし，PIT の仕様では，1 msec に割込み周期を設定できないため，設定可能な周期で最も近い 0.999847 msec を用いた．

5.5　実験結果と考察

5.5.1　デバイス割込みに関する実験

実験 (1) から (4) の結果を**図 8** と**図 9** に示す．図 8 は，計測結果から最大処理時間と平均処理時間，最小処理時間を表したものである．また，図 9 は，計測結果から割込み通知時間の標準偏差を表したものである．

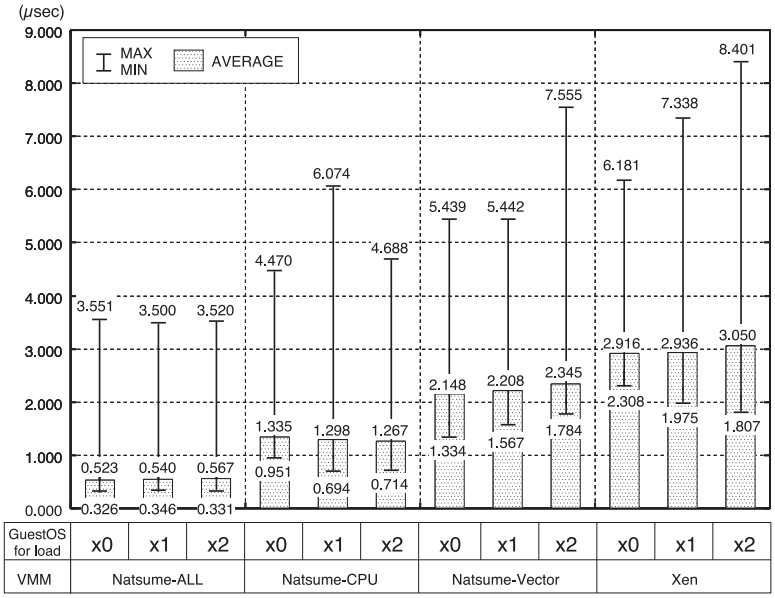

図 8　デバイス割込みの平均・最小・最大処理時間（実験 (1)–(4)）

Fig. 8　Average, minimum and maximum processing time of device interrupts (Experiment (1)–(4)).

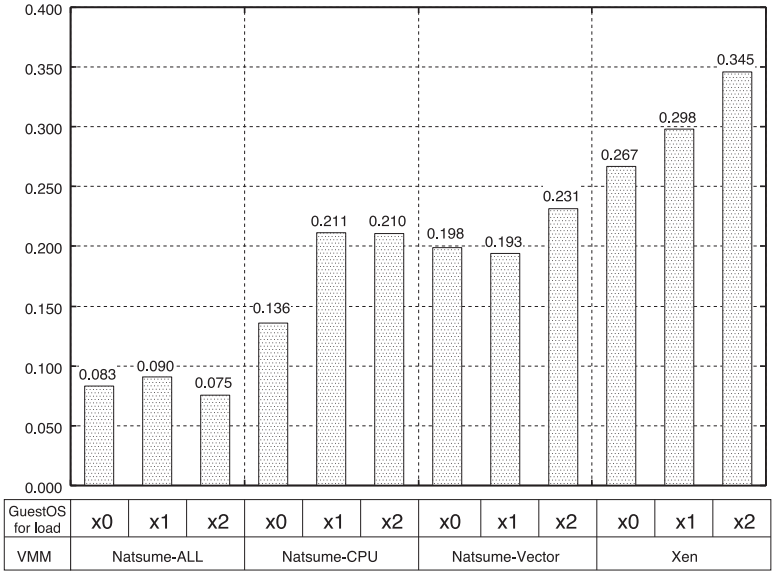

図 9　標準偏差（実験 (1)–(4)）

Fig. 9　Standard deviation (Experiment (1)–(4)).

図 8 から，Xen では負荷の増大にともなって最大処理時間が増加しているが，Natsume-ALL では，最大処理時間・平均処理時間・最小処理時間ともに変化は見られないことが分かる．また，負荷用ゲスト OS が 2 つ存在する場合，平均処理時間は，Xen の 3.050 μsec に対し，Natsume は 0.567 μsec となり，処理時間が約 81%削減される結果になった．最大処理時間は，Xen の 8.401 μsec に対し，Natsume は 3.520 μsec となり，約 58%の削減となった．さらに，最小処理時間では，Xen の 2.308 μsec に対し，Natsume は 0.326 μsec となり，約 85%の削減となった．

同様に，図 8 から，負荷用ゲスト OS が 2 つ存在する場合，Xen は割込み通知に最大 8.4 μsec 要することが分かる．一方，Natsume-ALL では最大 3.5 μsec まで抑制されており，Xen における平均処理時間の約 0.5 μsec 差まで抑制されている．すなわち，Natsume は，Xen と比較して RTOS によるシステム最適化などで対策できる余地が大きく，RTOS を動作させるプラットフォームとして，有効であることが確認できる．

次に，図 9 から，標準偏差を比較すると，Xen が負荷の状況により約 0.35 から 0.25 の範囲で遷移する結果に対して，Natsume-ALL では負荷の状況にかかわらず約 0.08 前後で安定する結果になった．したがって，Xen では負荷

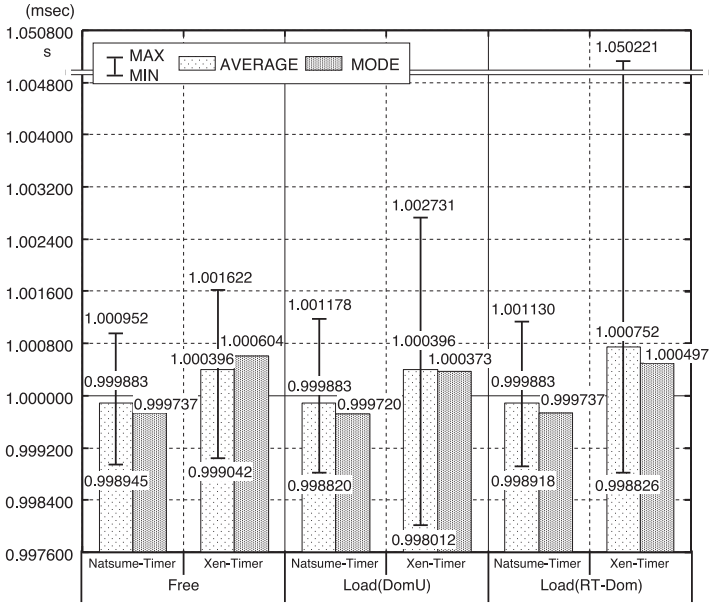

図 10　タイマ割込み間隔の平均値・最頻値・最大値・最小値（実験 (5)，(6)）
Fig. 10　Average, mode, maximum and minimum of timer interrupt interval (Experiment (5), (6)).

が増大するに連れて，処理時間の揺らぎが増大していることが分かる．一方，Natsume-ALL では負荷の大きさにかかわらず，処理時間の揺らぎに変化が見られない．また，負荷用ゲスト OS が 2 つ存在する場合，処理時間の揺らぎそのものの大きさも，Xen の 0.345 に対し，Natsume では 0.075 となり，約 22%に抑制されていることが分かる．このことより，Natsume は同時に動作する他の VM や VMM における負荷の影響を抑制し，リアルタイム性の保証に有効であると考えることができる．

最後に，実装した機構ごとの有効性について考察する．図 8 から，負荷用ゲスト OS が 2 つ存在する場合，平均処理時間では，Xen の 3.050 μsec に対して，Natsume-Vector で 2.345 μsec となり約 0.7 μsec の削減，Natsume-CPU で 1.267 μsec となり約 1.8 μsec の削減が確認できる．このことから，Natsume-CPU，Natsume-Vector ともに性能が改善され，かつ Natsume-CPU の方がより性能が高いことが分かる．すなわち，提案手法のうち，オーバヘッドの削減は，CPU の再割当てを防止する機構の効果が大きいと考えられる．

一方，図 9 から，標準偏差に着目すると，Natsume-CPU は約 0.2 から 0.13 で遷移し，Natsume-Vector は約 0.2 前後で安定する結果になり，Natsume-CPU では負荷のない状況において，処理時間の揺らぎが抑制されていることが分かる．しかし，負荷が発生すると，負荷のない状態と比較して，処理時間の揺らぎが増大していることが分かる．一方，Natsume-Vector では，Natsume-CPU と比較して負荷にかかわらず処理時間の揺らぎに大きな変化が見られないことが分かる．このことから，割込み負荷の影響を抑制する効果は，専用の割込みベクタを割り当てる機構と Natsume ハンドラの効果が大きいことが分かる．つまり，提案モデルのアプローチである割込み通知経路の専有は，リアルタイム性の保証に有効であるといえる．

5.5.2　タイマ割込みに関する実験

次に，実験 (5) と (6) の結果を**図 10** と**図 11** に示す．図 10 は，計測結果から最大値，最小値，平均値，最頻値を表したものである．また，図 11 は，標準偏差を表したものである．

まず，図 10 より，平均値において，Xen-Timer では Domain-U に割込み負荷がある場合に 1.000396 msec，RT-Domain に割込み負荷がある場合に 1.000752 msec となり，仮想タイマの周期である 1 msec より約 0.4 μsec から 0.75 μsec 程度遅延していることが分かる．一方，Natsume-Timer では，負荷の種類にかかわらず，0.999883 msec となり，PIT の周期である 0.999847 msec より，0.04 μsec 程度の遅延となった．以上より，タイマ割込みにおける遅延が最大で約 95%削減される結果となった．

また，最大値において，Xen-Timer では，Domain-U に割込み負荷がある場合に 2.731 μsec，RT-Domain に割込み負荷がある場合に 50.221 μsec の遅延が発生するが，Natsume-Timer では，RT-Domain に割込みがある場合でも 1.331 μsec まで抑制され，遅延が約 97%削減される結果となった．さらに，Natsume-Timer の最頻値に着目すると，PIT の周期と比べて約 0.1 μsec の差となっており，割込み周期に大きな歪みがないことが分かる．

最後に，図 11 より，標準偏差に着目すると Xen-Timer では，0.000459 から 0.001218 の間で遷移するが，Natsume-

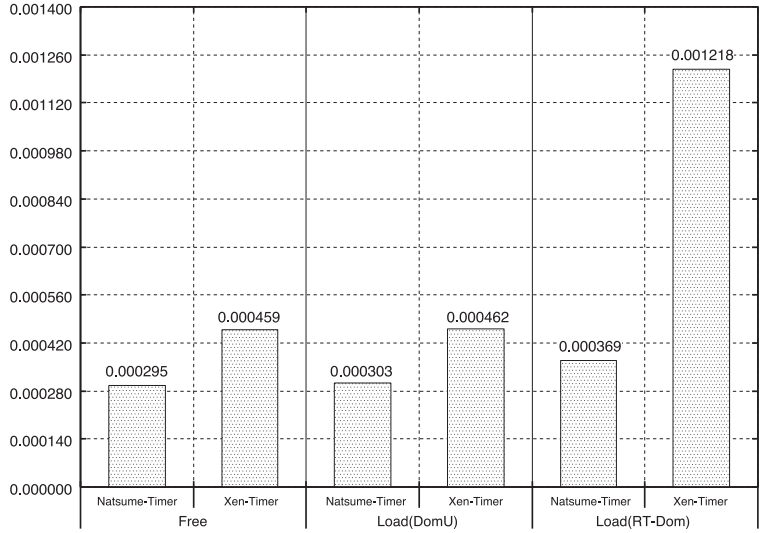

図 11　標準偏差（実験 (5)，(6)）
Fig. 11　Standard deviation (Experiment (5), (6)).

表 3　既存の RTOS における割込み応答時間
Table 3　Maximum response time of existing RTOSes.

RTOS	用途	割込み応答時間
NORTi	FA，業務用機器	数 μsec
Windows Embedded CE	小型情報端末	10.2 μsec
Xen 上の mini-os（負荷発生時）	-	8.401 μsec
Natsume-Xen 上の mini-os（負荷発生時）	-	3.520 μsec

Timer では，最大で 0.000369 となり，揺らぎが約 30%に抑制される結果になった．これらの結果から，提案機構はデバイス割込みだけでなく，タイマ割込みにおいても，同時に動作している VM や VMM における負荷を抑制し，リアルタイム性の保証に有効であると考えられる．

5.5.3　適用可能性に関する考察

文献 [32] と [33] より，既存の RTOS における割込み応答性能をまとめたものを**表 3** に示す．FA 用途に広く用いられている RTOS である「NORTi」の割込み応答時間は数 μsec であり，小型情報端末に用いられる「Windows Embedded CE」の割込み応答時間は最大 10.2 μsec である．一方，5.5.1 項で述べた実験結果より，割込み負荷発生時における Natsume-Xen の最大割込み処理時間は 3.520 μsec である．以上より，Natsume-Xen は FA システムや小型情報端末へ適用可能であると考えられる．また，同時に動作する高機能 OS が増加した際，Xen では最大割込み処理時間が 6.181 μsec，7.338 μsec，8.401 μsec と増大するが，Natsume-Xen では 3.551 μsec，3.500 μsec，3.520 μsec と変化が見られない．このことから，最悪応答時間の予測が容易で，かつ高機能 OS の数に対してスケーラブルであると考えられ，Xen と比較した場合でも Natsume-Xen が有効であると考える．

最後に，Natsume-Xen を実際のシステムに適用する際の，構成例について検討する．Natsume-Xen を FA に適用した場合，RTOS が動作しロボットを制御する制御用端末と，高機能 OS が動作し制御用端末をモニタリングする情報用端末の統合によるダウンサイジングが達成できる．具体的には，複数の情報用端末における CPU，ストレージ，ネットワークの共有や，制御用端末のコンフィグレーションやログを格納するストレージなど，ベストエフォートで利用される資源の共有により，低コスト化・省スペース化が実現できる．一方で，Fieldbus に代表される制御用ネットワークなどリアルタイム性が重要となるデバイスに対しては，PCI Pass-through による資源の専有が効果的である．このような資源割当てにより，小型化を達成しつつ，リアルタイム性と高機能性を両立させることができると考えられる．

6.　おわりに

本論文では，組込みシステムにおけるリアルタイム性と高機能性の両立という要求に対して，仮想化技術を活用し，単一計算機上で RTOS と高機能 OS の同時動作を実現する，仮想計算機モニタ「Natsume-Xen」を提案した．また，Natsume-Xen の実現に向けて，ゲスト OS による割込み通知の専有を可能にする，RTOS 向け割込み通知機構の設計と実装を行い，評価用 Xen と mini-os を用いた性能評価を行った．その結果，Natsume-Xen の基である Xen と比較して，平均処理時間が最大約 81%，最大処理時間が最

大約 58%，最小処理時間が最大約 85%削減され，処理時間の揺らぎも約 22%まで抑制されることを確認した．また，仮想タイマ割込みでは，Xen の仮想タイマと比較して，タイマ割込みの遅延の平均値で最大約 95%，最大値で最大約 97%削減し，処理時間の揺らぎも約 30%まで抑制されることを確認した．また，実装した機構別に評価を行い，提案機構のアプローチである割込み通知の専有は，割込み負荷による影響の抑制に寄与しており，リアルタイム性の保証において有効であることを確認した．

参考文献

[1] TRON Association: μITRON4.0 Specification (2007), ⟨http://www.t-engine.org/wp-content/themes/wp.vicuna/pdf/specifications/en_US/WG024-S001-04.03.00_en.pdf⟩.

[2] Wind River: VxWorks (2011), available from ⟨http://www.windriver.com/announces/vxworks6.9/⟩.

[3] QNX SOFTWARE SYSTEMS: QNX Realtime Operating System (RTOS) software, middleware, development tools and services for superior embedded design (2011), available from ⟨http://www.qnx.com/⟩.

[4] SYSGO: PikeOS RTOS and Virtualization Concept (2012), available from ⟨http://www.sysgo.com/products/pikeos-rtos-and-virtualization-concept/⟩.

[5] 新井利明，関口知紀，佐藤雅英，井上太郎，中村智明，岩尾秀樹：ナノカーネル方式による異種 OS 共存技術「DARMA」の提案，全国大会講演論文集，Vol.59, No.1, pp.139–140 (1999).

[6] Carpenter, B., Roman, M., Vasilatos, N. and Zimmerman, M.: The RTX Real-Time Subsystem for Windows NT, *Proc. USENIX Windows NT Workshop*, USENIX Association, pp.33–37 (1997).

[7] Renesas Electronics: SH-Mobile (2012), available from ⟨http://www.renesas.com/products/mpumcu/sh_mobile/⟩.

[8] Texas Instruments: OMAP Technology (2011), available from ⟨http://focus.ti.com/general/docs/gencontent.tsp?contentId=46946⟩.

[9] Renesas Electronics: Multi-Core Application Examples (SH4A-MULTI), (2012), available from ⟨http://www.renesas.com/products/mpumcu/multi_core/child/application_sh4a_multi.jsp⟩.

[10] ARM: The ARM Cortex-A9 Processors white paper (2009), available from ⟨http://www.arm.com/files/pdf/ARMCortexA-9Processors.pdf⟩.

[11] ARM: Cortex-A15 Processor (2011), available from ⟨http://www.arm.com/products/processors/cortex-a/cortex-a15.php⟩.

[12] Lammers, G. and Systems, R.-T.: Embedded Real-Time Virtualization and Partitioning, *Small Form Factor Boards Conference* (2009).

[13] Kanda, W., Yumura, Y., Kinebuchi, Y., Makijima, K. and Nakajima, T.: SPUMONE: Lightweight CPU Virtualization Layer for Embedded Systems, *Proc. IEEE/IFIP International Conference on Embedded and Ubiquitous Computing, 2008, EUC '08* (2008).

[14] 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*, pp.164–177, ACM (2003).

[15] Citrix Systems, Inc.: XenPCIpassthrough – Xen Wiki (2012), available from ⟨http://wiki.xen.org/wiki/Xen_PCI_Passthrough⟩.

[16] Yoo, S., Liu, Y., Hong, C.-H., Yoo, C. and Zhang, Y.: MobiVMM: A virtual machine monitor for mobile phones, *Proc. 1st Workshop on Virtualization in Mobile Computing* (*MobiVirt '08*), pp.1–5, ACM (2008).

[17] Yoo, S., Park, M. and Yoo, C.: A step to support real-time in virtual machine, *Proc. 6th IEEE Conference on Consumer Communications and Networking Conference* (*CCNC '09*), pp.405–411, ACM (2009).

[18] Borden, T.L., Hennessy, J.P. and Rymarczyk, J.W.: Multiple operating systems on one processor complex, *IBM Systems Journal*, Vol.28, pp.104–123 (1989).

[19] Neiger, G., Santoni, A., Leung, F., Rodgers, D. and Uhlig, R.: Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization, *Intel Technology Journal*, Vol.10, Issue 03 (2006).

[20] Dong, Y., Yu, Z. and Rose, G.: SR-IOV networking in Xen: architecture, design and implementation, *Proc. 1st Conference on I/O Virtualization, WIOV'08*, p.10, USENIX Association (2008).

[21] Dong, Y., Yang, X., Li, X., Li, J., Tian, K. and Guan, H.: High performance network virtualization with SR-IOV, *Proc. 2010 IEEE 16th International Symposium on High Performance Computer Architecture* (*HPCA*), pp.1–10 (2010).

[22] Dong, Y., Xu, D., Zhang, Y. and Liao, G.: Optimizing Network I/O Virtualization with Efficient Interrupt Coalescing and Virtual Receive Side Scaling, *Proc. 2011 IEEE International Conference on Cluster Computing* (*CLUSTER*), pp.26–34 (2011).

[23] Ahmad, I., Gulati, A. and Mashtizadeh, A.: vIC: Interrupt coalescing for virtual machine storage device IO, *Proc. 2011 USENIX Conference on USENIX Annual Technical Conference*, *USENIXATC'11*, p.4, USENIX Association (2011).

[24] Willmann, P., Shafer, J., Carr, D., Menon, A., Rixner, S., Cox, A. and Zwaenepoel, W.: Concurrent Direct Network Access for Virtual Machine Monitors, *Proc. IEEE 13th International Symposium on High Performance Computer Architecture, 2007, HPCA 2007*, pp.306–317 (2007).

[25] Landau, A., Ben-Yehuda, M. and Gordon, A.: SplitX: Split guest/hypervisor execution on multi-core, *Proc. 3rd Conference on I/O Virtualization, WIOV'11*, p.1, USENIX Association (2011).

[26] William von Hagen: *Professional Xen Virtualization*, Wiley Publishing Inc. (2008).

[27] Free Software Foundation: GNU GRUB (2011), available from ⟨http://www.gnu.org/software/grub/index.html⟩.

[28] PCI-SIG: *PCI Local Bus Specification*, Revision 3.0 edition (2002).

[29] ARM: *ARM PrimeCell Vectored Interrupt Controller* (*PL192*) *Technical Reference Manual*, A edition (2002).

[30] Intel Corporation: 8254/82C54: Introduction to Programmable Interval Timer (2011), available from ⟨http://www.intel.com/design/archives/periphrl/docs/7203.htm⟩.

[31] Intel Corporation: Intel I/O Controller Hub 8/9/10 and 82566/82567/82562V Software Developer's Manual (2009), available from ⟨http://download.intel.com/design/network/manuals/322409.pdf⟩.

[32] 電子情報技術産業協会：組み込み OS 技術動向調査報告書 (2006)，available from ⟨http://home.jeita.or.jp/is/

committee/tech-std/micro/pdf/06-kei-5.pdf⟩.
[33] Dedicated Systems Experts: Independent Real Time Report for Windows Embedded: WINDOWS EMBEDDED CE6.0 R2 ON AN X86 PLATFORM (2009), available from ⟨http://www.microsoft.com/windowsembedded/en-us/develop/ce-on-x86-platform.aspx⟩.

渡邉 和樹　（正会員）

1987 年生．2010 年立命館大学情報理工学部情報システム学科卒業，2012 年同大学大学院理工学研究科博士前期課程情報理工学専攻修了，同年三菱電機（株）情報技術総合研究所，現在に至る．オペレーティングシステム，仮想化技術，組込みシステム，リアルタイムシステム等に興味を持つ．

片山 吉章　（正会員）

1971 年生．1994 年立命館大学理工学部情報工学科卒業，1996 年同大学大学院理工学研究科修士課程情報システム学専攻修了，同年三菱電機（株）情報技術総合研究所，現在に至る．リアルタイムシステム，オペレーティングシステム，仮想計算機システム，携帯電話システム，組込みソフトウェア基盤技術の研究開発に従事．

松本 利夫

1958 年生．1985 年東京大学基礎科学科卒業，同年三菱電機（株）入社．固定磁気ディスク用薄膜磁気ヘッド・薄膜媒体，PC サーバ用 BIOS，高速データベース処理装置の開発に従事．2002 年より同社情報技術総合研究所にて映像配信システム，携帯電話システム，組込みソフトウェア基盤技術の研究開発に従事．電子情報通信学会会員．

瀧本 栄二　（正会員）

1976 年生．1999 年立命館大学理工学部情報学科卒業，2001 年同大学大学院理工学研究科博士前期課程修了，2005 年同研究科博士後期課程単位取得退学，同年（株）ATR 適応コミュニケーション研究所専任研究員，2010 年立命館大学情報理工学部情報システム学科助手，現在に至る．主にシステムソフトウェア，無線通信に関する研究に従事．

樫山 武浩　（正会員）

1983 年生．2005 年関西大学総合情報学部卒業，2007 年同大学大学院総合情報学研究科博士課程前期課程修了，2010 年同研究科博士課程後期課程修了，同年関西大学総合情報学部ポストドクトラルフェロー，立命館大学立命館グローバル・イノベーション研究機構ポストドクトラルフェローとなり，現在に至る．オペレーティングシステム，コンピュータセキュリティ等の研究に従事．また，2004～2010 年（株）関西総合情報研究所．CAD/CG，GIS，画像処理等の研究業務に従事．博士（情報学）．土木学会会員．

毛利 公一　（正会員）

1972 年生．1994 年立命館大学理工学部情報工学科卒業，1996 年同大学大学院理工学研究科修士課程情報システム学専攻修了，1999 年同研究科博士課程後期課程総合理工学専攻修了．同年東京農工大学工学部情報コミュニケーション工学科助手，2002 年立命館大学理工学部情報学科講師，2004 年同大学情報理工学部情報システム学科講師，2008 年同准教授となり，現在に至る．博士（工学）．オペレーティングシステム，仮想化技術，コンピュータセキュリティ等の研究に従事．電子情報通信学会，日本ソフトウェア科学会，ACM，IEEE-CS，USENIX 各会員．