仮想計算機モニタ Xen における RTOS 向け割込み通知機構

渡邉和樹[†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]

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，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，最大実行時間で約 60%，平均実行時間で約 85%の性能改善を達成した．また，実行時間の揺らぎも約 21%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．

Management of interrupts for RTOS in Xen Hypervisor

Kazuki WATANABE,[†1] Yoshiaki KATAYAMA,[†2]
Toshio MATSUMOTO,[†2] Eiji TAKIMOTO,[†3]
Takehiro KASHIYAMA[†4] and Koichi MOURI [†3]

We have extended 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 with causes delay and fluctuation 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 shortening the average interrupt response time of 85% and the maximum interrupt response time of 60%. In addition, we succeed in reducing about 21% of execution time fluctuations.

1. はじめに

近年，携帯端末や車載システムといった組込みシステムでは，製品の多機能化に伴い，リアルタイム性に加えて，高い機能性の両立が求められている．中でも，スマートフォンや PDA といった情報端末では，小型化や省電力化が求められている．従来から，組込みシステムで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，機器の制御を主目的とした最小限の機能提供により，処理時間の予測可能性を高め，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，処理時間の予測可能性は低い．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する手法として，2 通りの既存の手法がある．一つは，PikeOS[4]，Darma[5]，RTX[6] に代表される，RTOS と高機能 OS 両方の特性を併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．もう一つは，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．

†1 立命館大学大学院理工学研究科
Graduate School of Science and Engineering, Ritsumeikan University
†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
†3 立命館大学情報理工学部
College of Information Science and Engineering, Ritsumeikan University
†4 立命館大学グローバル・イノベーション研究機構
Ritsumeikan Global Innovation Research Organization, Ritsumeikan University

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，小型化や省電力化が達成できない．

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，小型化や省電力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．これに伴い，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，ゲスト OS に対して，全ての計算機資源を排他的に割り当てる必要があったり，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たり，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挙げられる．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 の処理時間の予測可能性を低下させ，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は RTOS に占有させることで，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，全てのデバイスで共通に処理されている．そこで，デバイス毎の割込み通知を実現し，デバイスの完全な占有を可能にする 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 の問題点を踏まえ，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から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] や SPUMONE[13]，MobiVMM[16),17] が挙げられる．また，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] が挙げられる．

RTS-Hypervisor[12](以下，RTS) は，Intel VT[18] 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 では全ての資源をゲスト 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 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を除いて，全ての機能を停止する．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 は全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 や OMAP などの複数の計算機を搭載する手法と同様に，小型化や省電力化が困難になるという問題を有する．

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 が占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 毎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 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，$20msec$ 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，一度全ての割込みを RTX がハンドルした上で処理するため，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．

以上で述べたような既存研究における問題点を解決するために，Natsume では既存の VMM として実績のある Xen にリアルタイム性を付加するアプローチを採る．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，Domain と呼ばれる 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することができる．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．その結果，システムの小型化や省電力化の実現に寄与できる．
- **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．

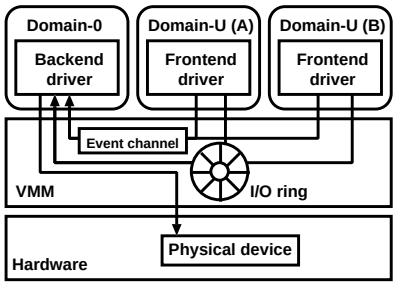

図 1 Xen の仮想デバイスドライバによるアクセス
Fig. 1 Access to I/O device using virtual device driver.

- **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あり，かつ新たにゲスト OS や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 から成る．

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 へ占有させ，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[19] を用いることで，割り当てる VM を固定し，特定のゲスト OS による占有を実現できる．ただし， Xen ではゲスト OS がアイドル状態になった場合，XenTools による設定内容に関わらず，CPU 資源が切り離され，Idle-Domain と呼ばれる仮想 VM に割り当てられる．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．

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 による処理の遅延や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] と呼ばれるもうひと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 処理の占有が実現できる．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関わらず，全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 の割込みは全て，両 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入出力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デバイスと平等に扱われたり，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．

これらの問題を解決するためには，RTOS に対して常に CPU 資源を割り当てつつ，割込み通知機構を占有させることで，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 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，小型化や省電力化に寄与する．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，管理用インター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実現する．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，各資源の占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 における割込みの占有を実現できない．そこで，ゲスト OS による割込みの占有を実現する 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占有を実現するために，デバイス毎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) 共有デバイスと占有デバイスにおける割込み通知機構を分離するために，占有デバイスを割込

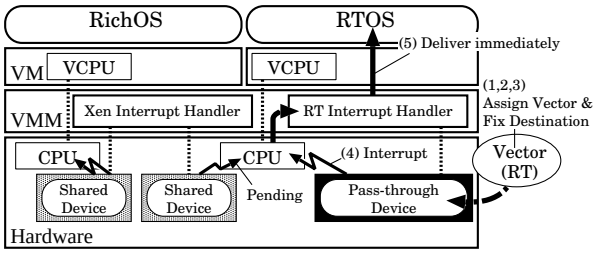

図 2 RTOS 向け割込み通知モデル
Fig. 2 Interrupt delivery model for RTOS.

みレベルで明確に識別する．

(2) 占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) でハンドルする．Natsume ハンドラは，占有デバイスと 1 対 1 で対応付けられており，占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) ハンドル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 資源を常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，占有デバイスの情報を取得し，占有デバイスを識別する．

(2) 占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) 占有デバイスからの割込みの通知先を RTOS が占有する CPU に固定する．

(4) 占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割込み処理を横取りし，RTOS の割込みハンドラに即座に制御を移す．

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占有デバイスに関する情報を取得し，占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[20] 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

占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，占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，占有デバイス・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 ハンドラは占有デバイス毎に提供される．これにより，占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 によって割り当てられる 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 は全ての PCI デバイスで最大 4 つに制限される[21]．すなわち，デバイス毎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[21](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[22](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 が占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

占有デバイスからの割込みの通知先を，常に RTOS が占有する CPU に固定する機構を実装した．Xen では全ての割込みが，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，全てのゲスト 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 を占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 が占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伴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．例えば，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，専用のベクタと占有デバイスは 1 対 1 で対応付けられているため，

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

	VMM	IRQ	CPU
実験 (1)	Xen	共有 (再現)	Idle 可
実験 (2)	Natsume-CPU	共有 (再現)	固定
実験 (3)	Natsume-Vector	固定 (Natsume)	Idle 可
実験 (4)	Natsume-ALL	固定 (Natsume)	固定

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 資源の占有を常に実現する目的で，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．

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関わらず，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，最大実行時間や処理時間の揺らぎに影響が無いことを確認できれば良い．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 3 つの異なるパラメータを用いて計測する．

(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: 提案手法すべてを適用した Natsume-Xen において，割込み処理性能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 0 から 2 まで変化させ，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．

5.2 実 験 手 法

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察する．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．
(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，ゲスト OS として最小限の機能しか持たない mini-os を用いる．
(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，割込みをハンドルした瞬間の 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．
(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 10000 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 10000 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時，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，平均実行時間と最大実行時間を算出する．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．

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 間のインター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．

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/RAM512MB 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/RAM512MB 割当て)

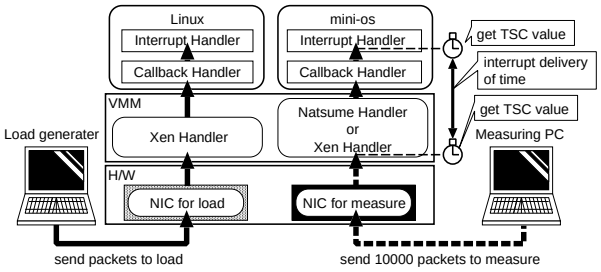

図 3 性能評価実験の概要
Fig. 3 Overview of performance evaluation.

- **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ーフェース[23] 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時，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ーフェース**
  10000 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，既存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

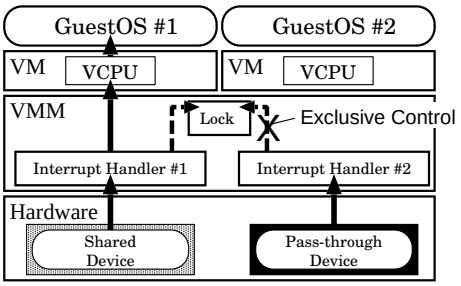

図 4 ロックを共有した割込みハンドラ
Fig. 4 Interrupt handlers that share locks.

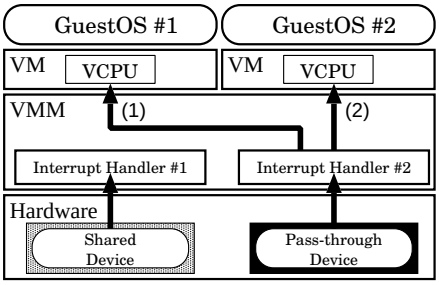

図 5 全ゲスト OS への割込み通知
Fig. 5 Delivery interrupts to all guest OSes.

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 4** と**図 5** に示す機構を追加した．

- **割込みハンドラのロックを共有する機構 (図 4)**
  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制御して実行される．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．
- **割込みを全てのゲスト OS に通知する機構 (図 5)**
  複数のデバイスで IRQ が共有されている場合，割込みをハンドルした時点では，割込み発生元のデバイスを特定することはできない．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全てのゲスト 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様に実装している．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 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 に性能評価用 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 4，図 5 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．

5.4.2 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．

5.4.3 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．

5.4.4 実験 4(Natsume-ALL)

この実験では，提案手法の全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 共に，提案手法の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．

5.5 実験結果と考察

実験結果を**図 6** と**図 7** に示す．図 6 は，計測結果から最大実行時間と平均実行時間，最小実行時間を表したものである．また，図 7 は，測定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 6 から，Xen では負荷の増大に伴って最大実行時間が増加しているが，Natsume-ALL では，最大実行時間・平均実行時間・最小実行時間共に変化はみられないことが分かる．また，処理時間も Xen と比較して，平均実行時間で約 2.5μsec の高速化となり，約 85%の性能改善，最大実行時間で約 5μsec から 2.5μsec の高速化となり，約 60%から 45%の性能改善を確認できた．さらに，最小実行時間では約 2μsec から 1.5μsec の高速化となり，約 76%から 72%の性能改善を確認できた．

次に，図 7 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 では負荷の状況に関わらず約 0.08 前後で安定する結果になった．このことから， Xen では負荷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関わらず，処理時間の揺らぎに変化が見られず，かつ処理時間の揺らぎそのものの大きさも Xen と比較して約 21%まで抑制でき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．

また，図 6 から，Xen は割込み通知に最大 8.4μsec 要することが分かる．一方，Natsume-ALL では最大 3.5μsec まで抑制されており，Xen における平均処理時間の約 0.5μsec 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．

最後に，実装した機構毎の有効性について考察する．図 6 から，Xen と比較して，平均実行時間において，Natsume-Vector で約 0.7μsec，Natsume-CPU で約 1.5μsec の高速化を確認できる．このことから，Natsume-CPU，Natsume-Vector 共に性能が改善され，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 7 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，負荷の無い状態と比較して，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 と比較して負荷に関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占有は，リアルタイム性の保証に有効であると言える．

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 による割込み通知機構の占有を可能にする，RTOS 向け割込み通知機構の設計と実装を行い，評価

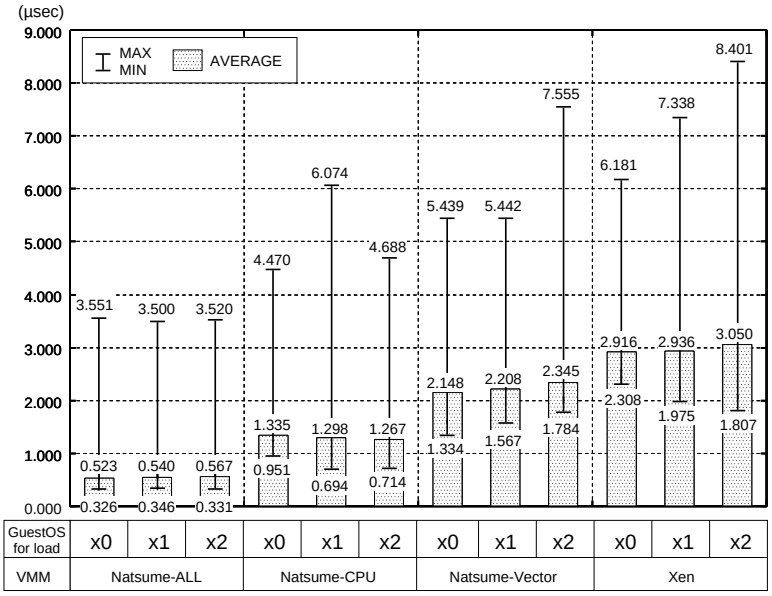

図 6 平均・最小・最大実行時間
Fig. 6 Average, minimum and maximum response time.

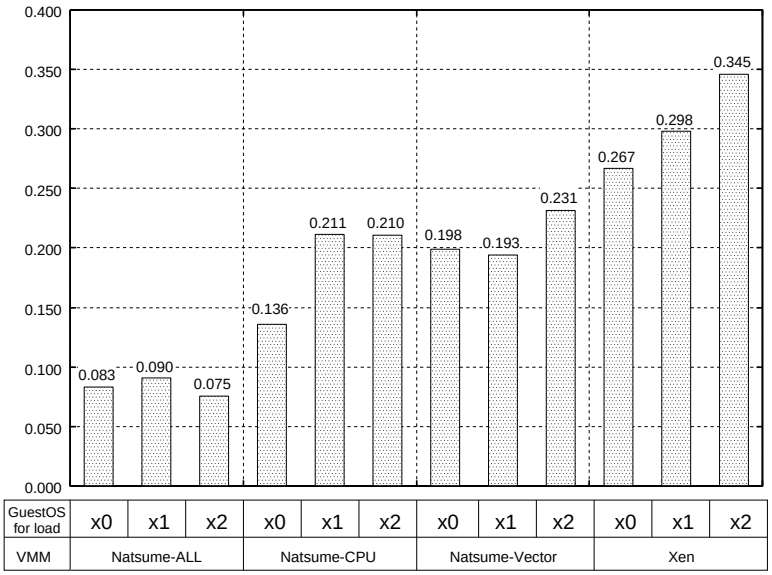

図 7 標準偏差
Fig. 7 Standard deviation.

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，平均実行時間が最大約 85%，最大実行時間が最大約 60%，最小実行時間が最大約 76%の性能改善を確認し，処理時間の揺らぎも約 21%まで削減することを確認した．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占有は，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．

現在，タイマ割込みを提案機構を用いて通知する仕組みを検討している．Xen のタイマ割込みは，入出力資源とは異なる経路によりゲスト OS へ割込みが通知される．タイマの割込み通知経路を提案機構に変更することで，さらなるリアルタイム性の向上を実現できると考える．

参 考 文 献

1) TRON Association: μITRON4.0 Specification, http://www.assoc.tron.org/spec/itron/itron403e/mitron-403e.pdf (2007).
2) Wind River: VxWorks, http://www.windriver.com/announces/vxworks6.9/ (2011).
3) QNX SOFTWARE SYSTEMS: QNX Real-time Operating System (RTOS) software, middleware, development tools and services for superior embedded design, http://www.qnx.com/ (2011).
4) SYSGO: PikeOS RTOS and Virtualization Concept, http://www.sysgo.com/products/pikeos-rtos-technology/ (2011).
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, *In Proceedings of the USENIX Windows NT Workshop*, USENIX Association, pp. 33–37 (1997).
7) Renesas Electronics: SH-Mobile, http://www.renesas.com/products/mpumcu/sh_mobile/sh_mobile_landing.jsp (2011).
8) Texas Instruments: OMAP Technology, http://focus.ti.com/general/docs/gencontent.tsp?contentId=46946 (2011).
9) Renesas Electronics: Application Excamples (SH4A-MULTI), http://www.renesas.eu/products/mpumcu/multi_core/child_folder/application_sh4a_multi.jsp (2011).
10) ARM: The ARM Cortex-A9 Processors white paper, http://www.arm.com/files/pdf/ARMCortexA-9Processors.pdf (2009).
11) ARM: Cortex-A15 Processor, http://www.arm.com/products/processors/cortex-a/cortex-a15.php (2011).
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, *In Proceedings of Embedded and Ubiquitous Computing, 2008. EUC '08. IEEE/IFIP International Conference on* (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, *In Proceedings of the nineteenth ACM symposium on Operating systems principles*, ACM, pp. 164–177 (2003).
15) Citrix Systems, Inc.: XenPCIpassthrough - Xen Wiki, http://wiki.xen.org/xenwiki/XenPCIpassthrough (2011).
16) Yoo, S., Liu, Y., Hong, C.-H., Yoo, C. and Zhang, Y.: MobiVMM: a virtual machine monitor for mobile phones, *In Proceedings of the First Workshop on Virtualization in Mobile Computing (MobiVirt '08)*, ACM, pp. 1–5 (2008).
17) Yoo, S., Park, M. and Yoo, C.: A step to support real-time in virtual machine, *In Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference (CCNC '09)*, ACM, pp.405–411 (2009).
18) Neiger, G., Santoni, A., Leung, F., Rodgers, D. and Uhlig, R.: Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization, *Intel Technology Journal*, Vol. Volume.10, No. Issue.03 (2006).
19) William von Hagen: *Professional Xen Virtualization*, Wiley Pubnlishing.Inc (2008).
20) Free Software Foundation: GNU GRUB, http://www.gnu.org/software/grub/index.html (2011).
21) PCI-SIG: *PCI Local Bus Specification*, Revision 3.0 edition (2002).
22) ARM: *ARM PrimeCell Vectored Interrupt Controller (PL192) Technical Reference Manual*, A edition (2002).
23) Intel Corporation: Intel I.O Controller Hub 8/9/10 and 82566/82567/82562V Software Developer's Manual, http://download.intel.com/design/network/manuals/322409.pdf (2009).