Vol.2012-OS-122 No.12
2012/8/1

VMMを用いた実デバイスドライバの検証環境の設計

島田 恭平[†1,a)]　表 祐志[†1,b)]　品川 高廣[†2]　加藤 和彦[†1]

概要：Windows や Linux などの主要な OS では，システムクラッシュの原因の多くはデバイスドライバの不具合にある．そのため，システムの安定性を維持するためにはデバイスドライバの検証が重要である．しかし，従来のデバイスドライバの検証方法では，デバイスドライバのソースコードが必要であったり，あるいは実マシン上での検証が難しい，といった課題があった．本論文では，デバイスの仮想化を行わない VMM を用いて，実デバイスドライバの検証を行うシステムを提案する．まれにしか発生しない状況での検証を容易に行うために，デバイスの仕様にもとづいて状態遷移を監視するオートマトンを VMM 内に保持し，予め指定したタイミングで I/O やレジスタの値を強制的に変更する．これにより通常はほとんど発生しない状態遷移を意図的に引き起こして，デバイスドライバが正しく動作するか検証する．VMM で I/O を監視・書き換えを行う手法であるため，デバイスドライバのソースコードが不要であるほか，実マシン上のデバイスドライバに対する検証が可能になる．本研究では，ディスクアクセスに使われる AHCI デバイスのデバイスドライバに対して，実際に指定した状態遷移を強制的に引き起こすシステムを実装し，Windows7 及び Ubuntu10.10 のデバイスドライバの検証を行った．

1. はじめに

現在一般的に利用されている Windows や Linux のなどのオペレーティングシステム（OS）において，システムクラッシュの原因の多くは，デバイスドライバにある．例えば，Linux カーネルではデバイスドライバのコードが約 70%を占め，その不具合の密度はデバイスドライバ以外のコードに比べて，3 から 7 倍であることが知られている [1]．そのため，運用するシステムや OS の安定性を維持するために，デバイスドライバの検証を行うことが重要である．しかし，特定のまれな条件やタイミングでのみ起こるような不具合は，再現することが難しく．十分に検証を行えない場合がある．

従来のデバイスドライバの検証を行う手法として，ソースコードを元にした静的解析 [2] があるが，この手法では，デバイスドライバのソースコードが公開されている必要がある．また，検証用のコードを入れるためにデバイスドライバのコードに修正が必要であり，修正することで不具合が再現しなくなる可能性がある．仮想マシン上に作成した仮想デバイスを用いてデバイスドライバのテストを行う手法 [3] もあるが，この手法では，予め検証対象となるデバイスと同じ挙動を示す仮想デバイスを用意する必要がある．すなわち，これらの手法ではデバイスドライバに変更を加えずに，物理マシン上で直接動作するデバイス（実デバイス）のデバイスドライバ（実デバイスドライバ）の検証を行うことが困難である．

本研究では仮想マシンモニタ（VMM）を用いて，OS やハードウェア，デバイスドライバのソースコードに依存せずに，一般的なハードウェア上で実デバイスドライバの検証を行う手法を提案する．まず，デバイスドライバとデバイスの状態を VMM で把握するために，検証対象となるデバイスドライバが動作する OS を，準パススルー型 VMM[4] 上で動作させることにより，実デバイスドライバの I/O を監視する．また，検証対象となるデバイスの状態遷移を監視するために，デバイスの仕様に基づいて状態遷移を監視するオートマトンを VMM 内で動作させる．特定のまれな条件やタイミングで不具合が発生しないことを検証するために，VMM で指定したタイミングで I/O やレジスタの値を強制的に変更する．これにより，仕様では規定されているものの通常はめったに発生しない状態遷移を意図的に引き起こし，デバイスドライバが正しく動作することを検証する．

本研究では，準パススルー型 VMM である BitVisor[4] をベースとして，HDD や SSD などのディスクアクセスの際に使われる標準的なインターフェイスである AHCI（Advanced Host Controller Interface）1.3[5] のデバイスド

†1 現在，筑波大学
Presently with University of Tsukuba
†2 現在，東京大学
Presently with University of Tokyo
a) shimada@osss.cs.tsukuba.ac.jp
b) omote@osss.cs.tsukuba.ac.jp

© 2012 Information Processing Society of Japan

ライバを検証するシステムを実装した．仕様書をもとに状態遷移を監視するオートマトンを実装し，Windows 7 及び Ubuntu 10.10 のデバイスドライバがデバイスにアクセスする際の状態遷移に正しく追随できることを確認した．また，実際に VMM で強制的に状態遷移を発生させて，デバイスドライバが正しく動作することを確認した．

本論文の構成は以下のとおりである．まず 2 章では本研究の提案する検証について述べ，その方法について説明する．3 章では，提案する方法による検証環境のシステム設計について説明し，4 章では，設計した環境の実装の説明を行う．5 章では，その実装した検証環境の動作を確認するための実験を行った結果を示す．5 章では，本研究との関連研究について述べ，最後に 6 章で，本研究の結論を述べ，今後の課題についての検討を行う．

2. デバイスドライバの検証

本章では，提案手法において用いる，デバイスドライバのモデル化について説明した後，デバイスドライバの検証について定義する．またその検証方法について説明する．

2.1 デバイスドライバの状態遷移モデル

デバイスドライバは，OS からデバイスへの I/O やデバイスの動作に応じて，デバイスドライバの状態が変化していると考える．検証対象となるデバイスドライバの各状態をそれぞれ $S_1, S_2, S_3, \ldots$ とし，状態 S_i から S_j への遷移条件を C_{ij} とした，決定性有限オートマトンとして表せる（**図 1**）．本研究では，このデバイスドライバの動作を状態遷移モデルとする．

2.2 デバイスドライバの検証方法

1 つのデバイスドライバの検証項目について，ある状態から，特定の処理が行われた場合に，期待通りの状態に遷移することができた場合に検証が正しいとする．この方法で検証を行うためには，まずデバイスドライバの I/O やレジスタの値を監視し，現在の状態を知る必要がある．また，任意の検証項目について検証を行うために，その検証項目に応じデバイスドライバの I/O に強制的に変更を加える必要がある．

OS やデバイスドライバのソースコードに依存せずにデバイスドライバの検証を行うために，本研究では VMM を用いる．VMM 上で，現在の状態 S_k から，遷移条件 C_{kl} を満たすように，デバイスとデバイス間の I/O に強制的に変更を加える．これによって VMM がデバイスドライバの状態を任意に遷移させることが可能となり，デバイスドライバの状態遷移が正しく行われていることを検証する．

3. システム設計

本研究の提案する，VMM を用いた実デバイスドライバの検証を行うためのシステムの概要を示す．また提案手法を用いた検証方法について説明した後，システムの各モジュールについての説明する．

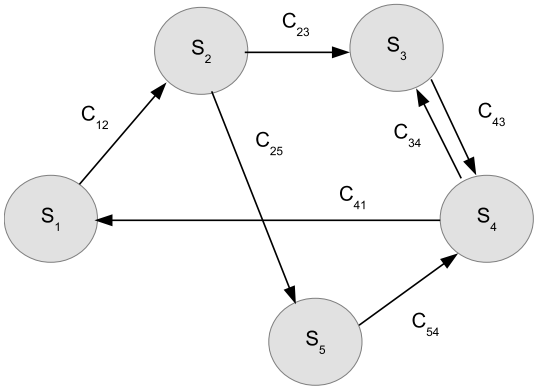

図 1 デバイスドライバの状態遷移例
Fig. 1 An example of state transition model for a device driver.

3.1 システム概要

VMM を用いてデバイスの I/O を監視することで OS や他のデバイスに依存することなく検証を行う．

本システムは**図 2** のように VMM 内にデバイスドライバを検証するためのモジュール（検証用モジュール）を持たせる．検証用モジュールは，検証対象のデバイスドライバからデバイスへの I/O を監視する．監視された検証対象のデバイスドライバの I/O は，検証項目に応じて I/O の内容を書き換える．また必要に応じてデバッグ出力の表示を行う．一方で，その他のデバイスドライバからデバイスへの I/O は，特別な処理は行わず，直接ハードウェアにアクセスする．

このようなシステムを構築することによって，任意のデバイスドライバの I/O を制御し，検証項目に応じた状態を再現することが可能となり，かつデバッグ出力を行える環境が構築できる．この方法を用いることで，実デバイスドライバの検証を，デバイスドライバのソースコードを必要とせずに行うことが可能となる．

3.2 検証用モジュール

検証用モジュールは，さらに役割ごとに，I/O 制御モジュール，状態制御モジュール，デバッグ出力モジュール，シナリオコードの 4 つに分かれる．それぞれの機能および役割の詳細は次の通りである．

- シナリオコード
  シナリオコードは，検証対象となるデバイスドライバの検証内容に合わせて，デバイスドライバの I/O の変更条件や，変更内容を記述したプログラムである．
- I/O 制御モジュール
  I/O 制御モジュールは，検証対象となるデバイスドラ

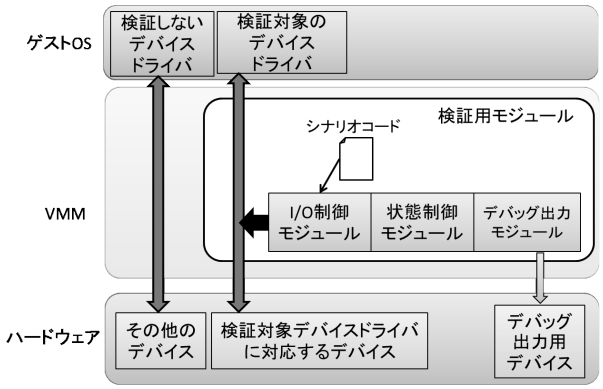

図 2 システム概要図
Fig. 2 The system architecture

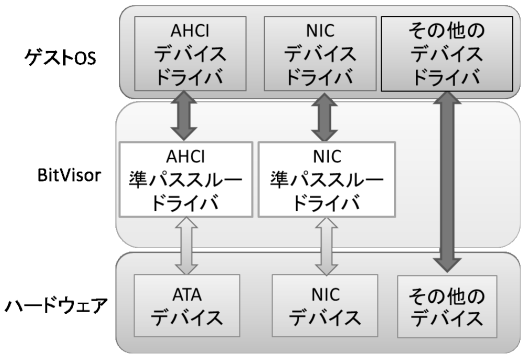

図 3 準パススルー型アーキテクチャ概要図
Fig. 3 Parapass-through architecture.

イバの I/O を監視する．検証項目によって用意されるシナリオコードに応じて，I/O の内容に変更を加えるモジュールでである．シナリオコードは検証項目に応じて，状態遷移を行うために I/O の変更条件等を記述したプログラムである．

- 状態制御モジュール
  検証対象のデバイスドライバの現在の状態を調べ，その状態から別の状態への遷移があるかを調べ，その制御を行うためのモジュールである．
- デバッグ出力モジュール
  デバッグ出力モジュールは，検証対象のデバイスドライバの情報や VMM の情報を確認するための付加的な情報を出力するために使用する．これを用いることで，レジスタの値や，検証対象のデバイスドライバの I/O のより詳細な情報の確認を行える．デバッグ出力は，検証対象のデバイスと直接関係のないデバイスを通じて行う．これは，検証対象のデバイスの動作が，デバッグ出力によって変化してしまう事を防ぐためである．

4. 実装

本システムの具体的な実装について述べる．実装に用いる BitVisor の設計や，具体的な機能について説明し，本システムの実装内容を説明する．ここでは例として，AHCI のデバイスドライバを検証対象とした場合の実装について示す．

4.1 準パススルー型アーキテクチャ

前章のシステム設計に基づき，実装を行う際に，VMM において検証対象デバイスドライバの I/O のみを監視する必要がある．そのため VMM は，準パススルー型アーキテクチャを用いる．準パススルー型アーキテクチャは必要最低限のデバイスとの I/O のみ仮想化を行い，それ以外のデバイスへの I/O は直接ハードウェアにアクセスする設計手法である．

4.2 BitVisor

BitVisor は本来，セキュリティ強化の目的のために作成された VMM である．BitVisor は準パススルー型アーキテクチャに基づいて設計されており，ゲスト OS とは別に，**図 3** のように，一部のデバイスの I/O のみを仮想化するための準パススルードライバを持つ．また，BitVisor は Type I 型 VMM であるため，ホスト OS を必要とせず，ゲスト OS は 1 つである．また準パススルー型アーキテクチャである BitVisor を用いることで，検証対象デバイスドライバ以外の I/O はハードウェアに直接アクセスし，検証対象となるわずかな I/O にのみ処理を加えるため，実デバイスドライバを実マシン上に極力近い環境で検証を行うことが可能である．本研究では，この BitVisor の準パススルードライバ中に，検証対象となるデバイスドライバ用の検証モジュールの実装を行った．

4.3 実装の詳細

本システムの実装は，BitVisor の AHCI 準パススルードライバを改変し，検証用モジュールを実装することで実現する．検証用モジュールは I/O 制御モジュール，状態確認モジュール，デバッグ出力モジュールの 3 つに分けられる．これら 3 つのモジュールは BitVisor の初期化にあわせて初期化を行う．各モジュールの詳細な実装については次の通りである．

- I/O 制御モジュール
  I/O 監視モジュールは，**図 4** のような手順で，検証対象となる AHCI のデバイスドライバの I/O を制御する．まず検証項目の必要に応じて作成されたシナリオコードをもとに，検証対象のデバイスドライバで発生した MMIO（Memory Mapped I/O）に変更，追加が必要であるかを確認する．MMIO の変更が必要であればシナリオコードに記述された処理に基づき，MMIO を強制的に変更，追加する．
- 状態制御モジュール
  状態制御モジュールは，現在デバイスドライバがどの状態であるかを保持する．また現在の状態から別の状

© 2012 Information Processing Society of Japan

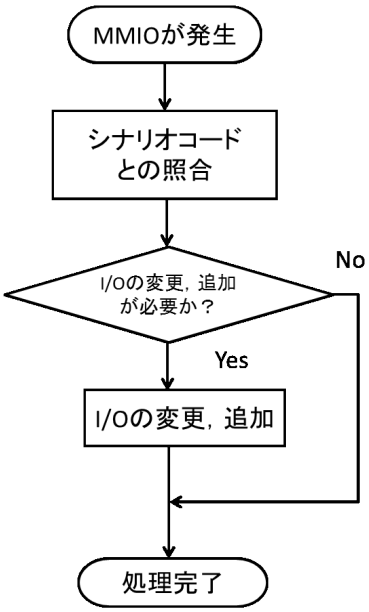

図 4 I/O 制御モジュールのフローチャート
Fig. 4 A flowchart of the I/O control module.

態への遷移があるかを調べ，状態遷移が行われた場合は，遷移した状態とその時間を記録する．記録したデータは VMM のメモリ上に保持される．また検証対象のデバイスドライバによっては，デバイスドライバから取得できない内部変数を持つものがあるため，その変数を擬似的に表す変数が必要な場合は，その変数も VMM 上で保持し．このモジュールで管理する．状態の遷移を記録した内容は，VMCALL を通じてゲスト OS で取得する．

- デバッグ出力モジュール
  BitVisor にはデバッグ出力を行うための機構が用意されている．BitVisor はネットワークを介して別のマシンに UDP によってデバッグ文字列を送信することができる．これを利用することで VMM の情報や，監視した各 I/O の情報を検証者が知ることができる．

5. 動作実験

実装した提案システムを用いて，Windows 7, Ubuntu 10.10 それぞれの OS 上で動作する AHCI のデバイスドライバについて，実際にデバイスドライバの状態遷移が正しく行われることを確認するための動作実験を行った．まず，VMM において I/O を変化させずに本システムを動作させ，デバイスドライバの状態遷移の様子を確認する．次に，VMM で I/O を変化させ，本システムでデバイスドライバの状態遷移を強制的に行えることを確認する．

5.1 実験環境

本実験において検証対象となるデバイスドライバは，AHCI のデバイスドライバを例に実験を行う．OS はストレージに Windows 7 および Ubuntu10.10 のデュアルブート環境をインストールし，それぞれの環境で同様の実験を行う．実験に用いた PC のスペックを表 1 に示す．また検証を行うデバイスドライバは表 2 に示す．AHCI はポートごとにそれぞれ状態を持つため，今回の実験はポート 0 を例に対して状態の確認を行った．

表 1 実験に用いたマシンのハードウェアの構成
Table 1 The hardware structure for experimentation,

CPU	Intel Core 2 Duo CPU E8600 @ 3.33GHz
メモリ	4GB
ストレージ	Intel X25-M Mainstream SSDSA2MH080G1C5 80GB

表 2 検証対象デバイスドライバ
Table 2 Target device drivers of the verification.

OS	デバイスドライバ
Windows 7	Standard AHCI 1.0 Serial ATA Controller
Ubuntu 10.10	Linux kernel 2.6.35-22 AHCI driver

5.2 実験 1 通常のデバイスドライバの状態遷移

5.2.1 実験内容

通常の OS 起動時における状態遷移の情報を得るために，コンピュータ起動時から各ゲスト OS が起動するまでの状態遷移を本システムを用いて調査を行う．このとき VMM で I/O の変更は行わず，OS 起動後，すぐに状態遷移の履歴を VMM から取得するためのコードを実行させた場合の，AHCI のデバイスドライバの状態遷移の流れを確認する．

5.2.2 実験結果

VMM において，I/O の変更を行わない場合の状態遷移は Windows7，Ubuntu10.10 のそれぞれにおいて，同様に次のような順序で状態遷移の結果が得られ，各状態に遷移した回数は，表 3 のようになった．各状態の名称は AHCI の仕様書 [5] における各状態の名称を使用した．

(1) `P:Init`（初期状態）
(2) `P:NotRunning`
(3) `P:RegFisPostToMem`
(4) `P:NotRunning`
(5) `P:Idle`
(6) `P:StartBitCleared`
(7) `P:NotRunning`
(8) `P:RegFisPostToMem`
(9) `P:NotRunning`
(10) `P:Idle`
(11) `NDR:Entry`
(12) `NDR:Accept`
(13) (6) から (12) の繰り返しが続く

このように初期状態を除き，特定のいくつかの状態を繰り返し遷移していることが確認できた．

表 3　各状態への到達回数
(VMM による I/O の変更を行わない場合)

Table 3　Counts of transition to states
(without forced change I/Os)

状態名	Windows 7	Ubuntu 10.10
NDR:Accept	3448	4444
NDR:Entry	3448	4444
P:Idle	3450	4446
P:Init	1	1
P:NotRunning	6901	8893
P:RegFisPostToMem	3450	4446
P:StartBitCleared	3450	4446

表 4　各状態の到達回数
(VMM による I/O の変更を行った場合)

Table 4　Counts of transition to states
(with forced change I/Os)

状態名	Windows 7	Ubuntu 10.10
NDR:Accept	3451	4415
NDR:Entry	3455	4420
P:Idle	3457	4422
P:Init	1	1
P:NotRunning	6915	8845
P:RegFisPostToMem	3457	4422
P:StartBitCleared	3457	4422

5.3　実験 2　デバイスドライバの状態の強制遷移

5.3.1　実験内容

AHCI の仕様書 [5] に基づくと状態 NDR:Entry は無条件で次の条件 NDR:Accept に遷移する．そこで本システムにおいて，現在のデバイスドライバの状態が NDR:Entry の場合において，状態 P:StartBitCleared に強制的に遷移させるため，ソフトウェアから特定のレジスタの特定のビット値を，1 から 0 に書き換える．この遷移が発生する確率は，現在の状態が NDR:Entry の時に 0.1 とし，この処理をパターンコードとして記述した．それ以外の条件は実験 1 と同様の環境で行い，VMM で I/O を変更させた場合を確認し，VMM で I/O を変更しない場合と比較する．

5.3.2　実験結果

各状態の遷移した回数を表 4 のような結果となった．また VMM のデバッグ出力において，Windows7 においては 4 回，Ubuntu10.10 においては 5 回レジスタの書きかえが行われたことが確認された．これは NDR:Accept の到達回数から NDR:Entry の到達回数を引いたものと等しくなっている．このことから状態遷移が強制的に行われていることがわかる．

状態 P:StartBitCleared に VMM で強制的に遷移させた場合を除き，状態遷移の結果は実験 1 と同様のものとなった．また，状態 P:StartBitCleared に VMM で強制的に遷移させた場合，仕様どおり P:StartBitCleared から P:NotRunning に遷移していることも確認できた．このことから，Windows 7, Ubuntu 10.10 上の各デバイスドライバは，本実験でパターンコードに記述した検証項目について，正常に動作していることが確認できる．

6.　関連研究

本研究の関連研究として，様々な手法でデバイスドライバを検証を行う方法が提案されている．Carburizer[6] はデバイスドライバのソースコードを解析し，バグを修復した新しいソースコードを生成する．それをコンパイルして作成した新しいドライバを使用する．また Carburizer はドライバの実行時の不具合を検出し，自動的で復旧させる．Carburizer はデバイスドライバの検証を行うが，あらかじめデバイスドライバソースコードが必要となるが，本研究ではデバイスドライバのソースコードを必要とせず，デバイスドライバの修正も行わない．

DDT[3] はバイナリデバイスドライバをテストを行うシステムである．DDT は，実行パスがバッファオーバーフローやカーネルがクラッシュするような予期せぬ振る舞いをするものかどうかを確認する．DDT は特別な VM を用意し，VM レベルでのメモリアクセスチェックとゲスト OS レベルでの検証を行う．DDT はバイナリのデバイスドライバにおいても検証を行えるが，仮想マシン上に専用の仮想デバイスを必要とし，実マシン上では検証を行えない．一方本研究では，実マシン上で検証を行う．

独自のランタイムを用いてデバイスドライバの耐障害性を向上させる研究 [7] がある．これは，本来ハードウェア資源にアクセスするためにカーネルモードで動作するデバイスドライバを，ユーザモードで実行させるものである．ユーザモードで実行されたデバイスドライバとデバイスの I/O は RVM（reference validation mechanism）[8] を経由させる．またデバイスドライバは実際のハードウェアからは隔離する．RVM が安全でないような I/O を確認することで，デバイスドライバをユーザモードで安全に使用することが可能となる．この手法はデバイスドライバの I/O を監視するという点で同一のアプローチである．しかし，デバイスの検証を行わずに，デバイスドライバの安全でない I/O を抑止することで，デバイスドライバの耐障害性を補助的に向上させることを目的としており．本研究にの目的である，デバイスドライバそのものの検証を行うものではない．

7.　結論

本研究では，従来のデバイスドライバの検証方法の問題について論じ，既存の研究とその解決手法について調査を行った．そして VMM を用いた実デバイスドライバを検証する方法を提案した．そして，実際に VMM を用いて実デバイスと実デバイスドライバ間の I/O の監視，変更を行う

© 2012 Information Processing Society of Japan

ことで，任意の処理を行い，OS やデバイスドライバのソースコード，その他のハードウェア環境に依存しない実デバイスドライバの検証環境を構築した．その構築した環境を用いて，実際に AHCI を対象とした実デバイスドライバを用いて実験を行った．実験を行った結果，OS に依存せず，またデバイスドライバのソースコードを必要とせずに，実デバイスドライバの状態遷移の変化を確認することができた．さらに，状態を VMM を用いて強制的に遷移させることができた．

本研究の課題としては，デバイスドライバの仕様から，シナリオコードを作成する方法を検討することが挙げられる．今回の実験では，ランダムなどの単純な I/O の書き換えしか行っておらず，また手動で BitVisor のソースコードを書き換える必要がある．より複雑な検証を行うために，デバイスの仕様から検証項目を見つけ，適切なシナリオコードを作成することが必要である．

参考文献

[1] Audy Chou, Junfeng Yang, Benjamin Chelf, Seth Hallem, and Dawson Engler.: *An empirical study of operating systems errors*, In Proceedings of the eighteenth ACM symposium on Operating systems princhiples, SOSP'01, pp73-88, New York, NY, USA, 2011.

[2] Tomas Ball, Ella Bounimova, Byron Cook, Vladimir Levin, Jakob Lichtenberg, Con McGravery, Bohus Ondrusek, Sriram K. Rajamani, and Abdullah Ustuner.: *Throught Static Analysis of Device Drivers*, In Proceeding of the 1st ACM SIGOPS/EuroSys European Conference on Computer Sysytems 2006, EuroSys '06, pp73-86, New York, NY, USA, 2006.

[3] Volodymyr Kuznetsov, Vitaly Chipounov, and George Candea.: *Testing Closed-Source Binary Device Drivers with DDT*, In Proceeding of the 2010 USENIX conference on USENIX annual technical conference, USENIX-ATC'10, USENIX Association Berkeley, CA, USA, 2010.

[4] Takahiro Shinagawa, Hideki Eiraku, Kouichi Tanimoto, Kazumasa Omote, Shoichi Hasegawa, Takashi Horie, Manabu Hirano, Kenichi Kourai, Yoshihiro Oyama, Eiji Kawai, Kengi Kono, Shigeru Chiba, Yasushi Shinjo, and Kazuhiko Kato.: *BitVisor: A Thin Hypervisor for Enforcing I/O Device Security*, In Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conerence on Virtual execution environments, VEE'09, pp121-130, New York, NY, USA, 2009.

[5] Intel Corporation:
Serial ATA AHCI: Specification, Rev. 1.3（online），入手先 ⟨http://www.intel.com/content/www/us/en/io/serial-ata/serial-ata-ahci-spec-rev1_3.html⟩ (2012.06.26).

[6] Asim Kadav, Matthew J. Renzelmann, and Michael M. Swift.: *Tolerating Hardware Device Failures in Software*, In Proceedings of the ACM SIGOPS 22nd symposium on Operatng systems principles, SOSP'09, pp59-71, New York, NY, USA, 2009.

[7] Dan Williams, Patrick Reynolds, Kevin Walsh, Emin Gü n Sirer, Fred B. Schneider. *Device driver safety through a reference validation mechanism*, In Proceedings of the 8th USENIX conference on Operating systems design and implementation, OSDI'08, USENIX Association Berkeley, CA, USA, 2008.

[8] James P. Anderson.: *Computer security technology planning study Vol.2*, , Technical Report ESD-TR-73-51, Electronic Systems Division, AFSC, L.G. Hanscom Field, Bedford, Massachusetts, Oct. 1972.

© 2012 Information Processing Society of Japan