Vol.2011-ARC-195 No.27
Vol.2011-OS-117 No.27
2011/4/14

仮想計算機モニタ Xen における MSI を用いた RTOS 向け割込み通知機構

渡邉和樹[†1]　瀧本栄二[†2]
樫山武浩[†3]　毛利公一[†2]

現在，仮想計算機 (VM) 上のゲスト OS としてリアルタイムタイム OS(RTOS) と高機能 OS を同時に動作させることを目的に，Xen の拡張を行っている．本論文では，VM 上ではデバイス割込みが仮想化され，RTOS の信頼性が損なわれる可能性に着目し，これの解決を試みたので，その経過を報告する．具体的には，MSI と専用の割込みハンドラを活用し，割込みを効率的に通知する割込み通知機構を考案し，その一部を実装した．また，実際に Linux を用いた性能評価を行った．

Management of interrupt for RTOS using MSI in Xen Hypervisor

Kazuki WATANABE,[†1] Eiji TAKIMOTO,[†2]
Takehiro KASHIYAMA[†3] and Koichi MOURI [†2]

We have been extended Xen hypervisor's capability to execute real time operating systems(RTOS) on it. On virtual machines, interrupts triggered by devices are also virtualized. Therefore performance and responce time of RTOS are degraded. To solve this problem, we improved interrupt management mechanism of Xen by MSI and lightweight interrupt handler. In this paper, we discuss the concrete method of the improvement, and its evaluation.

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

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，小型化や省電力化によるランニングコストの削減が求められている．従来，組込みシステムでは，μITRON[1] や VxWorks[2] に代表されるリアルタイム OS(以下，RTOS) が利用される．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する．一方，高度な情報処理を行うシステムでは，Windows や Linux に代表される高機能 OS が利用される．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をしており，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段として，2 通りの既存の手法がある．一つは，PikeOS[3] や Darma[4]，RTX[5] 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，その設計は容易ではなく，ソフトウェア開発コストの増加を招く原因となる．もう一つは，OMAP プラットフォーム[6] 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，物理的な計算機の増加や，計算機間の通信機構の複雑化に伴い，ハードウェア開発コスト増加を招く原因となる．これらの問題を解決する手段として，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，開発コストやランニングコストの削減を図ることができる．また一方で，組込み機器向けのプロセッサ市場において，Cortex-A9 MPCore[7] のようにマルチコア化されたものや，Cortex-A15[8] 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[9] や SPUMONE[10] 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[11] を拡張し，リアルタイム性の保証に影響しない資源の共有や，ゲスト OS 間の安全性確保

Vol.2011-ARC-195 No.27
Vol.2011-OS-117 No.27
2011/4/14

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．既存研究である RTS-Hypervisor は，実環境と同様の資源を各ゲスト OS に割り当てることで，実環境と同様のリアルタイム性と高機能性を実現する．しかし，この手法は，複数の計算機を搭載する手法と，同等のハードウェア開発コストが必要になる．そこで，リアルタイム性が要求されない処理に用いる資源や，高機能 OS に割り当てる資源の共有を可能にするアプローチを取る．また，SPUMONE は，CPU 以外の仮想化を行わず，必要最小限の機能を提供し，VMM を軽量化することで，実環境に近いパフォーマンスを実現する．しかし，ゲスト OS 間におけるメモリの保護を行わないため，高機能 OS 上の不具合が，RTOS に影響する可能性がある．そこで，各ゲスト OS を保護しつつ，計算機資源を適切に管理するアプローチによって，安全性と性能の両立を目指す．

VM 上で 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 上では，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を RTOS に占有させることで，この解決を試みる．

以下，本論文では，2 章で仮想計算機モニタ Natsume の概要とその基としている Xen について述べる．次に，3 章で Natsume における資源管理を実現する RTOS 向け割込み通知モデルについて述べる．そして，4 章で性能評価の手順と，その結果から行った考察を述べる．そして，5 章で関連研究について述べ，最後に 6 章でまとめる．

2. 仮想計算機モニタ Natsume-Xen

2.1 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．Natsume は，管理用インターフェース持つ特権 Domain，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．

Natsume は，リアルタイム性が重要な処理を 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を両立を可能にする．また，単一の計算機上で複数の OS を動作させることで，ハードウェア開発コストやランニングコストの削減を実現する．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開発コストの削減を実現する．

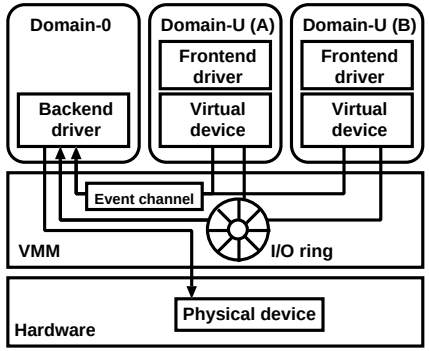

図 1　Xen の仮想デバイスドライバによるアクセス

2.2 仮想計算機モニタ Xen

Xen は，オープンソースによって開発されている 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する．Xen の準仮想化では，Xen 自身の制御や VM の管理を行うための Domain を Domain-0，その他の Domain を Domain-U と呼ぶ．

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，仮想デバイスドライバを用いる．仮想デバイスドライバは，物理資源に複数の仮想資源を関連付けることで，ゲスト OS 間で物理資源の共有を実現する．Domain-0 が持つ仮想デバイスドライバをバックエンドドライバ，Domain-U が持つ仮想デバイスドライバをフロントエンドドライバと呼ぶ．

ゲスト OS が仮想デバイスドライバを利用してデバイスへアクセスする場合，**図 1** に示すように，フロントエンドドライバから VMM の機構であるイベントチャネルや I/O リングを経由し，Domain-0 で動作するゲスト OS が実際の処理を行う．しかし，これは VMM や Domain-0 に負荷がかかった場合，要求の完了が遅延する可能性がある．また，イベントチャネルによる割込み通知機構は，デバイスが IRQ を共有した場合，割込み通知機構が共有され，その性能が著しく低下する．つまり，仮想デバイスドライバによる資源の割当ては，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．そのため，Xen 上で RTOS を動作させる場合，RTOS が VMM や Domain-0 の影響を受けないようにする資源管理機構が必要になる．

Vol.2011-ARC-195 No.27
Vol.2011-OS-117 No.27
2011/4/14

2.3 Natsume-Xen における資源管理

VM 上で RTOS を動作させ，リアルタイム性を保証する場合，計算機資源の管理が重要になる．仮想デバイスドライバを用いて資源を割り当てた場合，資源が VM 間で共有されるため，性能が他の VM や VMM の負荷の影響を受ける．これは，RTOS のリアルタイム性の保証を困難にする．そこで，RTOS に資源を占有させることで，これを解決する．

Natsume では，VM 上で RTOS を動作可能にするために，3 つの資源を占有させる．CPU と主記憶の占有は，Xen が提供する XenTools[12] を用いて実現する．入出力の占有は，Xen の資源管理機構のひとつである PCI Pass-through と我々が提案する RTOS 向け割込み通知モデルを用いて実現する．

2.4 PCI Pass-through と課題点

PCI Pass-through は，Xen が提供する資源管理機構であり，VM に PCI デバイスを排他的に割り当てる．対象デバイスは，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．

PCI Pass-through は，ゲスト OS による I/O 処理の占有を実現する．しかし，デバイスから発生する割込みは，仮想デバイスドライバによる割り当てと同様に，イベントチャネルを経由して行われる．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 に通知することができない．

すなわち，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，他の VM や VMM における負荷の影響を受ける．これは，割込みの遅延や損失の原因となり，リアルタイム性の保証を困難にする．これを解決するために，PCI が提供する割込み通知方式である MSI(Message Signaled Interrupts)[13] と専用割込みハンドラを用いた RTOS 向け割込み通知モデルを提案する．

3. RTOS 向け割込み通知モデル

3.1 概　　要

提案モデルは，RTOS によるリアルタイム性の保証を可能にする目的で， I/O 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．提案モデルと PCI Pass-through により，完全なデバイスの占有を可能にする．その概要を**図 2** に示す．

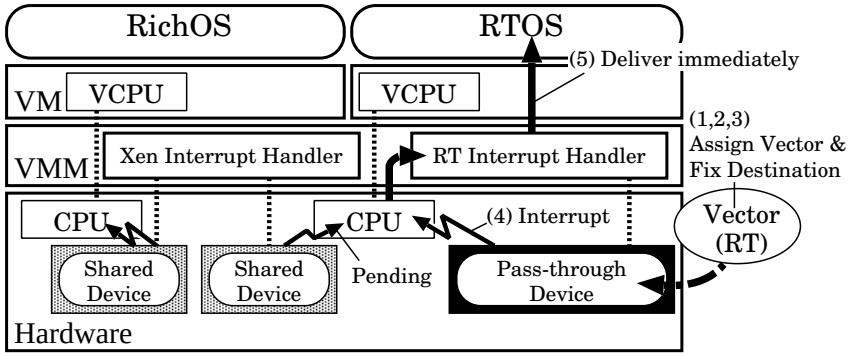

図 2 RTOS 向け割込みモデルの概要

(1) システム起動時に，占有デバイスの PCI バス・デバイス・機能番号を取得し，占有デバイスを識別する

(2) MSI を用いて，占有デバイスに専用の Vector を割り当てることで，他デバイスと割込みレベルで区別する

(3) 同様に，MSI を用いて，占有デバイスの割込み通知先を RTOS が占有する CPU に固定する

(4) 専用 Vector を持った割込み，すなわち占有デバイスからの割込みを，専用の割込みハンドラ (以下，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，ゲスト OS の割込みハンドラに即座に制御を移す

3.2 現在の実装状況

提案モデルの実現に向けて，以下の機構を実装した．

- **デバイス識別機能**
  Xen の起動オプションから，占有デバイスの情報を取得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[14] から**図 3** に示すオプション文字列を取得し，PCI Pass-through 対象デバイスの PCI バス・デバイス・機能番号を得る．
- **Vector と割込み通知先 CPU の固定機構**
  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，デバイスに専用の Vector を割り当てる機構を実装した．通常，PCI デバイスの Vector は BIOS によって

Vol.2011-ARC-195 No.27
Vol.2011-OS-117 No.27
2011/4/14

```
1: title  Xen-Natsume
2: root   (hd0,5)
3: kernel /boot/xen-3.4.3-natsume.gz
4: module /boot/vmlinuz-2.6.18.8-xen
5:    root=UUID=fa9065b1-54df-4cc7-a4b6-72a3f0406708 ro
6:    reassigndev=0000:05:00.0,0000:04:00.0,0000:00.19.0
7:    pciback.peermissive pciback.hide=(05:00.0)(04:00.0)(00:19.0)
8: module /boot/initrd-2.6.18.8-xen-2.6.18.8-xen
9: boot
                                  (BusNo:DeviceNo:FunctionNo)
```

図 3 Xen の起動オプション

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，割り当て可能な IRQ は全ての PCI デバイスで最大 4 つと制限される[13]．そこで，MSI による IRQ に依存しない Vector 割当てを採用する．MSI により，任意の Vector を各デバイスに割り当てることを可能にし，全ての占有デバイスで，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．

また，割込みコントローラに APIC[15] を利用している環境では，MSI を用いて割込み通知先 CPU を特定の CPU に固定することができる．割込み通知先 CPU を RTOS が占有するものに固定することで，割込みの転送に伴うオーバヘッドを削減することができる．

- **Natsume ハンドラ**
  既存の割込みハンドラから，割込みの共有に関する処理を削減し，軽量化を施した．割込みの共有とは，デバイスによる IRQ・Vector の共有と，ゲスト OS によるデバイスの共有である．なお，既存機構から割込みを横取りする機能は未実装である．
- **CPU の再割当てを防止する機構**
  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実装した．Xen では，XenTools によって，特定の VM が物理 CPU を占有する設定にした場合でも，VM がアイドル状態になった際，物理 CPU が IdleDomain と呼ばれる仮想 VM に割り当てられる．これにより，割込み発生時に物理 CPU の再割当てが発生し，オーバヘッドの原因となる．これを回避する目的で，VM 上で低優先度で動作する CPU バウンドなユーザプログラムを実装した．

表 1 評価環境

CPU	Core i7 920
Mainboard	DX58SO
評価用デバイス	82567LM-2 Gigabit Network Connection
負荷用デバイス	82541GI Gigabit Ethernet Controller
評価用ゲスト OS	linux-2.6.18-xen(CPU1 個/RAM2GB)
負荷用ゲスト OS	linux-2.6.18-xen(CPU3 個/RAM4GB)

4. 評　　価

4.1 実験内容

提案モデルによって，他のデバイスやゲスト OS における割込み負荷による影響を抑制できることを示す目的で，先述した機構を実際に適用し，性能評価を行った．評価環境を**表 1** と**図 4** に示し，具体的な手順を以下に示す．

(1) 負荷用 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，評価用 OS に PCI Pass-through を用いて評価用デバイスを割り当てる
(2) 割込みが発生した瞬間を検知するため，計測用デバイスの MSI を有効にし，計測用の Vector を割当てる (計測用 Vector は，Natsume ハンドラ，もしくは Xen のハンドラに TSC の値を記録する機能を追加した計測用割込みハンドラに関連付ける)
(3) 負荷用デバイスに対して，外部から大量の UDP パケットを送信し，割込みによる負荷をかける
(4) 計測用デバイスに対して，外部から 10msec に 1 回の間隔で，10000 個の UDP パケットを送信する
(5) 10000 個のパケットについて，VMM の Natsume ハンドラ・計測用割込みハンドラと，ゲスト OS の割込みハンドラで TSC の値を記録し，差分から所要時間を算出する

計測用デバイスからの割込みが発生した時刻を正確に取得するには，計測対象となる割込みの Vector を固定する必要がある．そのため，計測では提案モデルの一部である MSI による Vector 固定機構を，オリジナルの Xen にも適用した．しかし，この状態では本来の Xen の性能評価が行えないため，デバイスの Vector や IRQ が固定されておらず，かつ計測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組み込み，性能評価用 Xen を作成した．

Vol.2011-ARC-195 No.27
Vol.2011-OS-117 No.27
2011/4/14

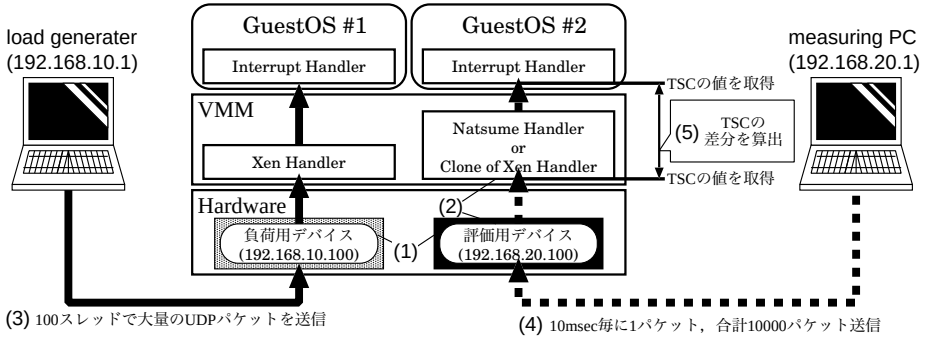

図 4 性能評価実験の概要

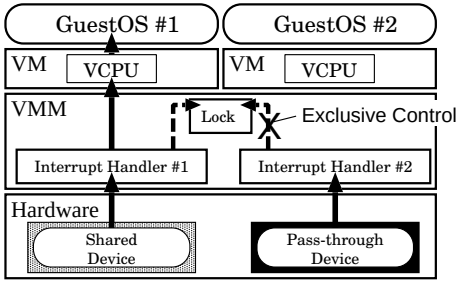

図 5 ロックを共有した割込みハンドラ

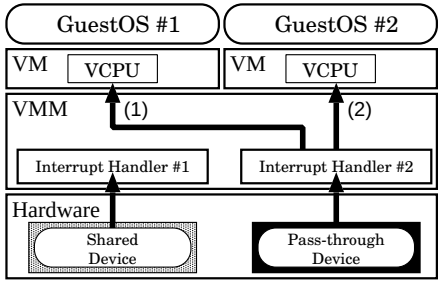

図 6 割込みを全てのゲスト OS へ通知する様子

4.2 性能評価用 Xen の概要

Natsume と性能比較を行う目的で，オリジナルの Xen に Vector 固定機構と，IRQ を共有している状況を再現する機構を追加した．具体的には，Vector 固定機構の他に**図 5** と**図 6** 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 5)**
  イベントチャネルでは，同一の Vector 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，複数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 6)**
  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，IRQ を共有したデバイスを割当てた場合，IRQ を共有しているデバイスを割当てている全てのゲスト 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，計測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，割込みを転送する機構を実装した．なお，負荷用デバイスの割込みを計測用デバイスに転送する機構は，計測用ゲスト 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．

表 2 評価環境

VMM	Vector	Natsume ハンドラ	CPU の再割当て防止
Xen	共有 (再現)	-	-
Natsume-CPU	共有 (再現)	-	適用
Natsume-Vector	固定	適用	-
Natsume-ALL	固定	適用	適用

4.3 実験結果

性能評価用 Xen と Natsume を用いて**表 2** に示す環境を用意し，実験を行った．具体的には，性能評価用 Xen と Natsume に対して，CPU の再割当てを防止する機構を適用したパターンと適用しないパターンについて実験を行った．

実験結果を**図 7** から**図 11** に示す．図 7 は，計測結果から各環境における最悪実行時間・平均実行時間・最小の実行時間を算出したものである．また，**図 8** から図 11 は，各環境における計測した実データを表したものである．

図 7 より，性能評価用 Xen と全ての機構を適用した Natsume(以下，Natsume-ALL) を比較すると，最悪実行時間で約 45μsec，平均実行時間で約 2.8μsec の高速化となり，約 85%から 95%の改善を確認できた．また，Xen と CPU の再割当て防止する機構を適用した Natsume(以下，Natsume-CPU) と比較すると，最悪実行時間で約 42μsec，平均実行時間で約 1.8μsec の高速化となった．そして，Natsume ハンドラのみを適用した Natsume(以下，Natsume-Vector) と比較すると，最悪実行時間で約 2μsec，平均実行時間で約 0.8μsec の高速化となった．また，最小の実行時間では，Natsume-CPU が約 6μsec の高速化，Nataume-Vector が約 7μsec の高速化，Natsume-ALL が約 10μsec の高速化という結果になった．

次に，図 8 から図 11 より，割込み通知に要したクロック数は，Xen で概ね 5000 から 25000 クロック，Natsume-CPU では 2000 から 12000 クロック，Natsume-Vector では 2000 から 20000 クロック，Natsume-ALL では，1000 から 10000 クロックの範囲で推移しており，

Vol.2011-ARC-195 No.27
Vol.2011-OS-117 No.27
2011/4/14

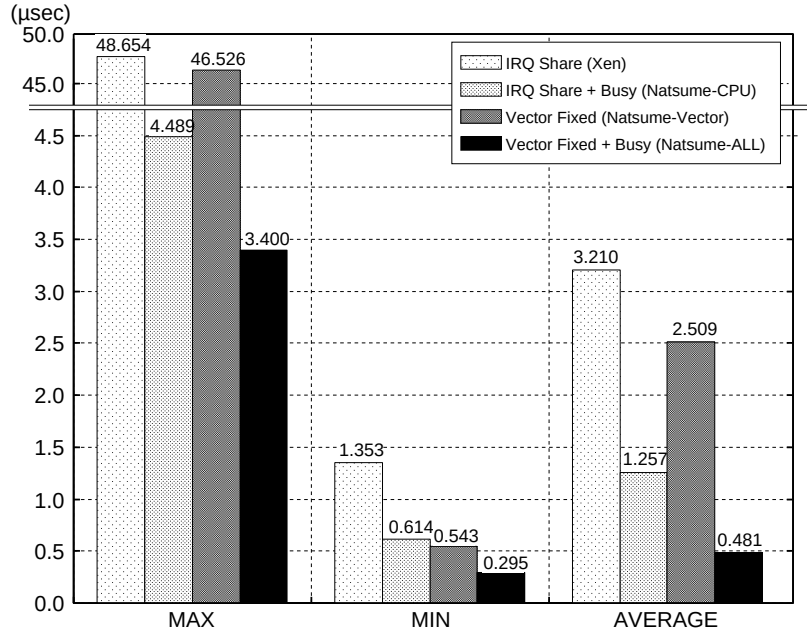

図 7 性能評価結果 (統計)

Natsume-CPU と Natsume-ALL において，処理時間のゆらぎが約 50%抑制されているという結果になった．また，図 8 と**図 10** より，Xen と Natsume-Vector において，非常に大きなクロックが 1 サンプル計測された．

4.4 考　　察

実験結果から，Natsume(Natsume-ALL) はオリジナルの Xen と比較して，最悪実行時間・平均実行時間共に，約 90%の高速化となり，大きく性能が改善されていると言える．さらに，処理時間のゆらぎも約 50%抑制されており，処理時間の予測可能性という観点からも，リアルタイム性の保証に対して有効であると考えられる．

一方，実装した機能毎の性能改善に着目すると，最小の処理時間すなわちオーバヘッドに関する性能改善は，Natsume ハンドラによるものが大きく，最悪実行時間に関する性能改善は，CPU の再割当てを防止する機構によるものが大きいと考えられる．これは，現状の Natsume ハンドラの実体が，Xen の割込みハンドラを軽量化したものであり，イベントチャ

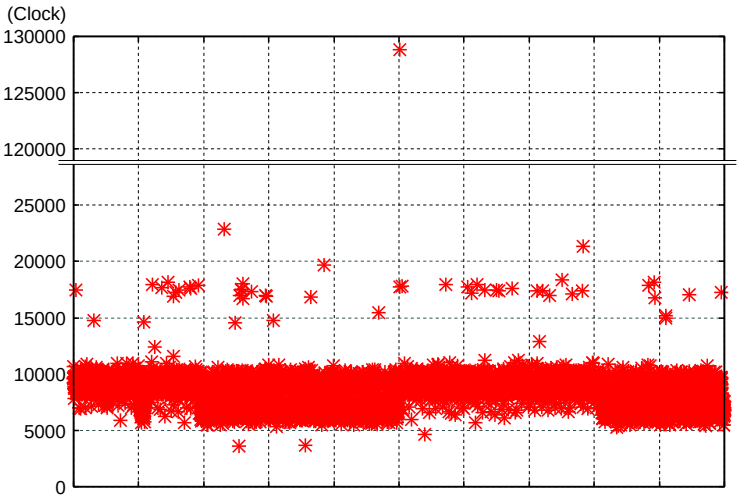

図 8 性能評価用 Xen

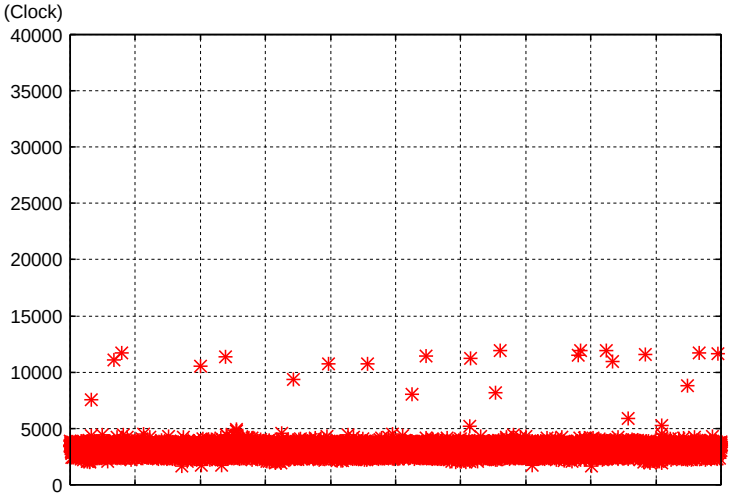

図 9 CPU 再割当て防止機構のみ適用 (Natsume-CPU)

Vol.2011-ARC-195 No.27
Vol.2011-OS-117 No.27
2011/4/14

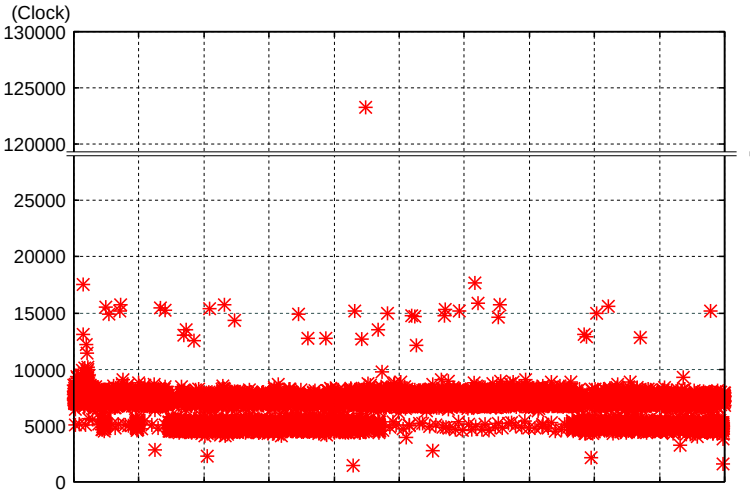

図 10 Natsume ハンドラのみを適用 (Natsume-Vector)

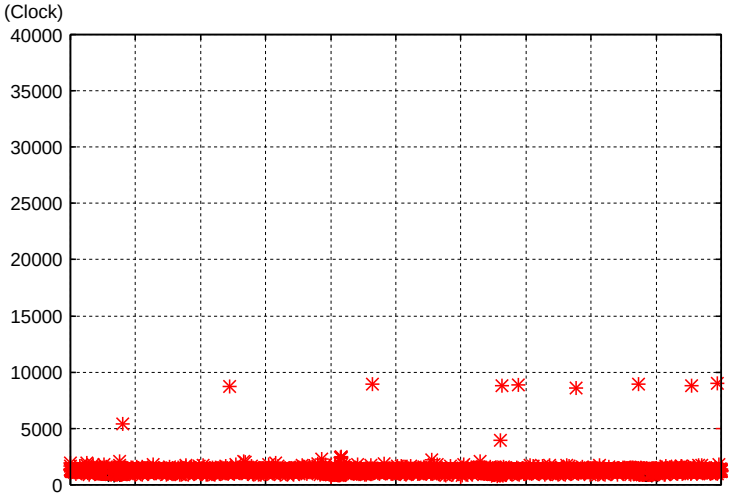

図 11 Natsume ハンドラ＋ CPU 再割当て防止機構を適用 (Natsume-ALL)

ネルから処理を横取りする機構が適用されていないためであると考えられる．この機構を実装することで，さらなる性能改善が可能だと考える．一方，性能評価用 Xen と Natsume ハンドラのみを適用した環境において，1 サンプルだけ非常に大きなクロックを要したものが計測された．これは割込み処理中に，CPU が偶発的に別の処理へ遷移したためと考え，現在詳細な調査を行なっている．

5. 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 や SPUMONE があげられる．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 の例として，RTX があげられる．

RTS-Hypervisor(以下，RTS) は，Intel VT[16] 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 の構造を**図 12** に示す．RTS ではあらゆる資源をゲスト OS に対して排他的，かつ直接割当てることで，デバイスドライバからの直接操作を可能にする．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，レイテンシをほぼ 0 に抑え，RTOS によるリアルタイム性の保証を可能にする．しかし，RTS はその仕組み上，動作するゲスト OS に対して，全ての計算機資源を用意する必要がある．これは，既存の解決手段である，複数の計算機を搭載する手法とほぼ同等のハードウェア開発コストが必要になる．一方，Natsume は高機能 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であり，ハードウェア開発コストを削減することができる．

SPUMONE は，準仮想化型の軽量な VMM である．SPUMONE の構造を**図 13** に示す．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．そして，各ゲスト OS に優先度を付加し，優先度の高いゲスト OS から順に実行する．SPUMONE は，割込みをゲスト OS 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，SPUMONE は CPU 資源以外の仮想化を行わないため，ゲスト OS のメモリ保護を行わない．また，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．Natsume は，既存の VMM である Xen を基にしているため，ゲスト OS 間のメモリ保護や，リアルタイム性の保証に重要でない資源をゲスト OS 間で共有することができる．また，SPUMONE の準仮

Vol.2011-ARC-195 No.27
Vol.2011-OS-117 No.27
2011/4/14

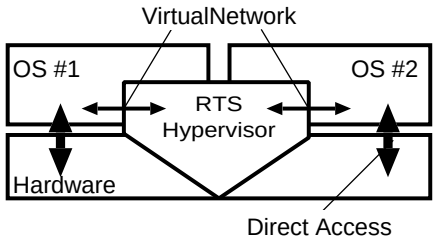

図 12 RTS-Hypervisor の構造

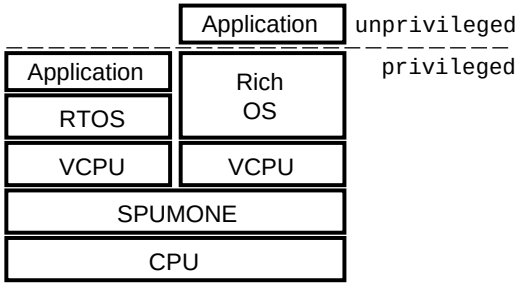

図 13 SPUMONE の構造

想化は，ゲスト OS の修正が少ないという特徴がある．Natsume では，既に VMM として実績のある Xen を基にしており，既に Xen に対応済みである豊富なソフトウェア資産をそのまま利用することができる．

RTX は，WindowsNT・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．リアルタイムタスクのスケジューラは，Windows のプロセススケジューラとは独立して提供され，全てのリアルタイムタスクが終了，もしくは待ち状態になるまで，Windows タスクは実行されない．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 を利用できないため，Linux のようにオープンなプラットフォームが提供する豊富なソフトウェア資産の活用が容易ではない．Natsume は，VMM を利用して RTOS と高機能 OS を独立した形で提供するため，豊富なソフトウェア資産を有効活用できる．

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 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 向け割込み通知モデルの設計と実装を行い，性能評価用 Xen と Linux を用いた性能評価を行った．その結果，Natsume の基である Xen と比較して最悪実行時間で 95%，平均実行時間で 85%の性能改善を確認し，処理時間のゆらぎが約 50%改善されていることを確認した．

今後，現在未実装であるイベントチャネルから処理を横取りする機構の実装を進め，実際の RTOS を用いた評価を行う予定である．

参 考 文 献

1) TRON Association. μITRON4.0 Specificastion. 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) SYSGO. PikeOS RTOS and Virtualization Concept. http://www.sysgo.com/products/pikeos-rtos-technology/, 2011.
4) 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. 1–139 – 1–140, 1999.
5) Bill Carpenter, Mark Roman, Nick Vasilatos, and Myron Zimmerman. The RTX Real-Time Subsystem for Windows NT. *In Proceedings of the USENIX Windows NT Workshop*, pp. 33–37, 1997.
6) Texas Instruments. OMAP Technology. http://focus.ti.com/general/docs/gencontent.tsp?contentId=46946, 2011.
7) ARM. The ARM Cortex-A9 Processors white paper. http://www.arm.com/files/pdf/ARMCortexA-9Processors.pdf, 2009.
8) ARM. Cortex-A15 Processor. http://www.arm.com/products/processors/cortex-a/cortex-a15.php, 2011.
9) Real-TimeSystems GerdLammers. Embedded Real-Time Virtualization and Partitioning. *Small Form Factor Boards Conference*, 2009.
10) Wataru Kanda, YuYumura, Yuki Kinebuchi, Kazuo Makijima, and Tatsuo Nakajima. SPUMONE: Lightweight CPU Virtualization Layer for Embedded Systems. *Embedded and Ubiquitous Computing, IEEE/IFIP International Conference on*, 2008.
11) Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the Art of Virtualization. *ACM Symposium on Operating Systems Principles*, 2003.
12) William von Hagen. *Professional Xen Virtualization*. Wiley Pubnlishing.Inc, 2008.
13) PCI-SIG. *PCI Local Bus Specification*, Revision 3.0 edition, aug 2002.
14) FreeSoftware Foundation. GNU GRUB. http://www.gnu.org/software/grub/

Vol.2011-ARC-195 No.27
Vol.2011-OS-117 No.27
2011/4/14

`index.html`, 2011.

15) Intel Corporation. *82093AA I/O ADVANCED PROGRAMMABLE INTERRUPT CONTROLLER (IOAPIC)*, may 1996.

16) Gil Neiger, Amy Santoni, Felix Leung, Dion Rodgers, and Rich Uhlig. Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization. *Intel Technology Journal*, Vol. Volume.10, No. Issue.03, 2006.