安全なソフトウェア基盤の構築

米澤 明憲
情報理工学系研究科 コンピュータ科学専攻

1 はじめに

現代の計算機システムにおいて，不正アクセスによる個人情報流出，サービス拒否攻撃による業務妨害，金融機関のシステム障害，個人用計算機の不安定さといった問題は，ほぼすべてソフトウェアの欠陥が原因である．我々は，そのような欠陥を防止するためのプログラミング言語技術やシステムソフトウェア技術についての研究を推進している．本年度も多くの研究成果をあげたが，それらの一部を挙げると，安全な言語処理系を構築するための技術，型つきアセンブリ言語を利用して安全性を保証する先進的なオペレーティングシステム，サービス拒否攻撃の防御手法，マルチプロセッサを仮想的に実現して複数の計算機を安全かつ効率的に使用することを可能にする仮想マシンなどがある．以下ではその中から，仮想マシンに関する研究成果について述べる．

2 マルチプロセッサを仮想的に実現するソフトウェア

仮想マシンとは，ハードウェア（CPU・メモリ・デバイス等）のエミュレーションをソフトウェアで行い，実マシンと同等の処理（例えばOSの実行）が可能なソフトウェアである．例えば，有名な仮想マシンとしてはVMware Workstationがあるが，これはWindowsのインストールされた実マシン上で，ユーザアプリケーションとしてLinuxを走らせることが可能である．

仮想マシンは非常に有用なソフトウェアであり，様々な目的のために使用することができる．最もオーソドックスには，異なるOS上で動作するアプリケーションを1台の実マシン上で同時に実行するために使用するが，それ以外にも，マシンを破壊する恐れのあるソフトウェア（例えば，Webやメールを経由して取得した，ウィルスである可能性のあるソフトウェア）を安全に実行するためのサンドボックスとしても利用することもできる．また，実行状態のスナップショットの取得・復元が可能であることを利用して，ソフトウェアのインストールによってOSに不具合が生じたらインストール前の状態にOSを復旧する，といったことも可能である． こうした利点を併せもつ仮想マシンであるが，ハードウェアの性能向上にしたがって実用に耐えうる速度で利用可能な環境が増えるにつれて，目覚ましい普及をみせている．

しかし，実用に耐えうる速度とはいっても，仮想化のオーバヘッドは非常に大きい．例えば，ソフトウェアでのエミュレーションを必要とする処理（例えばI/O）を頻繁に行うプログラムをVMware上で実行すると，実機と比較して80%近い性能低下となる．

そこで，ネットワークにつながった**複数の実マシン**上で**1台の仮想マルチプロセッサマシン**を実現するシステムを設計・実装する．この仮想マシンは，複数のマシンのCPUを有効利用することによって高速に処理を行うことを可能に

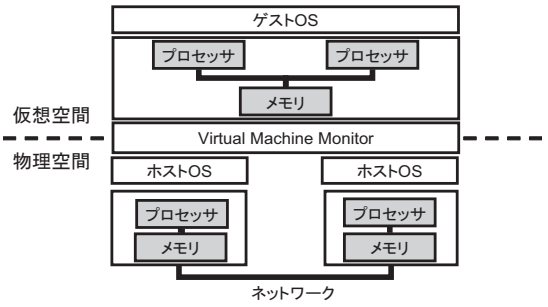

図 1: システムの構成

図 2: 動作イメージ：2 台の実マシン (Windows) 上での仮想デュアルプロセッサマシン (Linux) の実行

する．

このシステムは以下のモジュールから構成される（図 1 参照）．

ホスト OS 実マシン上で動作する OS.

ゲスト OS 仮想マシン上でユーザレベルで動作する OS．ホスト OS のユーザプロセス．

Virtual Machine Monitor (VMM) ハードウェアの仮想化を行なうレイヤー．仮想マシン上の処理で仮想化が必要なもの（例えばシステムコール呼び出し）を捕捉し，エミュレーションを行なう．

VMM は，実マシンと仮想マシンのプロセッサとメモリを以下のように対応づける[1]．

- 個々の実マシンのプロセッサは，仮想マシンのプロセッサの一つ一つと対応
- 複数の実マシンのメモリは，仮想マシンの（一つの）共有メモリと対応

ゲスト OS が通常のマルチプロセッサと同様にスケジューリングを行ないプロセスを仮想マシンの各プロセッサに割り当てると，個々の実マシン上でそのプロセスが走ることとなる．例えば，図 2 のように 2 台の Windows がホスト OS である実マシン上で，デュアルプロセッサマシンをエミュレーションし，ゲスト OS として Linux を走らせることができる．そして．Linux のプロセススケジューリングにしたがって，各実マシンにプロセスが割り当てられる．これにより，実行可能なプロセスが多数存在するとき（例えば，`make` コマンドの実行時）などに，高速に処理を行なうことができる．

また，以上の説明からも分かるように，このソフトウェアは，PC クラスタなどの分散した計算資源を簡便に有効利用するためのシステムとしても有用である．既存の Windows や Linux といった OS のインターフェースのままで，ネットワークにつながって分散している計算資源を扱うことができるからである．

Virtual Machine Monitor の実装における課題としては，主に，(1) CPU の特権レベルやデバイス等の仮想化と (2) 共有メモリのエミュレーションの二つが挙げられる．(1) の CPU の特権レベル等の仮想化は，仮想マシンを実装する上で共通の問題であり，効率的な実装に関する既存研究も多く存在するため，ここでは詳しく述べない．以下では，(2) を中心に述べる．

3 IA-32 のメモリモデルの概要

今回対象とする CPU は，最も普及していると思われる IA-32 とする．つまり実マシン，仮想

[1] その他のハードウェア（例えばネットワークインターフェース）をどう仮想化するかは，今後の課題である．

マシンともにIA-32であるとする．この章では，IA-32のメモリ順序モデルの概要について述べる．メモリ順序モデルとは，与えられたプログラムに対してメモリの読み書きがどのような順番で反映されなければいけないかを表すものである．基本的には，IA-32のメモリ順序モデルはプロセッサ順メモリモデル (Processor-ordered Memory Model) と呼ばれるものであり，

- 一つのプロセッサ内では，そのプロセッサに与えられたプログラムと同じ順番で，メモリの読み書きは反映される．
- ローカルプロセッサに書き込みが反映されたのと同じ順番で，遠隔プロセッサに対して書き込みは反映される．

となっている．以下に詳しい説明を行なう．

IA-32では読み込みは投機的に実行されうる．書き込みは原則的には[2]投機的に実行されない．

IA-32はストアバッファをもつ．これは，書き込み命令が連続するようなときに，次々に発生する書き込みデータをそのアドレスと共に一時貯めておいて，実際にメモリへの書き込み動作が完了するのを待たずに命令の制御を先に進めてしまう機構である．

ストアバッファに溜っている書き込みデータは，ローカルプロセッサにのみ反映される．遠隔プロセッサに対して書き込みが反映されるのは，ストアバッファからはき出だされたときである．

IA-32は，以下の2種類のメモリモデルを強める命令を用意している．

アトミック命令 read-modify-write命令などがアトミックに行なわれることをを保証する．

直列化命令 ストアバッファに溜っていた書き込み命令が反映される．つまり，プログラムの文面上でこの命令より前に位置するメモリアクセス命令が，全プロセッサに反映される．

[2]例えば例外として，String命令はOut-of-order実行されうる

4 仮想マシンのメモリモデル

仮想マシンが提供するメモリモデルは，既存のIA-32用のプログラムを変更を加えることなく動作可能にするため，前章で述べたIA-32の仕様を満たす必要がある．ただし，仕様を満たしながらもソフトウェアで効率的にエミュレーション可能なものでなければいけない．例えば，全ての書き込み命令を捕捉し，毎回遠隔マシンへと反映させるというは非効率である．そこで，基本的にはIA-32の仕様に従うが,「ストアバッファは無限長で，かつ，全ての書き込み命令は直列化命令が実行されるまでストアバッファに格納され続ける」というようにメモリモデルを設計する．これによって，書き込み命令を遠隔マシンへと反映させるのは，直列化命令実行時だけで済むようになる．

5 Virtual Machine Monitorの実装

前章で述べたメモリモデルを提供する仮想マシンを試作した．通常の命令はnativeに実機上で実行する．共有メモリのエミュレーションに必要な最低限の命令のみ，ソフトウェアでエミュレーション実行する．

Native実行中にエミュレーションの必要な同期命令を捕捉する必要がある．そのため，今現在の実装では，実行プログラムを事前に変換を行なっている[3]．具体的には，同期命令の直前（直後）に不正な命令を挿入する．例えば，

```
mfence
```

といったアセンブリプログラムが与えられたら，

[3]今現在は，この処理は手動で行なっている．

```
.byte 0x8e, 0xc8
mfence
```

と変換する．この変換によって，不正な命令を実行した際にシグナルが発生するようになるので，それを捕捉してエミュレーションの開始に移る．

また，ローカルマシン上で行なわれた書き込みの遠隔マシンへの反映方法であるが，基本的にはオーソドックスなソフトウェア分散共有メモリの実装手法に従う．mprotect を利用し各ページのアクセス保護を変更することによって実現している．

6 予備実験

今回実装した仮想マシンを，(1) 命令のエミュレーションにかかるオーバヘッド，(2) エミュレーションの必要な命令が全実行命令中に占める割合の二つを測定することよって評価した．

命令のエミュレーションにかかるオーバヘッドがたとえ大きくても，エミュレーションの必要な命令の比率が少ないことが実験から分かれば，仮想マシンの高速化が有望であるといえる．

6.1 命令エミュレーションのオーバヘッド

直列化命令（MFENCE 命令）を 10,000 回実行した時の実行時間を，実マシン上と仮想マシン上とで測定する．仮想マシンにおいては，MFENCE 命令実行間で書き込みが行なわれる場合とそうでない場合のそれぞれにおいて測定を行なった．また，実験には Pentium4 Xeon 2.4GHz を用いた．

表 1: 直列化命令の仮想化のオーバヘッド：MFENCE 命令を 10,000 回実行した時の実行時間

動作環境	書き込み	実行時間（秒）
実マシン	なし	0.00063
仮想マシン	なし	0.84
仮想マシン	あり	2.57

表 1 はこの実験の結果を示している．仮想マシン上での実行時間は実マシンのそれと比較して，書き込みなしの場合とありの場合でそれぞれ 1325 倍と 4052 倍となっている． 書き込みが行なわれた場合の方が，書き込み結果を遠隔マシンに反映させる処理のため，オーバヘッドがより大きくなっている．

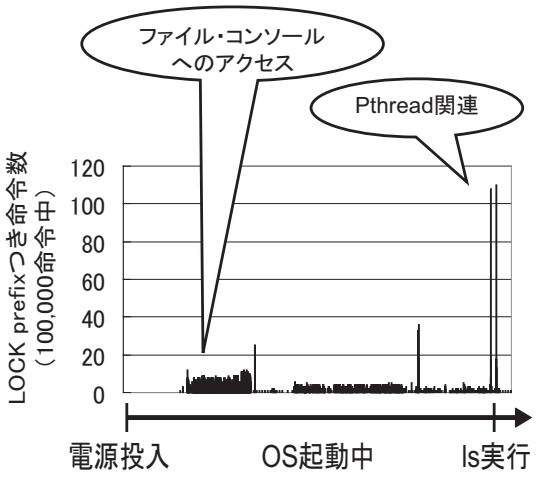

図 3: LOCK プレフィックスつき命令の，全実行命令中に占める割合

6.2 エミュレーションの必要な命令の頻度

Linux 2.2.19 を起動時してから起動後 ls コマンドを実行するまでの間に，LOCK プレフィックスのついた命令が実行された回数を測定した．この実験は，Bochs（x86 エミュレータ）上で行なった．

図 3 はこの実験の結果を示している．エミュレーション命令は，そのオーバーヘッドに対して，十分高速化が望めるほどの頻度でしか行なわれないことが分かる（ただし，これは実行アプリケーションにも依存する）．