2007－OS－104 (3)
2007／1／30

カーネル・レベル・コードによるユーザ・レベルVMMの移植性の向上

榮樂 英樹† 新城 靖†‡ 加藤 和彦†‡

†筑波大学大学院システム情報工学研究科コンピュータサイエンス専攻
‡科学技術振興機構 CREST

要旨

LilyVMは，静的なコード変換技術と準仮想化技術を用いて，実機用のOSをユーザ・プロセスとして動作させる仮想マシン・モニタである．従来のLilyVMでは，多くの特殊な機能をホストOSに要求しており，他のOSへの移植性が低く，性能にも問題があった．この論文では，LilyVMにおいてカーネル・レベル・コードを用いることにより，他のOSへの移植性を高めることについて述べている．カーネル・レベル・コードは，高い特権レベルで動作し，CPU例外のリダイレクトをOSに依存せずに実行する．

Improving Portability of a User-Level VMM by Kernel-Level Code

Hideki Eiraku† Yasushi Shinjo†‡ Kazuhiko Kato†‡

†Department of Computer Science, University of Tsukuba
‡Japan Science and Technology Agency CREST

Abstract

LilyVM is a virtual machine monitor (VMM) which runs an operating system (OS) for a real machine as a user process using a static code translation technique and a paravirtualization technique. Former LilyVM required many special features of host OSes. Therefore, it had less portability to other OSes, and also had a performance problem. In this paper, we discuss about improving portability of LilyVM to other OSes by using a kernel-level code. The kernel-level code runs in a higher privilege level and redirects CPU exceptions without dependence on a host OS.

1 はじめに

1台のコンピュータで複数のオペレーティング・システム（Operating Systems，以下OSと略す）を同時に動作させることが，一般的に行われるようになってきている．その目的としては，UnixとWindowsのように，その上で動作する異なるタイプの応用プログラムを組み合わせて利用することや，Mac OS 9とMac OS Xのようにオペレーティング・システムの進化を円滑に進めることなどがあげられる．この論文では，1つの基盤となるOSを動作させ，その上で別のOSを動作させることについて述べる．基盤となるOSを**ホストOS**，ホストOS上で実行されるOSを**ゲストOS**と呼ぶ．

ソフトウェアによるエミュレーションで仮想化された実行環境を**仮想マシン**と呼ぶ．ゲストOSによる仮想的なハードウェアとの入出力をモニタし，エミュレートするものを，**仮想マシン・モニタ**（Virtual Machine Monitor，以下**VMM**と略す）と呼ぶ．一般にVMMは，実際のハードウェアをエミュレートすることにより，実機用の任意のOSを実行する

ことができる．このような仮想化技術を，**完全仮想化**と呼ぶ．完全仮想化に基づくVMMの欠点としては，CPUをほぼ完全にエミュレートしなければならないので，実現が難しいこと，高速化のためにハードウェアによる支援や機械語命令の動的な書き換えが必要になってしまうこと，および，動的な書き換えに伴うオーバーヘッドが存在することがある．完全仮想化に対して，仮想化に都合の良いアーキテクチャを定義し，ゲストOSをそのアーキテクチャに移植することによって，仮想マシン上でOSを実行できるようにしたものがある．これは，**準仮想化**（paravirtualization）と呼ばれる．準仮想化の問題点は，実機用のゲストOSを移植しなければならないことである．

我々は，IA-32を対象として準仮想化技術を用いたVMMであるLilyVMを開発している．LilyVMの特徴は，機械語命令の静的な変換により，ゲストOSの移植を不要にしている点にある[4]．LilyVMでは，ゲストOSとしてNetBSD, FreeBSDおよびLinuxが動作している．また，LilyVMは，ホストOSとしてNetBSD, FreeBSDおよびLinuxの上で動作している．従来のLilyVMは，ユーザ・レベルのコードのみを用いて実現されており，以下のようなホストOSが提供している機能を使用していた．

- 指定したアドレスにメモリをマップする機能．
- システム・コールを検出する機能．
- 例外の詳細情報（割り込み番号や，エラー・コードなど）を取得する機能．

これらは一般的な機能ではないため，移植性が低いという問題があった．また，以下のようなオーバヘッドが大きかった．

- フォールトおよびトラップのエミュレーション．
- MMUのエミュレーション．
- FPUレジスタ切り替え．

この論文では，ホストOSにカーネル・レベル・コードを追加することにより，これらの問題を解決する方法について述べる．カーネル・レベル・コードが，ホストOSを介さずにCPUを制御することにより，ホストOSへの依存度を小さくする．これによって，従来のLilyVMで実現していなかった，Solarisへの移植[6]およびWindows XPへの移植を実現した．また，カーネル・レベル・コードを利用することで上に示したオーバヘッドを削減する．フォールトおよびトラップは，カーネル・レベルのVMMでエミュレーションを行うことで高速に処理する．MMUのエミュレーションは，カーネル・レベル・コードでページ・テーブルを操作することで，オーバヘッドを削減する．FPUレジスタの切り替えは，ハイパバイザ・コールを用いて，FPUレジスタを使用している場合にのみ行われるようにする．これにより，Linuxカーネルの構築にかかる時間を約半分に短縮している．

2 関連研究

VMware Workstation[8]は，IA-32，および，x86-64（AMD64, Intel 64）を対象とした完全仮想化に基づくVMMである．ホストOSとして，Linux，Windows，および，Mac OS X上で動作する．VMware Workstationでは実際のハードウェアをエミュレートしており，実機用の様々なOSをゲストOSとしてそのまま実行することができる．VMware Workstationは，本方式と同じように，カーネル・レベルのVMMを追加しているが，本方式では，ゲストOSとホストOSが協調して動作するような機能を容易に追加できるようにしている点が異なる．例えば，LilyVMではユーザ・レベルOSからホストOSのTCP/IPスタックを利用することができる[5]．

coLinux[1]は，IA-32を対象として，ゲストOSとして動作するように移植されたLinuxである．ホストOSとして，Windows，および，Linux上で動作する．coLinuxでは，ゲストOSをカーネル・レベルで動作させているため，ゲストOSがクラッシュすると，ホストOSを含めてシステム全体がクラッシュすることがある．本方式は，ゲストOSに対して特権を与えていないため，システム全体がクラッシュすることがない．また，本方式では，ゲストOSに対する修正を言語処理系で自動化している点が異なる．

Xen[3]は，IA-32，x86-64，および，IA-64などを対象とした準仮想化に基づくVMMである．Xenはホスト OS上で動作するのではなく，実機上で動作する．Xen 2.0で実現される仮想マシンのハードウェアは，実際のハードウェアとは異なるため，ゲストOSに対する修正が必要となる．Xen 3.0では，CPUによる仮想化機能に対応し，従来の準仮想化による仮想マシンに加えて，完全仮想化による仮想マシンも作成できるようになった[9]．ただし完全

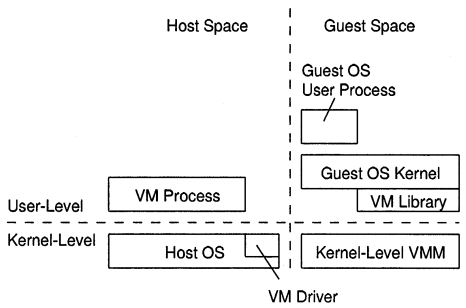

図 1: 本研究で提案する仮想マシン・モニタの構成

仮想化を用いると性能が低下する．本方式は，ホスト OS 上で動作するという点と，ゲスト OS に対する修正を言語処理系で自動化している点が異なる．

User-mode Linux[2] は，Linux を，他の Linux に移植したユーザ・レベル OS である．本方式は，ゲスト OS に対する修正を言語処理系で自動化している点が異なる．

3 基本設計

本研究で提案する VMM の構成を図 1 に示す．図において，ホスト空間（Host Space）は，ホスト OS のカーネルとそのユーザ・プロセスが動作するアドレス空間を表し，ゲスト空間（Guest Space）は，ゲスト OS のカーネル，そのユーザ・プロセスと，VMM が動作するアドレス空間を表している．LilyVM は以下の 4 つの要素から構成される．

- VM プロセス（VM process）．ホスト OS 上でユーザ・プロセスとして動作し，仮想マシン環境の資源を持つ．
- VM ドライバ（VM Driver）．VM プロセス，および，カーネル・レベル VMM との間の通信を行う．
- カーネル・レベル VMM（Kernel-level VMM）．ゲスト空間において高い特権レベルで動作し，例外，および，割り込み処理を行う．
- VM ライブラリ（VM library）．ゲスト OS カーネルの特権命令などのエミュレーションを行う．

ゲスト OS のカーネル（Guest OS Kernel）は，従来の LilyVM で用いているものと同様に，機械語命令の静的な変換を行ったものである．すなわち，特権命令や，センシティブな非特権命令が，VM ライブラリ内の手続きを呼び出す命令に変換されている．また，カーネル・レベル VMM，および，VM ライブラリを同じアドレス空間に入れるため，アドレス空間の定数を変更している．また，ゲスト OS のカーネルは，入出力のための特別なデバイス・ドライバを含む．

また，ゲスト OS のユーザ・プロセス（Guest OS User Process）は，従来の LilyVM で用いているものと同様に，実機用のものをそのまま使用する．これにより，実機用のバイナリとして配布されている豊富なアプリケーションをそのまま活用することができる．

LilyVM の構成要素の中で，ホスト OS に依存している部分は，VM ドライバ，および，VM プロセスのメモリを確保する部分と，VM ドライバとの通信部分である．VM プロセスのその他の部分と，カーネル・レベル VMM，および，VM ライブラリは，ホスト OS に依存していない．このように，多くの構成要素をホスト OS に依存しないように作成することで，移植性を高めている．以下では，これらの構成要素の働きについて述べる．

3.1 VM プロセス

VM プロセスは，以下のような処理を行うユーザ・プロセスである．

- 仮想マシン環境の資源（CPU 時間やメモリ）を持つ．
- ゲスト OS を実行するための準備を行う．例えば，仮想マシン環境のためのメモリを確保したり，ゲスト OS カーネルを読み込んだりする．
- ホスト OS のシステム・コールを発行し，入出力を行う．
- 仮想マシンのハードウェア割り込みをエミュレートする．

VM プロセスは，ゲスト OS 実行の準備ができたら，VM ドライバに対してゲスト OS の実行を命令する．すると，ハードウェア割り込みが発生するか，ゲスト OS において停止命令が実行されるか，ディスクドライブ，ネットワーク等への入出力要求があるまでの間，ゲスト OS が実行される．VM プロセスは，ゲスト OS の実行が停止すると，仮想マシンのハードウェア割り込みをエミュレートするか，入出力の要求を処理し，VM ドライバに対してゲスト

OSの実行を続行させる．VMプロセスが入出力要求を処理することにより，入出力を標準的なAPIで実装できるため，移植性が向上している．

3.2 VMドライバ

VMドライバは，ホストOSのデバイス・ドライバとして高い特権レベルで動作する．VMドライバは，VMプロセス，および，カーネル・レベルVMMと通信し，割り込みベクタ・テーブルやセグメント等を切り替え，ゲスト空間に切り替える機能を持つ．

VMドライバは，VMプロセスからゲストOSの実行を命令されると，アドレス空間や割り込みベクタ・テーブルなどを，ゲスト空間に切り替える．実行がゲスト空間から戻ると，VMドライバは，ゲストOSが停止した原因を調べる．もし，ハードウェア割り込みが原因であった場合は，本来のホストOSの割り込み処理ルーチンを呼び出す．

VMドライバは，ホストOSに依存している．実際に必要な機能の一覧を表1に示す．表には，Linuxにおける関数名を示した．これらの機能は，一般的なOSのカーネルで提供されているものであり，ホストOSへの依存度は小さくなっている．

3.3 カーネル・レベルVMM

カーネル・レベルVMMは，ゲストOSを実行している間に発生した例外や，ゲストOSユーザ・プロセスが発行するシステム・コールを，ゲストOSへリダイレクトする．この処理は，ホストOSの例外ハンドラに依存しないので，移植性が向上している．

ゲストOSを実行している間にハードウェア割り込みが発生すると，カーネル・レベルVMMの割り込みハンドラに制御が移る．ハードウェア割り込みを処理するのはホストOSであるため，割り込みハンドラは，アドレス空間や割り込みベクタ・テーブルなどをホスト空間に切り替える．その後，3.2節で述べたように，VMドライバによってホストOSカーネルの割り込みハンドラが呼び出される．

カーネル・レベルVMMは，VMライブラリにハイパバイザ・コールを提供する．ハイパバイザ・コールについて詳しくは4.3節で述べる．

表1: VMドライバが使用しているホストOSカーネルの機能

ホストOS（Linux）の機能	説明
virt_to_phys(), phys_to_virt()	アドレスの変換
copy_from_user(), copy_to_user(), memcpy()	メモリのコピー
kmalloc(), kfree()	メモリ管理
register_chrdev(), unregister_chrdev()	デバイス・ドライバの登録

3.4 VMライブラリ

VMライブラリは，従来のLilyVMと同様に，ゲストOSと同じアドレス空間にロードされ，特権命令や，センシティブな非特権命令のエミュレーションを行う．MMUの操作などは，カーネル・レベルVMMを呼び出す．

VMライブラリは，ゲストOSカーネルからホストOSの機能を呼び出すための手続きも持っている．従来のLilyVMでは，それらの手続きの中で，VMライブラリから直接ホストOSのシステム・コールを発行していた．本方式では，入出力を行うハイパバイザ・コールは，VMプロセスで処理される．このため，オーバヘッドが大きくなっているが，移植性は維持される．

4 カーネル・レベルVMMとVMライブラリの実装

ここでは，ホストOSに依存していないカーネル・レベルVMMと，カーネル・レベルVMMを呼び出すVMライブラリの実装について述べる．

4.1 MMUのエミュレーション

従来のLilyVMでは，ホストOSのmmap()システム・コールを用いて，ホストOSが持つページ・テーブルへの反映を行っていた．mmap()システム・コールでは，以下に示すような，非標準の機能を使わなければならなかったり，ホストOSによって制限がある場合があった．

- 引数に指定したアドレスへの割り当てを強制する`MAP_FIXED`フラグ．
- `MAP_FIXED`フラグを使用した0番地への割り当て．

- 割り当て可能な領域の数の制限．たとえばLinuxでは，1つのプロセスが割り当てられる領域は約6万個までに制限されている．

カーネル・レベルVMMでは，ホストOSの機能を使わずに，MMUを直接操作することにより，これらの問題を解決できるため，移植性が向上する．

ゲストOSは，内部に自分自身で管理しているページ・テーブルを持っている．このページ・テーブルは，仮想マシンの仮想アドレスから，仮想マシンの物理アドレスへのマッピングを保持している．プログラムを実行するときに必要なのは，仮想マシンの仮想アドレスから実マシンの物理アドレスへのマッピングであり，カーネル・レベルVMMは，このページ・テーブルを持つ．

現在のカーネル・レベルVMMでは，Xenと同様に，ゲストOS内にあるページ・テーブルのシャドウをVMM内に持っている．ゲストOSはプロセスごとにページ・テーブルを作成し，それらを切り替えながら動作する．ゲストOSがページ・テーブルを切り替える時に，カーネル・レベルVMMはシャドウを切り替える．

MMUのエミュレーションのために，ページ・フォールトを利用している部分がある．これについては4.2節で述べる．

4.2 ハードウェア割り込み，例外処理とシステム・コール

従来のLilyVMでは，例外処理とシステム・コールのリダイレクションを，ptrace() システム・コールとシグナルにより実現していた．これらの機能はOSに依存していた．例えば，ホストOSがLinuxの場合，ptrace() を用いてシステム・コールの発行を検出し，そのシステム・コールを無効化することができたが，NetBSDおよびFreeBSDではそのようなことができなかった．

本方式では，ゲストOSの実行中に発生した例外やシステム・コールは，カーネル・レベルVMMが受け取り，ゲストOSにリダイレクトする．具体的には，まず，例外やシステム・コールのスタック・フレームをゲストOSのメモリ上に作成し，スタック・ポインタを変更する．次に，プログラム・カウンタをゲストOSカーネルのハンドラに変更する．

カーネル・レベルVMMは，基本的に例外をそのままゲストOSカーネルへリダイレクトする．しかし，下で述べる一般保護例外，浮動小数点演算に関する例外，および，ページ・フォールトについては，カーネル・レベルVMMで例外の内容を調べ，その他の処理をする場合がある．

IA-32では，256種類の割り込みがある [7]．そのうち，割り込み番号0から31までは，CPUの例外などに使用されるため予約されている．残りの割り込み番号32から255までが，ハードウェア割り込みやトラップに使用される．しかし，使用される割り込み番号は，OSによって異なる．カーネル・レベルVMMは，ホストOSに依存しないようにするため，IA-32において予約されていない割り込み番号32から255までの割り込みを，ハードウェア割り込みとして扱う必要がある．しかし，Linuxのユーザ・プロセスがシステム・コールを発行する時，トラップ命令により割り込み番号128の割り込みが生成される．したがって，すべての割り込みを無条件にハードウェア割り込みとして扱うことはできない．

この問題を解決するために，割り込み番号32から255の割り込みを生成するトラップ命令をユーザ・レベルから実行できないように特権レベルを設定し，一般保護例外を通じて，システム・コールを検出するようにした．また，現在，ハイパバイザ・コールのために割り込み番号255を使用しており，その検出にも一般保護例外を使用している．

一般保護例外は，上で述べたように，トラップの検出に使用している．トラップでない場合や，ゲストOSによりユーザ・プロセスからの生成が禁止された割り込みである場合は，一般に，ユーザ・プロセスが不正な処理を行ったことを意味する．従って，例外をゲストOSに処理させるために，ゲストOSカーネルへのリダイレクションを行う．

ページ・フォールトは，MMUのエミュレーションにおいて重要なイベントのひとつである．ページ・フォールトが発生したときに，ゲストOSのページ・テーブルをアクセスし，必要に応じてシャドウへの反映を行う．また，ページ・フォールトによって，ゲストOSカーネルがページ・テーブルを変更しようとしていることも検出し，シャドウへの反映を行う．ゲストOSのページ・テーブルの情報を調べた結果，ページが不在であったり，アクセス権のないページへのアクセスであった場合は，ゲストOSカーネルにページ・フォールトを通知する．

浮動小数点演算に関する例外は，FPUレジスタの

保存と読み込みと，ゲストOSの遅延評価機能（制御レジスタCR0のTSビット）のために使用する．従来のLilyVMでは，FPUを使わない場合でも，ゲストOSのコンテキスト切り替え時にFPUレジスタの内容を入れ替える必要があった．その理由は，IA-32の遅延評価機能をユーザ・プロセスから利用することができないためである．カーネル・レベルVMMでは，ゲストOSに対して遅延評価機能を提供することにより，この問題を解決している．

4.3 VMライブラリとハイパバイザ・コール

カーネル・レベルVMMは，VMライブラリに対してハイパバイザ・コールを提供する．実装したハイパバイザ・コールの一覧を表2に示す．これらは，従来のLilyVMで使用していたシステム・コールよりも，CPUの機能に近い，すなわち，抽象度の低い機能としている．これによって，カーネル・レベルVMMが多くの処理を行うことができ，性能が向上する．

VMライブラリは，従来のLilyVMで用いていたものをベースとし，一部を修正して使用した．そのまま利用したのは，命令のエミュレーションを行う部分の一部である．たとえば，セグメント・レジスタやフラグ・レジスタの値を読み出す命令のエミュレーションを行う部分は，ホストOSに依存していないため，そのまま利用した．修正した部分は，以下の点である．

- MMUのエミュレーション．
- セグメント機能のエミュレーション．
- シグナル処理．
- 時計，および，タイマーのエミュレーション．
- ディスク，ネットワーク，および，シリアル・ポートの入出力処理．

MMUのエミュレーションについては，性能を上げるため，そのほとんどをカーネル・レベルVMMに実装している．そのため，MMUを制御する命令のエミュレーションにおいて，ゲストOSのページ・テーブルにアクセスし，ホストOSのmmap()システム・コールを使用していた部分をすべて削除し，ハイパバイザ・コールに置き換えた．

セグメント機能のエミュレーションについては，従来のLilyVMでは，VMライブラリからシステム・コールを発行していたために，次のような移植性を低下させる問題があった．

- FreeBSDやLinuxなどは，LDT（Local Descriptor Table）を変更する機能を持っているが，OSによってAPIが大きく異なる．
- Linux 2.6系列には，GDT（Global Descriptor Table）の一部を変更するための機能である`set_thread_area()`システム・コールが追加されているが，Linux 2.4以前や，FreeBSD，NetBSDなどにそのような機能はない．

VMライブラリでハイパバイザ・コールを使用し，カーネル・レベルVMMでGDTとLDTを変更することにより，これらの問題を解決した．ハイパバイザ・コールはホストOSに依存しないように設計しているため，VMライブラリの移植性が向上している．

従来のLilyVMでは，例外を処理するために，VMライブラリにおいてホストOSに依存したシグナル機能を利用していた．本方式では，カーネル・レベルVMMが例外をすべて処理するようにした．

VMライブラリにおいて，時計，および，タイマーのエミュレーションは，システム・コールを発行していた部分を，ハイパバイザ・コールに置き換えている．ハイパバイザ・コールの処理においてカーネル・レベルVMMは，実際のタイマーを制御してカウンタの値を取得し，その値と，ホスト空間からゲスト空間に切り替える時に得た時刻情報を元に，時刻を計算し，VMライブラリに結果を返す．

表2: ハイパバイザ・コールの一覧

ハイパバイザ・コール	説明
set_cr3()	ページ・テーブル (CR3レジスタ) をセットする
invalidate_page()	指定されたページのTLBをフラッシュしページをマップする
set_gdt()	GDTのエントリを変更する
set_ldt()	LDTのエントリを変更する
run_user()	ゲストOSのユーザ・プロセスに制御を移す
call_host()	ホストOS上のVMプロセスを呼び出す
gettimeofday()	時刻情報を得る

ディスク，ネットワーク，および，シリアル・ポートの入出力処理についても，システム・コールを発行していた部分を，ハイパバイザ・コールに置き換えている．カーネル・レベルVMMは，VMプロセスに入出力要求を伝える．

5 VMドライバの実装

VMドライバは，3章で述べたように，カーネル・レベル・コードを用いるLilyVMにおいて，ホストOSに存在する部分である．本研究では，ホストOSとしてLinux，Solaris，および，Windows XPに対してVMドライバの実装を行った．ここでは，それぞれのホストOSに対する実装について述べる．

5.1 Linux

まずLinuxをホストOSとしてVMドライバを実装した．VMドライバは，VMプロセスからioctl()システム・コールを使用して呼び出される．VMドライバは，ホストOSの機能として，表1に示す機能を使用している．

5.2 Solaris

Linuxの次に，SolarisをホストOSとしてVMドライバを実装した．Solarisにおいても，VMドライバは，VMプロセスからioctl()システム・コールを使用して呼び出される．そのため，VMプロセスはLinuxのものと同一である．

VMドライバは，表1に示す機能として，Linuxのkmalloc()に対応するkmem_alloc()関数，kfree()に対応するkmem_free()関数を使用している．また，virt_to_phys()およびphys_to_virt()に直接対応する関数はないが，vmem_alloc(), hat_devload(), hat_unload(), vmem_free()関数を用いて対応する機能を実装した．

5.3 Windows XP

次に，Windows XPをホストOSとしてVMドライバを実装した．Windows XPにおいては，Windows XPのAPIはUnixのものと大きく異なるため，Unixと互換性のあるAPIを実装しているCygwinを使用した．しかし，Cygwinにおいても，ioctl()によるデバイス・ドライバの呼び出しができない．そこで，VMプロセスとVMドライバの間の通信にはioctl()ではなく，Windows XPのDeviceIoControl() APIを使用した．そのため，VMプロセスは，ioctl()の部分を除いてLinux版と同一である．

VMドライバは，表1に示す機能として，Linuxのvirt_to_phys()に対応するMmGetPhysicalAddress()関数，kmalloc()に対応するMmAllocateContiguousMemory()関数，kfree()に対応するMmFreeContiguousMemory()関数を使用している．また，phys_to_virt()に直接対応する関数はないが，MmMapIoSpace()，MmUnmapIoSpace()関数を用いて対応する機能を実装した．

Windows XPへの実装において問題となるのは，アドレス空間である．全体で4GB（1GB=2^{30}バイト）あるアドレス空間のうち，カーネル用に1GBを確保しているLinux，0.5GBを確保しているSolarisと異なり，Windows XPでは2GBが確保されている．しかし，ゲストOSを実行するには残りの2GBでは足りないため，カーネル空間を残したままゲストOSを実行することができない．そこで，ゲストOSを実行する時は，アドレス空間全体を切り替え，ホストOSのカーネルがアドレス空間に残らないようにした．

6 実験

カーネル・レベルVMMの性能を測定し従来のLilyVMとの比較を行った．実験で用いたハードウェア，および，OSは以下の通りである．

CPU Pentium 4 3.00GHz
ホストOSのメイン・メモリ 1GB
ゲストOSのメイン・メモリ 64MB
ホストOSカーネル Linux 2.6.15.4
ゲストOSカーネル Linux 2.6.15.4

設定として，SMPサポートを無効とし，ホストOSとゲストOSのタイマー割り込みの頻度を100Hzとした．また，VMプロセスに割り当てられたメモリがページ・アウトされないようにした．

実験では，実機（ホストOS），従来方式，および，本方式において，標準設定のLinux 2.6.16のコンパイルを行った．そのときのmakeコマンドの実行時間を測定した．結果を図2に示す．図2では，

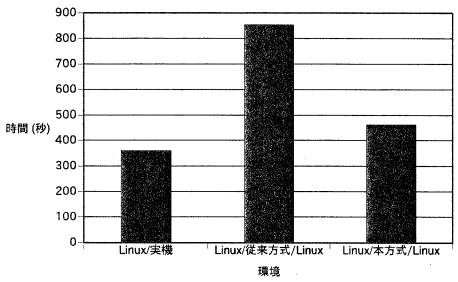

図 2: アプリケーションベンチマークの結果 (Linux 2.6.16 のコンパイル時間)

左から順に実機，従来方式，そして，本方式における実行時間を示している．このように，実行時間が従来の LilyVM と比べて約半分に短縮している．これは，カーネル・レベル VMM が例外のリダイレクションと MMU のエミュレーションを行うことにより，オーバヘッドを削減したためである．

また，本方式は実機の 1.3 倍程度遅くなっている．これは，ゲスト空間とホスト空間の間でアドレス空間や割り込みベクタ・テーブルなどを切り替えることによるオーバヘッドが大きいためである．

7 おわりに

この論文では，ホスト OS にカーネル・レベル・コードを追加することにより，ユーザ・レベル VMM の移植性を向上させる方法について述べた．その特徴は，カーネル・レベル・コードが，ホスト OS を介さずに CPU を制御することにより，ホスト OS への依存度を小さくしている点にある．LilyVM は，この方法により，新たに Solaris，および，Windows XP 上で動作するようになっている．

また，カーネル・レベル・コードを利用することで，フォールトおよびトラップのエミュレーションや，MMU のエミュレーションなどのオーバヘッドを削減している．これにより，Linux カーネルの構築にかかる時間を約半分に短縮している．

今後の課題は，MMU のエミュレーションにおいて，ホスト OS の VM プロセスに割り当てられたメモリがページ・アウトされた場合に対応することである．また，FreeBSD，NetBSD，および，Mac OS X といった OS への移植も実現したい．

謝辞

LilyVM の開発の一部は，萩谷プロジェクト・マネージャのもと，情報処理振興事業協会（IPA）平成 15 年度未踏ソフトウェア創造事業の援助を受けて行いました．萩谷プロジェクト・マネージャには，さまざまなご指導をいただきました．心より感謝いたします．

参考文献

[1] Aloni, D.: Cooperative linux, *Proceedings of the Linux Symposium*, pp. 23–31 (July 2004).

[2] Dike, J.: A user-mode port of the Linux kernel, *the 4th Annual Linux Showcase & Conference* (2000).

[3] Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Pratt, I., Warfield, A., Barham, P. and Neugebauer, R.: Xen and the Art of Virtualization (2003).

[4] Eiraku, H. and Shinjo, Y.: Running BSD Kernels as User Processes by Partial Emulation and Rewriting of Machine Instructions, *USENIX BSDCon 2003 Conference (BSDCon'03)* (September 2003).

[5] 榮樂英樹, 新城靖, 加藤和彦: ユーザレベル OS のためのユーザレベルネットワーク機能, 第 3 回情報科学技術フォーラム (FIT2004) (2004 年 9 月).

[6] 榮樂英樹, 新城靖, 加藤和彦: 複合ハイパバイザによる仮想計算機の高速化, 第 17 回コンピュータシステム・シンポジウム (ComSys2005) ポスター & デモ・セッション (2005 年 11 月).

[7] Intel Corporation: *IA-32 Intel Architecture Software Developer's Manual* (2006).

[8] Sugerman, J., Venkitachalam, G. and Lim, B.-H.: Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor, *Proceedings of the General Track: 2002 USENIX Annual Technical Conference*, Berkeley, CA, USA, USENIX Association, pp. 1–14 (2001).

[9] University of Cambridge, UK et al.: *Xen 3.0 Users' Manual*, http://www.cl.cam.ac.uk/research/srg/netos/xen/readmes/user/user.html (2006).