6A-5

仮想計算機上での I/O デバイス仮想化方式

宮内　大†　　伊藤　孝之†　　鶴　薫†

†三菱電機株式会社　情報技術総合研究所

1. はじめに

近年，物理リソースの有効活用のため，計算機を仮想化することが一般的となってきている．その中で物理 I/O デバイスを仮想化する手法としては，完全仮想化と準仮想化が存在する（KVM[1]や Xen[2]など）．これらの方法で仮想化された I/O デバイスをゲスト OS から利用することができ，これによって，H/W 更新時にゲスト OS 側の環境を変更することなく，システムを更新できるメリットを享受することが可能となる．

しかし，一般的には仮想化されていない I/O デバイスを独自に仮想化する場合，従来の方式では，仮想化の対象とする物理 I/O デバイスごとに仮想マシンモニタ（VMM）の内部構造に即した開発が必要であるという課題がある．本稿では，この課題を解決する方式について述べる．

2. 従来の方式と課題

本章では，従来の完全仮想化と準仮想化による物理 I/O デバイスの仮想化方式と課題について述べる．

2.1. 完全仮想化

完全仮想化とは，実在するハードウェアを VMM で完全にエミュレートしてゲスト OS に認識させる方式である．I/O デバイスについても同様であり，ゲスト OS からは，通常のデバイスドライバ（標準ドライバ）で，仮想 I/O デバイスを制御し，仮想 I/O デバイスが，ホスト OS 上の標準ドライバを利用して物理 I/O デバイスを制御する（図 1）．

完全仮想化を利用する場合，実在する物理 I/O デバイスを完全にエミュレートする機能の開発が必要であるが，これは，VMM の内部構造や対象の物理 I/O デバイスの詳細を把握した上での開発となる．そのため，仮想化の対象とする物理 I/O デバイスごとに複雑なエミュレート機能を開発する必要があるという課題がある．

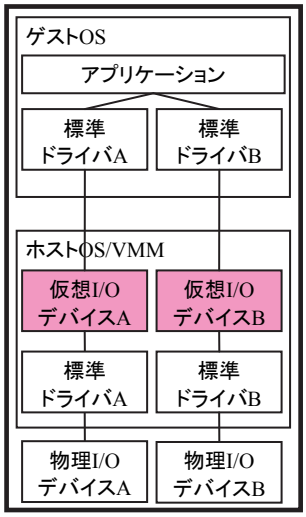

図 1. 完全仮想化

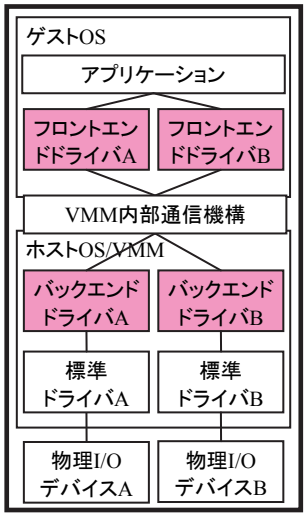

図 2. 準仮想化

2.2. 準仮想化

準仮想化とは，ゲスト OS やドライバに仮想化を意識した処理を行うための修正を加え，仮想化によるオーバーヘッドを減らす方式である．そのため，ゲスト OS とホスト OS に VMM とのインタフェース（I/F）となるドライバを組み込む．このゲスト OS 側のドライバをフロントエンドドライバ，ホスト OS 側のドライバをバックエンドドライバと呼ぶ．ゲスト OS での I/O デバイスの制御は，まず，フロントエンドドライバから，VMM の内部通信機構を利用して，バックエンドドライバにデバイスの制御データを渡す．続いて，バックエンドドライバが標準ドライバ用のデータ構造に変換し，標準ドライバを介して，物理 I/O デバイスを制御する（図 2）．

準仮想化を利用する場合，仮想化の対象とする物理 I/O デバイスごとに特化したフロントエンドドライバとバックエンドドライバの開発が必要となる．これは，フロントエンドドライバからバックエンドドライバに渡された制御データを，バックエンドドライバが，物理 I/O デバイスを制御できるデータ構造に変換する必要があるためである．つまり，完全仮想化と同じく，VMM の内部構造の詳細を把握した上での対応が，仮想化の対象となる物理 I/O デバイスごとに必要であるという課題がある．

Virtualization method for I/O devices.
Dai Miyauchi†, Takayuki Ito† and Kaoru Tsuru†
†Information Technology R&D Center, Mitsubishi Electric Corporation.

3. 解決方式

本章では，2 章の課題を解決する方式を述べる．この方式により，対象の物理 I/O デバイスごとの VMM の内部構造を把握した開発を不要とし，かつ，H/W 更新時にゲスト OS 側の環境を変更することなくシステム更新が可能となる．

3.1. 構成

構成を図 3 に示す．ゲスト OS 側に準仮想化方式と同じように対象の I/O デバイスごとにフロントエンドドライバを設置する．さらに，仮想化する各 I/O デバイスで共通となる標準ドライバ呼出し機構①を設置する．ホスト OS 側には，対象の物理 I/O デバイスの標準ドライバと仮想化する各 I/O デバイスで共通となる標準ドライバ呼出し機構②を設置する．標準ドライバ呼出し機構は，ゲスト OS 上のフロントエンドドライバに対して，ホスト OS 上の標準ドライバを呼び出すことを可能とする共通の I/F を提供する．これにより，仮想化する I/O デバイスごとに複雑な VMM の内部構造を意識した開発が不要となる．また，アプリケーションとフロントエンドドライバ，標準ドライバ呼出し機構②と標準ドライバ間の I/F の仕様を同じとすることで，アプリケーション側での I/O デバイス制御においても仮想化を意識する必要はなくなる．

例えば，ゲスト OS のアプリケーションが open()を呼び出す処理を実行する場合，次のような動作となる．アプリケーションからフロントエンドドライバの open()が標準ドライバの I/F と同じ I/F で呼び出されると，フロントエンドドライバは，対応するホスト OS 上の標準ドライバのパスとアプリケーションから渡された open 時のフラグを標準ドライバ呼出し機構①を介して，ホスト OS 側の標準ドライバ呼出し機構②に渡す．標準ドライバ呼出し機構②は要求に応じてホスト OS の標準ドライバを呼び出す．標準ドライバから返されたファイルディスクリプタは，標準ドライバ呼出し機構を通して，フロントエンドドライバ，アプリケーションの順に返される．

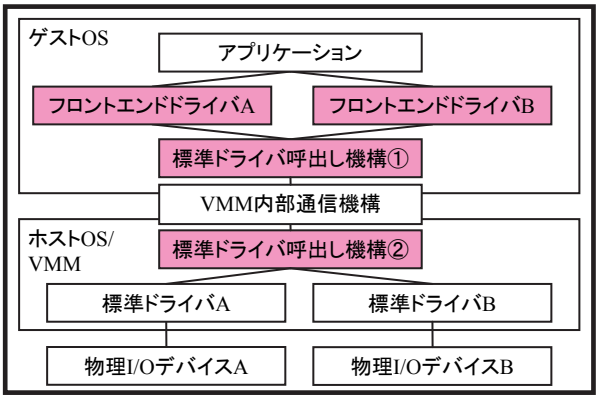

図 3. 解決方式

表 1. 評価項目

評価項目	評価方法	結果
一般のドライバ I/F の呼出し	ホスト OS の RAM ディスク/dev/ram0 へのアクセスによる open, close, read, write	○
ioctl のドライバ I/F の呼出し	ホスト OS の CD/DVD ドライバ /dev/sr0 を用いた，cdrecord コマンドによる CD-RW ドライブに対する ioctl	○
I/O デバイスの共有	複数のゲスト OS から，1 台の CD-RW ドライブを排他的に利用	○

3.2. データ受け渡し方式

デバイスの制御データには一般的にポインタが含まれることも多いが，3.1 の方式において，単純なゲスト OS とホスト OS 間の制御データの受け渡しでは問題が生じる．ゲスト OS 上のアプリケーションからフロントエンドドライバに渡される制御データ内のポインタはゲスト OS から参照可能な論理アドレスを参照することになるが，これはホスト OS の標準ドライバに渡されたときには参照できないためである．この問題を解決するため，標準ドライバ呼出し機構にメモリのマッピング処理を実装し，ホスト OS から参照可能な論理アドレスにマップし，制御データをこれに置き換えることで対処する．

4. 評価

VMM の 1 つである KVM 上で標準ドライバ呼出し機構の実装を行い，RAM ディスクおよび CD-RW/DVD-RW ドライブによる動作確認を実施した．結果，VMM の内部構造を意識せず，かつ，ホスト OS 側の標準ドライバを変更することなく，ゲスト OS から表 1 の制御が行えた．

5. おわりに

I/O デバイスの仮想化の際に，対象の I/O デバイスごとに VMM の内部構造を意識する必要のない実装方式を考案し，フロントエンドドライバから VMM の内部構造の隠蔽とバックエンドドライバの統一化に成功した．今後は，他の I/O デバイスについても本方式が有効であることを確認していく．

参考文献

[1] A. Kivity et al. : kvm: the Linux virtual machine monitor, Proceedings of the Linux Symposium, pp. 225-230, 2007.
[2] P. Barham et al. : Xen and the art of virtualization, Proceedings of the nineteenth ACM symposium on Operating systems principles, pp. 164-177, 2003.