情報処理学会論文誌 Vol. 50 No. 1 394–408 (Jan. 2009)

軽量仮想計算機モニタを利用したOSデバッガのロギング&リプレイ機能の提案

竹内 理[†1] 坂村 健[†2]

汎用 PC サーバに専用 OS を搭載し I/O を高性能化するアプライアンスサーバの普及と，高速 I/O デバイスの頻繁な技術革新にともない，専用 OS 向けデバイスドライバの開発効率向上，特に開発効率の良いデバイスドライバの開発環境（OS デバッガ）への期待が高まりつつある．従来の PC サーバ向け OS デバッガは，(1) OS 異常動作時の安定稼働保証，(2) 多様な I/O デバイスのドライバ開発への適用可能性，(3) ロギング&リプレイ機能（実行履歴を保存しながら実行を行い，再現性の低いバグが顕在化した際に当該実行の再現を行う機能）への対応，の同時実現が難しく，開発効率向上を阻害していた．本研究では，上記課題を解決する OS デバッガの実現を目指す．すでに (1)，(2) の課題は，筆者らが軽量仮想計算機を用いた OS デバッガを実装し解決した．そこで，上記 OS デバッガへのロギング&リプレイ機能の実装方式を新規に提案することで (3) の課題の解決を目指す．そして，実装したロギング&リプレイ機能の実行オーバヘッドの性能評価を行い，その現実性を評価する．実装・評価を行った結果，実装方式は，既存の仮想計算機モニタ（VMWare）と比べて，38%程度の性能低減でネットワークの受信処理（および当該処理のロギング処理）が行えること，およびその際のログ格納領域サイズの増大レートは，ネットワーク受信レートとほぼ同等のレートに抑えられることが明らかになった．

Logging and Replay Method for OS Debugger Using Lightweight Virtual Machine Monitor

Tadashi Takeuchi[†1] and Ken Sakamura[†2]

Recently, the number of appliance PC servers that realize high performance I/O by utilizing proprietary OS has been increasing. Besides, I/O technology innovation cycle has been shortening. As a result, efficient debugging environments for proprietary OS device drivers are expected by many vendors. However, conventional OS debugging environment on PC servers cannot improve the driver development efficiency so much, because they have not been able to satisfy the following demands at the same time: (1) debugging environment stability, (2) wide applicability that enable debugging for any I/O device drivers, (3) easiness to support logging and replay functionality (the functionality that records logs during OS execution and replay the execution by using the logs when a timing critical bug appears). In this paper, we propose a novel OS debugging environment to improve the efficiency of device driver development. We have already implemented an OS debugging environment using Lightweight Virtual Machine Monitor to solve the problem (1) and (2). We solve the problem (3) by proposing a novel logging and replay implementation methods for our debugging environments. Besides, we will evaluate the practicability of the methods by measuring the logging and replay performance. We implemented and evaluated the logging and replay functionality and found that network packet receive performance of the method is only 38% below that of conventional virtual machine monitor (VMware). We also found that logging memory region size is increasing only at the nearly same rate as network packet arrival.

1. はじめに

近年，汎用 PC サーバ（PC/AT 互換機）上に自社で開発した専用 OS やドライバ搭載し，高い I/O 性能を特徴に持たせたアプライアンスサーバが普及してきている．Streamonix 社の SX.Streamer[1)]，日立の Videonet.tv[2)]，Kesenna 社の High Performance Network Driver[3)] などはその例である．また一方で，ネットワーク技術の急激な進歩にともない，10 Gbps Ethernet や 8 Gbps FC HBA のような新規高速 I/O デバイス，Intel 社の I/OAT[4)] などの高速 I/O を支援する新技術が頻繁に出現しつつある．この結果，専用 OS の開発において，デバイスドライバの開発工数の占める比率が増大しつつあり，デバイスドライバの開発効率を上げられる OS デバッガのニーズが高まりつつある．

PC/AT 互換機には，ICE などのデバッグを支援するハードウェアが提供されていない．そのため，一般に以下のいずれかのソフトウェアデバッガを用いてデバイスドライバを開発する．

(1) リモートソフトウェアデバッガ[5)]
(2) OS 組み込みのソフトウェアデバッガ[6)]
(3) ハードウェアシミュレータ（もしくは仮想計算機モニタ）と連携して動作するソフトウェアデバッガ[7)–9)]

†1 株式会社日立製作所システム開発研究所
Hitachi Ltd., Systems Development Laboratory
†2 東京大学大学院学際情報学府
Tokyo University Graduate School of Interdisciplinary Information Studuies

しかし，(1)，(2) は，デバッグ対象 OS の異常動作時にもデバッガの安定稼動が保証できず，安定性に課題がある．また，ロギング&リプレイ機能[10]（OS 実行履歴を保存しながら実行を行い，再現性の低いバグが顕在化した際に当該実行の再現を行う機能）への対応が困難，という課題もある．一方，(3) は，仮想計算機モニタでエミュレートできる I/O デバイスの種類に限りがあり，新規 I/O デバイスのドライバ開発への適用が困難，という課題を持つ．

本研究では，(1) OS 異常動作時の安定稼動を保証する安定性，(2) いかなる I/O デバイスドライバの開発にも適用できる汎用性，(3) ロギング&リプレイ機能への対応，の 3 つを同時に実現可能な OS デバッガの実現を目標とする．すでに (1)，(2) の課題については，筆者らが軽量仮想計算機モニタを用いた OS デバッガ[11] を提案し，課題を解決している．そこで本研究では，この OS デバッガ向けのロギング&リプレイ機能の実装方式を新規に提案し，(3) の課題を解決する．軽量仮想計算機モニタを用いた OS デバッガでは，デバッグ対象の OS が NIC や HBA などのハードウェアに直接アクセスを行う．既存のロギング&リプレイ機能の実装方式は，すべての I/O 動作をエミュレートする既存のハードウェアシミュレータ（もしくは仮想計算機モニタ）を仮定しているため，その実装方式の適用は難しい．そこで，軽量仮想計算機モニタによるデバイスレジスタアクセスや DMA 転送などの履歴の保存・再生の実現を目指し，軽量仮想計算機モニタの起動契機付与方式，およびデバッグ対象 OS の直接アクセス動作の軽量仮想計算機モニタによる追跡方式を新規に設計・実装する．さらに，実装方式の性能評価を行い，提案方式が現実的なオーバヘッドでロギング&リプレイ機能を実現できることを定量的に示す．

以下，2 章では，すでに提案している軽量仮想計算機モニタを用いた OS デバッガの概要について説明する．3 章では，本研究で追加しようとしているロギング&リプレイ機能の概要について説明する．4 章では，軽量仮想計算機モニタを用いた OS デバッガにロギング&リプレイ機能を実現するための課題を示し，デバイスレジスタアクセス履歴の保存・再生処理におけるデバイス依存処理の分離・極小化方式，および DMA 転送履歴の保存・再生の全体実現方式の新規設計が必要であることを示す．5 章では，新規に提案する VM 層デバイスドライバを用いたデバイス依存処理の極小化・分離方式，および DMA 転送履歴の保存・再生の実現方式について説明する．6 章では，本研究で提案する OS デバッガの実装の概要について説明する．7 章で性能評価結果を示し，提案方式での履歴の保存・再生で生じるオーバヘッドを定量的に示す．8 章で提案した OS デバッガの利点/欠点と適用範囲を考察した後，最後に 9 章で関連研究について述べる．

2. 軽量仮想計算機モニタを用いた OS デバッガ

本章では，(1) OS 異常動作時の安定稼動を保証する安定性，(2) いかなる I/O デバイスドライバの開発にも適用できる汎用性を実現できる OS デバッガとして筆者らがすでに提案した，軽量仮想計算機モニタを用いた OS デバッガの概要について説明する．本デバッガの構成を**図 1** に示す．

本デバッガは従来のリモートデバッガの改善であり，従来同様，ホストマシン上にソフトウェアリモートデバッガが，ターゲットマシン上にデバッグ対象の OS およびスタブが動作する．リモートデバッガとスタブ間でデバッグコマンド（ターゲットマシンのレジスタ/メモリの参照/更新コマンドなど）を送受することでデバッグを実行する．しかし，従来のリモートデバッガとは異なり，スタブが使用するハードウェア資源（割込みコントローラ，タイマ，ページテーブル，割込みベクタテーブルなど）のみをエミュレートする軽量仮想計算機モニタもターゲットマシン上で動作する．デバッグ対象の OS は，これらのハードウェア資源に軽量仮想計算機モニタを介してアクセスするため，たとえ OS がバグにより異常動作を行ってもハードウェア資源の正常動作は保証され，リモートデバッグ機能の安定稼動を保証できる．

また一方で OS は，スタブが使用しない NIC や HBA などの I/O デバイスには軽量仮想計算機モニタを介さずに直接アクセスする．このため，軽量仮想計算機モニタはこれらの I/O デバイスのエミュレーションを行う必要はなく，いかなる I/O デバイスのドライバ開発にも本 OS デバッガの利用を可能にしている．また，デバッグ環境下での I/O 実行性能の劣化も防いでいる．

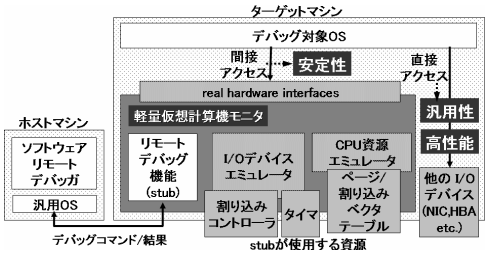

図 1　軽量仮想計算機モニタを用いた OS デバッガ
Fig. 1　OS debugger using lightweight virtual machine monitor.

3. ロギング&リプレイ機能

本研究では，2 章で説明した軽量仮想計算機モニタにロギング&リプレイ機能の追加を目指す．本章では，このロギング&リプレイ機能の概要と機能追加方針について説明する．

3.1 機 能 概 要

ロギング&リプレイ機能とは，デバッグ対象の OS の実行中に，初期状態と実行履歴をログとして残しておき，OS のバグが顕在化した際に，初期状態の回復と当該実行履歴に従った命令レベルでの OS 実行の再生を行う機能である．このような実行再生をサポートすることで，タイミング依存の再現性の低いバグが発生した際にも，何度でも当該バグを再現可能となり，効率的な OS のデバッグが可能になる．

3.2 機能追加方針

本研究では，デバッグ対象の OS をシングル CPU 上で動作する OS に限定する．マルチコア CPU や複数 CPU 上で動作する OS のロギング&リプレイ機能の実現については今後の課題とする．本研究で実装するロギング&リプレイ機能の概要を**図 2** に示す．

シングル CPU 上で動作する OS を命令レベルで再生するためには，初期状態と外部（I/O デバイス）からの入力履歴だけを OS 実行時に保存すればよい．外部（I/O デバイス）への出力などは，初期状態からの命令再実行を再生時に行うだけで実行時と同様の実行を行えるため，履歴として保存する必要はない．本研究で実装するロギング&リプレイ機能では，以下の 4 種類の履歴を保存する．CPU 能力以外の履歴保存帯域の制約（I/O 帯域制約など）をできる限り取り除くため，これらの履歴を主メモリ上に格納する方針をとる．

I/O ポートアクセス履歴　I/O ポートへの read 実行の結果読み込んだ値の履歴

デバイスレジスタアクセス履歴　PCI デバイスのデバイスレジスタへの read 実行の結果読み込んだ値の履歴

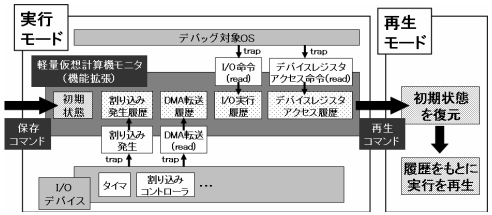

図 2　ロギング&リプレイ機能の概要
Fig. 2　Logging-and-replay functionality overview.

割込み発生履歴　割込み発生タイミングと，発生した割込み番号の履歴

DMA 転送履歴　DMA write 転送の発生タイミングと，転送されたデータの履歴

本研究で実装する OS デバッガからのロギング&リプレイ機能の起動手順は以下になる．

(1) ホストマシン上のソフトウェアデバッガから保存コマンドを実行する．この結果，ターゲットマシン上の OS の初期状態（メモリやレジスタのスナップショット）が保存される．

(2) ホストマシン上のソフトウェアデバッガから実行継続コマンドを実行する．ターゲットマシン上の OS は実行モードで動作を開始する．この間，上記履歴を軽量仮想計算機モニタ内部に保存する．

(3) タイミング依存のバグなどが発生した場合，ターゲットマシン上の OS は例外を発生し，実行を停止する．ホストマシン上のソフトウェアデバッガから回復コマンドを実行すると，OS のメモリ/レジスタ状態が初期状態に回復する．

(4) ターゲットマシン上のソフトウェアデバッガから実行継続コマンドを実行する．今度は，OS は再生モードで動作を開始し，(2) で保存した履歴を参照しながら，(2) での実行を忠実に再現する．そして，(3) と同じタイミングで例外が発生し，実行が停止する．

4. ロギング&リプレイ機能の実現上の課題

本章では，3 章で述べたロギング&リプレイ機能を，2 章で述べた軽量仮想計算機モニタ上に実現する際の課題について述べる．

4.1 解決したい課題

1 章で述べたとおり，本研究の目標は，(1) 安定稼動性，(2) いかなる I/O デバイスドライバの開発にも適用できる汎用性，(3) ロギング&リプレイ機能，を同時に実現する OS デバッガの実装である．そして，2 章で述べた軽量仮想計算機モニタにロギング&リプレイ機能の追加実装することで，(3) の要件を充足させようとしている．

また，3.2 節で述べたとおり，仮想計算機モニタにロギング& リプレイ機能を実装するためには，I/O デバイスからの入力履歴（I/O ポートアクセス履歴，デバイスレジスタアクセス履歴，割込み発生履歴，DMA 転送履歴）の保存・再生の実現ができればよい．

従来の仮想計算機モニタを用いた I/O 処理は，仮想計算機モニタが I/O デバイスのエミュレータ（仮想 I/O デバイス）を保持し，OS のデバイスドライバ（以下，「OS ドライバ」と略す）は仮想 I/O デバイスに対して I/O 要求を発行する．仮想計算機モニタ（デバイスエ

ミュレータ）が，物理デバイスを介した I/O 処理と，OS ドライバへの通知処理を行う．OS ドライバに対する受信データや I/O 完了の通知はすべて仮想計算機モニタが制御しているため，自モジュールの動作履歴の保存と当該履歴を用いた自モジュール動作の再生処理のみで履歴の保存・再生が実現できた．

しかし，軽量仮想計算機モニタを用いた I/O 処理は，OS ドライバが直接物理 I/O デバイスに要求を発行し，物理 I/O デバイスが受信データや I/O 完了通知を OS ドライバに対して行う．I/O 処理中に軽量仮想計算機モニタが動作する保証もない[⋆1]．そのため，従来の履歴の保存・再生方式では，OS ドライバと物理 I/O デバイスとの間でやりとりされる動作履歴の保存・再生を軽量仮想計算機モニタが行うことはできず，新規の方式設計が必要になる．特に，以下の課題解決の可否の検討が必要となる．

動作契機付与　軽量仮想計算機モニタによる履歴の保存・再生には，OS ドライバと物理デバイスとの間の I/O 処理実行中に当該モニタの動作契機が必要になる．この動作契機を確実に与える方式を設計する．

OS ドライバの動作追跡　再生動作に必要十分な履歴を収集するために，OS ドライバの動作を軽量仮想計算機モニタが追跡する機構も新規に設計する．たとえば，NIC からの受信処理の履歴を保存・再生するためには，OS ドライバと物理 I/O デバイスとの間で共有しているデータ構造（NIC の受信ディスクリプタなど）を軽量仮想計算機モニタがスヌープして，OS が受信したデータの内容や，OS が I/O 完了通知を受けたタイミングの検出などをしなければならない．

デバイス依存処理の極小化・分離　OS ドライバの動作追跡などはデバイス依存処理であり，軽量仮想計算機モニタ層にデバイス依存コードを実装しなければならない．しかし，本コードサイズや開発工数が大きくなると，本 OS デバッガの目標 (2) の充足が難しくなる．そのため，デバイス依存処理の極小化，および，当該コードの分離による新規デバイスへの対応の容易化が必要となる．

以下の節では，3.2 節であげた各履歴に関して，仮想計算機モニタによる履歴の保存や再生を行うための動作契機の付与方式や，OS ドライバの動作追跡に基づいた履歴の保存・再生方式の設計・実装が容易であるか否か，および各履歴の保存・再生にデバイス依存処理があるか否を検証する．

⋆1 従来方式と同様に軽量仮想計算機モニタ内部にデバイスエミュレータを保持させれば動作契機の保証はできるが，本 OS デバッガの目標 (2) の充足はできなくなる．

4.2　I/O ポートアクセス履歴の保存・再生

Intel VT [12] などの CPU 機構を利用すれば，OS が I/O ポートへのアクセス命令を発行した際に仮想計算機モニタを起動でき，以下の方式で I/O ポートアクセス履歴の保存・再生をデバイス非依存の処理で容易に実現できる．実行モードで起動された仮想計算機モニタは，当該アクセス命令を実行して I/O ポートから read した値をレジスタに格納する．あわせて，read された値を履歴に保存する．また，再生モードでは，アクセス命令を実行する代わりに，保存した履歴から read されるべき値を読み出しレジスタに格納する．

4.3　デバイスレジスタアクセス履歴の保存・再生

2 章で示した OS デバッガを用いた場合，NIC や HBA などの I/O デバイスレジスタへのアクセスは，OS が仮想計算機モニタを介さずに行う．PCI デバイスレジスタアクセスはメモリアクセス命令（mov 命令など）で行われるため，デバイスレジスタのアクセス履歴の保存・再生には，軽量仮想計算機モニタの初期化時に当該デバイスレジスタ領域のページを不在化し，メモリアクセス命令実行時にページ例外を強制発行させればよい．これにより軽量仮想計算機モニタの動作契機が保証され，I/O ポートと同様のアルゴリズムで履歴の保存・再生ができる．

しかし，デバイスレジスタアクセス領域を取得し，該当するページを不在化する処理は，PCI 構成空間から読み込んだデバイス ID の識別などの I/O デバイス依存の処理を含む．そのため，本履歴の保存・再生におけるデバイス依存処理の極小化と分離が課題となる．

4.4　割込み発生履歴の保存・再生

Intel VT などの CPU 機構を利用すれば，実行モードにおいて割込みが発生した際に仮想計算機モニタを起動でき，以下の方式で割込み発生履歴の保存・再生をデバイス非依存の処理で容易に実現できる．実行モードにて起動された仮想計算機モニタは，発生した割込み番号，割込み発生時の命令ポインタ，累積実行命令カウンタの値を読み込み履歴に残す．再生モードでは，仮想計算機モニタは，割込み禁止状態で OS を動作させる．そして，履歴に記載された命令ポインタにブレークポイントを設定し，当該命令実行時にブレークポイント例外により仮想計算機モニタを起動させる．起動された際に，累積実行命令カウンタが履歴に記載された値と同じであるか否かを判別し，同じであれば，履歴に記載されている番号の割込み発生を OS に通知する．

4.5　DMA 転送履歴の保存・再生

通常の仮想計算機モニタは，OS が認識する DMA 転送そのものをソフトウェアがエミュレートするため，DMA 転送履歴の保存・再生は容易である．しかし，提案する OS デバッ

ガを用いた場合，NIC や HBA などの I/O デバイスからの DMA 転送は，OS ドライバが仮想計算機モニタを介さずに制御する．そのため，提案する OS デバッガでは，OS が制御するハードウェア機構を用いて行われる DMA 転送状況を仮想計算機モニタがスヌープし，その結果をもとに履歴を生成・保存する（もしくは，履歴をもとに DMA 転送を再生する）必要がある．また，このスヌープを行える契機は，OS によるデバイスレジスタアクセスと割込み発生に限られる．そのため，OS ドライバの動作のモデル化と，当該モデルに基づく履歴の保存・再生機構の設計が必須となる．また，上記スヌープ処理はデバイス依存処理となるため，デバイス依存処理の極小化と分離が課題となる．

以上より，

(1) デバイスレジスタアクセス履歴の保存・再生におけるデバイス依存処理の極小化・分離方式

(2) DMA 転送履歴の保存・再生の全体実現方式（当該保存・再生処理における軽量仮想計算機モニタの動作契機付与方式，OS ドライバの動作追跡および当該追跡結果を利用した履歴の保存・再生方式，デバイス依存処理の極小化・分離方式）

について詳細設計が課題となることが分かる．これらの課題の解決方式を次章で述べる．

5. 課題の解決方式

本章では，4 章で述べたロギング&リプレイ機能の実現上の課題の解決方式として，デバイスレジスタアクセス履歴の保存・再生におけるデバイス依存処理の極小化・分離方式，および DMA 転送履歴の保存・再生の実現方式を説明する．

5.1 デバイスレジスタアクセス履歴の保存・再生方式

4 章で述べたとおり，デバイスレジスタアクセス履歴の保存・再生のために，軽量仮想計算機モニタの初期化時に PCI デバイスのレジスタ領域を不在にする．そして，OS ドライバがデバイスレジスタにアクセスした際に軽量仮想計算機モニタの動作契機を与え，軽量仮想計算機モニタが，I/O ポート履歴の保存・再生と同様の手法でデバイスレジスタアクセス履歴の保存・再生を行えるようにする．

上記履歴の保存・再生処理におけるデバイス依存処理の極小化と分離を行うため，**図 3** に示すとおり，VM 層デバイスドライバ（OS 層ドライバとは別のドライバ）を新規に実装し，当該ドライバでデバイス依存処理を実現した．

初期化時のデバイスレジスタ領域不在化のために，PCI 構成情報を読み出し，デバイスレジスタ領域のアドレス取得を行う．本 PCI 構成情報取得処理はデバイス依存処理であるため，軽量仮想計算機モニタとの間に PCI 構成情報取得インタフェースを作成し，VM 層デバイスドライバで処理本体を実装する．一方，履歴の保存・再生のための命令エミュレーションや read された履歴の保存処理などはデバイス非依存処理であるため，VM 層ドライバではなく仮想計算機モニタ内部に実装し，デバイス依存部分を極小化する．

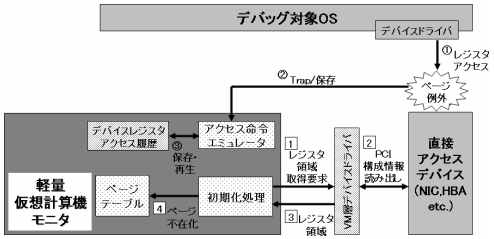

図 3　デバイスレジスタアクセス時の仮想計算機モニタ機能方式
Fig. 3　Virtual machine monitor driving method after device register access.

前章で述べたとおり，VM 層デバイスドライバのコード規模や開発工数が，本研究で実装する OS デバッガの汎用性に大きく影響する．この解析結果については 6 章で述べる．

また，上記方式によるデバイスレジスタアクセス履歴の保存・再生機能を提案する OS デバッガに追加することで，OS によるデバイスレジスタアクセスのたびの仮想計算機モニタ起動が余計に必要になり，OS の I/O 性能が大きく低減する懸念がある．I/O 性能劣化についての定量的な評価結果については 7 章で述べる．

5.2 DMA 転送履歴の保存・再生方式

4 章で述べたとおり，DMA 転送履歴の保存・再生には，OS ドライバ（NIC ドライバ）のモデル化と，当該モデルに基づく履歴の保存・再生機構の設計が必要となる．特に NIC 受信時の DMA 転送は，OS ドライバの指示ではなく，OS ドライバとは独立のタイミングで DMA 転送が起動されるため，この設計が複雑になる．本節では，NIC の受信時の DMA 転送の保存・再生の実現方式について説明する．まず，提案する OS デバッガで仮定した NIC ドライバの動作モデルを説明した後に，当該モデルに基づいた履歴の保存・再生手順，およびその手順を実現するために VM 層デバイスドライバに実装すべきデバイス依存部分について説明する．

5.2.1 NIC ドライバのモデル化

提案する OS デバッガで想定している NIC ドライバの受信時の動作モデルを策定するため，筆者らが開発した HiTactix [13)–15)] で実装されている PCI バスに接続可能な NIC のド

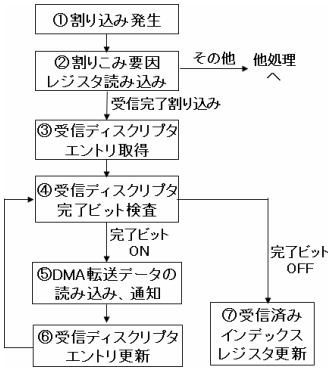

図 4　NIC ドライバ受信時の動作モデル
Fig. 4　Network packet receive model of NIC driver.

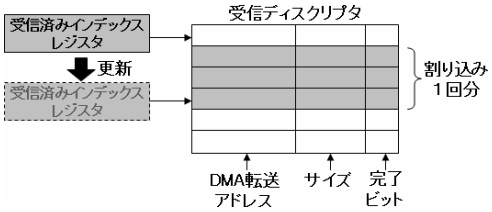

図 5　受信ディスクリプタの構成
Fig. 5　Data structure of receive descriptor.

ライバ実装コードの解析を行った．その結果，これらの NIC ドライバの受信処理は**図 4** に示すフローでモデル化できることが分かった．以下のモデルの説明では，**図 5** に記載されている受信ディスクリプタの使用を仮定する（HiTactix でデバイスドライバを実装している NIC は，本ディスクリプタと同じ，もしくは同等のデータ構造を利用している）．受信ディスクリプタは，DMA 転送先アドレス，サイズのほかに，受信済みインデックスレジスタ，完了ビットを保持する．ディスクリプタのエントリは，OS ドライバが reader，NIC が writer の共有リングバッファになっている．OS ドライバは，受信処理を完了した最終エントリを受信済みインデックスレジスタに格納することで，NIC に read 状況を通知する．一方，NIC は，当該エントリへの受信データの DMA 転送が完了すると完了ビットを ON にすることで，OS ドライバに write 状況を通知する．

NIC ドライバの受信時の動作手順を以下に示す．

(1)　割込み発生により NIC ドライバが起動する．
(2)　割込み要因レジスタを読み込み，割込み要因が受信完了かそれ以外であるかを判別する．受信完了であれば処理を継続する．
(3)　受信ディスクリプタのエントリを順に参照し，(4)〜(6) の処理を行う．
(4)　完了ビットが立っていれば (5) に，立っていなければ (7) にジャンプする．
(5)　データ受信をドライバの上位層に通知する．
(6)　新しい DMA 転送アドレス（受信バッファアドレス），サイズを設定する．完了ビットをクリアする．
(7)　受信済みインデックスレジスタを更新する．

上記モデル化により，以下の 2 点が明らかになる．次項で述べる履歴の保存・再生方式では，この 2 点を仮定する．

- DMA 転送データは，割込み要因レジスタの参照後，かつ，受信済みインデックスレジスタの更新前に読み込まれる．また，これら 2 つの処理の間には仮想計算機モニタの起動契機はない．すなわち，履歴の保存は受信済みインデックスレジスタ更新時に，1 回の割込みで処理した DMA データの転送履歴をまとめて保存する必要がある．また，割込み要因レジスタの参照時に，1 回の割込みで処理した DMA データの転送をまとめてエミュレート（履歴を用いて再生）する必要がある．
- 受信ディスクリプタの更新内容と受信済みインデックスレジスタの更新履歴をたどることで，1 回の割込み処理でデータ受信を通知した DMA 転送データのアドレス，サイズ群を取得できる．

5.2.2　履歴の保存・再生方式

DMA 転送履歴を用いた再生のために，以下を履歴に保存する．

- 割込み要因レジスタの累積参照回数
- DMA 転送アドレス
- DMA 転送サイズ
- DMA 転送データ

これらの情報があれば，割込み要因レジスタ参照を契機に起動した仮想計算機モニタが以下を実行すれば，DMA 転送のエミュレート（履歴を用いた再生）が実現できる．

(1)　累積参照回数が一致する履歴のエントリ群を検索する[*1].
(2)　上記エントリ群に格納されている DMA 転送アドレスに，格納されている DMA 転送データを，格納されているサイズ分だけコピーする（DMA 転送本体のエミュレート）.
(3)　受信ディスクリプタエントリの DMA 転送サイズフィールドに履歴に格納されている値を設定する．あわせて同エントリの完了ビットを設定する（受信ディスクリプタの更新のエミュレート）.

以下では，実行モードにおいて上記各情報を取得・保存する方法，および当該方法の処理のうち VM 層デバイスドライバに実装すべきデバイス依存部分について説明する.

割込み要因レジスタの累積参照回数　仮想計算機モニタは初期化時に割込み要因レジスタアドレスを取得する．5.1 節で述べた方式により，OS が割込み要因レジスタを参照した際に仮想計算機モニタを起動できる．仮想計算機モニタは，起動の際にページ例外発生アドレスと初期化時に取得したアドレスと比較することで，当該レジスタへの累積参照回数を管理する．保存時には，現在の累積参照回数を履歴にコピーする.

DMA 転送アドレス　DMA 転送アドレスの取得および履歴への保存は，受信済みインデックスレジスタ更新時に受信ディスクリプタを参照しても実現できない．5.1 節で示したとおり，受信済みインデックスレジスタ更新（5.1 節のステップ (7)）前に，OS ドライバは，対応するエントリの DMA 転送アドレスを新しいアドレスに更新（5.1 節のステップ (6)）するためである.

提案する OS デバッガでは，仮想計算機モニタ内にシャドウ受信ディスクリプタ（DMA 転送アドレスフィールド部分のみ）と，シャドウ受信済みインデックスレジスタを保持する．軽量仮想計算機モニタは，初期化時に受信ディスクリプタと受信済みインデックスレジスタのコピーを作成する.

受信済みインデックスレジスタ更新時に，シャドウ受信済みインデックスレジスタと更新後の受信済みインデックスレジスタの値を比較し，対応する割込みで受信処理が行われた受信ディスクリプタエントリを取得する．そして，シャドウ受信ディスクリプタから，DMA 転送アドレスを取得し，履歴に保存するとともに，新しい DMA 転送アドレスをシャドウ受信ディスクリプタに設定する.

DMA 転送サイズ　DMA 転送サイズも，DMA 転送アドレスの項で述べた理由で，受信済みインデックスレジスタ更新時に受信ディスクリプタを参照しても取得できない．代わりに，提案する OS デバッガでは，取得した DMA 転送アドレスに格納されている受信データのヘッダを解析し，受信データサイズを算出，算出した値を履歴に保存する.

DMA 転送データ　上記方式で取得した DMA 転送アドレス，DMA 転送サイズをもとに主メモリ上に配置されている DMA 転送データを取得，履歴に保存する.

上記履歴の保存・再生方式において，デバイス依存部分を極小化するため，以下の方針で軽量仮想計算機モニタと VM 層デバイスドライバのインタフェースを設計する.

- 割込み要因レジスタの累積参照回数を取得するため，当該レジスタアドレス取得インタフェースを設ける．アドレス取得処理は VM 層デバイスドライバで行うが，取得できたアドレスとページ例外アドレスとの比較や累積参照回数の管理は軽量仮想計算機モニタが行う.
- DMA 転送アドレスを保存するため，受信済みインデックスレジスタアドレス・格納値の取得，ディスクリプタエントリの取得・更新など，シャドウ受信ディスクリプタとシャドウ受信済みインデックスレジスタの管理に必要なインタフェースを設ける．VM 層デバイスドライバでは指定情報の取得処理のみを行い，シャドウの管理は軽量仮想計算機モニタが行う.

6. 実装の詳細

本章では，実装した OS デバッガの構成，実装上の課題，VM 層デバイスドライバの実装容易性の解析結果について示す.

6.1 構　　成

本研究で実装したシステムの構成を**図 6** に示す．本システムで実装した軽量仮想計算機モニタは，Intel VT 機構を持つ CPU（Intel 社 Xeon）上で動作する．既存実装機能に，履歴保存機能と再生機能を追加で実装している．さらに，VM 層ドライバとして，Intel 社 NIC のドライバを実装している．本ドライバの実装容易性の解析結果は 6.3 節に示す．軽量仮想計算機モニタは，Intel 社 Xeon[*2] と完全互換なハードウェアインタフェースを OS に提供している．ただし，アドレス変換機構のみは Intel VT 機構でサポートされていないため実装していない．本軽量仮想計算機モニタ上で動作する OS がページ変換機構を利用しないこ

⋆1 本エミュレーション方式では，割込み要因レジスタの参照直後に 1 回の割込みで処理した DMA データの転送をまとめてエミュレートする．仮定した NIC ドライバの動作モデルでは，1 回の割込み処理につき 1 度の割込み要因レジスタの参照を行うため，割込み要因レジスタの累積参照回数から，対応する DMA データの転送が処理された割込み処理，すなわち DMA 転送のエミュレートの実行タイミングが特定できる.

⋆2 Xeon は米国 Intel 社の登録商標です.

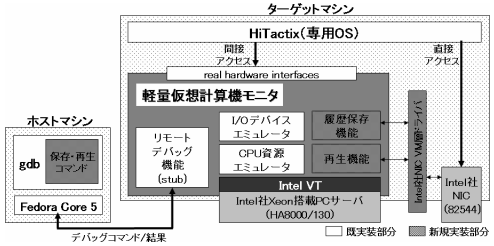

図 6　実装システムの構成
Fig. 6　Implemented system architecture.

とを前提としている．現在の実装では，軽量仮想計算機モニタ上に，筆者らが開発した専用 OS HiTactix を動作させている．ターゲットマシン上では，Fedora Core 5 上で gdb（Gnu Debugger）を動作させている．gdb には，3.2 節で述べた保存・再生コマンドを追加で実装している．

6.2　実装上の課題

Intel 社 NIC の VM 層ドライバの実装にあたり，以下の 2 つの課題に直面した．

- 受信エラーの再生
- 送信完了通知のエミュレート

以下では，上記課題とその解決方式の概要について説明する．

6.2.1　受信エラーの再生

5.2.2 項で示したとおり，提案した OS デバッガの軽量仮想計算機モニタは，受信ディスクリプタ更新のエミュレートのため，受信ディスクリプタエントリの DMA 転送サイズフィールド，完了ビットの設定を行う．

しかし，Intel 社 NIC（および他の多くの NIC）は，受信処理時にチェックサムエラーなどの受信エラーが発生した際に，エラー発生を通知するエラーフィールドのビットを立てる．そのため，軽量仮想計算機モニタによる再生処理は，DMA 転送サイズフィールド，完了ビット以外に，エラーフィールド更新のエミュレートも行わなければならない．しかし，5.2.1 項で述べた NIC 動作モデルの割込み要因レジスタ読み込み後に，NIC が本フィールドを更新する可能性がある（当該レジスタ読み込み後に NIC が行った受信処理でエラーが発生した場合）．一方，受信済みインデックスレジスタ読み込み時には，ドライバは当該受信ディスクリプタエントリを再初期化した後であるため，実行モードにて軽量仮想計算機モニタは当該フィールドの内容を読み込み，履歴に保存することができない．

現在の実装では，エラー発生時に OS ドライバがディスクリプタエントリに設定されている DMA 転送アドレスを更新しないこと，および OS ドライバは，通知されたエラーの種類により処理を変えないことを仮定している．そして，実行モードにおける受信済みインデックスレジスタ更新時に，受信ディスクリプタエントリの DMA 転送アドレスの更新が行われていないことを軽量仮想計算機モニタが検知した場合に，特定の種類の受信エラーが発生したと認識し，履歴に保存する．再生モードで，当該履歴に従い特定の種類の受信エラー発生を通知するために受信ディスクリプタの更新を行う．

より厳密な受信エラーの再生には，OS ドライバとより密な連携が必要である．この実現は今後の課題とする．

6.2.2　送信完了通知のエミュレート

OS ドライバにより送信要求が発行されると，NIC は送信処理を開始する．そして，送信処理完了時に，NIC が OS ドライバに完了を通知する．本通知は，デバイスレジスタ（送信済みインデックスレジスタ）への値の設定により行われる場合と，送信ディスクリプタの完了ビット更新により行われる場合がある．

前者の場合は，OS ドライバによるデバイスレジスタアクセス時に仮想計算機モニタが起動できるため，実行モードにおけるデバイスレジスタのアクセス履歴の保存，および再生モードにおけるアクセスの再現のみで，上記送信完了通知をエミュレートできる．しかし，後者の場合は，送信ディスクリプタ領域全体をページ不在にしない限り，完了ビットアクセス時に仮想計算機モニタの起動が行えず，送信完了通知のエミュレートを行えない．そして，送信ディスクリプタ領域全体のページ不在化は，他の DMA 転送アドレスフィールド設定などのたびにページ例外発生を引き起こし，大きな I/O 性能劣化を招く．

現在の実装では，VM 層デバイスドライバが，初期化処理時に，対象 NIC が上記どちらのモードで動作するかを軽量仮想計算機モニタに通知している．そして，軽量仮想計算機モニタは，上記モードによって送信完了通知のエミュレート方法を変える．実装した Intel 社 NIC の VM 層ドライバは後者のモードでの動作を選択しているため，送信性能の大きな劣化が発生している．本性能劣化の回避方法も今後の課題とする．

6.3　VM 層デバイスドライバの実装容易性

VM 層デバイスドライバは，NIC からの DMA 転送の履歴を保存・再生するために**表 1** に示すインタフェースを提供する．どのインタフェースも，PCI 構成空間レジスタ，デバイスレジスタ，ディスクリプタ領域の読み込み，または更新の組合せのみで実現でき，内部状態管理はいっさい必要ない．

表 1　VM 層デバイスドライバの提供インタフェース
Table 1　Interfaces provided by VM-layer device driver.

I/F 名	概要
Init	VM 層デバイスドライバの初期化処理を行う．PCI 構成空間から自デバイス ID を検索し，当該デバイスのデバイスレジスタ領域のリスト，割込み要因レジスタのアドレス，受信済みインデックスレジスタアドレス，送信完了通知の動作モードをリターンする．
GetDescRange	送信/受信ディスクリプタ領域をリターンする．
GetRxIndexReg	受信済みインデックスレジスタの値をリターンする．
GetRxMaxIndex	受信済みインデックスレジスタの最大値（受信済みインデックスレジスタのエントリ数）をリターンする．
GetRxDmaAddr	指定インデックスを持つ受信ディスクリプタエントリの DMA 転送先アドレスをリターンする．
SetRxUpdateEntry	指定インデックスを持つ受信ディスクリプタエントリの DMA 転送サイズ，完了ビットフィールドを設定する．

Intel 社 NIC の VM 層デバイスドライバを実装したところ，約 600 行（コメント行を含む）程度で実装できた．OS ドライバ約 14,000 行と比較すると，その 4.3%の追加コードの開発でロギング&リプレイ機能を利用可能になっており，大きな追加開発は必要ない．

7. 性能評価

本章では，本研究で実装したロギング&リプレイ機能で，履歴の保存・再生に要するオーバヘッドを定量的に評価する．そして，提案方式による履歴の保存・再生がどの程度の現実的なオーバヘッドで実現可能であるかを明らかにする．オーバヘッドの定量評価は，履歴の保存・再生に要する CPU 負荷とメモリ消費量の 2 つについて行う．

7.1 CPU 負荷の評価

履歴の保存・再生に要する CPU 負荷を測定するため，**図 7** に示す評価環境で評価を行った．本環境では，(1) 履歴の保存機能がない軽量仮想計算機モニタ，(2) 履歴の保存機能を持つ軽量仮想計算機モニタ，(3) VMware Workstation 6 [16]（Linux 版）上で，我々が開発した専用 OS HiTactix，および UDP 受信プログラムを動作させる．そして，UDP 受信プログラムに対して，他のサーバ上で動作する UDP 送信プログラムから可変レートで UDP パケットを送信する．UDP パケットの送信レートを変動させた際のそれぞれの仮想計算機モニタが動作するサーバの CPU 負荷の変動を測定することで，ネットワーク受信処理の履歴保存に要する CPU 負荷の評価を行った[⋆1]．また，履歴保存機能つきおよび履歴保存機能なしの軽量仮想計算機モニタと従来の仮想計算機モニタのネットワーク受信性能の比較も行った．

測定結果を**図 8** に示す．図中の横軸は UDP 送信プログラムによる UDP パケットの送信レートを，縦軸がそれぞれの仮想計算機モニタが動作するサーバの CPU 負荷を示す．本結果から以下が明らかになった．

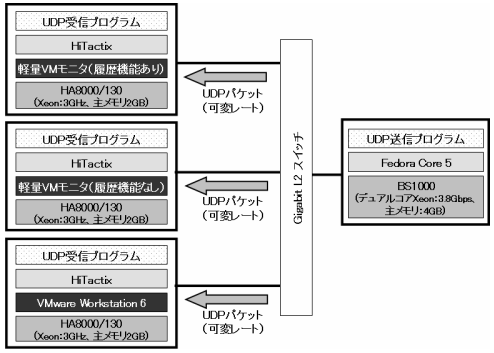

図 7　評価環境
Fig. 7　Evaluation system architecture.

(1)　履歴保存を行うことで，ネットワーク受信に要する CPU 負荷が約 3.5 倍に上昇する．
(2)　履歴保存を行わない軽量仮想計算機モニタは，従来の仮想計算機モニタ（VMware）の約 38%の CPU 負荷でネットワークの受信処理が実現できる．一方，履歴保存を行う軽量仮想計算機モニタは，VMware よりネットワーク受信処理に要する CPU 負

⋆1 HiTactix ではアイドルスレッドが消費する CPU 時間を利用してシステム全体の CPU 使用率を算出する．アイドルスレッド走行中に軽量仮想計算機モニタが動作した場合，当該モニタが動作したことが HiTactix から認識できないため，当該モニタの走行時間もアイドルスレッドの走行時間の一部としてカウントされる．これによる測定誤差を防ぐため，本測定では，(1) アイドルスレッド走行時間の総計，(2) アイドルスレッド実行中の軽量仮想計算機モニタの走行時間の総計，(3) それ以外のスレッド実行中の軽量仮想計算機モニタの走行時間の総計を計測し，(1) から (2) を引いた時間をシステムがアイドル状態になっている時間の総計と判定した．

表 2　CPU 使用率の内訳の測定結果
Table 2　Measurement results of detailed CPU usage.

処理	履歴保存機能あり（実測）		履歴保存機能なし（実測）		VMware（推測）	
	起動回数（回/s）	CPU 使用率（%）	起動回数（回/s）	CPU 使用率（%）	起動回数（回/s）	CPU 使用率（%）
デバイスメモリアクセス&DMA 転送履歴保存	34,430.0	56.44%	-	-	21,282.2	26.70%
（うち DMA 転送履歴保存のためのデータコピー）	（33,929.2）	（13.25%）	-	-	-	-
割込み発生履歴保存・割込み通知 emulation	1,839.1	2.12%	1,754.7	2.08%	1,839.1	2.12%
I/O ポートアクセス履歴保存・ICU emulation	8,153.8	8.62%	8,305.7	8.97%	8,153.8	8.62%
CPU クロック読み出し履歴保存	4,966.4	4.85%	-	-	-	-
コンテクストスイッチ emulation	558.3	0.56%	900.5	0.89%	558.3	0.56%
OS ドライバ処理，UDP 受信プログラム	-	13.39%	-	10.80%	-	13.39%
Linux ドライバ，World Switch など	-	-	-	-	-	8.48%
合計		85.98%		22.73%		59.97%

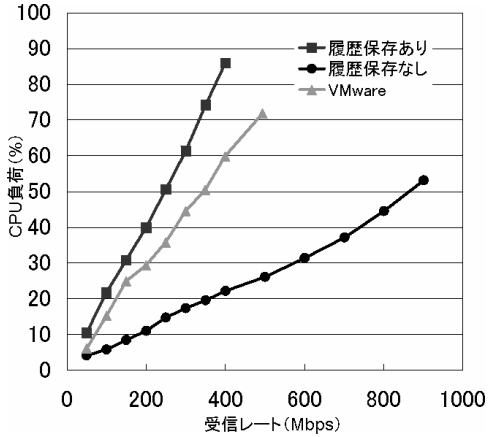

図 8　CPU 負荷の評価結果
Fig. 8　CPU load evaluation results.

荷が約 38%上昇する．

上記 (1) の要因を分析するため，履歴保存機能を持つ軽量仮想計算機モニタと持たないモニタを使用しつつ 400 Mbps の UDP パケットを受信した場合について，軽量仮想計算機モニタ，HiTactix，UDP 受信プログラムが消費する CPU 使用率の内訳を測定した．測定は以下により行った．

- 軽量仮想計算機モニタの起動時と終了時に CPU クロックを取得するコードを埋め込み，計量仮想計算機モニタの起動要因ごとに，起動回数，および消費 CPU 使用率を求めた．
- 上記計量仮想計算機モニタの起動/終了時に，Intel VT が提供する VMexit/VMresume 命令の実行，および VMread/VMwrite 命令を用いたコンテキストの退避/回復も実行される．上記起動回数にこれらの命令の実行時間をかけ，消費 CPU 使用率を求めた（命令の実行時間はベンチマークプログラムを作成し実測した．VMexit/VMresume 命令合計で約 3.22 マイクロ秒，VMread/VMwrite 命令によるコンテキストの退避/回復合計で約 6.65 マイクロ秒の実行時間を消費するとの結果を得た）．
- 上記から軽量仮想計算機モニタの起動/走行/終了に要する CPU 使用率の総計が求まる．全体の CPU 使用率より上記 CPU 使用率の総計を引いた値を UDP 受信プログラムおよび OS（HiTactix）が消費する CPU 使用率と推定した．

測定結果を**表 2** に示す．この結果より以下が分かった．

- デバイスメモリのアクセス履歴保存処理により，CPU 使用率が 56.44%上昇している．これは全体の CPU 使用率上昇量の約 89%に相当し，CPU 使用率上昇の主要因だといえる．なお，この 56.44%の上昇のうち，VMexit/VMresume/VMread/VMwrite 命令実行により 33.98%（(3.22 + 6.65) マイクロ秒 × 34,430 回 = 339.8 ミリ秒），DMA 転送データ履歴保存のためのコピー処理により 13.25%の CPU 使用率上昇が起こり，これらの処理時間を低減させることができなければ，本 CPU 使用率上昇を低減させることができない．

- 上記のほかに，CPU クロックの読み出し履歴保存[⋆1]により約 4.85%の CPU 使用率上昇が起こる．さらに，UDP 受信プログラム・HiTactix による CPU 使用率も約 2.59%上昇している．後者は，頻繁な VMexit/VMresume 命令実行によるキャッシュヒット率低下が要因と考えられる．

また，上記 (2) の要因を推測するため，表 2 の測定結果から VMware による受信処理の CPU 使用率の内訳を以下の方法で推定した．推測結果も表 2 に追記してある．

- HiTactix は軽量仮想計算機モニタ上で動作する場合には Intel 社 NIC ドライバを，VMware 上で動作する場合には Lance ドライバを利用して受信処理を行う．前者は，1 回の割込み要因処理につき（到達パケット数）+ 1 回の，後者は 3 回のデバイスメモリアクセスが発生する．Lance ドライバが動作した場合のデバイスメモリアクセスエミュレーション処理の起動回数を推測するため，Intel 社の NIC ドライバを Lance ドライバと同様のデバイスメモリアクセス回数となるように改変し，デバイスメモリアクセスエミュレーション処理の起動回数を測定した．その結果 21,282 回/秒の起動が発生した．これにより，デバイスメモリアクセスエミュレーションの消費 CPU 使用率は，$(56.44 - 13.25) \times 21{,}282/34{,}430 = 26.70\%$だと予測した（デバイスメモリアクセスの履歴保存に要する CPU 使用率は小さいため，本推測では無視した）．
- その他の各種エミュレーション処理は，軽量仮想計算機モニタの履歴保存処理とエミュレーション処理の合計と同等の CPU 使用率を消費すると推定した．
- 残った CPU 使用率が，world switch，バッファのリマッピング，Linux のデバイスドライバの実行により消費された CPU 使用率だと想定し，約 8.48%だと推測した．

上記推測結果より以下が明らかになった．

- 履歴保存なしの軽量仮想計算機モニタを用いた場合，VMware と比べ，約 26.70%の CPU 使用率を消費するデバイスメモリアクセスエミュレーションが不要になる．これが全体の CPU 使用率低減量の約 72%を占め，CPU 低減の主要因になっている．
- 履歴保存ありの軽量仮想計算機モニタを用いた場合は，約 8.48%の world switch，バッファのリマッピング，Linux のデバイスドライバの動作が不要になるものの，DMA 転送履歴の保存にためのデータコピー処理，CPU クロック読み取り処理で約 18.10%の CPU 使用率を余計に消費し，全体の消費 CPU 使用率が上昇する．さらに本実験では，使用しているデバイスドライバの違いによりデバイスメモリアクセス回数が増えたため，さらなる CPU 使用率上昇が発生した．

⋆1 HiTactix の場合，OS 実行中に rdtsc 命令を実行する．実行モードで読み出した値を再生モードで読み出し可能にするため，軽量仮想計算機モニタは本命令の実行結果も履歴に保存している．

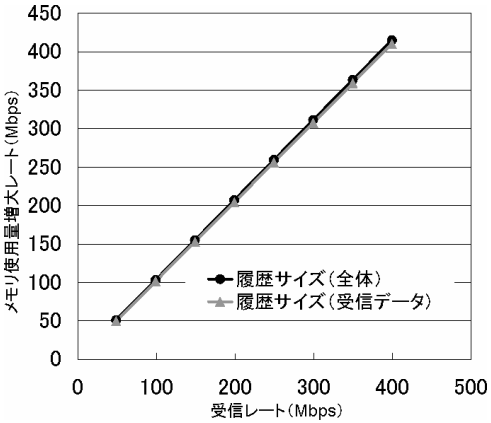

図 9　メモリ消費量の評価結果
Fig. 9　Used memory size evaluation results.

7.2　メモリ消費量の評価

履歴の保存のために消費するメモリ消費量を評価するため，図 7 の評価環境で，UDP 送信プログラムによる UDP パケットの送信レートを変動させた場合に，履歴保存機能付き軽量仮想計算機モニタが履歴保存のために消費するメモリ使用量の増加レートを測定した．

評価結果を**図 9** に示す．図の横軸が UDP パケットの送信レートを，縦軸が履歴保存のためのメモリ使用量の増加レートを示す．メモリ使用量の測定は，全履歴を格納するためのメモリ使用量と，DMA 転送データの履歴を格納するためのメモリ使用量のそれぞれについて行った．その結果，全メモリ消費量の 98.5%が DMA 転送データの履歴保存に使用されていることが分かった．

提案した OS デバッガで，長時間にわたる実行の履歴の保存（およびその履歴に基づく再生）を行うためには，上記メモリ消費量増加レートで wirte 可能な大容量記憶媒体が必要になる．上記の結果より，ギガビットオーダの write 性能を持つ大容量記憶媒体（HDD を多数搭載した RAID 装置など）を使用すれば，上記の履歴の保存・再生も可能になると予測できる．

8. 考　　察

本章では，提案デバッガの利点と欠点，および提案デバッガの I/O デバイス依存性についての考察を行う．

8.1 提案デバッガの利点/欠点

提案デバッガは，汎用性（いかなる I/O デバイスドライバの開発にも適用可能）と安定稼動を確保しつつ，ロギング&リプレイ機能を利用したデバッグ機構を提供する．特にロギング&リプレイ機能に関しては，物理 I/O デバイスから OS ドライバが受けた入力履歴を保存し，当該履歴の忠実な再生ができる点で従来のロギング&リプレイ機能とは異なる．この特徴のために，提案デバッガは以下の利点を提供できる．

- 対象 I/O デバイスにかかわらず，再現性の低いバグ（たとえば，特定の割込み発生タイミング，特定の受信データなど限定的な条件下でのみ顕在化するバグやメモリ破壊）が顕在化した場合に，その原因の着実な追求が可能になる．
- OS ドライバと I/O デバイスの並列動作タイミングに起因するバグの着実な原因追求が可能になる．軽量仮想計算機モニタの場合，OS ドライバ実行中も I/O デバイスは並列動作し，ハードウェア状態を変える．一方，従来の仮想計算機モニタのハードウェアエミュレータは，この状態変更を OS ドライバ実行中に行う保証はない．そのため，たとえば，OS ドライバの受信処理中に受信ディスクリプタの更新が発生した場合にのみ顕在化するバグなどの原因追求が可能になる．
- I/O デバイスのハードウェアエラー発生など，特別なハードウェア状態下でのみ顕在化するバグの原因追求が可能になる．通常，仮想計算機モニタのエミュレータはハードウェアの異常動作をエミュレートしない[⋆1]．

一方で提案デバッガは，履歴の保存・再生処理での大きな CPU 負荷発生，I/O ドライバのモデル化，仮想計算機モニタ層の実装の I/O デバイス依存性，などの前提を持っているため，以下の欠点を持つ．

- 軽量仮想計算機モニタ上で動作させる場合とさせない場合で OS ドライバの実行タイミングが大きく変わる．そのため本モニタ上で顕在化しないバグがある．たとえば，一定時間内に特定区間を走行した場合にのみ健在化するバグなどは顕在化しない可能性がある．
- 軽量仮想計算機モニタの履歴の保存・再生に大きな NIC の受信性能低下をもたらす．その結果，ネットワーク帯域飽和より前に CPU 飽和が発生し，OS ドライバによるネットワーク帯域飽和時の異常処理のデバッグが困難になる．デバッグのためには，軽量仮想計算機モニタによる擬似ネットワーク負荷の生成など，追加のエミュレーション処理の実装が必要になる．
- 軽量仮想計算機モニタは OS ドライバの動作をモデル化している．OS ドライバが当該モデルに従わない動作を行うなどの重大なバグがあった場合，正しい履歴の保存・再生ができない．間違った履歴の再生は，OS ドライバ実行中の例外発生としてデバッガは通知する．しかし，再生履歴の誤り，OS ドライバのバグのどちらが例外発生要因であるかの特定は難しい．
- 提案デバッガを使用するためには VM 層デバイスドライバの開発が必要になる．VM 層デバイスドライバのバグも，誤った履歴再生を引き起こすうえに，バグ追求が難しい．

8.2 提案 NIC 受信モデルの適用可能性

軽量仮想計算機モニタを用いた OS デバッガは，NIC の受信処理のモデル化を行い，DMA 転送履歴の保存・再生を行う．本モデルの適用可否は NIC が保持する DMA 転送機能仕様と OS ドライバの実装方式により決まる．本節では，本モデル化の適用範囲について考察し，本モデルが一般的な NIC や OS ドライバに広く適用可能であることを示す．

8.2.1 NIC の DMA 転送仕様の仮定事項と適用範囲

NIC の DMA 転送機能仕様に関して提案した受信処理モデルで仮定した事項について述べ，その仮定が一般的な NIC で成立することを示す．

リングバッファ状受信ディスククリプタ　OS ドライバと NIC 間で共有リングバッファ状の受信ディスクリプタを持ち，当該ディスクリプタで DMA 転送先アドレス，サイズを通知を行うことを仮定している．本機能は，OS ドライバと NIC の並列動作を可能にし，高速ネットワークからの受信処理時のパケット取りこぼしを防ぐための必須機能であり，高速ネットワークに接続可能な NIC に一般的に搭載されている．

完了ビット　DMA 転送が完了した受信ディスクリプタのエントリを完了ビット，もしくはそれに相当する手段で NIC から OS ドライバに通知することを仮定している．一般に本通知は，(1) 受信ディスクリプタのビット（完了ビット），(2) デバイスレジスタ，のいずれかで行われる．後者の方式を持った NIC でも提案受信モデルの適用は容易である．VM 層ドライバの SetRxUpdateEntry インタフェースの実装において，受信ディ

⋆1 ただし，現在の実装では 6.2.1 項に示す制限があるため，提案デバッガを用いてデバッグできるハードエラー処理には制限がある．

スクリプタの更新の代わりにデバイスレジスタの更新を行うことで提案デバッガとの連携ができる．

受信インデックスレジスタ　DMA 転送データの読み込みが完了したエントリを，受信インデックスレジスタ，もしくはそれに相当する手段により OS ドライバから NIC が受け取ることを仮定している．一般に本通知は，(1) 受信ディスクリプタのビット，(2) デバイスレジスタ（受信済みインデックスレジスタ），のいずれかで行われる．前者の方式を持った NIC でも提案受信モデルの適用は可能である．前者方式を持つ NIC は，一般に，新たな受信割込みを受け付け可能になったことを OS ドライバからデバイスレジスタ更新により通知を受ける．そのため，VM 層デバイスドライバで，当該デバイスレジスタアドレスを Init インタフェースでリターンして登録するとともに，当該レジスタアクセス時に呼び出される GetRxIndexReg インタフェースで受信済みインデックスレジスタの代わりに受信ディスクリプタのビットを検査することで，提案デバッガとの連携が可能になる．

8.2.2　OS ドライバ実装方式の仮定事項と適用範囲

OS ドライバの実装方式に関して提案した受信処理モデルで仮定した事項について述べ，その仮定が一般的な OS ドライバで成立することを示す．

割込み要因レジスタの読み出し契機　OS ドライバが，割込み処理の先頭で割込み要因レジスタを読み出すことを仮定している．NIC には割込み要因が複数あり，OS ドライバがその要因別に処理を変える必要があるため，一般的な OS ドライバでこの仮定は成立する．

DMA 転送データの読み込み契機　OS ドライバが，割込み要因レジスタの読み込み後，かつ受信済みインデックスレジスタの更新前に DMA 転送データを読み込み，かつディスクリプタの読み込みエントリを更新することを仮定している．OS ドライバの受信処理は，割込み要因の読み込みにより起動されること，および受信インデックスレジスタ更新前のディスクリプタエントリ更新を怠ると NIC による受信済みデータの上書きが発生しうることから，一般的な OS ドライバでこの仮定は成立する．

9. 関 連 研 究

本論文が目的としている効率的な OS のデバッグの実現を目的とした関連研究やソフトウェアは数多く存在する．しかし，1 章で示したとおり，kgdb [5]（リモートソフトウェアデバッガ），kdb [6]（OS 組み込みのソフトウェアデバッガ），Temporal Debugger [7]（ハードウェアシミュレータと連携して動作する OS デバッガ）は，デバイスドライバ開発時の安定稼動の保証，または新規の I/O デバイスドライバ開発への適用が課題である．一方，本研究で提案したデバッグ方式は，スタブが使用するハードウェア資源のみをエミュレートする軽量仮想計算機モニタを使用することで，安定稼動保証と新規 I/O デバイスドライバへの適用可能性を同時に実現している．さらに，8.1 節に示した利点も提供している．

また，本研究で新規に実装したロギング&リプレイ機能に関する研究も多数存在する．

Flight Data Recorder（FDR）[17] は，メモリの更新履歴，割込み発生履歴，I/O ポートアクセス履歴，DMA 転送履歴などを保存できるモジュールを組み込んだ専用ハードウェアを実装し，OS 動作の完全な再生（ロギング&リプレイ機能）を可能にしている．しかし，本研究でターゲットとしている汎用 PC サーバは，チップセット，バスアーキテクチャ，周辺 I/O デバイスが頻繁に更新され，ハードウェアにロギング&リプレイ機能を組み込む方式は開発工数がかかる．軽量仮想計算機モニタによるロギング&リプレイ機能は，汎用 PC サーバが提供するインタフェースが変わらない（PC/AT 仕様に準拠し続ける）限り，VM 層デバイスドライバの新規開発のみで上記更新にも対応できる．

ReVirt [10]，ExecRecorder [18] は，通常の仮想計算機モニタ上でロギング&リプレイ機能を実現し，OS 動作の完全再生を可能にしている．仮想計算機モニタにて I/O デバイスの完全なエミュレーションを行っているため，デバイスレジスタアクセス履歴や DMA 転送履歴の保存・再生は容易に実現できるものの，仮想計算機モニタでエミュレートできる I/O デバイスに限りがあり，新規 I/O デバイスのドライバ開発への適用が難しい．軽量仮想計算機モニタは，デバイスレジスタアクセスなどの捕捉は行うがエミュレーションは行わないことで，新規 I/O デバイスのドライバ開発への適用を可能にしている．

BugNet [19]，FlashBack [20] は，専用ハードウェアもしくは OS の機能拡張により，ユーザレベルのコードのロギング&リプレイを可能にしている．デバッグ対象をユーザレベルのコードに限定しているため，割込み発生履歴や DMA 転送履歴の保存・再生は行わない．軽量仮想計算機モニタのロギング&リプレイ機能は，これらの履歴の保存・再生もサポートし，OS 動作の完全再生を実現している．

10. ま と め

本研究では，(1) OS 異常動作時の安定稼働を保証する安定性，(2) いかなる I/O デバイスドライバの開発にも適用できる汎用性，(3) ロギング&リプレイ機能への対応，の 3 つを同時に実現可能な OS デバッガの実現を目指し，筆者らがすでに提案している軽量仮想計算

機モニタを用いた OS デバッガへのロギング&リプレイ機能の追加を行った．

軽量仮想計算機モニタを用いた OS デバッガでは，デバッグ対象の OS が NIC や HBA などのハードウェアに直接アクセスを行い，デバイスレジスタアクセスや DMA 転送時に仮想計算機モニタを介さない．ロギング&リプレイ機能の追加を行うためには，これらの履歴の保存・再生が必要になるため，軽量仮想計算機モニタの起動契機付与方式，およびデバッグ対象 OS による直接アクセス動作の追跡方式を新規に設計・実装した．また，これらの方式実現処理におけるデバイス依存処理を極小化・分離する方式も新規に設計・実装した．

起動契機付与方式，および直接アクセス動作の追跡方式の設計は DMA 転送履歴の保存・再生の際に問題となる．本研究では，NIC 受信時の動作を中心に設計・実装を行った．NIC ドライバ動作をモデル化した結果，(1) デバッグ対象 OS（NIC ドライバ）割込み要因レジスタ参照時，受信インデックスレジスタ更新時に軽量仮想計算機モニタの起動契機を与える，(2) 受信ディスクリプタ，受信済みインデックスレジスタの更新履歴の追跡により，1 回の割込みにおいて受信処理された DMA 転送アドレス，サイズ，データ群の取得を行う方式にて履歴の保存・再生が可能であることが分かった．

デバイスレジスタアクセスおよび DMA 転送履歴の保存・再生処理には I/O デバイスに依存する処理が含まれる．そこで，デバイス依存部分を VM 層デバイスドライバとして実装する方針をとった．さらに，これらの履歴の保存・再生処理内容を検証し，本ドライバ実装規模を最小化できるドライバインタフェースを設計してデバイス依存処理の極小化・分離を実現した．Intel 社の NIC 用の VM 層デバイスドライバを実装した結果，約 600 行で実装可能であり，多様な I/O デバイスドライバの開発への本 OS デバッガの適用が容易であることが分かった．

実装した軽量仮想計算機モニタによるロギング&リプレイ機能の実用性を評価するため，ネットワーク受信処理の履歴保存に要する CPU 負荷と消費メモリ量の測定を行った．評価の結果，ロギング&リプレイ処理を行うと，当該処理を行わない場合に比べて CPU 負荷が約 3.5 倍に上昇するが，VMWare によるネットワーク受信処理と比較すると約 38%程度の CPU 負荷の上昇に抑えられることが分かった．また，メモリ消費量の約 98.5%は受信データ履歴の保存であり，ネットワーク受信レートと同等の帯域を確保できる大容量記憶装置があれば，長時間の履歴の保存・再生が可能になることが分かった．

参 考 文 献

1) Streamonix: Streamonix and MikroM Announce Alliance to Provide Video Over IP Streaming Solutions at NAB2005. http://www.streamonix.com/press/Streamonix_and_MikroM_Announce_Alliance.pdf
2) 日立：高性能映像配信サーバシステム「Videonet.tv」を販売開始. http://www.hitachi.co.jp/News/cnews/month/2007/06/0611.html
3) Kasenna: Cost-Effective Video Delivery for Large-Scale VOD Solutions (White Paper). http://www.kasenna.com/downloads/white_papers/LargeScaleVODDeliveryWhitepaper.1.8.03.pdf
4) Intel: Accelerating High-Speed Networking with Intel I/O Acceleration Technology (White Paper). http://download.intel.com/technology/comms/perfnet/download/98856.pdf
5) Kale, A.S.: kgdb: linux kernel source level debugger. http://kgdb.sourceforge.net
6) SGI 社：KDB (Built-in Kernel Debugger). http://oss.sgi.com/projects/kdb
7) Albertsson, L. and Magnusson, P.S.: Using Complete System Simulation for Temporal Debugging of General Purpose Operating Systems and Workloads, *IEEE 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecomunication Systems*, pp.191–198 (2000).
8) 安田絹子ほか：オペレーティングシステム開発環境の設計と実装（言語処理/OS 支援アーキテクチャ，および一般），情報処理学会研究報告「システムソフトウェアとオペレーティング・システム」，Vol.072-011, pp.872–879 (1996).
9) 清水正明ほか：OS デバッグ環境の設計と実装，情報処理学会研究報告「オペレーティング・システム」，Vol.057-001, pp.1–8 (1992).
10) Dunlap, G.W., King, S.T., Cinar, S., Basrai, M.A. and Chen, P.M.: Revirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay, *Proc. 5th Symposium on Operating Systems Design and Implementation*, pp.211–224 (2002).
11) 竹内　理ほか：軽量仮想計算機モニタを用いた OS デバッグ方式の提案，情報処理学会論文誌，Vol.46, No.7, pp.1735–1751 (2005).
12) Intel: Intel 64 and IA-32 Intel Architecture Software Developer's Manual Volumes 3B (2006).
13) Iwasaki, M., Takeuchi, T., Nakano, T. and Nakahara, M.: Isochronous Scheduling and its Application to Traffic Control, *19th IEEE Real-Time System Symposium*, pp.14–25 (1998).
14) 竹内　理ほか：HiTactix-BSD 連動システムを応用した大規模双方向ストリームサーバの設計と実装，情報処理学会論文誌，Vol.43, No.1, pp.137–145 (2002).
15) 竹内　理ほか：外付け I/O エンジン方式を用いたストリームサーバの実現，情報処理学会論文誌，Vol.44, No.7, pp.1680–1694 (2003).
16) Sugerman, J., Venkiachalam, G. and Lim, B.-H.: Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor, *USENIX Annual Technical Conference*, pp.1–14 (2001).

17) Xu, M., Bodik, R. and Hill, M.D.: A “FlightData Recorder” for Enabling Full-system Multiprocessor Deterministic Replay, *Proc. 30th Annual International Symposium on Computer Architecture*, pp.122–135 (2003).
18) de Oliveira, D.A.S., Crandall, J.R., Wassermann, G., Wu, S.F., Su, Z. and Chong, F.T.: ExecRecorder: VM-Based Full-System Replay for Attach Analysis and System Recovery, *Proc. 1st Workshop on Architectural and System Support for Improving Software Dependability*, pp.66–71 (2006).
19) Narayanasamy, S., Pokam, G. and Calder, B.: BugNet: Continuously Recording Program Execution for Deterministic Replay Debugging, *Proc. 32nd Annual International Symposium on Computer Architecture*, pp.284–295 (2005).
20) Srinivasan, S.M., Kandula, S., Andrews, C.R. and Zhou, Y.: Flashback: A Lightweight Extension for Rollback and Deterministic Replay Debugging, *Proc. USENIX Annual Technical Conference 2004*, pp.29–44 (2004).

(平成 20 年 2 月 7 日受付)
(平成 20 年 10 月 7 日採録)

竹内　理（正会員）

1969 年生．1992 年東京大学理学部情報科学科卒業．1994 年同大学大学院理学系研究科情報科学専攻修士課程修了．同年より現在まで（株）日立製作所システム開発研究所．東京大学大学院学際情報学府博士課程在学中．連続メディア処理向きマイクロカーネルの研究，特にリアルタイムスケジューリング方式，リアルタイム通信方式，異種 OS 共存技術，ストリームサーバ構成方式，OS デバッグ方式，仮想計算機モニタの研究に従事．2006 年情報処理学会論文賞受賞．

坂村　健（正会員）

1951 年生．東京大学大学院情報学環教授．工学博士．1984 年からオープンなコンピュータアーキテクチャTRON を構築．現在，TRON はユビキタス環境を実現する重要な組み込み OS として世界で多数使われている．さらに，コンピュータを使った電気製品，家具，住宅，ビル，都市，ミュージアム等広範なデザイン展開を行っている．2002 年 1 月より YRP ユビキタス・ネットワーキング研究所長を兼任．『ユビキタスとは何か』，『変われる国，日本へ』等著書多数．IEEE フェロー，ゴールデンコアメンバ．第 33 回市村学術賞特別賞受賞．2001 年武田賞受賞．2003 年紫綬褒章．2006 年日本学士院賞．