Vol.2009-ARC-183 No.14
Vol.2009-OS-111 No.14
2009/4/23

OS 共存環境におけるカーネル間隔離機能の性能への影響

追川 修一[†1]

信頼性やセキュリティ向上のためには，オペレーティングシステム（OS）カーネルの内部状態をカーネル外部から別のソフトウェアにより監視することが有効である．単一システム内でそのような監視を行うためには，監視対象となる OS と監視ソフトウェアを実行する OS からなる，複数 OS の共存環境が必要となる．OS 共存環境は，本来 OS で行うべき処理を阻害しないように，できるだけ軽量であることが望ましい．本論文では，OS 共存環境におけるカーネル間の隔離機能の有無による性能差について調査するため，IA-32 システム上で OS 共存環境を実装し，実験を行った結果について述べる．実験結果から，カーネル間の隔離機能の有無による性能差は大きいこと，また性能差は CPU によって変化することがわかった．

Performance Impact by OS Isolation on OS Colocation Environments

Shuichi Oikawa[†1]

Monitoring the internal conditions of the operating system (OS) kernel is an effective technique to improve reliability and security. It requires the colocation of two OSes running on a single system, a monitored one and the other that executes monitoring software. Enabling such colocation should be lightweight in order not to disturb the execution of the monitored OS. We implemented the OS colocation software layers with and without the OS isolation mechanism on IA-32 systems. This paper investigates the performance impact imposed by OS isolation at the OS colocation software layers.

†1 筑波大学 システム情報工学研究科 コンピュータサイエンス専攻
University of Tsukuba, Department of Computer Science

1. はじめに

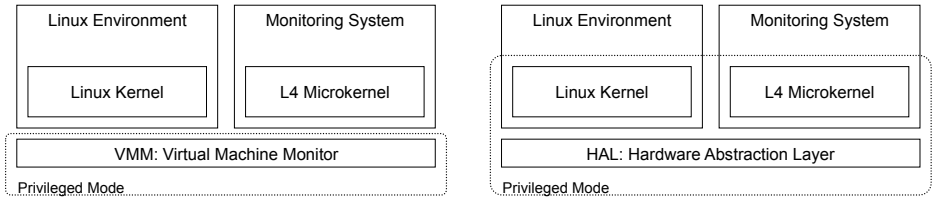
オペレーティングシステム（OS）の信頼性やセキュリティ向上のためには，OS カーネル外部から別のソフトウェアにより，OS カーネルの内部状態や入力されるデータを実行時に監視することが有効である[11),14)]．単一システム内でそのような監視を行うためには，監視対象となる OS の他に監視ソフトウェアを実行する OS も実行する必要があるため，複数 OS の共存環境が必要となる．OS 共存環境としては，OS を実行する仮想マシン（VM）を提供する仮想マシンモニタ（VMM）[5),13)] や，カーネルの機能拡張によるもの[1),3),4),15),16)] がある．

信頼性やセキュリティ向上は重要であるが，監視対象となる OS で行う処理が本来行うべき処理であり，監視のオーバヘッドは必要最小限であるべきである．そのため，OS 共存環境はできるだけ軽量であることがま望しい．VMM による OS 共存環境では，OS を VMM よりも低い特権レベル（非特権モード）で実行する．ハードウェアを操作する命令は基本的には特権命令であるが，非特権モードでは特権命令を直接実行できない．VMM は特権命令をエミュレーションすることで，代わりに VM を操作する．従って，特権命令による VM 操作毎にモード切替が発生するため，OS の実行にはオーバヘッドを伴う．一方，カーネルの機能拡張による OS 共存環境では，共存しない場合と同じく，OS カーネルを特権モードで実行する．従って，OS カーネルは特権命令によりハードウェアを直接操作可能であるため，OS 共存のオーバヘッドは小さい．しかしながら，VMM による OS 共存環境は，共存するカーネル間の隔離ができるため，監視 OS 側が監視対象 OS の不具合による影響を受けにくいという利点を持つ．

本論文は，監視 OS を含むアーキテクチャ設計において性能面での指針を提供することを目的とし，OS 共存環境におけるカーネル間の隔離機能の有無による性能差について調査する．そのために，IA-32 システム上に 2 つの OS 共存環境を実装し，実験を行った結果について述べる．実験結果から，カーネル間の隔離機能の有無による性能差は大きいこと，また性能差は CPU によって変化することがわかった．

以下，本論文の構成は次のとおりである．2 章では，本論文で実現した OS 共存環境についてまとめる．3 章では実験結果について述べ，4 章で考察を行う．5 章で関連研究について述べ，6 章でまとめと今後の課題について述べる．

Vol.2009-ARC-183 No.14
Vol.2009-OS-111 No.14
2009/4/23



(a) VMM による OS 共存環境　(b) カーネルの機能拡張による OS 共存環境

図 1 OS 共存環境におけるカーネル間隔離機能の有無

2. OS 共存環境

VMM によるカーネル間隔離機能を持つ OS 共存環境と，カーネルの機能拡張によるカーネル間隔離機能を持たない OS 共存環境の，2 つの OS 共存環境について，概要，それぞれの設計と実装の詳細，および差異について述べる．

2.1 概　　要

図 1 に本論文で IA-32 システム上に実現した 2 つの OS 共存環境を示す． 図 1(a) が VMM によるカーネル間隔離機能を持つ OS 共存環境であり，図 1(b) がカーネルの機能拡張によるカーネル間隔離機能を持たない OS 共存環境である．本論文では，図 1(b) に示すカーネルの機能拡張による OS 共存環境において，OS 共存を可能にするソフトウェアレイヤをハードウェア抽象化レイヤ（HAL）と呼ぶ．

まず，これら 2 つの OS 共存環境の共通点について述べる．本論文で述べる OS 共存環境は，OS 監視システムを実装するために実装されている．OS 監視システムは L4 マイクロカーネル上に構築されており，Linux を監視対象 OS としている．そのため，どちらの OS 共存環境も Linux と L4 それぞれ 1 つの共存を可能にするものとなっている．L4 上の OS 監視システムは，Linux カーネルのデータ構造体を監視し，異常を検知したら修復を試み，修復できない場合は Linux カーネルを再起動するものである[11),14)]．L4 上では OS 監視システム以外は必要最小限のプログラムのみを実行する．

現在のところ，1 CPU システムのみをサポートしている．また，メモリおよびデバイスについてはコンフィグレーション時に分割し，共有しないことを前提としている[⋆1]．そのため，メモリおよびデバイスの仮想化はしていない．CPU は 2 つの OS により共有されるため，次のようにスケジューリングを行う．監視対象 OS の負荷に関わりなく監視システムが起動するように，L4 を優先的に実行するようにスケジュールする．即ち，L4 が受け取るべき割り込みを契機として L4 はディスパッチされる．必要に応じて監視ソフトウェアが実行された後，L4 はアイドル状態になる．その後，Linux に実行が移される．L4 が実行中に発生した Linux が受け取るべき割り込みは，L4 がアイドル状態になった後に Linux がディスパッチされた時に通知される．

これら 2 つの OS 共存環境における最も顕著な違いは OS カーネルの実行モードにある．図 1(a) の VMM によるカーネル間隔離機能を持つ OS 共存環境では，OS カーネルを非特権モードで実行する．一方，図 1(b) のカーネルの機能拡張によるカーネル間隔離機能を持たない OS 共存環境では，OS カーネルを特権モードで実行する．VMM による OS 共存環境では，OS カーネルを非特権モードで実行するため，OS カーネルは CPU の実行環境を制御する特権命令を直接実行できない．そのため，OS カーネルは CPU の仮想記憶機能を直接管理することができない．代わりに VMM が管理し，OS 毎に別々のページテーブルを用い，物理メモリを分割して割り当てることにより，カーネル間の隔離が実現できる．一方，カーネルの機能拡張による OS 共存環境では，OS カーネルを特権モードで実行し，OS カーネルが CPU の仮想記憶機能を直接管理する．各 OS カーネルはコンフィグレーション時に割り当てられた物理メモリを使用して実行するため，通常動作時には問題が起こることはない．しかし，ハードウェアによる保護は提供されないため，他 OS に割り当てられた物理メモリへのアクセスを検出することはできない．

2.2 仮想マシンモニタ

VMM の設計と実装について述べる．まず，VM 環境の実現について述べ，次に OS 共存環境の実現について述べる．本論文で用いた VMM は，Gandalf VMM[8),9)] をベースに，1 CPU 上で OS 共存を可能にするように変更したものである．

2.2.1 仮想マシン環境の実現

本論文で述べる VMM は，OS カーネルにわずかの変更を加えることで効率の良い VM 環境を実現する Mesovirtualization[8)] という手法を用いている．

VMM は IA-32 CPU（以下単に IA-32）の保護機能を用いてカーネル間隔離機能を持つ OS 共存環境を実現している[⋆2]．IA-32 は保護機能として 4 レベルの特権レベル（保護リング）を提供しているが，本論文で述べる

⋆1 割り込みコントローラ（Intel 8259A PIC）も共有されるが，Linux と L4 でハードウェア割り込みのベースアドレスなどの設定が共通であるため，ハードウェア割り込み番号（IRQ）をコンフィグレーション時に分割し，割り込みディスクリプタテーブル（IDT）の切り替えのみを行うことで対応している．

⋆2 現在の IA-32 はハードウェアによる仮想化支援機能として Intel VT-x を提供しているが，本論文で述べる

Vol.2009-ARC-183 No.14
Vol.2009-OS-111 No.14
2009/4/23

グ）を提供している．特権レベルは 0 から 3 まであり，レベル 0 が特権モードを表す．特権命令はレベル 0 でのみ実行可能であり，レベル 0 以外で実行しようとすると一般保護例外（General Protection Fault）を起こす．OS カーネルは特権命令を用いてハードウェア資源を管理するため，通常は OS カーネルをレベル 0，ユーザプロセスをレベル 3 で実行する．VMM による OS 共存環境では，VMM をレベル 0，OS カーネルをレベル 1，ユーザプロセスをレベル 3 で実行する．

VMM は一般保護例外により OS カーネルの特権命令の実行を検知し，適切に特権命令のエミュレーションを行う．OS カーネルを特権レベル 1 で実行することで，OS カーネルによる特権命令の実行は一般保護例外を起こす．VMM は一般保護例外により VMM の例外ハンドラが起動するように割り込みディスクリプタテーブル（IDT）を設定しておく．これにより，OS カーネルによる特権命令の実行により VMM の一般保護例外ハンドラが起動する．VMM の一般保護例外ハンドラは例外が発生した命令アドレスから命令を読み込み，命令をデコードし，OS カーネルが実行しようした特権命令を識別する．VMM は，その特権命令に対応した処理を行い必要ならば VM に反映することで，特権命令のエミュレーションを行う．

一部，特権命令の実行の検知だけでは，OS カーネルが行う実行環境への変更を VM に反映することが難しい部分については，カーネルにハイパーコールを追加することにより，明示的に変更を VMM に通知し，VM に反映する．例えば，OS カーネルはプロセス切り替え時にカーネルスタックを変更するが，ユーザプロセスからのモード切替時に変更したカーネルスタックが使用されるよう，タスクステートセグメント（TSS）の該当エントリも変更する．この変更は単なるメモリへの書き込みであり特権命令ではないため，特権命令の実行の検知では対応できない．該当アドレスを含むページを書き込み不可にすることで検出可能であるが，ここではカーネルがハイパーコールを発行するコードを追加し，ハイパーコールにより変更を VMM に通知することで，VM に反映する．また，Linux カーネルは LIDT 命令により IDT を登録した後に，IDT の該当エントリを変更し割り込みハンドラを追加するため，同様にハイパーコールを追加することで対応している．

IA-32 の仮想記憶機能はページテーブルを用いている．OS カーネルは割り当てられた物理メモリを仮想メモリ空間にマップするためにページテーブルを管理する．OS カーネルのページテーブルをそのまま CPU が参照すると，任意の物理メモリをマップできてしまうため，カーネル間隔離が実現できない．そのため，VMM は OS カーネルのページテーブルを参照し，VMM が管理するページテーブルを作成し，CPU は VMM が管理するページテーブルを参照するようにする．この VMM が管理するページテーブルのことをシャドウページテーブルと呼ぶ．OS カーネルはプロセス毎にページテーブルを作成するが，本論文で述べる VMM は，現在のところ簡単のため，1 つのシャドウページテーブルのみを管理し，ページテーブルの切り替え毎に作成し直す実装にしている．

VMM と OS は同じ仮想メモリ空間上に置かれる．VMM は仮想メモリ空間の上位 64MB を占有し，OS はそれ以下を使用する．OS カーネルのページテーブルには VMM はマップされておらず，VMM がシャドウページテーブルに自身をマップするエントリを追加する．ある仮想アドレスにマップされる物理ページフレームを指定するページテーブルエントリは，そのページにアクセスできる特権レベルを指定する U/S フラグを含む．U/S フラグは 1 ビットのフラグであり，0 の時は特権レベルが 0 から 2 の時のみアクセス可能であり，1 の時には全ての特権レベルでアクセス可能である．通常，OS カーネルが使用する仮想アドレス空間にマップするページテーブルエントリの U/S フラグは 0 とし，ユーザプロセスのは 1 とすることで，ユーザプロセスから OS カーネルの仮想アドレス空間を保護している．VMM による OS 共存環境では，VMM をレベル 0，OS カーネルをレベル 1 で実行するため，OS カーネルから VMM の仮想アドレス空間がアクセス可能になってしまう．そのため，VMM の仮想アドレス空間を OS カーネルから保護するために，OS カーネルのセグメントから上位 64MB を除外する．

Linux カーネルは使用可能な物理メモリ領域をブートローダから渡される情報から取得する．ブートローダは BIOS を呼び出すことでこの情報を得る．VMM による OS 共存環境では，ブートローダから渡された情報を加工し，VMM および L4 が使用する物理メモリ領域を除外した情報を Linux カーネルに渡すことで，物理メモリの分割を行う．L4 カーネルは使用する物理メモリをコンフィグレーションファイルに記述する方式なため，このファイルを変更し，VMM および Linux が使用する物理メモリと重ならないように，L4 が使用する物理メモリを指定する．

Linux-2.6.27.5 カーネルへの変更は 9 箇所 50 行（#ifdef，#else，#endif マクロを含む）であるが，3 箇所 22 行は OS の仮想アドレス空間を制限するための変更であり，実行時に対応することで変更を不要にすることも可能である．OKL4-1.4.11 カーネルへの変更は 18 箇所 35 行である．

VMM では使用していない．

Vol.2009-ARC-183 No.14
Vol.2009-OS-111 No.14
2009/4/23

2.2.2 OS 共存環境の実現

VMM による OS 共存環境では，VMM のみが特権モードで実行する．そのため，ブートローダは VMM をカーネルとしてメモリ上にロードし，実行を開始する．VMM のロード時に，Linux カーネルと L4 カーネルも付加モジュールとしてメモリ上にロードする．物理メモリ上では，下位アドレスから Linux，L4，VMM の順に使用するため，実行を開始した VMM が，自身と Linux カーネル，L4 カーネルを適切な位置に移動する．OS カーネルが物理メモリとみなして使用できるよう物理メモリと仮想メモリのアドレスを 1 対 1 対応させた初期ページテーブルを作成するなどの初期設定を行った後に，L4，Linux の順にブートする．

2.1 節で述べたように，L4 を優先的に実行するようにスケジュールするため，L4 から Linux への切り替えは L4 の HLT 命令の実行を，Linux から L4 への切り替えは割り込みを契機として行う．HLT 命令は特権命令であるため，特権レベル 1 で実行している OS カーネルがアイドル状態になり HLT 命令を実行すると一般保護例外を起こし，VMM が起動し切り替え処理を行う．Linux 実行時に Linux が受け取るべき割り込みが発生した場合は，VMM を介さずに直接 Linux カーネルに通知される．しかし，L4 が受け取るべき割り込みが発生した場合は，VMM の割り込みハンドラを起動するように IDT を設定しておき，Linux から L4 への切り替え処理を行う．同様に，L4 実行中に Linux が受け取るべき割り込みが発生した場合は，VMM の割り込みハンドラを起動するように IDT を設定しておくが，この場合は切り替え処理を行うのではなく，L4 の HLT 命令の実行により切り替え処理が行われる時に，Linux の割り込みハンドラが呼び出されるように設定するだけの処理を行う．

OS 切り替え時には，VM 環境を構成する制御レジスタの入れ換えを行う．基本的に制御レジスタは特権命令のエミュレーション時にその値を取得しているため，保存する必要のあるレジスタは実行時に変化するレジスタのみになる．また，VMM 起動時に保存されないセグメントレジスタの保存，入れ換えも行う．VM 環境の実行再開のための情報は継続を用いて管理しており，継続を表す命令ポインタと VMM 起動時に VMM スタック上に保存された VM コンテキストへのポインタのみを保存し，その他の汎用レジスタの保存復帰は不要としている．上記の，Linux への切り替え時に Linux の割り込みハンドラが呼び出されるように設定する処理は，継続を表す命令ポインタと VMM スタック上に保存された VM コンテキストの変更によって行われる．

2.3 ハードウェア抽象化レイヤ

カーネルの機能拡張による OS 共存環境の設計と実装について述べる．本論文では，図 1(b) に示すカーネルの機能拡張による OS 共存環境において，OS 共存を可能にするソフトウェアレイヤを，ハードウェア抽象化レイヤ（HAL）と呼ぶ．HAL は，既存の Linux 環境に簡単に OS 共存環境を導入できるようにするとの目的で，設計，実装された．そのため，現在は 2.2 節で述べた VMM とは全く別個の実装となっている．

カーネルの機能拡張による OS 共存環境では，OS カーネルは特権モードで動作し，また HAL も特権モードで動作する．従って，2.2.1 節で述べた VM 環境は HAL 上には存在せず，OS カーネルの実行する特権命令はそのまま CPU により直接実行され，実行環境に反映される．そのため，モード切替やエミュレーションが不要になり，オーバヘッドの少ない OS 共存環境が構築できる．

2.3.1 OS 共存環境の実現

HAL の基本的な機能は OS の切り替えである．2.1 節で述べたように，L4 を優先的に実行するようにスケジュールするため，L4 がアイドル状態になった時に Linux への切り替えを，Linux が実行中に L4 に通知すべき割り込みが発生した時に L4 への切り替えを行う必要がある．割り込みを契機に OS 切り替えを行うため，IDT を HAL にものに入れ換える必要がある．OS カーネルの IDT を HAL を呼び出すように書き換えることもできるが，OS カーネルにより上書きされる可能性もあるため，HAL が IDT を用意し CPU に登録する．また，L4 がアイドル状態になった時に通常は HLT 命令を実行するが，特権モードで動作する L4 は HLT 命令をそのまま実行できてしまうため，それを HAL で検知することは出来ない．従って，明示的に HAL を呼び出すように変更する必要がある．

2.3.2 OS 共存環境の導入および起動

上述したように，現在の HAL は，既存の Linux 環境に簡単に OS 共存環境を導入できるようにするとの目的で，設計，実装された．簡単に導入することができるように，HAL は Linux のブート後に導入されるローダブルカーネルモジュール（LKM）により，L4 とともにロードされる．LKM は Linux のデバイスドライバとして実装されており，/dev/x86hal というデバイスファイルを作成する．このデバイスファイルに対し ioctl システムコールを発行し，HAL と L4 を Linux カーネル空間内にロードする．

HAL がロードされる領域は，Linux カーネルが使用することがなく，しかし仮想メモリのマッピングは保障されている必要がある．また，HAL がロードされた領域の物理メモリは，L4 の仮想メモリにマップする必要があるため，簡単のためには物理メモリとしても連

Vol.2009-ARC-183 No.14
Vol.2009-OS-111 No.14
2009/4/23

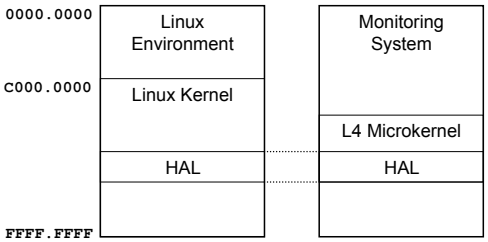


図 2 HAL による OS 共存環境における仮想アドレスマップと HAL の配置

続した領域であることが望ましい．そこで，crashkernel カーネルオプションにより確保される領域を使用する[6]．例えば，ブートローダとして GRUB を用いる場合，以下のようにブートする Linux カーネルイメージの後にカーネルオプションを付けることができる．

```
title Linux-2.6.27.5
        kernel /boot/vmlinuz-2.6.27.5 crashkernel=48M@16M
```

この場合は，物理メモリ上で開始アドレス 16MB，サイズ 48MB の領域を確保することを意味する．HAL をロードするデバイスドライバは，この領域に HAL と L4 を読み込む．LKM へのオプションなどで明示的に確保した領域のパラメータを渡すことも可能であるが，crashkernel カーネルオプションで指定されたアドレスとサイズは crashk_res.start および crashk_res.end に保存されるため，現在はこれの値を使用している⋆1．

HAL は，一部の割り込みを受け取る必要があるため，Linux と L4 の仮想メモリにマップされる必要がある．上述の crashkernel カーネルオプションにより確保される領域は，Linux カーネル空間の一部であるため，この領域に HAL をロードすることにより，HAL の Linux の仮想メモリへのマップは保障される．L4 の仮想メモリにも，Linux と同じアドレスにマップすることができれば，Linux と L4 で HAL を共有することが出来る．そのために，L4 を変更した．図 2 に，Linux と L4 の仮想アドレスマップにおける L4 と HAL の配置を示す．L4 は，HAL と重ならないように，また Linux 内で確保された物理メモリを使用するように変更した．

⋆1 crashk_res はグローバル変数であるがエキスポートシンボルではないので，LKM からは使用できない．そこで，現在はエキスポートシンボルとする変更を加えている．

HAL および L4 のロード後に，HAL を起動する．HAL はグローバルディスクリプタテーブル（GDT）のコピーの作成，IDT の入れ換えなどの初期化後に，ページングを停止した状態に設定し直し L4 をブートする．

2.3.3 OS 切り替え

OS 切り替え時には，OS 実行環境を構成する制御レジスタおよび汎用レジスタの入れ換えを行う．VMM による OS 共存環境とは異なり，特権命令のエミュレーションにより制御レジスタの値を取得することができないため，ブート後は変更の可能性があまりない制御レジスタについても，OS 切り替え時に保存する必要がある．また，汎用レジスタも保存する．

OS 実行環境を構成する制御レジスタは，相互に依存した関係になっているため，入れ換えには注意を要する．セグメントや TSS は GDT に依存しているが，GDT は仮想アドレスを用いて指定するため，ページテーブルへのポインタを保持する CR3 に依存している．CR3 を切り替えると，GDT, IDT, TSS がマップされていない状態になり，また CR3 を切り替える前に GDT をロードすると，セグメントが無効な値になる．そこで，OS 間で共有する中間 GDT を HAL 内に作成し，中間 GDT を経由して制御レジスタの入れ換えを行う．この中間 GDT を作成するために，HAL は起動後に Linux の GDT のコピーを作成し，L4 起動後に L4 の GDT の内容も追加する．中間 GDT の問題点は，Linux と L4 が使用する GDT エントリに重なりがあってはならないところにある．実際，いくつかのエントリは重なってしまっており，L4 側の使用するエントリを変更した．

2.3.4 OS への変更点

OS 共存環境の導入のための OS への変更点をまとめる．Linux カーネルには，HAL と L4 が使用する領域のパラメータを LKM から使用できるようにする変更を加えているが必須ではないため，基本的には無変更で使用できる．L4 には，アイドルループから L4 の呼び出しのための変更，中間 GDT を作成できるように使用する GDT エントリが Linux と重ならないように変更，使用する物理アドレスと仮想アドレスを変更を加えた．

3. 実　　験

本論文で実現した，VMM によるカーネル間隔離機能を持つ OS 共存環境と，カーネルの機能拡張によるカーネル間隔離機能を持たない OS 共存環境の，2 つの OS 共存環境を比較する実験を行い，その結果について述べ，考察を行う．

実験には L2 キャッシュサイズの異なる CPU を持つ 2 台の PC として，Intel Core 2 Duo

Vol.2009-ARC-183 No.14
Vol.2009-OS-111 No.14
2009/4/23

表 1 計測結果（Core 2 Duo T7300 2.0GHz 4MB L2 キャッシュ）

	平均値			最悪値		
環境	クロック数	時間（us）	比率	クロック数	時間（us）	比率
L4	120	0.060		150	0.075	
HAL	330	0.165	2.75	410	0.206	2.73
VMM	3250	1.630	27.08	3430	1.720	22.87

表 2 計測結果（Atom N270 1.60GHz 512KB L2 キャッシュ）

	平均値			最悪値		
環境	クロック数	時間（us）	比率	クロック数	時間（us）	比率
L4	114	0.071		216	0.135	
HAL	864	0.540	7.58	876	0.548	4.06
VMM	5014	3.134	43.98	7704	4.815	35.67

T7300 2.0GHz[⋆1] 4MB L2 キャッシュを搭載する Dell Latitude D630，および Intel Atom N270 1.60GHz[⋆2] 512KB L2 キャッシュを搭載する Intel Atom Processor N270 and Mobile Intel 945GSE Express Chipset Customer Reference Board である．Core 2 Duo のデュアルコア機能，Atom のハイパースレッディング機能は BIOS で不使用の設定にし，どちらも 1CPU の状態で計測した．メモリはカーネルオプションで 256MB を指定した．Linux カーネルのバージョンは 2.6.27.5，L4 は OKL4 1.4.1.1 である．

実験としては割り込みハンドラの起動コストを RDTSC 命令を用いて計測した．計測したのは，RTC（Real-Time Clock）からの割り込みが発生，IDT に登録されたハンドラが起動し復帰に必要なレジスタを保存，セグメントの設定をした時点から，L4 のタイマ割り込みハンドラ（timer_interrupt 関数[⋆3]）が呼び出された時点までの時間である．この起動コストを，オリジナルの L4（L4），カーネルの機能拡張によるカーネル間隔離機能を持たない OS 共存環境（HAL），VMM によるカーネル間隔離機能を持つ OS 共存環境（VMM）について，2 台の PC 上で計測した．計測はアイドル状態で行った．HAL と VMM の環境ではアイドル状態では Linux が起動しているため，HAL と VMM の起動コストには，OS 切り替えのコストが含まれる．表 1，2 に 1 万回計測した平均値と最悪値を示す．比率はオリジナルの L4 を 1 とした場合の値である．

計測結果から，HAL による OS 共存環境での割り込みハンドラ起動コストが，VMM による OS 共存環境でのコストよりもかなり小さいことがわかる．Core 2 Duo では 9.85 倍，Atom では 5.80 倍，VMM の方がコストが大きい．HAL と VMM のどちらも OS 切り替えを行う．OS 切り替えでは，レジスタの入れ換えやページテーブルの切り替えが行われ，ページテーブルの切り替え時に TLB はフラッシュされる．カーネルはプロセス間で共有されるため，グローバルページとしてマップされ，ページテーブルの切り替え時にもその TLB はフラッシュされないが，OS 切り替えでは全ての TLB がフラッシュされるように，CR4.GPE フラグを一度落としてから CR3 へ次に実行する OS のページテーブルへのポインタを書き込む．また HAL は中間 GDT を使用するため，GDT のロードは 1 回多く行われる．しかし，モード切替を伴う VMM のコストの方がかなり大きい．

また，2 つの CPU 間で比率を比較すると，Atom の方が OS 共存環境導入によるオーバヘッド増加の割合が大きいことがわかる．Core 2 Duo と比較し，Atom は HAL ではオーバヘッド増加の割合が 2.76 倍，VMM では 1.62 倍となっている．これは比較の基準となっているオリジナルの L4 のコストが，Atom では小さいことが影響している．Core 2 Duo と比較し，Atom は，時間で 1.18 倍，クロック数では 0.95 倍となっている．これが HAL では時間で 3.27 倍，クロック数では 2.62 倍，VMM では時間で 1.92 倍，クロック数では 1.54 倍となっている．

4. 考　　察

まず，実装に関して考察を行う．HAL は VMM と比較し単純な機能を持ち，そのため実装量もより少ない．カーネル間の隔離機能を除いて，どちらもほぼ同等の機能を持っていると考えられるが，本論文執筆時点で HAL は 1149 行，VMM は 4344 行[⋆4]からなり，VM 環境を実現する VMM の方がかなり規模が大きい．しかし，VMM と OS カーネルの関係は，OS のカーネルとユーザプロセスの関係に相当し，IA-32 上では 4 レベルの特権レベルに VMM，OS カーネル，ユーザプロセスを当てはめることができるため，比較的容易に OS 切り替え部分を実装することができた．しかし，HAL では同一の特権レベルで OS カーネル自身の環境を定義する実行環境の切り替えを行う必要がある．そのため，経験不足も手伝い，L4 の変更を伴う中間 GDT を置く必要があるなど，やや工夫を要した．今後，L4 への

⋆1 /proc/cpuinfo は 1994.397MHz と表示するため，この値をクロック数から秒への変換に用いた．
⋆2 /proc/cpuinfo は 1599.990MHz と表示するため，この値をクロック数から秒への変換に用いた．
⋆3 マクロにより，IDT に登録されるラッパ関数とハンドラ本体の関数の 2 つが定義されるが，C で記述された本体側の関数を指す．
⋆4 VMM，HAL ともに空行，コメントを含む．

Vol.2009-ARC-183 No.14
Vol.2009-OS-111 No.14
2009/4/23

変更が少ない方法で OS 切り替えを可能にする方法を検討する必要がある．

次に，実験結果について考察を行う．実験は，L2 キャッシュサイズの異なる 2 種類の CPU，Intel Core 2 Duo と Atom を用いて行った．Atom の方が OS 共存環境導入によるオーバヘッド増加の割合が大きく，特に HAL 導入のオーバヘッド増加の割合が大きい．Core 2 Duo と Atom はキャッシュの大きさだけでなく，Core 2 Duo はアウト・オブ・オーダ実行を採用し投機実行も行い，また Atom ではイン・オーダ実行で投機実行を行わないなど，CPU としての機能の差も大きい．そのため，この差は単純にキャッシュの大きさだけでなく，CPU としての機能の差も影響していると思われる．しかしより正確な理由の解明には，Performance Monitoring Counter を用いるなどして，より詳細な測定を行い，分析を行う必要がある．

5. 関 連 研 究

OS 共存環境として VMM[5),13)] は長い歴史を持つ．VMM による OS 共存環境では，特権命令のエミュレーションや例外や割り込み発生時のモード切替などのオーバヘッドが伴うため実行効率は落ちるが，OS カーネルを VMM よりも低い特権レベルで実行することで OS 間や VMM の保護を可能である．また，実機ではなく仮想化に適した VM を提供することで実行時のオーバヘッドを小さくする手法として準仮想化[2)] がある．本論文で述べた VMM は，基本的には VM に影響を与えるイベントをトラップしエミュレーションすることにより実現されているが，ハイパーコールによる明示的なイベント通知と組み合わせることで，わずかの変更で効率の良い VM 環境を実現する Mesovirtualization[8)] という手法を用いている．

特権モードで複数 OS を共存させる手法は，OS 共存環境の導入によるオーバヘッドを抑えたいリアルタイムシステムでは一般的な手法である[3),4),16)]．これらのシステムでは，リアルタイムカーネルと汎用 OS の共存を実現しているが，リアルタイムカーネルはそのアプリケーションタスクも特権モードで実行するのが一般的である．本論文で述べた HAL は，アプリケーションタスクは非特権モードで実行する L4 と汎用 OS である Linux の共存を実現しており，リアルタイムカーネルとの共存とは異なっている．汎用 OS の共存という点では，coLinux[1)] や TwinOS[15)] により類似している．しかし，Linux と共存する L4 の用途を監視用に特化しているという点で異なっている．

6. まとめと今後の課題

信頼性やセキュリティ向上のためには，オペレーティングシステム（OS）カーネルの内部状態をカーネル外部から別のソフトウェアにより監視することが有効である．単一システム内でそのような監視を行うためには，監視対象となる OS と監視ソフトウェアを実行する OS からなる，複数 OS の共存環境が必要となる．OS 共存環境は，本来 OS で行うべき処理を阻害しないように，できるだけ軽量であることが望ましい．本論文では，VMM によるカーネル間隔離機能を持つ OS 共存環境とカーネルの機能拡張によるカーネル間隔離機能を持たない OS 共存環境の 2 つの環境を IA-32 システム上に構築し，OS 共存環境におけるカーネル間の隔離機能の有無による性能差について調査した．実験結果から，カーネル間の隔離機能の有無による性能差は大きいこと，また性能差は CPU によって変化することがわかった．

今後の課題は，HAL，VMM ともにより多くの実験を行うことが出来るように開発を進めること，Performance Monitoring Counter を用いるなどしてより詳細な測定を行い性能差について分析を行うこと，Intel VT-x に対応した VMM を開発し，それとの比較を行うことなどがある．

謝　　辞

本研究の一部は JST-CREST「実用化を目指した組込みシステム用ディペンダブル・オペレーティングシステム」の助成による．

参 考 文 献

1) D.Aloni. Cooperative Linux. In *Proceedings of the Linux Symposium*, Vol. 1, pp. 23-32, 2004.
2) P.Barham, B.Dragovic, K.Fraser, S.Hand, T.Harris, A.Ho, R.Neugebauer, I.Pratt, and A.Warfield. Xen and the Art of Virtualization. In *Proceedings of the 19th ACM Symposium on Operating System Principles*, pp. 164–177, October 2003.
3) G. Bollella and K. Jeffay. Support for Real-Time Computing within General Purpose Operating Systems. In *Proceedings of the IEEE Real-Time Technology and Applications Symposium*, May 1995.
4) B.Carpenter, M.Roman, N.Vasilatos, and M.Zimmerman. The RTX real-time subsystem for windows NT. In Proceedings of the USENIX Windows NT Workshop on The USENIX Windows NT Workshop, August 1997.

Vol.2009-ARC-183 No.14
Vol.2009-OS-111 No.14
2009/4/23

5) R.P.Goldberg. Survey of Virtual Machine Research. *IEEE Computer*, pp. 34–45, June 1974.
6) V.Goyal, E.W.Biederman, H.Nellitheertha. Kdump, A Kexec Based Kernel Crash Dumping Mechanism. In *Proceedings of the Linux Symposium*, July 2005
7) IntelCorporation. IA-32 Intel Architecture Software Developer's Manual.
8) M.Ito and S.Oikawa. Mesovirtualization: Lightweight Virtualization Technique for Embedded Systems. In *Proceedings of the 5th IFIP Workshop on Software Technologies for Future Embedded & Ubiquitous Systems*, Springer-Verlag LNCS 4761, pp. 496-505, May 2007.
9) 伊藤 愛, 追川 修一. Gandalf VMM におけるシャドウページングの実装と評価. 情報処理学会論文誌：コンピューティングシステム，ACS21，2008.
10) P. Kamp and R. Watson. Jails: Confining the Omnipotent Root. In *Proceedings of the 2nd International System Administration and Networking Conference*, May 2000.
11) T. Nakajima, H. Ishikawa, Y. Kinebuchi, M. Sugaya, S. Lei, A. Courbot, A. Zee, A.Aalto, and K.Duk. An Operating System Architecture for Future Information Appliances. In *Proceedings of the 6th IFIP International Workshop on Software technologies for future Embedded and Ubiquitous Systems (SEUS 2008)*, Springer-Verlag LNCS 5287, pp. 292–303, 2008.
12) OKL4. http://www.ok-labs.com/.
13) M.Rosenblum and T.Garfinkel. Virtual Machine Monitors: Current Technology and Future Trends. *IEEE Computer*, pp. 39–47, May 2005.
14) L.Sun, D.K.Nilsson, T.Katori and T.Nakajima. Online Self-Healing Support for Embedded Systems. In *Proceedings of the 12th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2009)*, 2009. (to appear)
15) 田渕 正樹，伊藤 健一，乃村 能成，谷口 秀夫. 二つの Linux を共存走行させる機能の設計と評価. 電子情報通信学会論文誌（D-I），Vol.J88-D-I，No.2，2005.
16) V. Yodaiken and M. Barabanov. Real-Time Linux. In *Proceedings of Linux Applications Development and Deployment Conference (USELINUX)*, January 1997.