SPE Observer: Cell/B.E. のSPEを用いたOS監視システム

永田 卓也[†1]　光来 健一[†1]

近年、カーネルルートキットを用いたOSの改ざんが問題となっている。OSが攻撃されると、OSの上で動作するセキュリティソフトの正常な動作が保証されなくなるためOSの完全性を保証する必要がある。本稿ではCell/B.E.のSPE上で安全にOS監視を行うSPE Observerを提案する。SPE Observerは、SPE Isolationモードを用いることでOS監視システムの完全性と機密性を保証し、外部のセキュリティプロキシからのハートビートによりOS監視システムの動作状況を監視する。また、SPEを占有しないようにスケジューリングを行う。実験により、SPE Observerを用いてOSの改ざんを検知することができ、OS監視のスケジューリングを行うことで演算性能の低下が大幅に抑制できることが分かった。

SPE Observer: An OS Monitoring System Running on an SPE in Cell/B.E.

Takuya Nagata[†1] and Kenichi Kourai[†1]

Recently, operating systems (OSes) are often altered by kernel rootkits. Since security software running on a compromised OS may not run correctly, it is necessary to guarantee the integrity of OSes. In this paper, we propose SPE Observer, which securely monitors an OS from an SPE in Cell/B.E. SPE Observer guarantees the integrity and the confidentiality of the OS monitoring system with the SPE isolation mode. It monitors the running status of the OS monitoring system with heartbeats from the external security proxy. It also schedules the OS monitoring system to prevent one SPE from being occupied. According to our experiments, it was shown that SPE Observer enables detecting the alteration of an OS and that the performance is improved by scheduling the OS monitoring system.

†1 九州工業大学

1. はじめに

近年、ネットワークを経由して計算機が攻撃されることが増えている。各ユーザはAntiVirus等のセキュリティソフトウェアを利用してそれらの攻撃に備えるのが一般的である。しかし、セキュリティソフトウェアはOSの機能を利用して監視を行っており、adoreやknark、SucKIT等のカーネルルートキットを用いた攻撃によってOSが改ざんされてしまった場合、セキュリティソフトウェアの実行結果を信頼することはできなくなる。

セキュリティソフトウェアの信頼性を向上させるためにはOSが正しく動作していることを保証すべきである。TPM等のハードウェアを用いることで、OSの完全性を保証することができる。しかし完全性チェックはブート時のみに行われるため、システム実行中のOSが改ざんされていないことを保証するのは困難である。カーネルルートキットはシステムの実行中に感染するため、システムが動作している間もOSの完全性を保証する必要がある。

本稿では、Cell/B.E.のSPE上でOS監視システムを安全に動作させることができるシステムSPE Observerを提案する。SPE ObserverはSPE Isolationモードを用いることにより、OS監視システムの完全性および機密性を保証する。さらに、外部のセキュリティプロキシからOS監視システムに定期的にハートビートを送ることで、OS監視システムが停止させられていないかどうか監視する。また、スケジューリングを行うことでOS監視システムを必要な時だけ動かし、SPEをアプリケーションに解放することができる。

我々はPlayStation 3 Linux上にSPE Observerを実装し、SPE上でカーネルメモリをチェックするOS監視システムを開発した。このOS監視システムを用いて、カーネルの改ざん検知と性能評価を行った。実験により、カーネルの改ざんを検知できることが確認できた。OS監視用にSPEを占有した場合アプリケーションの演算性能が低下する場合があるが、OS監視システムの実行をスケジューリングすることで性能低下を抑えることができる。

以下、2章で従来のOS監視システムとその問題点について述べる。3章でSPE Observerを提案し、4章でその実装について述べる。5章でSPE Observerの性能について述べる。6章で関連研究に触れ、7章で本稿をまとめる。

2. 従来のOS監視システム

一般的なシステム構成ではOSの上でセキュリティソフトウェアが動いており、OSに対してシステムコールを発行しながらセキュリティチェックを行っている。攻撃者がカーネルルートキットを使用してOSを改ざんした場合、システムコールの処理内容を変えられてし

まう恐れがある。そのため、改ざんされた OS 上で動くセキュリティソフトウェアは正常な動作をしているとは言えない。

例えば、ウィルススキャンソフトウェアはファイルを開き、その内容をチェックし、結果をユーザに通知するという処理を行っている。ファイルを開く動作や結果の出力は OS へのシステムコールを使って実行されている。OS が改ざんされた場合には、攻撃に使うファイルが存在しないように見せられたり、チェックするファイルを他のものにすり替えられたりする可能性がある。また、異常を検知してもユーザ側に通知できないという恐れもある。

セキュリティソフトウェアの信頼性を向上させるには、OS が改ざんされていないことをチェックする必要がある。しかし、OS 監視システムが OS の上で動いている場合、OS 監視システムも改ざんされたシステムコールを使用してしまうため、OS が改ざんされていないことを保証するのは難しい。カーネルモジュールなどを用いて OS 内部に OS 監視システムを埋め込むことも可能だが、改ざんされた OS と同一階層で動いている OS 監視システムが正しく動くという保証はない。

OS より下層にある TPM [1] 等のハードウェアを用いることで、OS が改ざんされていないかどうかをチェックすることができる。ハードウェアから chain of trust が繋がっていれば、ハードウェアそのものは改ざんすることができないため、OS の完全性を保証することができる。しかし、TPM が OS の整合性をチェックするのは起動時のみであるため、システムを動かしている間に OS が改ざんされたことを検知することはできない。

仮想マシンを用いて OS より下層で監視を行う手法 [2] も存在する。仮想マシン技術は、ゲスト OS の下に仮想マシンモニタ（VMM）と呼ばれるソフトウェアを走らせ、ゲスト OS からハードウェアへのアクセスを仮想化する技術である。この技術を用いると、ゲスト OS よりも下層で動く VMM がゲスト OS の監視を行うことで、ゲスト OS が改ざんされていないことを保証することができる。しかし、VMM はソフトウェアであるため、複数のバグが存在する可能性がある。例えば、ゲスト OS から VMM に対して攻撃を行えるバグも報告されている [3] [4]。VMM が改ざんされてしまうと、その中で動く監視システムも正常に動く保証がなくなる。

3. SPE Observer

本稿では、Cell/B.E. を対象として OS が動いている PPE から物理的に隔離された SPE 上で OS の監視を行う SPE Observer を提案する。SPE Isolation モードを用いることにより OS 監視システムの完全性と機密性が保証される。また、外部のセキュリティプロキシから OS 監視システムに対してハートビートを送ることで OS 監視システムの動作状況を監視する。図 1 に SPE Observer の全体図を示す。

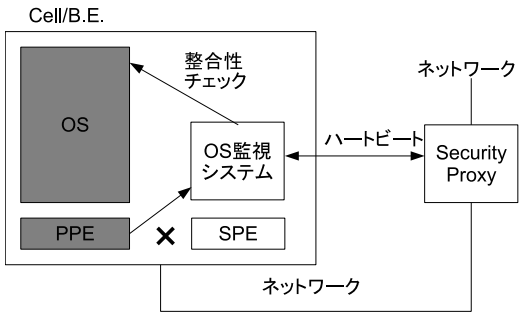

図 1 SPE Observer のシステム構成

3.1 Cell/B.E.

Cell/B.E. [5] は OS や通常のプロセスを動かす PPE という制御コアと、演算処理に特化した SPE という演算コアから構成されるヘテロジニアス・マルチコアプロセッサである。各コアは EIB と呼ばれるリングバスでつながれており、EIB を通じてメインメモリにアクセスを行う。SPE はローカルストア（LS）と呼ばれるメモリをコア内部に持ち、LS はメインメモリから物理的に分離されている。SPE は LS にプログラムをロードし、DMA 転送を行ってメインメモリと LS の間で情報をやり取りしながら処理を行う。

3.2 Isolation モードを用いた安全な実行

SPE Isolation モードは OS 監視システムを安全に実行することを可能にする。Isolation モードは SPE が内部に持つ LS に対し PPE や他の SPE からのアクセスを禁止する。SPE 上で動かすプログラムや SPE 内部で処理されるデータは全て LS の内部に置かれるため、Isolation モードを用いれば LS 内部にある実行中のプログラムやデータを外部から書き換えることはできず、OS 監視システムの完全性を保証できる。また、実行中のプログラム内部で扱っている暗号鍵等の機密情報を読み出されることもないため、OS 監視システムの機密性も保証できる。

Isolation モードで動く OS 監視システムは Secure Loader [6] によって安全に SPE にロードされる。まず、暗号化された SecureLoader が SPE にロードされ、SPE はハードウェアに埋め込まれた暗号鍵を用いて Secure Loader の署名を検証した後、Secure Loader

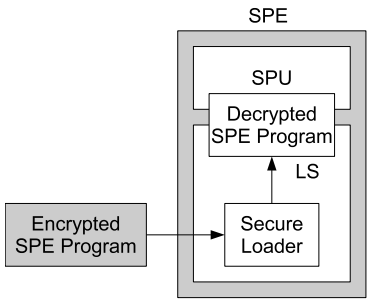

図 2 Secure Loader による安全なロード

を復号化する。次に、図 2 のように SecureLoaser が暗号化された OS 監視システムを LS にロードし、Secure Loader が持つ暗号鍵を用いて署名を検証した後、OS 監視システムを復号化する。

ハードウェアによって完全性が保証された Secure Loader が監視プログラムの完全性をチェックするため、攻撃者は改ざんした監視プログラムをロードすることができない。さらに、ディスク上に置かれている監視プログラムの機密性も保証される。監視プログラムを復号化する暗号鍵は Secure Loader が持ち、Secure Loader を復号化する暗号鍵はハードウェア内部にあるため、攻撃者は監視プログラムを復号化することができない。

Isolation モードで動いている SPE 上のプログラムの完全性と機密性は保たれるが、PPE は SPE 上の OS 監視システムを停止させることができる。これは PPE 上の OS がシステム全体を管理しているためである。しかし、OS 監視システムを停止されたとしても機密性を保つことができる。プログラムを停止して Isolation モードを解除すると、ハードウェアによって LS の内容がすべて消去されるためである。PPE から別のプログラムをロードして LS 上の機密情報を取得することはできない。

3.3 セキュリティプロキシ

OS 監視システムが停止されたことを検知できるようにするために、外部のセキュリティプロキシから SPE 上の OS 監視システムに定期的にハートビートを送る。PPE から SPE 内部のプログラムを終了させられるると OS の改ざんを検知できなくなり、その上で動くセキュリティソフトウェアの信頼性も保証できなくなる。ハートビートへの応答がなくなれば、セキュリティプロキシは OS 監視システムが停止されたと判断して、即座に対処を行うことができる。また、攻撃者に OS 監視システムを止めさせないようにするための抑止力にもなる。

正しい OS 監視システム以外が応答を行えないようにするために、セキュリティプロキシはハートビートを送る際に暗号通信を行う。Isolation モードにより SPE 内部の暗号鍵を攻撃者が取得することはできないので、ネットワーク上を流れるハートビートを解読することは困難である。そのため、ハートビートに対して正しい応答を返すことができるのは OS 監視システムが動いている SPE のみであることが保証できる。

セキュリティプロキシはハートビートに対して正しくない応答を返されるか一定時間応答がなかった場合、監視対象マシンに出入りするパケットを遮断する。セキュリティプロキシは監視対象マシンに出入りするパケットを中継しているため、すべてのパケットは必ずセキュリティプロキシを通る。パケットを遮断されると攻撃者はネットワークを経由した攻撃を継続することができなくなる。攻撃プログラムは動き続けるが、監視対象マシンはネットワークから遮断されているため、被害をこのマシンにとどめることができる。外部への攻撃を継続するためには OS 監視システムを動かさざるを得ないため、OS が改ざんされていないかチェックを継続することができる。

3.4 OS 監視のスケジューリング

SPE Observer は OS 監視システム動かすために、SPE を少なくとも 1 つ使用する。OS の常時監視を行う必要がある場合には、SPE を完全に占有することになる。Cell/B.E. は SPE を用いて並列処理を行うことで性能を引き出すプロセッサであるため OS 監視のために SPE を占有すると性能への影響が大きい。

そこで、SPE Observer はスケジューリングを行うことで必要な時にだけ OS 監視システムを動かすことを可能にしている。例えば、OS カーネルの改ざんの検査は 1 秒おきに行えば十分なことが多い。スケジューリングを行うと、毎回 OS 監視システムを起動するオーバーヘッドがかかるが、監視を行っていない時は OS 監視用に使用されていた SPE を他のアプリケーションに割り当てることができる。これにより、アプリケーションの性能低下を抑えることができる。スケジューリングは OS の機能にも依存するが、改ざんされた OS が監視システムを正しく実行しない場合には、セキュリティプロキシがそのことを検出することができる

4. 実　　装

IBM が提供している Cell/B.E. SDK 3.1 と Cell/B.E. Security SDK を用いて SPE Observer を実装した。対象とした OS は Linux 2.6.27 である。

4.1 SPE での安全な実行

ハードウェアが提供する SPE Isolation モードを使用することができなかったため、Security SDK によって提供されている Isolation モードのエミュレーションを利用した。ハードウェアの Isolation モードとは違い、暗号化されていない Secure Loader が通常のモードで SPE にロードされる。ハードウェアの Isolation モードと同様に SPE を占有させるために、SPE_NOSCHED フラグをつけて実行するようにした。このフラグをつけることで OS 監視システムを実行している SPE ではコンテキストスイッチが行われないようにすることができる。

4.2 カーネルメモリの監視

SPE にカーネルメモリへのアクセス権限を持たせるために、SPE の持つ Memory Flow Controller（MFC）の状態レジスタ 1 の Problem-State ビットをクリアする。MFC は各 SPE 内にあり、SPE がメインメモリに情報を送受信する際に DMA 転送を行っている SPE から MFC のレジスタにデータを書き込むことで DMA 転送を行うことができる。

通常、SPE は PPE 上のプロセスのアドレス空間にしかアクセスできないので、SPE がカーネルアドレス空間にアクセスできるようにするために、SPE の持つ Segment Lookaside Buffer (SLB) にマッピングを登録する。Cell/B.E. ではカーネルメモリを含むメモリアドレスが実効アドレスで管理されている。実効アドレスは仮想アドレスにマップされ、仮想アドレスは物理アドレスにマップされている。各 SPE が持つ SLB は実効アドレスから仮想アドレスへの変換テーブルであり、SLB に実効アドレスからのマッピングを登録することでメインメモリにアクセスすることができる。

カーネルメモリからの DMA 転送と監視処理をオーバーラップさせるために、SPE 上の OS 監視システムはダブルバッファリングを行う。DMA 転送を行うには時間がかかるが、転送の完了を待っている間に別の処理を行うことができる。具体的には、DMA 転送の完了を待っている間に直前に取得したメモリブロックに対して監視処理を行う。一例として、DMA 転送によりカーネルのテキスト領域と読み込み専用データ領域を取得し、その SHA-1 ハッシュ値をあらかじめ計算した値と比較する OS 監視システムを実装した。

4.3 ハートビート

セキュリティプロキシから SPE 上の OS 監視システムにハートビートを送るために、PPE 上でリレープロセスを動かしてパケットを中継させる。SPE にプロトコルスタックを実装し、直接 NIC にアクセスすることで SPE が直接通信を行うことも可能だが、ネットワーク処理のために LS を大量に消費してしまい、監視処理を行うのが難しくなる。

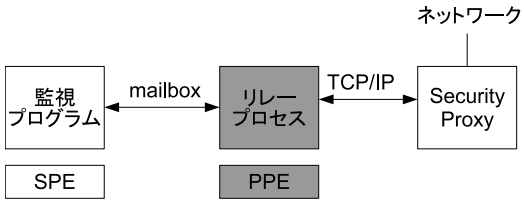

図 3 セキュリティプロキシからのハートビート

セキュリティプロキシとリレープロセスの間の通信には TCP/IP を用い、PPE と SPE の間の通信にはメールボックスを用いる。メールボックスとは SPE の MFC が持つメールボックスチャンネルを介して 32bit のデータをやり取りする機能である。この機能を用いれば、SPE が Isolation モードで動作しても PPE と SPE の間でデータを送受信することが可能である。SPE がメッセージを書き込むチャンネルと、メッセージを読み込むチャンネルが用意されている。

セキュリティプロキシはハートビートで送るメッセージを乱数で生成し、あらかじめ共有しておいた暗号鍵を用いて暗号化して監視対象マシンに送る。このメッセージを PPE で動くリレープロセスが一旦受け取り、メールボックスを用いて SPE に渡す。SPE 上の OS 監視システムはメールボックスの受信キューに格納されたメッセージを受け取り、復号化と応答メッセージの生成を行う。作成した応答メッセージを暗号化し、メールボックスの送信キューが空になったら PPE に送信する。PPE 上のリレープロセスが受け取った応答メッセージをセキュリティプロキシに送ると、セキュリティプロキシはそれを復号化してチェックを行う。

4.4 OS 監視のスケジューリング

OS 監視システムのスケジューリングはセキュリティプロキシが行う。OS 監視システムを起動する時には、セキュリティプロキシが PPE 上のリレープロセスに対して起動メッセージを送る。起動メッセージを受け取ったリレープロセスは OS 監視システムを SPE にロードし、SPE に対して起動メッセージを送る。SPE で OS 監視システムが起動すると、開始メッセージをセキュリティプロキシに送る。一連の OS 監視が終了すると、OS 監視システムは PPE 上のリレープロセスを経由してセキュリティプロキシに終了メッセージを送り、SPE を解放する。終了メッセージを受け取ったセキュリティプロキシは、次に OS 監視システムを起動する時刻まで待機する。これらのメッセージのやりとりはハートビートと同様

に暗号化されて安全に行われる。

SPE に OS 監視システムをロードする時には Linux カーネルが SPE のスケジューリングを行う。SPE スケジューラはまず、空いている SPE が存在するか調べ、空きがある場合はその SPE に OS 監視システムをロードする。SPE に空きがない場合はアプリケーションが使っている SPE を 1 つ選び、LS の内容をメインメモリに保存してコンテキストスイッチを行う。

OS 監視システムを適切にスケジューリングできるようにするために、我々は Linux カーネル内の SPE スケジューラに修正を行った。既存の SPE スケジューラでは、SPE_NOSCHED フラグを立ててプログラムを実行した場合、SPE に空きがないといつまでもスケジューリングされないというバグがあった。このバグを回避するために、SPE_NOSCHED フラグが立っている場合には優先度を最大にすることで即座にスケジューリングされるようにした。また、特定の SPE のみをコンテキストスイッチする実装になっていたため、SPE 間で公平にコンテキストスイッチが起こるように修正した。

5. 実　　験

実験には Sony Computer Entertainment 社製の PlayStation3 80GB モデルを用いた。このマシンで使用可能な SPE 数は 6 つである。OS として Fedora 9 をインストールした。

5.1 カーネル改ざんの検知

SPE Observer を用いて OS 監視システムを動作させ、カーネルの改ざんを検出できるかどうか調べた。OS 監視システムにはカーネルのテキスト領域と読み込み専用領域のハッシュ値を計算するプログラムを用いた。実験対象として、オリジナルのカーネル、システムコールを追加したカーネル、システムコール内部に printk を追加したカーネルの 3 つを用意した。実験の結果、オリジナルのカーネルを動かした時のみ、ハッシュ値があらかじめ計算しておいた値と同じになることが確かめられた。

5.2 監視の実行時間

OS 監視システムが 1 回の監視を行うのにかかる時間を測定した。1 回の監視で 12 MB のカーネルメモリを DMA 転送によって取得し、その SHA-1 ハッシュ値を計算する。この OS 監視システムの実行時間は 24 ミリ秒であった。用いた SHA-1 のライブラリでは SIMD 命令を利用していなかったため、SIMD 命令を利用することで実行時間を短くすることが可能だと考えられる。

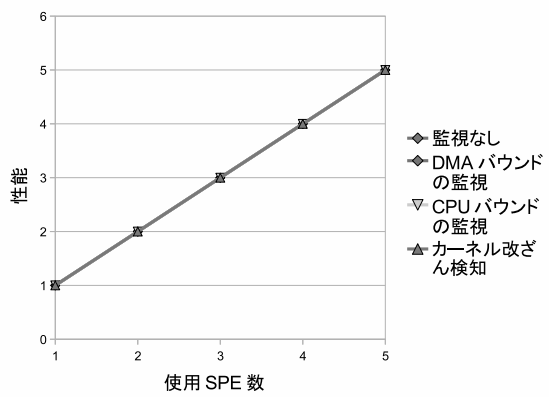

図 4　CPU バウンドのアプリケーションへの影響

5.3 監視を行うことによる影響

1 つの SPE を占有して様々な OS 監視システムを動作させ、アプリケーションの性能にどのような影響があるかを調べた。OS 監視システムとして、上記のカーネル改ざんを検知するプログラムおよび、比較のために、計算のみを繰り返す CPU バウンドのプログラム、DMA 転送のみを繰り返す DMA バウンドのプログラムを用いた。また、監視を行わない場合についても実験を行った。

5.3.1 CPU バウンドのアプリケーション

まず、並列に計算を行う CPU バウンドのアプリケーションを用いて実験を行った。このアプリケーションが使用する SPE の数を 1 から 5 まで 1 ずつ増やして、一定の計算を完了するのにかかる時間を測定した。図 4 に様々な OS 監視システムを動かした時のアプリケーションの性能を示す。1 つの SPE のみを使って実行した時の性能を 1 としている。アプリケーションが使用する SPE の数が増えると、それに比例して性能も向上していることが分かる。一方、どのような OS 監視システムを動かしても、アプリケーションの性能は監視を行わなかった場合と変わらなかった。

5.3.2 DMA バウンドのアプリケーション

次に、並列にメインメモリから DMA 転送を行う DMA バウンドのアプリケーションを用いて実験を行った。このアプリケーションが一定のメモリを取得するのにかかる時間を測定した。図 5 に様々な OS 監視システムを動かした時のアプリケーションの性能の変化を

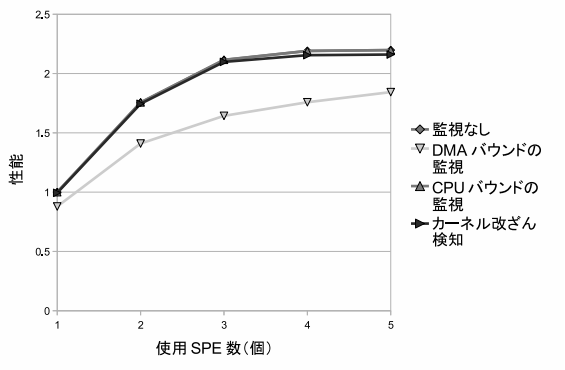

図 5 DMA バウンドのアプリケーションへの影響

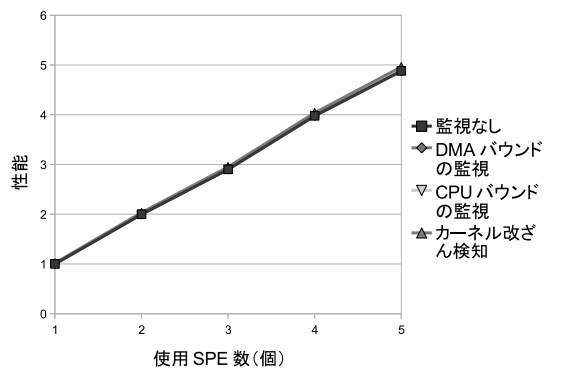

図 6 実アプリケーションへの影響

示す。 監視を行っていない場合でもアプリケーションの性能向上は 2.2 倍にとどまっている。これは DMA の帯域を使い切ってしまうためである。その結果として、OS 監視システムが DMA バウンドの場合、DMA 転送の干渉のためにアプリケーションの性能が 12 %〜22 %低下している。

一方、OS 監視システムが CPU バウンドの場合、監視を行っていない場合と性能がほぼ等しくなっている。これはアプリケーションの行っている DMA 転送と干渉しないためである。同様に、カーネルの改ざん検知を行う OS 監視システムを動かした場合もアプリケーションの性能はほとんど低下しなかった。カーネルの改ざん検知ではカーネルメモリから DMA 転送を行っているが、ハッシュ計算にかかる時間のほうが支配的になっているためである。

5.3.3 実アプリケーション

実アプリケーションとして、IBM が Cell/B.E. のデモとして提供している行列の乗算を行うプログラムを用いて実験を行った。このアプリケーションは指定したサイズの行列に対して、指定した個数の SPE を使って同期をとりながら並列に計算を行う。図 6 に実験結果を示す。CPU バウンドのアプリケーションと同様に、OS 監視システムを動かすことによる影響はほぼ見られなかった。

5.4 SPE を占有することによる影響

OS 監視システムが SPE を 1 つ占有している時に、SPE を 6 つ使用するように設計されているアプリケーションを実行することの影響を調べた。この場合、アプリケーションが使

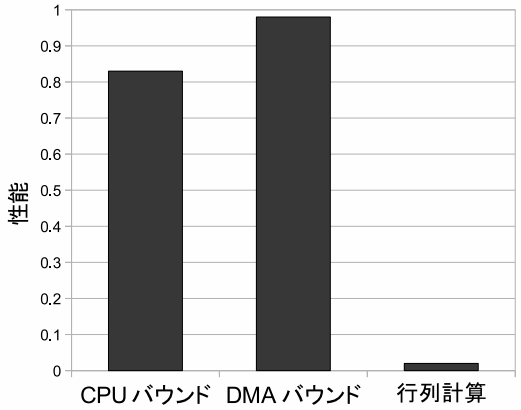

図 7 コンテキストスイッチによる性能低下

える SPE は 5 つとなる。アプリケーションには 5.3 節の 3 種類を用い、SPE スレッドを 6 つ作成した場合の性能を測定した。OS 監視システムにはカーネルの改ざん検知を行うプログラムを用いた。図 7 に OS 監視システムを動作させていない状態で測定した各アプリケーションの性能を 1 とした時の性能を示す。

CPU バウンドのアプリケーションの場合、性能は 83 %に低下している。6 つの SPE スレッドを使うアプリケーションを 5 つの SPE で動かさなければならないため、Linux カー

ネルの SPE スケジューラがコンテキストを切り替えながら実行を行う。このコンテキストスイッチは 5 つの SPE 間で平等に行われ、アプリケーションが実行を完了するまでの時間は $\frac{5}{6}$ となった。

一方、DMA バウンドのアプリケーションの場合はほとんど性能が低下しなかった。つまり、この場合、6 つの SPE スレッドを 5 つの SPE で動かしても実行にかかる時間はほぼ同じということである。用いた OS 監視システムはアプリケーションの DMA 転送にほとんど影響を与えないため、5 つの SPE だけで DMA 転送を行うことになる。SPE を 5 つ使う場合、図 5 のように DMA 帯域を使い切っているため、6 つの SPE を使う場合と比べて各 SPE の性能が $\frac{6}{5}$ に向上する。そのため、使える SPE の数が $\frac{5}{6}$ に減った分の性能低下を補うことができている。

行列計算については著しく性能が低下している。このアプリケーションは 6 つの SPE スレッドが少し計算を行うたびに、メールボックスを用いて全体の同期を取っている。そのため、1 つでも SPE スレッドの処理が遅れると他のすべての SPE スレッドが待たされることになる。6 つの SPE スレッドをスケジューリングして 5 つの SPE で動かすとメールボックスでの待機時間が長くなるのが性能低下の原因である。

5.5 OS 監視のスケジューリングによる性能改善

OS 監視システムが 1 つの SPE を占有するのではなく、スケジューリングをを行うことでアプリケーションの性能を向上させられることを確かめる実験を行った。5.4 節の 3 種類のアプリケーションと OS 監視システムを用いて、OS 監視の起動間隔を変えて各アプリケーションの性能を測定した。図 8、図 9、図 10 に実験結果を示す。

すべてのアプリケーションにおいて、OS 監視システムが SPE を占有するよりもスケジューリングを行ったほうが性能の改善が見られた。行列計算は他のアプリケーションと異なり、同期を取るための待ち時間が発生するため、OS 監視システムの起動間隔が短いと性能がよくない。それでも、OS 監視に SPE を占有した場合よりも性能が改善されており、OS 監視システムの起動間隔を長くすれば十分に性能が改善している。起動間隔を 200 ミリ秒以上にすれば、性能低下を SPE1 つ分に相当する 17 %以下に抑えることができる。このことから、SPE を 6 つ使うように設計されたアプリケーションであっても大幅な性能低下を避けることができることが分かる。

6. 関 連 研 究

SPE Isolation モードを用い、PPE 側で実行するアプリケーションの完全性を SPE から保証する CodeVerification Service [6] が提案されている。このサービスは PPE がアプリケーションをシステムメモリにロードしたあと、Isolation モードで動いている Code VerificationService にアプリケーションの検証を依頼する。アプリケーションがロードされているシステムメモリを DMA 転送し、SPE 内の暗号鍵を使ってアプリケーションの署名を検証する。Code VerificationService から改ざんされていないという結果が返されたら、PPE はアプリケーションを安全に実行することができる。

HyperCheck [7] は Intel や AMD の CPU によって提供されている System Management

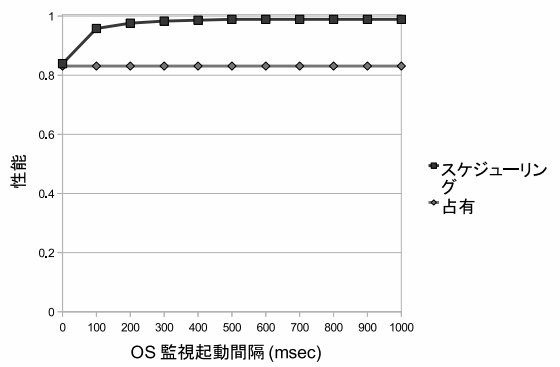

図 8 スケジューリングによる性能改善（CPU バウンド）

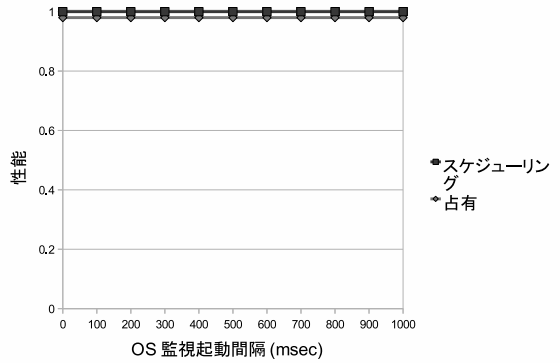

図 9 スケジューリングによる性能改善（DMA バウンド）

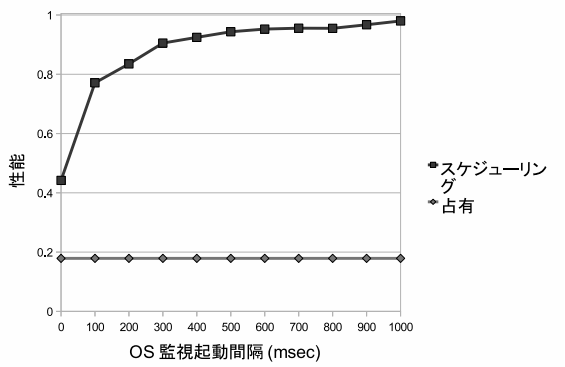

図 10 スケジューリングによる性能改善（実アプリケーション）

Mode（SMM）と呼ばれるモードを利用して OS の監視を行う。これらの CPU では。SMI と呼ばれる最高優先度の割り込みが発生した時に、SMRAM 上のプログラムが実行される。SMRAM は通常のモードではアクセスすることができないメモリ領域である。このモードを用いれば OS や他のアプリケーションから実行を邪魔されることなく、安全に OS 監視を実行することができる。ただし、基本的に BIOS からのみ利用可能であり、様々な OS 監視システムを使うのは容易ではない。

Flicker [8] は Intel TXT や AMD SVM などのハードウェア機構を使ってセキュリティの必要なアプリケーションを安全に動作させることを可能にする。このようなアプリケーションを動かす際には、すべてのコアでのプログラム実行を OS も含めて停止させ、TPM を用いてロードされるアプリケーションの完全性を検証する。そして、割り込みや DMA などを禁止した状態でアプリケーションを実行する。Flicker ではセキュアアプリケーションの実行中は他のプログラム実行が完全に停止するため、常時動作させる必要がある監視プログラムには向かない。また、SMM から攻撃が可能であることが指摘されている [9]。

7. ま と め

Cell/B.E. の SPE 上で安全に OS 監視を行うことができる SPE Observer を提案した。SPE Isolation モードを用いることにより、OS 監視システムの完全性と機密性が保証される。外部のセキュリティプロキシからハートビートを送ることにより OS 監視システムの動作状況を監視し、スケジューリングを行うことで OS 監視システムを必要な時だけ動かすことができる。実験により、SPE Observer を用いて OS の改ざんを検知でき、OS 監視システムがアプリケーション性能に及ぼす影響はスケジューリングによって改善可能であることが分かった。今後の課題は、より多くの種類の OS 監視システムを作成できるようにすることである。

謝　　辞

本研究の一部は、科学技術振興機構 戦略的創造研究推進事業（CREST）研究領域「実用化を目指した組み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」、および、科学研究費補助金（課題番号：21500039）による。

参 考 文 献

1) Trusted Computing Group: Trusted Platform Module, `http://www.trustedcomputinggroup.org/`.
2) Garfinkel, T., Rosenblum, M.: A Virtual Machine Introspection Based Architecture for Intrusion Detection, In Proc. Symp. Network and Distributed System Security, pp.191–206 (2003).
3) CVE: CVE-2008-4405 (2008).
4) CVE: CVE-2008-2004 (2008).
5) SCEI: Cell Broadband Engine Architecture Version 1.02 (2007).
6) Murase, M., Plouffe, W., Shimizu, K. and Sakamoto, M.: Effective Implementation of Cell Broadband Engine Isolation Loader, In Proc. Computer and Communications Security Conf. (2009).
7) J.Wang, A.S. and Ghosh, A.K.: HyperCheck: A Hardware-Assisted Integrity Monitor, In Proceedings of International Symposium on Recent Advances in Intrusion Detection (RAID) (2010).
8) McCune, J.M., Parno, B., Perrig, A., Reiter, M.K. and Isozaki, H.: Flicker: An Execution Infrastructure for TCB Minimization, In Proc. European Conf. Computer Systems, pp.315–328 (2008).
9) Wojtczuk, R. and Rutkowska, J.: Attacking Intel Trusted Execution Technology, Black Hat DC (2009).