2008－OS－107 (2)
2008／1／30

仮想計算機モニタを利用した ゲストOSの入出力要求監視手法

金城 聖† 荒木 裕靖† 元濱 努†† 片山 吉章†† 毛利 公一†††
† 立命館大学大学院理工学研究科 †† 三菱電機株式会社情報技術総合研究所
††† 立命館大学情報理工学部

仮想化技術を搭載したプロセッサの登場により，仮想計算機モニタ (VMM) の研究が盛んになっている．既存の VMM は，ゲスト OS に対する資源が起動時に割り当てられるなど固定的である．そこで，我々は動的な資源割当てを実現する協調型 VMM の研究を行っている．協調型 VMM とは，ゲスト OS の動作状況を監視し，ゲスト OS の負荷や資源利用状況に基づき，ゲスト OS と協調して計算機資源の割当てを実現する VMM である．協調型 VMM を用いることで，セキュリティの向上，リアルタイム処理の実現，動的な負荷分散，効率的なデバッグが可能となる．協調型 VMM における動作監視では，動作状況解析に基づいた資源割当てを目的としているため，ゲスト OS の動作全体を監視する必要がある．本論文では，協調型 VMM を実現する上で重要となるゲスト OS の動作監視のうち，特にゲスト OS の入出力要求監視について述べる．

A Monitoring Method for I/O Requests from Guest OSes by Virtual Machine Monitor

Akira KANASIRO† Hiroyasu ARAKI†
Tsutomu MOTOHAMA†† Yoshiaki KATAYAMA†† Koichi MOURI†††
†Graduate School of Science and Engineering, Ritsumeikan University
††Mitsubishi Electoric Corporation
†††College of Information Science and Engineering, Ritsumeikan University

Today, researches of Virtual Machine Monitor (VMM) is active because processors equipped with Virtualization Technology appeared. Existing VMM allocates computational resources to a guest OS fixedly, when the guest OS boots. So, We research on cooperative VMM to achieve dynamic resource allocation. Cooperative VMM monitors guest OS's behavior, and allocates resource to them dynamically according to analysis of the behavior. By cooperative VMM, security is improved, realtime processing is achieved and a dynamic load-balancing and efficient debugging become possible. Purpose of monitoring guest OS's behavior in cooperative VMM is to achieve resource allocation based on analysis of behavior. Therefore it is necessary to monitor guest OSes entirely. In this paper, a monitorig method for behavior of guest OSes, especially I/O request from guest OSes, is described.

1 はじめに

近年，プロセッサのマルチコア化や，仮想化技術を搭載したプロセッサの登場によって，新たな計算機システムの模索が可能となっている．仮想化技術を利用したソフトウェアの代表例として，Xen [1] や VMware [2] などの仮想計算機モニタ (VMM; Virtual Machine Monitor) が挙げられる．VMM は，物理的な計算機資源を仮想化し，仮想プロセッサや仮想メモリ，仮想入出力デバイスなどの資源を備えた環境を構築することで，複数のゲスト OS を同時に動作させることを可能とするシステムソフトウェアである．VMM 上のゲスト OS は，VMM の存在を意識するこ

となく動作することができる．我々は，この仮想化機能を用いることで，ゲストOSに対して，必要な資源を適応的に割り当て，システム全体として有限な資源を効率的に使う方式として協調型VMMの研究を進めている．協調型VMMは，これを実現するために，

1. VMMによるゲストOSの動作状況監視
2. 監視結果に基づく適切な資源量の割当て
3. 資源量の変化に適応可能なゲストOSにおける資源管理機能

が必要となる．本論文では1の動作状況監視手法のうち，現在検討している入出力動作の監視手法について述べる．

OSの動作監視は，ソフトウェアやOSの開発やデバッグなどに用いられてきた．従来のOS動作監視手法には，命令トレーサのように監視対象OS内部に監視機構を設置することで実現する手法や，ICE (In-Circuit Emulator) のような専用の高価なハードウェアを利用することで実現する手法がある．これらの手法は，短時間，かつ限られた範囲の監視で大きな効果を得ることが可能である．しかし，動作中のOSの監視を行う上では，オーバヘッドが大きく，監視対象の動作に大きな影響を与えたり，ハードウェア的に監視対象が制限されるといった問題がある．これらの問題を解決するためにVMMを利用する．VMMは，ゲストOSに対してVM資源を提供する．VMMは，必要であれば，ゲストOSの処理をフックして，その内容をチェックしたり，ゲストOS間の調整をしたり，エミュレーションしたりする．本論文で提案する手法では，このフックする段階で監視することで，ゲストOSに変更を加えることなく，ゲストOSの動作に影響を与えない監視を実現することを目的としている．

以下，本論文では，2章で協調型VMMの概要について述べ，3章で既存のVMMであるXenの入出力の実現法について述べる．4章でその仕組みを利用した入出力要求可視化手法について，5章で関連研究について，6章で今後の検討事項について述べ，7章でまとめとする．

2 協調型仮想計算機モニタ

既存のVMMは，VM構築の段階で計算機資源を固定的に割り当てている．そのため，ゲストOSの処理特性や，負荷に適応した資源の割り当てが実現されていない．協調型VMMは，ゲストOSの動作状況に応じて動的に計算機資源をVMに割り付けるVMMである．図1に示すように，協調型VMMでは，ゲストOSの監視によって，入出力デバイスに対する入出力要求，各VMに対して実プロセッサを割り付けるドメインスケジューラなどから現在のゲストOSの情報を取得し，資源利用状況を把握する．さらに，資源利用状況からゲストOSが必要とする資源を

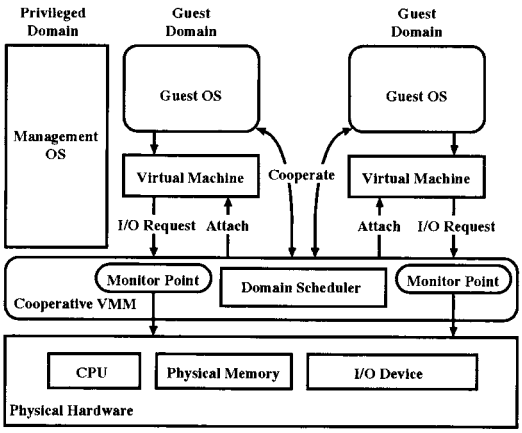

図1 協調型仮想計算機モニタ

分析し，VM資源を割り付ける．一方で，使用されていない資源についてはそれを取り上げて，別のVMに割り当てる．これによって，より効率の良い計算機資源の利用が可能となる．このような協調型VMMは，OSのデバッグに加え，負荷分散，システムのセキュリティの向上，さらにリアルタイム処理の実現の場面で有用である．

デバッグ

ゲストOSからの入出力要求は，全てVMMが処理を行うため，ゲストOSからの入出力要求を全て監視することが可能である．この特徴を利用することで，ゲストOSからの入出力要求が正しいものであるかどうかを判別したり，特定の要求が来ると停止させたり，値を変更したりすることが可能である．そのため，ICEデバッガなどを利用しなければできなかったデバイスドライバの開発・デバッグがOSの動作を動的に監視しながら行うことが可能となる．

負荷分散

複数のゲストOSが動作する環境では，実計算機上のプロセッサを複数のゲストOSで共有しており，VMM内部においてゲストOSに対するプロセッサの割当てをスケジューリングすることで複数のゲストOSを動作させている．そのため，現在のゲストOSによる実プロセッサの使用率を知ることが可能である．また，ゲストOSの負荷状況を動的に入手できるため，ゲストOS上で動作するアプリケーションの特性を把握することも可能である．これを元に，CPU利用率の低いゲストOSへの割当てを減らしたり，逆に負荷の高いゲストOSへの割当てを増やしたりすることができ，負荷を分散できる．

セキュリティの向上

ネットワークを経由して受信されるデータは，最初に VMM が受信する．VMM は，そのデータを最終送信先である仮想計算機に送信する．一方で，仮想計算機側からのデータの送信は，VMM を介して外部ネットワークへと送信される．VMM 内部においてネットワークに関する入出力を監視し，受信したデータを分析することでウイルスやワームなどによる悪質なデータの送受信を検出し，それに起因するトラブルを未然に防ぐことが可能である．

リアルタイム処理

現在の VMM では，仮想計算機上でリアルタイム OS の動作を想定しておらず，そのリアルタイム性を保証していない．その原因としては，ゲスト OS がリアルタイム性を必要とする処理を行う際に CPU が割り付けられるようなドメインのスケジューリングがされていないことや，入出力処理に関するオーバヘッドが挙げられる．しかし，ゲスト OS の監視を行うことで，現在必要としている資源を把握し，それをゲスト OS に割り付けることが可能となる．またゲスト OS 間での優先度を考慮することもできる．これによって，ゲスト OS のリアルタイム要求を満たすことが可能となる．

本論文では，以下，資源利用状況に関わるゲスト OS の入出力に焦点をあて，その入出力分析方法の検討と入出力要求の可視化手法について述べる．

3 Xen における入出力仮想化の実現法

3.1 Xen の概要

協調型 VMM は，既存の VMM である Xen に変更を加えることで実現する．Xen は，オープンソースソフトウェアとして開発されている．Xen では，CPU の仮想化技術 (Intel VT-x や AMD Virtulization) を利用して完全仮想化された VM 環境の構築が可能である [3][4]．完全仮想化された VM 上では，ゲスト OS に変更を加えることなく動作させることが可能である．

Xen は，VM をドメインという単位で管理している．ドメインには，完全仮想化ゲストドメイン (domHVM) と特権ドメイン (Dom0) の 2 種類がある (図 2 参照)．以下に，各ドメインの特徴を示す．

- 完全仮想化ゲストドメイン
  ゲスト OS が動作する VM．ゲストドメイン上では，変更を加えられていないゲスト OS が動作する．ゲスト OS からは実計算機と同様のものとして認識され，VMM の存在が完全に隠蔽された状態で動作する．そのため，ゲスト OS はそれ本来の動作をする．すなわち，VMM の動作はゲスト OS の動作に影響を与えない．

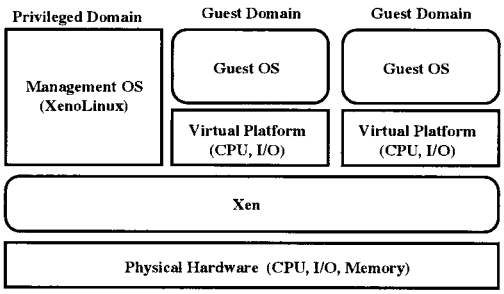

図 2　Xen の構成

- 特権ドメイン
  ゲストドメインの生成，削除，起動，終了を行う管理用 VM．特権ドメインは Xen に対してハイパーバイザコールを発行することで管理を行う．そのため，ゲストドメインにはハイパーバイザコールが利用可能となるように変更を加えられた Linux 2.6 カーネルベースの OS (XenoLinux) が利用されている．また，特権ドメインではデバイスモデルというデバイスエミュレーション機構が動作しており，ゲストドメインからの入出力要求は，全てこの機構によって処理される．

3.2 入出力の仮想化

Xen の入出力仮想化処理は，仮想プラットフォーム，共有ページ，イベントチャネル，デバイスモデル (ioemu) の機構を利用して実現されている．以下に，各機構の詳細を示す．

- 仮想プラットフォーム (Virtual Platform)
  ゲストドメインに対して提供する計算機ハードウェアを完全仮想化したレイヤである．ゲストドメイン上のゲスト OS からは，実計算機ハードウェアと同様のものとして認識される．入出力については，ゲスト OS で実行される入出力に関する命令を Xen の内部形式へ変換し，さらに処理結果を取得し，ゲスト OS に対して仮想割込みを発行する機能を実現している．

- 共有ページ (Shared Page)
  各ゲストドメインの仮想プラットフォームと，特権ドメイン間において入出力要求と処理結果の相互通信を行うためのメモリ空間．Xen 内部と後述のデバイスモデルからでは，参照の方法が異なる．詳細は，4 章で述べる．

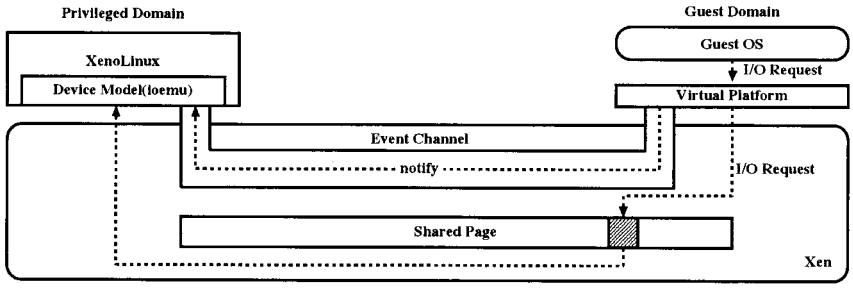

図3　入出力処理要求の流れ

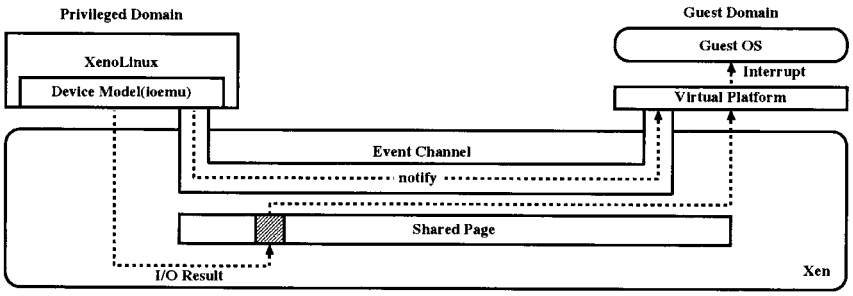

図4　入出力処理結果の流れ

- イベントチャネル (Event Channel)
  ドメイン間でのシグナリング機構．入出力要求に関しては，共有ページへ要求・結果が書き込まれた際に，それを送信先のドメインへ通知するために用いられる．
- デバイスモデル (Device Model)
  特権ドメイン上の管理用 OS (XenoLinux) でユーザプロセスとして動作するデバイスエミュレーション機構．ゲストドメインからの入出力要求の処理を行う．デバイスモデルは，計算機シミュレータ QEMU [5] の機能を利用して実現されている．

Xen は，入出力要求処理に Intel VT-x [6] を利用している．VT-x ではゲスト OS が動作するモード (VMX non-root) と VMM が動作するモード (VMX root) が用意されている．特権を必要とする処理は VMX root で実行される．VMX non-root から VMX root への状態遷移を VMExit といい，VMX root から VMX non-root への状態遷移を VMEntry という．入出力要求の処理は VMExit と VMEntry によって実現されている．図3の矢印は，入出力処理要求の流れを示しており，図4の矢印は入出力処理結果の流れを示している．以下に，詳細な手順を示す．

1. ゲスト OS が入出力要求を発行すると，VMExit が起こり，制御が Xen に移行する．ここで仮想プラットフォームを実現している Xen は，入出力要求を Xen の内部形式へ変換する．変換された入出力要求は共有ページに書き込まれ，イベントチャネルを利用して，特権ドメインに書き込みを通知する．
2. 特権ドメインは，書き込み通知を受け取ると，イベントハンドラがデバイスモデルを呼び出し，共有ページから要求を取り出す．取り出された要求に対応する入出力は，デバイスモデルによって実現されているデバイスエミュレーションによって処理される．
3. 入出力処理の結果は，再びデバイスモデルによって共有ページに書き込まれ，イベントチャネルを利用して，ゲストドメイン側の仮想プラットフォームに対して，書き込みを通知する．
4. 仮想プラットフォームは通知を受け取ると，共有ページから処理結果を取り出し，ゲスト OS に対して仮想割込みを発生させる．これによって，VMEntry が起こり，制御が Xen からゲスト OS へと移行する．

4　入出力要求の監視と可視化

4.1　入出力要求可視化方式

入出力要求可視を監視するにあたり，まずは，可視化を

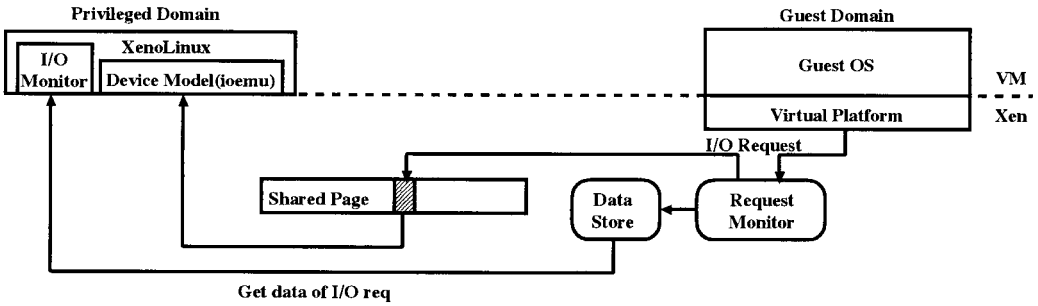

図 6　入出力要求可視化方式

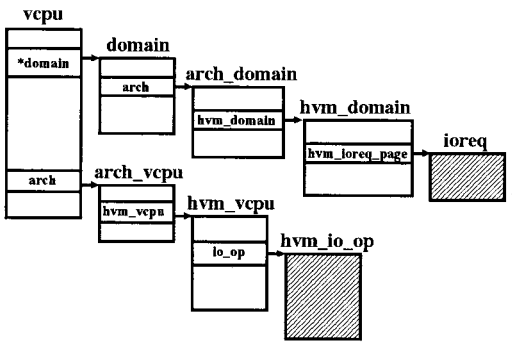

図 5　仮想 CPU 構造体

行った．可視化では，表示する内容として，入出力要求を発行した仮想 CPU，仮想 CPU を所有するドメイン，入出力要求の原因となった命令，その命令が発行された時点でのレジスタの値が挙げられる．これらは協調型 VMM を実現する場合に必要となる情報である．

ゲスト OS からの入出力要求は，Xen を介して，特権ドメイン上で動作するデバイスモデルへ伝えられ，処理が行われる．本手法では，この処理をフックし，要求の情報を取得することで，どのような入出力要求をゲスト OS が発行しているのか監視する．Xen 内部で扱われる入出力要求に関する情報は 2 種類あり，ゲスト OS から発行された入出力に関する命令の情報 (hvm_io_op) と，それを基に生成された入出力要求 (ioreq) である．それぞれ仮想 CPU(vcpu) 構造体によって管理されている (図 5 参照)．vcpu 構造体は，仮想 CPU のアーキテクチャ情報 (arch_vcpu)，仮想 CPU の構成 (hvm_vcpu) を管理しており，また，その仮想 CPU の所属するドメイン (domain) の情報も保持しており，ドメインの情報には，ドメインのアーキテクチャ情報 (arch_domain)，ドメイン構成 (hvm_domain) に関する情報がある．hvm_domain と hvm_cpu は，完全仮想化されたドメインと仮想 CPU に関する情報を管理している．本方式では，hvm_io_op と ioreq の両方の構造体から情報を取得する．

入出力要求の可視化を実現する上で，必要となる機構には次のものがある (図 6 参照)．

- 入出力要求処理フック機構 (Request Monitor)
  本来の Xen の動作では，ゲスト OS からの入出力要求は，Xen 内部でデバイスモデルで扱える形式に変換され，共有ページに書き込まれる．フック機構では，この処理をフックし，ゲスト OS の入出力要求から情報を取得する．取得した情報は，入出力要求ストア領域に格納される．

- 入出力要求ストア領域 (Data Store)
  フック機構によって取得した情報を格納するための領域は，入出力処理には利用せず，可視化のために利用する．

- 可視化ツール (I/O Monitor)
  Data Store に格納された情報を可視化するためのアプリケーション．

4.2　可視化ツール

仮想計算機内部での入出力可視化を実現する上で，まず特権ドメイン上で動作する入出力要求の可視化を実現するプログラムを作成した．入出力要求可視化プログラムは，特権ドメイン上で動作するデバイスモデルの機能を利用して実装を行った．デバイスモデルは，全てのゲストドメインからの入出力要求の処理を行う．

デバイスモデル内では，ゲストドメインからイベントチャネル経由して送信される入出力要求書込みの通知を受信すると，共有ページから入出力要求が取り出され，処理が行われる．可視化プログラムでは，入出力処理要求を取得後に，処理をフックし，ログファイルに書き出す (図 7 参照)．可視化プログラムは，表 1 の環境で実装を行っ

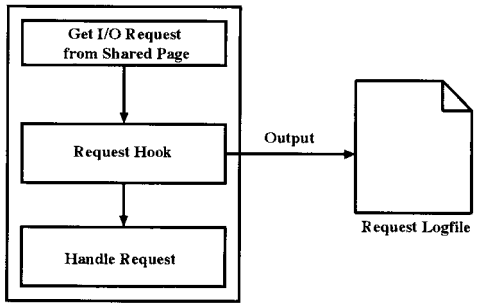

図 7　可視化処理の流れ

表 1　実装環境

CPU	Core 2 Duo T5600
メモリ	DDR2 SDRAM 1GB
OS	Fedora 7
VMM	Xen 3.1

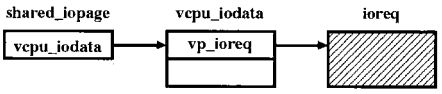

図 8　共有ページ構造体

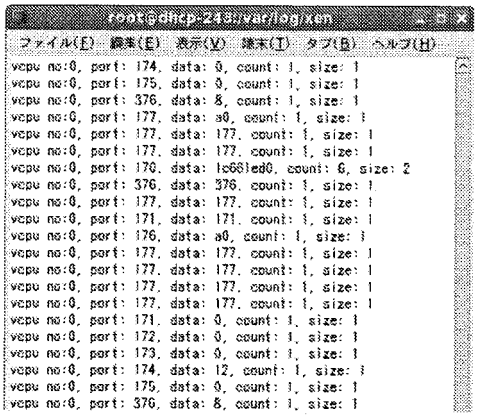

図 9　実行結果

た．デバイスモデルの機能を利用した可視化では，共有ページ (shared_iopage) 上の入出力要求を取得している (図 8 参照)．

現在の実装では，入出力要求に関する情報を shared_iopage から参照できる ioreq の情報を取得して表示している．具体的には，入出力要求を発行している仮想プロセッサ番号，I/O ポート番号，データ，カウント，データサイズの表示を行った (図 9 参照)．shared_iopage から取得できる情報は，図 5 の ioreq から取得できる情報と同じである．

5 関連研究

5.1 OS の動作状況取得手法

本論文では，協調型 VMM を実現するための OS の入出力処理動作観測について述べた．既存の OS の動作状況観測手法には，命令トレーサ鶴 [7] を利用する手法や，Xenoprof [8] を利用する手法がある．鶴を利用する手法では，監視対象 OS の内部に命令トレーサ機構を実装することで実現している．鶴は，OS 上で動作するアプリケーションの挙動を知ることを目的として開発された．アプリケーションの挙動を知るために，アプリケーションが処理を行っている間に取得する命令のログを解析し，パフォーマンスチューニングに利用したり，デバッグを行うために有利な情報を取得する．動作状況の取得は静的に行われる．Xenoprof は，Xen を利用したゲスト OS の動作状況取得手法である．Xenoprof では，準仮想化によって構築されたゲスト OS の静的性能評価が実現されている．Xen に Xenoprof 専用のハイパーバイザコールが追加されており，ゲスト OS はそのハイパーバイザコールを利用して，実 CPU のパフォーマンスモニタカウンタから値を取得する．Xen を介することによって発生するオーバヘッドを計測することで，Xen 上で動作するゲスト OS の性能評価を行っている．上記の手法は，観測対象 OS，または，その上で動作するプロセスから動作状況取得要求が発行し，性能評価に利用するデータや値を取得する．また，観測対象に観測データ取得のための変更が加えられている．本論文で提案した方式では，観測対象に変更を加えることなく，動的の観測データの取得を行うため，観測対象に与える影響を軽減できる．

5.2 複数 OS の共存システム

本論文では，複数のゲスト OS が動作する環境での資源の有効利用方法として，協調型 VMM を提案した．既存研究の中にも，豊富な計算機資源の有効利用方法として複数の OS 共存させて動作可能とすること目的とした研究が盛んである．三菱電機のマイクロクラスタリング OS [9][10] や日立の DARMA [11] などがその例として挙げられる．マイクロクラスタリング OS 環境は，シングルチップマルチプロセッサ上での動作を想定しており，各プロセッサコア上でリアルタイム OS と汎用 OS を独立に動作させること目的としている．DARMA は，DARMA ナノカーネルという仮想計算機モニタを利用した OS 共存システムである．DARMA の特徴としては，各 OS が独立して

利用可能である資源を資源分割機能によって実現している点である．また，OSの障害検知機能も備えている．本論文で提案した協調型VMMは，ゲストOSの動作を監視することでリアルタイムOSの動作の保証や，負荷分散を行うことを目指している．

6 今後の検討事項

協調型VMMを実現するには，図6に示してあるフック機構やストア領域以外に取得した入出力要求を分析し，各ドメインの要求ごとに優先度の設定を行う機構としての入出力要求分析機構や，入出力要求分析機構が設定した優先度に基づいたドメインスケジューリングを行うためのドメインスケジューラに対するフィードバック機構が必要となる．以下に，今後の検討事項を示す．

1. 入出力要求監視
   取得した入出力要求がどのドメインのどのCPUで発生したのかを判断するための方法の検討と，取得した要求を管理するための領域とデータ構造の決定が必要である．
2. 入出力要求分析
   監視結果を利用した資源割り付けを行うには，取得した要求の分析結果を基に，ドメイン間の優先度付けを行う必要がある．そのため，優先度付けを目的とした入出力要求の分析方法の検討が必要である．
3. 再スケジューリング
   入出力要求分析によって設定された優先度をもとに，ドメインの再スケジュールを行う必要がある．そのため，優先度情報を利用可能とするためにドメインスケジューラの改良が必要である．

7 おわりに

本稿では，ゲストOSの監視機構として利用するXenの入出力処理について述べ，ゲストOSの入出力監視手法について述べた．現在の進捗としては，ゲストOSの入出力要求を確認するために特権ドメインに対し送信されている要求を取得するプログラムを作成し，実際に入出力要求の情報が取得可能であることを確認した．具体的には，図9に示したように，入出力要求を発行している仮想プロセッサ番号，I/Oポート番号，データ，カウント，データサイズの表示を行い，ゲストOSでLinuxを起動させ実験した．今後は，6章で述べた内容について進めていく予定である．

参考文献

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfield: "Xen and the Art of Virtualization," In Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, pp. 164–177 (2003).

[2] VMware.Inc: VMware, http://www.vmware.com/.

[3] David Chisnall: "The Definitive Guide to the Xen Hypervisor," Prentice Hall Open Source Software Development Series (2007).

[4] Intel Corporation:Intel Virtualization Technology: "Extending Xen with Intel Virtualization Technology," Intel Technology Jounal, 2006, Volume 10, Issue 3, pp. 193–204 (2006).

[5] QEMU open source processor emulator, http://fabrice.bellard.free.fr/qemu/.

[6] Intel Corporation: "Intel Virtualization Technology Specification for the IA-32 Intel Architecture," http://www.intel.com/technology/ (2005).

[7] 森本 洋行, 小宮山彰一郎, 毛利公一, 吉澤康文 : "性能評価のための命令トレーサの開発," 電子情報通信学会論文誌 Vol. J84-D-I No. 6, pp. 584–pp.593 (2001).

[8] Aravind Menon, Jose Renato Santos, Yoshio Turner, and G. (John) Janakiraman, Willy Zwaenepoel: "Diagnosing Performance Overheads in the Xen Virtual Machine Environment," First ACM/USENIX Conference on Virtual Execution Environments (VEE '05), pp. 13–23 (2005).

[9] 遠藤 幸典, 菅井 尚人, 山口 義一, 近藤 弘郁: "シングルチップマルチプロセッサ上のハイブリッドOS環境の実現 –システムアーキテクチャ–," 第66回情報処理学会全国大会講演論文集, Vol.1, pp. 9–10 (2004).

[10] 菅井 尚人, 遠藤 幸典, 山口 義一, 近藤 弘郁: "シングルチップマルチプロセッサ上のハイブリッドOS環境の実現 –OS間インタフェースの実装–," 第66回情報処理学会全国大会講演論文集, Vol.1, pp. 11–12 (2004).

[11] 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," 第59回情報処理学会全国大会講演論文集, Vol.1, pp. 139–140 (1999).