仮想計算機のゲスト OS におけるファイルアクセスに対する高水準リンクの実現

豊岡拓[†] 新城靖[†,‡] 佐藤聡[†] 中井央[†] 板野肯三[†,‡]

筑波大学[†] 科学技術振興機構[‡]

1 はじめに

Type II の仮想計算機モニタ（Virtual Machine Monitor、以下 VMM と呼ぶ）において、仮想計算機内で動作する OS をゲスト OS、実機上で動作する OS をホスト OS と呼ぶ。ゲスト OS のファイルアクセスにおいては、VMM によるディスクデバイスのエミュレーションが重たい処理となっている。この問題を解決するために、準仮想化と呼ばれる手法がある。準仮想化では、ゲスト OS に専用のデバイスドライバを組み込むことでエミュレーションを不要にしている。この手法では、ホスト OS へバックエンドドライバという特殊なモジュールを組み込む。準仮想化において、デバイスドライバ以外のゲスト OS のモジュールを変更する方法も考えられるが、まだ試みられていない。

本研究では、ゲスト OS のファイルアクセスを、ゲスト OS のデバイスドライバを修正するのではなく、ゲスト OS とホスト OS の高水準なファイルシステム抽象化層を短絡することで高速化を実現する。この方式を**高水準リンク**と呼ぶ。この方式の利点は、ホスト OS のファイルシステム抽象化層という枠組みを利用できる点にある。

2 OS におけるファイルアクセス

Linux において、ユーザプロセスが read システムコールを発行する場合の制御の流れを以下に示す。まず、read システムコールが発行されることで、処理がカーネルに移る。カーネルでは、read システムコール処理において、VFS（VFS については後で詳しく述べる）の関数を呼び出す。VFS は、目的のファイルのあるファイルシステムにファイル読み出し要求を出す。ファイルシステムは、ファイルのデータが配置されているデバイスを制御するデバイスドライバに読み出し要求を出す。デバイスドライバは、ディスクなどのハードウェアに I/O 要求を出す。ハードウェアはデータをカーネルのバッファに転送し、割り込みを行う。以後、制御の流れは逆方向に上がっていく。

VFS（Virtual File System）とは、Linux を含む UNIX 系の OS におけるファイルシステム抽象化層であり、複数の

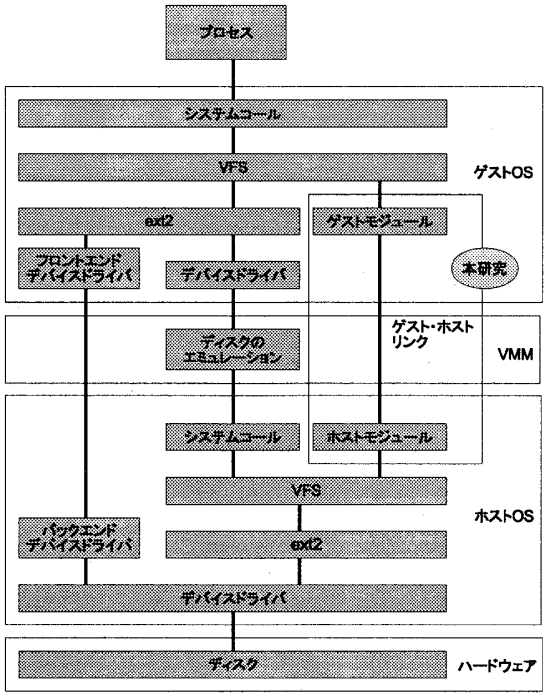

図 1 準仮想化を用いた場合（左）、ディスクのエミュレーションを用いた場合（中央）、および本研究（右）におけるファイルアクセスの比較

ファイルシステムの実装の違いを隠蔽する。例えば、ローカルファイルシステムとして使われる ext2 と、ネットワークファイルシステムである NFS（Network File System）[1] において、ファイルを操作する関数は異なるが、操作関数へのポインタを提供することで、同一のインタフェースでのファイルアクセスを実現している。VFS は、NFS を実装するときに考案された仕組みであり、これによりローカルディスクのファイルとリモートのサーバのファイルを透過的に操作することができるようになる。NFS では、クライアント側は VFS の下位のモジュール、サーバ側は VFS の上位のモジュールとして実現されている。

3 仮想計算機におけるファイルアクセス

VMware Workstation に代表される Type II VMM では、ゲスト OS からのファイルアクセスの流れは、図 1 の中央のようになっている。このように、2 章におけるシステムコールからデバイスドライバまでの流れを 2 つ、ディスクのエミュレーション部分を介してつなげた形になっている。ホス

Realization of high level links for file access in guest OSes running on Virtual Machines

Hiraku TOYOOKA, Yasushi SHINJO, Akira SATO, Hisashi NAKAI, Kouzo ITANO

† University of Tsukuba

‡ Japan Science and Technology Agency

トOSのデバイスドライバは実際のハードウェアであるディスクを制御するが、ゲストOSのデバイスドライバはVMMのディスクのエミュレーション部分にアクセスする。

それに対して準仮想化では、図1の左側のようになっている。このように、ゲストOSのフロントエンドドライバからホストOSのバックエンドドライバへ要求を送り、バックエンドドライバがデバイスドライバにアクセスしている。

4 ファイルアクセスに対する高水準リンク

本研究では、図1中央におけるゲストOSのデバイスドライバ、およびVMMのディスクのエミュレーション部分を使わず、ゲストOSのVFS層とホストOSのVFS層を短絡する（図1右側）。この方式を**高水準リンク**と呼ぶ。具体的にはゲストOSとホストOS両方のカーネルを書き換えて、2つのVFSの間にNFSに似た機構（ゲストモジュール、およびホストモジュール）を作り、さらにその間でデータのやりとりをできるような仕組み（ゲスト・ホストリンク）を作る。

ゲストモジュールは、NFSのクライアント側と同じようにVFSのインタフェースを実装する。NFSの実装においては、RPCを行う関数と行わない関数がある。本研究において、RPCを行う関数に相当する関数はホストモジュールに要求を送るが、RPCを行わない関数に相当する関数はホストモジュールに要求を送らず、ゲストOSだけで完結する。

ホストモジュールは、ゲストモジュールからの要求を受け取り、VFSのインタフェースを使ってホストOSのファイルを操作する。ホストモジュールでは、NFSのサーバ側のモジュールに相当する実装を行う。

5 実装

本研究では、4章で提案した仕組みを、ホストOSとゲストOSとしてLinuxを用いて行っている。仮想計算機としては、User Mode Linux[2]を使用している。また、ゲストモジュールはhostfsを改変して作成し、ホストモジュールはデバイスドライバとして作成している。ゲストモジュールからのホストモジュールへのアクセスはioctlシステムコールを用いて行っている。

5.1 ゲストモジュール

hostfsでは、個々のiノードにホストOSのファイルの名前やファイルディスクリプタなどの拡張情報を持たせることにより、ゲスト側のファイルとホスト側のファイルを対応付けている。我々はこれを変更し、iノードの拡張情報としてホスト側のiノード番号とスーパブロックのID(sbid)を記憶させるようにした。また、hostfsではVFSへのインタフェース関数からライブラリ関数呼び出しやシステムコール発行を行っている。我々はhostfsのVFSへのインタフェース関数に対して変更を行い、ホストモジュールの機能の呼び出しをioctlシステムコールで行うようにした。

5.2 ホストモジュール

ホストモジュールは、デバイスドライバとしてホストOSに登録され、NFSサーバのように内部でVFSの関数を呼び出す。一度lookupされたファイルはfile構造体としてハッシュテーブルに登録され、次に使用されるときはテーブルから高速にアクセスが可能となるようにした。もしハッシュテーブルで衝突が起こった場合は、既に存在するfile構造体を破棄し、新しいfile構造体を作り直して登録し、それを使用する。また、hostfsにおいて、ファイルをopen()できる回数には、仮想計算機がホストOS上の1プロセスであることによる制限がある。本研究のホストモジュールを用いることにより、この制限はなくなる。

5.3 実装を行った手続き

実装を行った手続きのうち、主要なものについて述べる。

5.3.1 lookup()

lookup()は、あるディレクトリ中から、指定した名前のファイルを見つけてくる処理である。ディレクトリは、ゲスト側で既に取得しているiノード番号とsbidで指定する。また、見つけることができたファイルの属性情報も併せて取得する。このことにより、ゲストモジュールにおけるVFSインタフェース関数のgetattr()からのホストモジュール呼出しの回数を抑えることができる。

5.3.2 read()/write()

read()/write()では、どのファイルに読み込み・書き込みを行うかを、ゲスト側でlookup()等によって既に取得したiノード番号とsbidを用いて行う。また、読み書きを行う開始位置も指定する。

6 関連研究

ディスクのエミュレーションを使わないホストOSのファイルアクセスの実装として、User Mode Linuxのhostfsがある。これは、本研究と同じくゲストOSのVFSの下位のモジュールを用いているが、ホストOSのシステムコールを呼び出すという実装になっている。本研究ではホストOS内のモジュールを用いるところが異なる。また、ホストOSのカーネル内で動作するcoLinux[3]のcofsでは、ゲストOSのモジュールが直接ホストOSのデバイスドライバ用のAPIにアクセスしている。本研究では、ユーザレベルで動作する仮想計算機も対象としているところが異なる。

7 おわりに

本稿では仮想計算機のゲストOSにおけるファイルアクセスを高水準リンクを用いて実現することについて述べた。本実装では、VFSにおいてゲストOSのモジュールとホストOSのモジュールを作成して短絡することでディスクデバイスのエミュレーションを不要にしている。今後は、提案方式の性能を評価する。また、他の仮想計算機の技術に本方式を適用し、準仮想化におけるフロントエンドドライバを用いた方式との機能と性能を比較する。

参考文献

[1] Sun MicroSystems, Inc.: "Network Programming", 1988.

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

[3] Dan Aloni: "Cooperative Linux", *Proceedings of the Ottawa Linux Symposium*, July 2004.