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」
研究課題「高機能情報家電のためのディペンダ
ブルオペレーティングシステム」

研究終了報告書

研究期間　平成18年10月～平成23年3月

研究代表者：中島　達夫
（早稲田大学理工学術院　教授）

§1 研究実施の概要

(1)実施概要

本研究チームでは，DEOS フレームワーク D-RE の内の D-Visor と D-System Monitor を担当している．本報告書では，過去５年間の我々の取り組みに関して報告する．中島チームでは，4 つの研究課題に関して取り組んだ．１つ目の課題は，SPUMONE という仮想化層，２つ目は，モニタリングシステム，３つ目は，D-Visor 上で D-System Monitor を動作させるための標準 API の策定，４つ目は，ロギングシステム，5 つ目は，仮想化層向けの検証技術である．

SPUMONE 仮想化層はマルチコアプロセッサを用いた組込みシステム向けの仮想化層である．仮想化層は複数のオペレーティングシステムを同一のマルチコアプロセッサ上で動作することを可能とすることと，オペレーティングシステムの監視をするための D-System Monitor のセキュアな実行環境を提供する．SPUMONE は従来のクラウド環境向けの仮想化層と異なり，ハードウエア仮想化支援機能をサポートしていないプロセッサを利用した場合でも十分な性能が確保出来ること，リアルタイム性を保証出来ることを目指している．本研究課題では，RP1 と呼ばれる SH4 ベースのコアを 4 つ持つマルチコアプロセッサを利用して，提案するアイデアが実際に動作し，有効であることを示した．また，DEOS プロジェクト内で共同開発しているフレームワークである D-RE のコンポーネントの１つである D-Visor の１つとしても利用可能であることを示した．また，SPUMONE は研究コミュニティだけではなく，産業界からも興味を持たれ，CEATEC における講演やいくつかの国内企業の内部セミナーでも講演をおこない，研究の有効性を報告し，我々のノウハウを元に企業が提供する製品の要求にカスタマイズした仮想化層の構築の支援をおこなった．さらに，IEEE ASPDAC 等の著名な国際会議の招待論文としての発表やベル研究所，カーネギメロン大学，ラトガーズ大学等でも SPUMONE を紹介する講演をおこなった．

２つ目の課題は，SPUMONE 上で動作するオペレーティングシステムを監視するための D-System Monitor を安全に動作するための支援機能である．仮想化層を用いた類似研究は多数あるが，従来のアプローチはハードウエア仮想化支援機能がサポートされていない場合は著しくオペレーティングシステムの性能を低下させてしまっていた．そのため，クラウド環境で広く使用されているインテルのチップではハードウエア仮想化支援機能のサポートが一般的となっている．しかし，組込みシステムは多様なプロセッサを考慮する必要があり，また，現状の組込みアプリケーションプログラムで多用されている特権命令の呼び出しの効率低下をもたらすことなく仮想化を実現することを可能としている．成果に関しては，海外の大学でもセキュリティ研究のプラットフォームとして利用されたり，SPUMONE の機能の一部として SPUMONE の成果と一緒に招待論文や内外の講演において紹介をおこなった．

3 つ目の課題は，D-Visor 上で D-System Monitor を動作させるための標準 API の策定である．本領域では，D-System Monitor として，複数のチームにより監視機能が開発されてきたが，それぞれ独立に開発されたため，独自の方法で D-Visor を変更し，監視に必要な監視対象 OS にアクセスしていた．D-System Monitor ランタイム API の策定により，監視対象 OS へのアクセス方法を統一することができ，また複数の監視機能が使用できるようになる．そのため，新たな攻撃方法に対応した監視機能の開発を迅速に行えるようになり，また原因究明が強化され得る．D-System Monitor ランタイム API により，VM サブコアチームの他のチームが開発したシステムと統合され，最終成果報告会では VM サブコアチームの成果を統合したデモンストレーションをおこなうことがで

きるようになった.

4つ目の課題は,Linux用のロギングシステムである.本課題は平成21年度までDEOSセンターと共同で研究をおこない,成果をDEOSセンターに移管したため,平成21年度で研究を終了した.

5つ目の課題は,仮想化層向けの検証技術である.仮想化層はプロセッサやデバイスなどのハードウェアに依存した部分がその実装の多くを占め,抽象化された資源管理など,これまで静的検証の対象となってきた機能は,特に組み込み向けの仮想化層には含まれない.仮想化層のソフトウェアに記述されるハードウェアへの操作はビット操作で表され,ソフトウェアには状態が保存されないため,そのままでは静的検証を適用できない.そのため,ビット操作を抽象化した構造体への操作として表し,状態をトレースすることで静的検証が適用可能になることを示した.また,ハードウェアをモデル化し,その状態をトレースすることで,ハードウェアが正常な状態に保たれる範囲で操作を行っているか,検証可能であることを明らかにした.

以上の成果はDEOSプロジェクトに所属する他のプロジェクトと共同してD-RE内のコンポーネントとして1つのまとまった成果として利用可能とした.

(2)顕著な成果

1.組込みシステム向け仮想化層 SPUMONE
成果概要: 組込みシステム向けに特化した仮想化層であるSPUMONEをSH4ベースのマルチコアプロセッサ上で開発をおこなった.成果は,招待論文や展示会の講演等において紹介をおこなった.

2.SPUMONE 上の D-System Monitor 実行環境
成果概要: 仮想化上で動作するオペレーティングの性能を低下させずに D-System Monitor 実行環境を提供する方式を提案しSPUMONE上に実装した.成果は,招待論文や展示会の講演等において紹介をおこなった.

3.VMサブコアチームにおける研究開発の推進
成果概要:VMサブコアチームが発足した平成21および22年度にリーダー,平成23年度はサブリーダーとして研究開発を推進し,White Paper V.2, LAAS レポートを取りまとめた.また,D-System Monitorランタイム APIを策定するための研究開発を行い,その成果をもとにVMサブコアチームの成果を統合するデモシステムの開発を行った.

§2. 研究構想

(1) 当初の研究構想

本研究チームでは，以下に示す 4 つの面から仮想化を利用した情報家電機器向けのディペンダブル OS の構築に取り組んだ．

1) 1 つ目の計画は組込みシステム向けの仮想化層である SPUMONE に関する研究開発である．SPUMONE はマルチコアプロセッサ上で動作する仮想化層であり，オーバヘッドが小さく，ゲスト OS の変更量が小さいことが主な特徴である．SPUMONE 上では，D-System Monitor を Linux の異常動作から保護するための D-System Monitor 動作環境と Linux を保護するためのモニタリングサービスも存在する．D-System Monitor 動作環境はモニタリングサービスを実行するための小型の OS とその OS を Linux から保護するためのセキュアページャから構成される．小型 OS としては様々なリアルタイム OS が利用可能であり，本研究では，Toppers JSP，xv6，L4 等を利用することが可能である．モニタリングサービスは，ソースコードから自動的に OS のインバリアントを発見するツールと発見したインバリアントの実行時チェックをおこなうための実行時環境から構成される．

2) 2 つ目の計画は仮想化層高信頼性のための検証である．仮想化層を基盤として，高信頼 OS を構築するには，仮想化層自身の信頼性が確保されなければならない．仮想化層はプロセッサやデバイスなどのハードウェアに依存した部分がその実装の多くを占め，ビット操作で表される処理に対しても有効な検証技術が必要である．そのような検証技術について研究開発を行い，静的検証技術を適用し，ハードウェアへの操作の正しさ，ハードウェアが正常な状態に保たれる範囲で操作を行っているか，検証可能であることを明らかにした．

3) 3 つ目の計画は他チームの成果との統合を進めるための領域全体の活動である．本領域では，当初より成果の統合が求められていたため，そのための活動が行われた．プロジェクト開始時は，統合した先にある全体のアーキテクチャを明確にするための議論を行った．アーキテクチャを構成するコンポーネントが明らかになるにつれ，関係する個々のコンポーネントに関する議論を行い，それを研究開発に反映した．

4) 4 つ目の計画の，ロギングサービスは Linux 内のリソース使用量をモニタリングすることによりアプリケーションの振る舞いの異常を発見するためのカーネル内のサブシステムである．Linux 内に細粒度にリソース使用量を監視するためのサブシステムであるアカウンティングシステムと，LLTng を利用したロギングシステム，ログを解析するためのシステムから構成される．

(2) 新たに追加・修正など変更した研究構想

1) 提案書では高信頼 OS となっていた物が，DEOS アーキテクチャを議論していった結果，D-System Monitor 実行環境となって発展した．特に，D-System Monitor を実行するため，高信頼 OS はリアルタイム OS としてよりも，攻撃される等によりおかしな挙動を示す Linux から D-System Monitor を保護するための役割に変化していった．

2) ロギングシステムは，平成 21 年度に開発を終了し，その後の DEOS プロジェクト内における利用の管理を DEOS センターに移管したため，本チーム内での開発は平成 21 年度で終了することにした．

3) 「DEOS プロセス・アーキテクチャ」に関する特筆すべき研究項目としては，VM サブコアチーム

をまとめて行く中で，D-Visor の役割を明確にし，D-System Monitor を DEOS アーキテクチャに導入したことにある．また，D-System Monitor の1つの例として OS のインテグリティを保証するためのモニタリングサービスを開発した．D-System Monitor は河野チームでも採用され，D-RE に河野チームの成果を統合するためにも貢献している．

§３ 研究実施体制

(1)「早稲田大学」グループ

①研究参加者

氏名	所属	役職	参加時期
中島　達夫	**早稲田大学理工学術院**	教授	H18.10～H24.3
Alexandre Courbot	同上	研究助手	H19.4～H23.3
山邉　哲生	同上	研究助手	H19.4～H24.3
杵渕　雄樹	同上	研究助手	H18.10～H24.3
木村　浩章	同上	研究助手	H18.10～H24.3
李 寧	同上	学生研究員	H20.10～H24.3
刘 野枫	同上	学生研究員	H20.10～H24.3
林　宗翰	同上	学生研究員	H20.4～H24.3
島田 裕正	同上	学生研究員	H19.4～H24.3
三嶽 仁	同上	学生研究員	H21.4～H24.3
石井 正将	同上	学生研究員	H22.4～H24.3
安川 要平	同上	学生研究員	H22.4～H24.3
福島 拓	同上	学生研究員	H22.4～H24.3
吉井 章人	同上	学生研究員	H22.4～H24.3
李 承益	同上	学生研究員	H23.4～H24.3
七戸 貴大	同上	学生研究員	H23.4～H24.3
舩橋 義雄	同上	学生研究員	H23.4～H24.3
八神 貴心	同上	学生研究員	H22.4～H24.3
山口 大輔	同上	学生研究員	H23.4～H24.3
矢島　匠	同上	学生研究員	H23.4～H24.3
川勝 由美子	同上	研究員	H18.10～H24.3
森田 拓司	同上	学生研究員	H20.4～H23.3
権 奇徳	同上	学生研究員	H18.10～H22.3
Andrej van Zee	同上	学生研究員	H20.4～H23.3
巻島 一雄	同上	学生研究員	H19.4～H22.3
大野 有輝	同上	学生研究員	H19.4～H22.3
孫　雷	同上	研究助手	H19.4～H22.3
石川　広男	同上	学生研究員	H18.10～H21.9
Aleksi Aalto	同上	学生研究員	H20.4～H21.3
Fahim Kawsar	同上	学生研究員	H19.4～H21.3
香取 知浩	同上	学生研究員	H19.4～H21.3
神田 渉	同上	学生研究員	H19.4～H21.3
湯村 悠	同上	学生研究員	H19.4～H21.3
日下 照英	同上	学生研究員	H20.4～H21.3
鈴木 英恵	同上	学生研究員	H20.4～H21.3
高山 千尋	同上	学生研究員	H20.4～H21.3
菅谷　みどり	同上	学生研究員	H18.10～H20.3

②研究項目

・　仮想化を利用したディペンダブルＯＳの構築

(2)「筑波大学」グループ

①研究参加者

氏名	所属	役職	参加時期
追川　修一	筑波大学大学院 システム情報工学研究科	准教授	H18.10～
伊藤　愛	筑波大学大学院 システム情報工学研究科	修士課程 2 年	H18.10～H20.3
青柳　信吾	筑波大学大学院 システム情報工学研究科	修士課程 1 年	H19.4～H20.3

②研究項目

・　組込みシステムに適したアイソレーション機能の研究

§4 研究実施内容及び成果

4．1仮想化を利用したディペンダブルＯＳの構築 （早稲田大学 中島グループ）

(1) 研究実施内容及び成果

1． 研究開発のねらい

1.A 概要

マルチコアプロセッサは性能の改善や消費電力の削減、開発コストの削減を目的に、ますます組込みシステムで採用されている。マルチコアプロセッサ上で複数のオペレーティングシステムを動作させることで、高機能な情報アプリケーションを開発する際にソフトウェアの再利用が可能となる。複数のOSが動作している環境では、製品上で 2 種類のOSを同時に使用することが出来る。複数OS環境を実現するためには、組込みシステムに特化した仮想化層が必要不可欠である[1]。その理由は、ほとんどの組込みシステム向けプロセッサで提供されている保護機能は 2 段階のみであり、仮想化のためのハードウェアを持たないためである。

従来の手法では、それぞれの OS を隔離して信頼性を向上させるために、カーネルをユーザーレベルで動作させていた。しかしこの手法は、組込みシステムでは稀であるハードウェアによる適切な仮想化のサポートが提供されていない限り、ゲスト OS の多大な修正が必要であった。このため、現状の仮想化手法は組込みシステム産業には好まれていなかった。

Armand と Gien は、以下に述べるように、組込みシステム向けの仮想化層に望まれているいくつかの最適な条件を提示している。

i. 既存の OS やその上で動くアプリケーションを仮想化環境に導入するにあたって、OS の修正や性能低下を最小限に抑えること。
ii. OS のバージョンの変更を容易に行うことが出来ること。特にこれは、Linux の頻繁な更新に対応するために重要となる。
iii. 現状の実行形態を修正せずにデバイスドライバーを再利用するべきである。
iv. 現存している従来のリアルタイム OS やアプリケーションをサポートすると同時に、それらの持つリアルタイム性を保証する必要がある。

我々の研究では SPUMONE と呼ぶマルチコアプロセッサ環境で動作する組込みシステム向けの仮想化層を構築することを目的としている。SPUMONE は SMP（対称型マルチコアプロセッサ）方式の環境を対象としており、それぞれのコアはローカルメモリを保持していると想定している。そのため、全てのコア間で容易にコードやデータの共有が可能であるが、同時に各コアはコアローカルメモリを利用することによってコードやデータを機密に保持することも可能である。SPUMONE はこのような性質を利用することによって、空間的な隔離を実現している。

SPUMONE では上記の要求を満たすと同時に、以下の 4 つの用件を満たすことを目標としている。

i. 仮想コアを動的に物理コアに割り当てることによって、リアルタイム制約と性能・消費電力に関するトレードオフのバランスを取る。
ii. シングルコアやマルチコアプロセッサ上において、リアルタイム性を損なうことなく割り込み遅延を小さくする。
iii. ゲスト OS をユーザーモードで動作させることによって、RTOS を GPOS から隔離する。
iv. カーネルへの完全性の侵害を検知し、カーネルを独立して再起動させることで回復を図る。

[1] 我々が研究している仮想化層は様々な組込みシステムに利用可能だが、現状では主に情報機器に注目して研究を行っている。

中島チームでは、以上の要求を満足する組込みシステム向け D-Visor としての SPUMONE、D-System Monitor を時間的、空間的に保護するための SPUMONE 上のメカニズムであるアイソレーションシステム、D-System Monitor の1つとして OS カーネルのインテグリティを汎用的に保証するための仕組みであるモニタリングサービスの3つのサブシステムに関する開発をおこなった。

1. B 仮想化の必要性

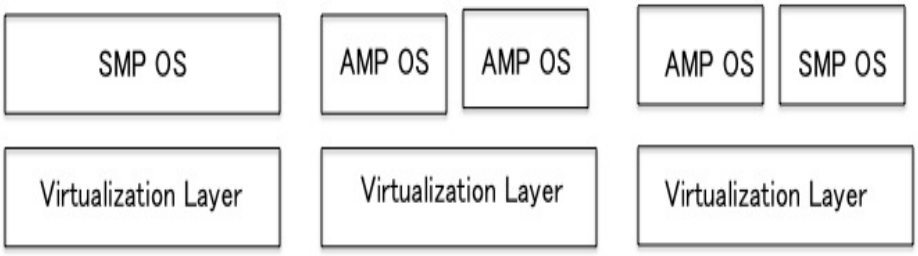

図 4.1.1: 複数の OS の共存

この節では、組込みシステムに仮想化層を用いることによって得られる 2 つの利点を紹介する。図 4.1.1 は仮想化層の 1 つめの利点を示している。モーターや無線通信の制御を初め、科学的制御など、組込みシステムでは一般的に何らかの処理制御を行っている。ソフトウェアを利用することでより柔軟な制御が可能になるため、最近の先端組込みシステムには柔軟な制御を実装するためのマイクロプロセッサが搭載されている。一方で、最近の組込みシステムは、人間の意思決定を補助するために様々な情報の処理をおこなう必要がある。そのため、最近の組込みシステムは制御機能と情報処理機能の両方を持ち合わせている。従来の組込みシステムでは、それぞれの処理に対して専用のプロセッサが割り当てられている。マルチコアプロセッサの利用はこのような複数の処理を単一のプロセッサ上で扱うことを可能にする。図 4.1.1 は、制御機能と情報処理機能がシングルプロセッサ上の仮想化層の上で動作している様子を示している。この手法では、従来よりも少数のプロセッサで実現することが可能であり、組込みシステムのコストの削減にもつながる。

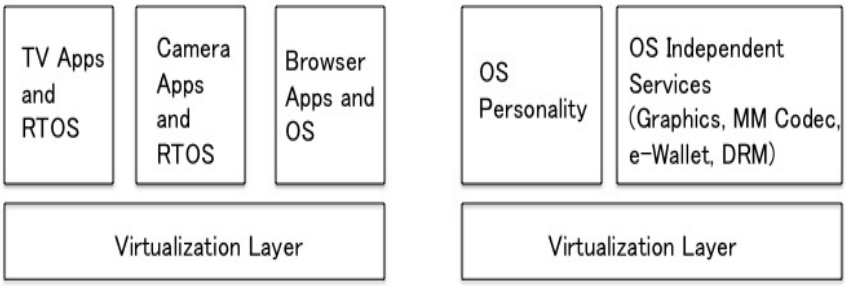

図 4.1.2: 既存のソフトウェアの再利用

図 4.1.2 では、仮想化層の 2 つ目の利点である既存のソフトウェアの再利用性を示している。左図では、3 つの独立した既存のソフトウェアが単一のシステムの上で動作している。仮想化層はソフトウェアを最小限の修正で組込むことを可能にする。左図では， Symbian から Android のように、OS 固有のユーザーインターフェースが変わったとしても、OS 非依存のサービスは引き続き使い続けることが出来る．様々な商業上の理由から、OS の種類を変更する必要が出てくる。もし組込みシステムの開発企業による追加のソフトウェアが OS 非依存のサービスとして開発されていれば、ソフトウェアを新しい OS に適応する必要は無くなる。

また、組込みシステムに仮想化層を採用する利点が他にもある。例えば、所有権のあるデバイ

スドライバーを、規約違反を犯さずに GPL コードの中に埋め込むことが出来る。これにより、組込みシステムに Linux を使用する際に発生する様々な商業的な問題を解決することが出来る。

2. 研究開発実施方法

2.A 組込みシステム向け D-Visor SPUMONE

1) ユーザーレベルゲスト OS とカーネルレベルゲスト OS

複数の機能を構成するために、複数の OS を単一のプロセッサ上で動かすには従来からいくつかの方法があった。 マイクロカーネルと仮想マシンモニタはゲスト OS のカーネルをユーザーレベルで実行する。マイクロカーネルの場合は、カーネルによるトラップや割り込みなど、様々な特権命令のコードを置き換えて仮想化する必要があった。また、カーネルはユーザーレベルのタスクとして実行されるので、アプリケーションのタスクはプロセス間通信を利用してカーネルと通信を行う必要がある。そのため，OS に膨大な量の修正が必要であった。

仮想マシンモニタは複数の OS を実行するために使われるもう一つの方法である。もしプロセッサがハードウェア的に仮想化のサポートを提供する場合は、全ての仮想化されるべき命令は、仮想マシンモニタにトラップを発生させる。これにより、どんな OS でも修正を行うことなく使用することが可能になる。しかし，仮想化のサポートがハードウェアで提供されていない場合は、いくつかの命令は仮想化するためにコードの一部を置き換える必要が出てくる。

組込みシステムで用いられるほとんどのプロセッサは 2 種類のみの保護レベルを提供しており、MMU は通常では特権レベルで使用することは出来ない。そのため、OS のカーネルが特権レベルに割り当てられている場合は、それらを隔離するのは困難である。一方で、OS のカーネルがユーザーレベルに配置されている場合は、カーネルに膨大な修正が必要になる。ほとんどの組込みシステム開発業者では、OS のコードを大量に修正する必要がある方法は避けるため、カーネルを特権レベルに配置するのは絶望的である。また、仮想化がソフトウェア的に実装されている場合、MMU の仮想化には膨大なオーバーヘッドが伴う。そのため、開発コストの削減やカーネルの信頼性の向上、また、マルチコアプロセッサの恩恵を活用するためには、他の選択肢が必要とされている。

ゲスト OS をユーザーレベルで実装する場合は、以下の 3 つの問題が深刻であると考えられる。

i. ユーザーレベルで動作する OS の実装には、OS のカーネルに大量の修正が必要となる。
ii. 割り込み禁止命令は、命令自体を置き換えることが出来ない限り、エミュレートの際に非常に負荷が高くなる。
iii. デバイスへのアクセス命令は、命令自体を置き換えることが出来ない限り、エミュレートの際に非常に負荷が高くなる。

典型的な RTOS では、カーネルとアプリケーションのコードはともに同じアドレス空間上で実行される。組込みシステムは全ての新製品に対して大幅に機能が追加されている。開発費用を削減するため、古いバージョンのアプリケーションのコードはアドホックな方法で再利用および拡張を許すべきである。ハードウェア資源の制約は開発コストを削減するために常に最も重要となる問題である。そのため、アプリケーションのコードは時折とてもアドホックなプログラミング手法を取ることがある。例えば、RTOS の上で動作するアプリケーションのコードは、割り込みの許可や禁止を行う命令など、通常たくさんの特権命令を含んでいる。また、デバイスドライバーもアプリケーションのコードの中に組込まれているかもしれない。このように、たとえアプリケーションのコードが入手可能であっても、それらに大量の修正を加えることなくユーザーレベルで動作するように移植するのは非常に難しい。このため、前節で述べられた要求を満たしつつ、アプリケーションのコードや RTOS をユーザーレベルで実行するのは困難である。そのため、プロセッサがハードウェアによる仮想化のサポートを提供しない場合、RTOS の実行は非常に難しくなる。たとえハードウェアによる適切な仮想化のサポートがあった場合でも、RTOS やアプリケーションの性能は大幅に低下すると考えられる。今回の手法では、OS のカーネルと仮想化層を同じ特権レベルで動作させることにした。この選択により OS のカーネルの修正は最小限になり、仮想化層の導入による性能の低下も防ぐことが出来る。しかしながら、次のようなとても深

刻な問題が 2 つ発生する。

i. 割り込み禁止命令は RTOS の割り込み遅延に深刻な影響を与える。
ii. OS のカーネル間を空間的に隔離する方法が無い。

上述の通り、RTOS やそのアプリケーションにとって、割り込み禁止命令を置き換えることは非常に困難であるため、1 つ目の問題は深刻である。また、OS のカーネルを仮想アドレス空間上で動作させるには、カーネルを大量に修正しなくてはならないので、2 つ目の問題も非常に大きな障害となる。このような問題を克服するため、SPUMONE では以下に述べる新しい仮想化層の設計方式の提案と、3)で述べるセキュアページャを提案し、D-Visor として利用可能なものとして実装をおこなった。

2) SPUMONE: 組込みシステム向けマルチコアプロセッサ用仮想化層

2-1)概要

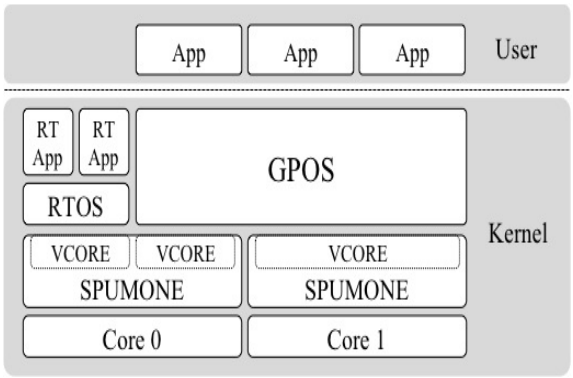

図 4.2.1: SPUMONE の基本構成

SPUMONE (Software Processing Unit、 Multiplexing ONE into two or more) は単一の物理 CPU コアを複数の仮想コアに複製するための小規模なソフトウェア層である（図 4.2.1）。SPUMONE が現状で対象としているプロセッサは SH4a である。SH4a は MIPS の構造に非常に似ており、日本の様々な組込みシステムの製品で採用されている。また、現状では標準の Linux と様々な RTOS が SH4a をサポートしている。現在のバージョンの SPUMONE は SH4a チップのシングルコアおよびマルチコア環境で動作している。現在、SMP Linux、TOPPERS JSP、そして L4 がゲスト OS として SPUMONE の上で動作している。

SPUMONE の基本概念は仮想コアにある。メモリやデバイスなど、その他のハードウェアの仮想化は付属機能である。様々なトレードオフを考慮し、それぞれの組込みシステムに最適な仮想化手法を選択する。前の章で述べた通り、SPUMONE では典型的なマイクロカーネルや仮想マシンモニタとは異なり、SPUMONE 自身と OS のカーネルは特権レベルで実行される。SPUMONE が提供するインターフェースと、下層のプロセッサが提供するインターフェースとの違いは僅かであるため、OS のカーネルの修正は単純である。この手法は、準仮想化と呼ばれている。このことは、いくつかの特権命令は SPUMONE の API を呼び出すための関数に置き換える必要があることを意味しているが、コードの置き換えは非常に少量で済む。このように、非常に簡単に新しいゲスト OS を SPUMONE に適応させることができ、既に適応している OS を改変することも容易になる。

複数のオペレーティングシステムを空間的に隔離するために、必要であれば、VIRTUS のように下層のプロセッサが各 OS の使用する物理メモリを保護する機構を提供するものと SPUMONE は想定する。この手法は、深刻なオーバーヘッドを発生させずに SPUMONE 上のゲスト OS の信頼性を向上させる最適な方法かもしれない。本報告書では、ゲスト OS 間の隔離を実現するた

めに、マルチコアプロセッサの機能を利用した新たな手段を提案する。この手法では、プロセッサにゲスト OS の空間的な隔離をサポートするための付加機能が存在しないものと想定している。

従来の I/O デバイスを仮想化する方法では、ほとんどの組込みシステムでは許容出来ないほどの重大なオーバーヘッドを発生させるため、SPUMONE では I/O デバイスの仮想化を行わない。SPUMONE では、デバイスドライバーはカーネルレベルで実装されているため、デバイスが複数の OS 間で共有されていない限り、デバイスドライバーの修正を行う必要は無い。

2-2) 割り込み/トラップの受け渡し

割り込みの仮想化はSPUMONEの重要な特徴である。割り込みは各ゲストOSに伝達される前に、SPUMONEに渡される。SPUMONEが割り込みを受け取った際は、割り込み配送テーブルを参照し、どのOSに伝達するべきかを判断する。目的地となる仮想プロセッサは、OSのカーネルがビルドされる際に、割り込みソース毎に静的に定義される。トラップも同様にまずSPUMONEに運ばれ、実行中のOSに直接転送される。

SPUMONEが割り込みを受け取るために、OSのカーネルのエントリーポイントを、SPUMONEのベクターテーブルに修正した。各OSのベクターテーブルを登録するために、各OSのエントリーポイントは仮想命令を経由してSPUMONEに通知される。割り込みは最初にSPUMONEの割り込みハンドラによって実行され、目的地となる仮想コアが決まり、関連するスケジューラが呼び出される。割り込みがOSの切り替えを引き起こした場合、現状のOSの全てのレジスタはスタックに退避される。次に、以前に動作していたOSにおいて、スタック内に保存されているレジスタが復帰される。最後に、目的地となるOSのエントリーポイントに実行が切り替わる。プロセッサはまるで本当に割り込みが発生したかのように割り込みを初期化するので、OSのエントリーポイントのソースコードを変更する必要は無い。

マルチコアプロセッサ環境における割り込みの伝達は、基本的にはシングルコアプロセッサ版と同じである。SPUMONEの各インスタンスは割り込みをそれらの目的地に伝達する。マルチコアプロセッサ環境では、仮想コアは物理コア間を移動しているかもしれない。割り込みを異なるコアの上で動作している仮想コアに伝達するために、割り込みと物理コアの割り当ては仮想コアの移動に応じて切り替えられる。

2-3) 仮想コアのスケジューリング

複数のOSが物理コアを多重化することにより動作している。OSの実行状態は仮想コアと呼ばれるデータ構造で管理されている。仮想コアの実行を切り替える場合には、全てのレジスタは関連する仮想コアのレジスタテーブルに格納され,次に実行される仮想コアのテーブルから復帰される。この仕組みは典型的なOSのプロセスの実装に似ているが、仮想コアは特権レベルの管理レジスタを含め、プロセッサ全体の状態を保存する。

仮想コアのスケジューリングは、固定的な優先度に基づくプリエンティブなスケジューリング方式である。RTOSとGPOSが同一の物理コアを共有している場合には、RTOSのリアルタイム性を維持するために、GPOSよりもRTOSの仮想コアの方が高い優先度を持つことになる。このことは、RTOSの仮想コアが待機状態であり、リアルタイム性を要求するタスクが存在しない時に限り、GPOSが実行されることを意味する。プロセスのスケジューリングはOSに一任されているので、各OSのスケジューリング方式を変更する必要は無い。待機状態のRTOSは、割り込みを受け取った場合、その実行を中断する。たとえGPOSが割り込みを禁止している場合でも、RTOSへの割り込みが発生した際は、直ちにGPOSの実行を中断するべきである。

GPOSに割り当てられている複数の仮想コアが,共有コアの上で実行されるように移動された場合は、それらのコアはタイムシェアリング方式に基づいてスケジューリングされる。

2-4) コア間通信

物理コアの上で動作しているSPUMONEのインスタンス間の通信は、共有メモリエリアの利用とコア間割り込み（ICI）の機構によって実装されている。最初に、送信者はデータを特定のメモリエリアに格納する。次に受信者に割り込みを送信し、受信者が共有メモリからデータをコピーする。

2-5）OS のカーネルの修正

以下で述べるように、プロセッサ以外のハードウェア資源はSPUMONEによって多重化されていないため、各ゲストOSは他のゲストOSの存在を認識するように修正されている。このように、OSのカーネルを修正することによって,プロセッサ以外のハードウェア資源は排他的に各OSに割り当てられている。以下では、SPUMONE上で動作するために、どのようにOSのカーネルが修正されているのかを説明する。

割り込みベクターテーブル登録命令: ベクターテーブルのアドレスを登録する命令は、アドレスをSPUMONEの割り込みマネージャーに通知するように修正されている。一般的には、この命令はOSの初期化時に一度だけ呼び出される。

ブートストラップ: シングルコアのSPUMONEによってサポートされている特性に加え、マルチコア環境ではベクターデバイスの仮想初期化機能が提供されている。この機能は、他のコアに属している仮想コアに対して、プログラムカウンタの初期化を行う役割を担っている。

物理メモリ: 各ゲストOSには固定サイズの物理メモリ空間が割り当てられる。各OSに割り当てられる物理アドレスは、設定ファイルやOSのソースコードを改変することによって簡単に変更することが出来る。物理メモリを仮想化することは、仮想化層の大規模化や性能低下に繋がる。さらに、企業向けシステムの仮想化層とは異なり、組込みシステムがサポートする必要のあるゲストOSの数は固定的である。これらの理由から、今回は各ゲストOSに固定量の物理メモリを割り当てている。

待機命令: 実際のプロセッサの待機命令では、プロセッサが割り込みを受け取るまで動作を一時停止させる。仮想化環境においては、待機命令は他のOSに物理コアを明け渡す時に使用される。今回はSPUMONEのAPIと置き換えることで、待機命令の実行を防いでいる。一般的に、待機命令はカーネル内において、平等に探しやすい特定の領域に配置されている。

周辺機器: 周辺機器はSPUMONEによって各OSに排他的に割り当てられる。これは、同じ外部機器を共有しないように各OSの設定を修正することで実現される。我々はほとんどのデバイスは各OSに排他的に割り当てることが出来ると想定している。ほとんどの組込みシステムでは、ゲストOSには別々の機能が割り当てられており、異なる物理デバイスを使用するので、この想定は妥当である。多くの組込みシステムはRTOSとGPOSで構成されており、RTOSは無線伝送機やデジタル信号処理プロセッサなどの特殊用途のデバイス管理を目的に使用され、GPOSは様々なヒューマンインタラクションを実現するデバイスや記憶装置などを始めとする一般的なデバイスを扱うために使用される。

しかしながら、一部のデバイスは各OS間で共有する必要があるため、排他的に割り当てることが出来ない。例として、我々が使用したプロセッサは割り込みコントローラーを1つしか提供しない。一般的には、ゲストOSは起動中に一部のレジスタを初期化するする必要がある。SPUMONE上で動作している場合には、2番目以降に起動するゲストOSは、以前に起動したOSの設定を初期化や上書きしないように注意する必要がある。例えば、今回はTOPPERSで保存された設定をLinuxが書き換えないように、Linuxの起動コードを修正している。

3）動的な複数コアの管理

マルチコアプロセッサ向けの SPUMONE は、Multikernel の手法に類似した分散型の設計となっている。独立した SPUMONE インスタンスはそれぞれの物理コアに割り当てられている。この設計は、複数の物理コアの同期の際に発生する予期せぬオーバーヘッドを防ぐことを可能とする。さらに、基本的なロック機構はシングルコアでもマルチコアでも同一であるため、SPUMONEの設計を単純にすることを可能とする。

SPUMONE は、１つの物理コア上に複数の仮想コアを多重化することが可能である。リアルタイム制約、性能、消費電力との間に存在するトレードオフを解決するために、物理コアと仮想コア間の関係を動的に変更することが出来る。また、SPUMONE は、状況に応じて仮想コアを他の物理コアに移動することが可能である。この手法は、以下で説明するようないくつかの利点をもたらす。

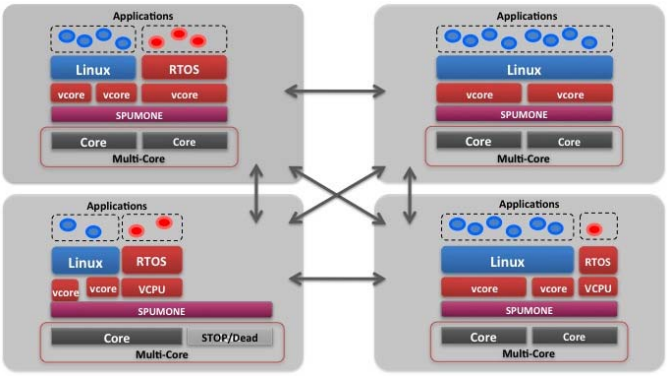

図 4-2-2: 動的マルチコアプロセッサ管理

1 つ目の利点は消費電力を減らすために仮想コアと物理コア間のマッピングを変更することである。図 4-2-2 で示すように、図内のプロセッサは 2 つの物理コアを提供することを仮定している。Linux は 2 つの仮想コアを使い、RTOS は一つの仮想コアを使う。RTOS の CPU 使用率が高い時は、Linux の 2 つの仮想コアは 1 つの物理コアにマップされる（左上)。RTOS が停止したとき、Linux の各仮想コアは異なる物理コアを使う(右上)。また、RTOS の CPU 使用率が低い時、一つの物理コアが Linux の仮想コアによって使われ、別の物理コアが Linux と RTOS によって共有される(右下)。最後に、消費電力削減が必要な時や物理コアの一つが停止した時、全ての仮想コアが同じ物理コアで動作する(左下)。このアプローチは、我々にリアルタイム制約、性能、消費電力のバランスをとるための積極的な方針を利用可能とする。

4) 性能とエンジニアリングコスト

テーブル4-2-1はSPUMONE上でToppersとLinuxを動作させる場合と、Linuxだけをネーティブに動作させる場合で、Linuxカーネルをビルドするのに必要な時間を示している。Toppersでは1ms毎にタイマー割り込みを受け取るだけで、他のタスクは存在しない。この結果は、SPUMONEとToppersがLinuxの性能に1.4%のオーバーヘッドを生じていることを示している。このオーバーヘッドはToppersによって消費されるサイクルも含んでいるため、この結果は、SPUMONEの仮想化によるシステムのスループットへのオーバーヘッドが十分に小さいことを示している。

テーブル4-2-1 Linuxカーネルビルドタイム

Configuration	Time	Overhead
Linux Only	68m5.9s	-
Linux and TOPPERS	69m3.1s	1.4%

我々は、各OSカーネルの修正したコード行数(LoC)を比較することでSPUMONEを利用する際のLinuxの変更に必要なエンジニアリングコストを評価した。テーブル4-2-2は、元々のLinuxカーネルに加えられた、あるいは削除したLoCを示している。ここでは、カーネル内通信のためのデバイスドライバは含んでいない、なぜなら、ライン数はそれらがサポートするプロトコルがどれくらいあるのか、それらの複雑さによって異なってくるからである。また、この結果はLinuxとRTOSの間、あるいはLinuxとサーバプロセス間の通信に提供される実用的なデバイスドライバのLoCも含んでいない。なぜなら，それはいくつのプロトコルがサポートされているのか、それらの実装がどれほど複雑かに依存するからである。

テーブルはまた，マルチOS環境をサポートするための以前のアプローチであるRTLinux、RTAI、OK Linuxのために修正されたLoCを示している。我々はSH4aプロセッサアーキテクチャ用のRTLinux、RTAI、OK Linuxを見つけることが出来なかったので、Intelアーキテクチャ向けに開発されたもので評価をおこなった。OK LinuxはL4マイクロカーネル上で動作する仮想化されたLinuxカーネルである。OK Linuxのために我々はアーキテクチャ依存ディレクトリにarch/l4と

include/asm-l4を加えたコードだけを数えた。その結果は、我々のアプローチがLinuxカーネルに極めて少量の修正を必要とすることを明確に示している。つまり、ゲストOSの修正量を減らすことと、SPUMONE実現のための要求を満たすためにプロセッサを仮想化するSPUMONEの戦略が効果的であることを示している。

テーブル4-2-2.　*.c, *.S, *.h, Makefilesの変更量

OS(Linux version)	Added LoC	Removed Loc
Linux/SPUMONE(2.6.24.3)	161	8
RTLinux 3.2(2.6.9)	2798	1131
RTAI 3.6.2 (2.6.19)	5920	163
OK Linux (2.6.24)	28149	-

2.B. アイソレーションシステム

アイソレーションシステムは、D-Visor 上で動作する D-System Monitor を時間的、空間的にアイソレーションするためのシステムである。本節では、SPUMONE 上でどのように時間的、空間的アイソレーションを実現したかに関して説明する。

1) 時間的アイソレーション

2.A 節で説明したように、もし各ゲスト OS やそのアプリケーションコードがリアルタイム性を重視するなら、割り込み停止命令を使うことは深刻な問題となる。この節では、以上の問題を克服するための新規技術を提案し、我々が提案した技術の有効性を示す。

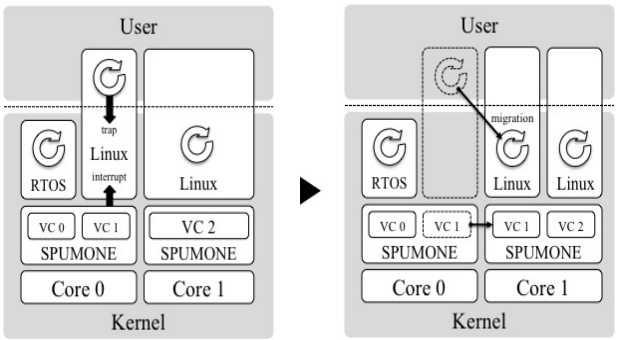

図 4-2-3: 仮想コアマイグレーション

提案手法は仮想コアマイグレーションを利用することで、この問題を解決する。TopperとLinuxをSPUMONE上に移植した時、Linuxカーネルのいくつかのパスにおいて最も高い割り込み優先度レベル(IPL)を得ることが発見された(例えばブートストラップやアイドルスレッド等)。これは、悪い作法でプログラムされているいくつかのデバイスドライバやカーネルモジュールがToppersの動作と干渉していることを示している。我々は、SPUMONEを、カーネルにトラップするか、割り込みがトリガーになる時、図4-2-3で示すように別の物理コアに仮想コアを移動するように修正した(それはToppersと物理コアを共有するLinuxに割り当てられる)。この方法により、Linuxのユーザレベルのコードだけが共有された物理コアで同時に実行されるようになる。それは、優先レベルを決して変更しないので、Toppersは割り込み可能/不可能命令を置き換えることなく、すぐにLinuxをプリエンプトすることが可能となる。

次に、我々はLinuxの負荷がToppersにおける周期タスクへのディスパッチレイテンシに与える影響を測定した。周期タスクは1ms毎に実行される。測定の間，周期タスクは，100,000回サンプ

リングされる。ディスパッチレイテンシは周期タスクがその実行を始めるまで割り込みをトリガーにして費やされる時間である。Toppers上の周期タスクは他のアクティビティに邪魔されないため、理想的にはディスパッチレイテンシは0となる。

図4-2-4と図4-2-5はNFS共有ファイルシステムへの連続したwrite呼び出しをおこなっている時に、仮想コアのマイグレーション技術の有無でのディスパッチレイテンシの分布を比較している。仮想コアマイグレーション技術なしの測定では、最大レイテンシは96μsを示しているが、仮想コアマイグレーション技術を有効にした場合では、最大レイテンシは39μsに減少する。

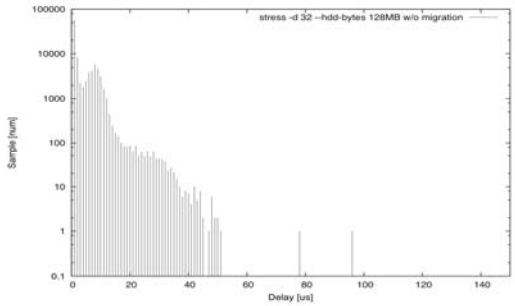

図4-2-4: ディスパッチレイテンシ(仮想コアマイグレーションを利用しない時にNFSストレスを実行)

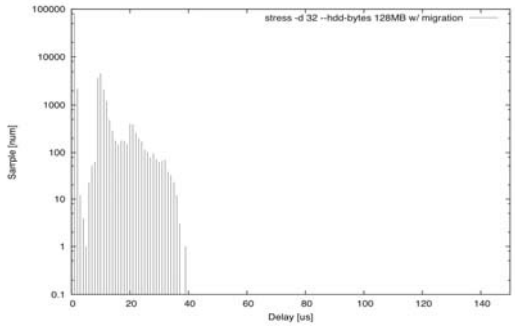

図 4-2-5: ディスパッチレイテンシ（仮想コアマイグレーションを利用した時に NFS ストレスを実行)

次に、Toppersのプロセッサ使用率がLinuxに与える影響を測定した。初めに、４つの専用コア上で動作するLinuxでDhrystoneベンチマークのスコアを測定した(図4-2-6で4コアとして示される)。また、Linuxは３つの専用コア上で動作し、１つのコアは様々なワークロード（図のxx%）のToppersと共有する際のスコアを測定した。最後に、３つの専用コアをLinuxが利用した際のスコアを測定した(図内で3コアとして示される)。Toppers上のリアルタイムタスクは、10msのサイクルで実行され、図で示される割合は周期タスクの実行時間の割合を示している(30%はリアルタイムタスクが3msで連続的に実行されることを意味している。)。

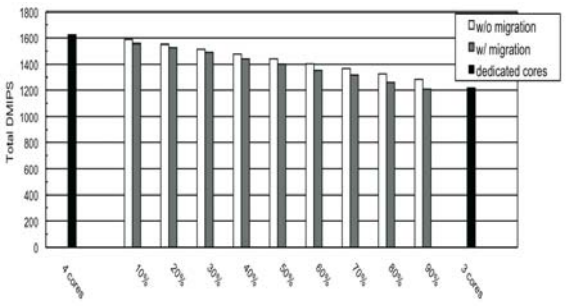

図4-2-6:. TOPPERSの負荷がLinuxのDMIPSスコアに与える影響

図4-2-6はDhrystoneベンチマークのトータルスコアを示している。左端のバーは３つの物理コアを利用してSPUMONE上で実行されるLinuxで実行されるベンチマークの評価のスコアを示している。周期タスクのワークロードが増大するに従い、Dhrystoneのスコアは低下する。90%の負荷で、結果は３つの専用コア構成の場合のスコアに近づくことを示している。その結果は、仮想コアマイグレーション技術のオーバーヘッドが、本評価で利用したベンチマークでは、重大でないことを示している。

2) 空間的アイソレーション

同じ特権アドレス空間で動作するGPOSからRTOSを孤立させるために、直感的な解決策はMMUベースのメモリ孤立性を利用することである。しかしながら、このアプローチは悪意のあるOSカーネルからページテーブルとページフォルトハンドラを保護するメカニズムを提供することが出来ず、D-Visorの要件を満足しないことになる。我々はその代わりにコアローカルメモリを利用する技術を提案することで、SPUMONEをD-Visorとして利用することを可能とする。提案アプローチは、将来マルチコアプロセッサがコアローカルメモリをサポートすることが一般的となると思われるので、実用的なアプローチであると考えられる。そして、我々は仮想化層を実装するために余計なハードウェアサポートを前提としない。コアローカルメモリはマルチコアプロセッサにおける各コアからのみアクセス可能なメモリである。コアローカルメモリへのアクセスレイテンシは共有メモリへのアクセスよりも早いため、コアローカルメモリの元々の目的は並列アプリケーションの性能を改良するために、各スレッドのメモリアクセス局所性を有効に使うためのものである。コアローカルメモリは他のコアからは見えず、アクセスすることは出来ないため、この性質を有効利用してD-System Monitorの保護を実現する。

2-1) コアローカルメモリ

各コアが図4-2-7で示されるように独立なコアローカルメモリを持つ、デュアルコアプロセッサ上で２つのOSカーネルが動作することを仮定する。もし次の仮定が満たされるなら、ターゲットのOSは他のOSカーネルから保護が可能となる。

i. ターゲットとなるOSのサイズはコアローカルメモリにフィットするほど十分に小さい。
ii. 各コアは他のコアのコアローカルメモリの内容を除去するリセットを制限する。
iii. ターゲットOSのブートイメージが変更されることがないことを保証する。セキュアブートローダがカーネルイメージを正しく共有メモリにロードすることを保証する。
iv. 各コアはI/Oデバイスにアクセスすることを制限する. コアによって管理されるI/Oデバイスは他のコアからアクセスされないように制御する必要がある。

2-2) ハッシュベース整合性管理

前節で示される解決策の問題は、コアローカルメモリのサイズである。現在、それらは最大でも数百KBであり、現在のRTOSをロードするには小さすぎる。ローカルメモリのサイズを仮想的に拡張するために、我々はコアローカルメモリのサイズをMMUを利用して仮想的なサイズを拡張するハッシュベース整合性管理メカニズムを提案する。また、本提案方式は、共有メモリ内のプログラムが破壊されたことを検出可能とすることで、複雑なD-System Monitorの保護を可能とする。元のカーネルイメージは共有メインメモリに格納され、カーネルイメージのサブセットがそのOSを実行する物理コアが持つコアローカルメモリにコピーされる。ターゲットOSのカーネルイメージの一部がコアローカルメモリにロードされる時、この部分が攻撃により変更されていないかをハッシュ値を用いて毎回確認する。

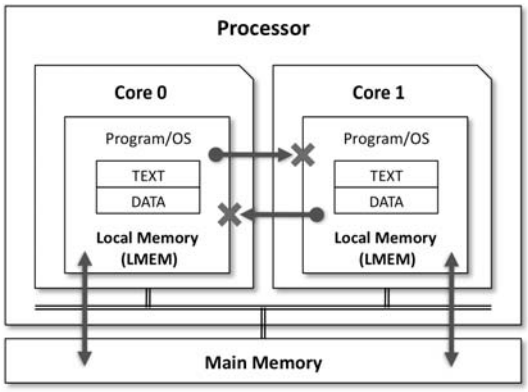

図 4-2-7: コアローカメモリ

ここでは､ハッシュベース整合性管理がどのように動作するのかを図4-2-8を用いて説明する。コアローカルメモリへのページアロケーションと暗号化ハッシュ値の計算はローカルメモリ（LMEM）マネージャによって管理される｡それは永続的にコアローカルメモリ内に格納させる。ここでは、他のOSカーネルから保護されるOSカーネルイメージを保護されるOS(pOS)と呼び、悪意のある動作によって汚染されるかもしれないOSカーネルは危険性のあるOS(vOS)と呼ぶ。pOSとvOSは異なるコアで動作する。

i. ブートローダはLMEMマネージャをコアローカルメモリにロードする。pOSとvOSのOSカーネルイメージはメインメモリに同時にロードされる｡LMEMマネジャーはpOSの各ページのハッシュ値を計算し、そしてまたコアローカルメモリに置かれるハッシュテーブルにそれを格納する｡LMEMマネージャはpOSのエントリポイントを含むメモリページをコアローカルメモリにロードする。そして、他のコアはvOSを実行することが可能となる。
ii. pOSのページは仮想アドレス空間にマッピングされ、仮想アドレス空間を管理するためのページテーブルはコアローカルメモリに置かれる。ページテーブルのサイズがコアローカルメモリのサイズより大きい場合、LMEMは共有メモリへ使用されていないページテーブルをスワップアウトすることが可能である。LMEMは、スワップされるページテーブルの整合性を維持するためにハッシュテーブルを管理する。LMEMはページテーブルが対応するページテーブルエントリを含まない時ページフォルト発生し、ページを共有メモリからコピーする。
iii. LMEMがページフォルトを処理する時、対応するページが共有メモリからコアローカルメモリにコピーされる。LMEMはページのハッシュ値を計算し、コアローカルメモリに格納されている、事前に計算されてあるハッシュ値と比較をおこなう。ハッシュ値のミスマッチは共有メモリ中のpOSのイメージが破損していることを意味する。もし，ミスマッチがないなら、ページフォルトは正しく完了し、pOSの実行を再開する。
iv. コアローカルメモリに利用可能なスペースがない時、LMEMは共有メモリにいくつかのページをスワップアウトする。LMEMはページが更新されているかどうかチェックし、もし更新されているならLMEMはそのページのハッシュ値を再計算し、ハッシュテーブルエントリを更新する｡そして､空いたメモリスペースは､他のページをロードするために使用される。

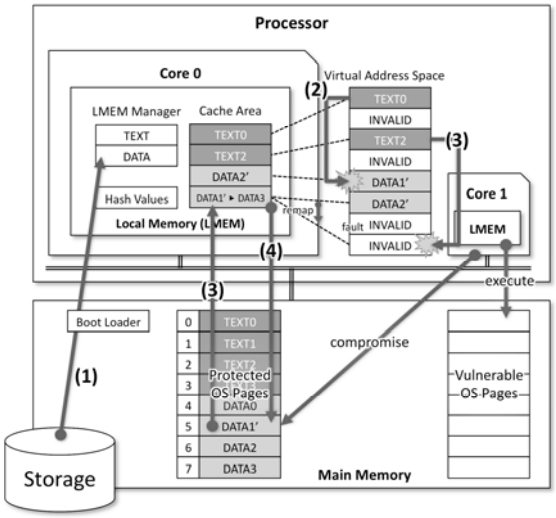

図 4-2-8: ハッシュベース整合性管理

このアプローチでは、共有メモリ中の pOS のイメージは vOS によって破損させられる可能性がある。我々の現在の方針では、安全なローダーによってダメージを受けていない新しいカーネルイメージを再ロードすることで pOS を再起動することによりダメージからの回復をおこなう。

2.C モニタリングサービス

空間的なアイソレーションだけでは、各ゲスト OS の安全性と信頼性を向上することは不可能である。悪意のある攻撃コードは Linux カーネルに挿入され、ウイルス検出器からそれらを削除するまでシステムは危険にさらされることになる。

SPUMONE においては、この問題を解決するための基本的なアプローチとしてモニタリングサービスを提供している。モニタリングサービスは D-System Monitor の 1 つであり、OS カーネル内のデータ構造体の整合性を定期的にチェックする。整合性は各データ構造体の制約として指定される。現状では、ソースコードから自動的にインバリアントを抽出することによりデータ構造に関する詳細を知らなくてもモニタリングすることを可能としている。もし、モニタリングサービスが制約の違反を検出すると、サービスは整合性をリカバリするための各データ構造体を定義されたリカバリ関数を呼び出す。修復手続きは完全にはシステムを修復しない可能性がある。つまり、整合性が回復出来ないガーベージデータがカーネル空間に残るか、修復手続きは失敗するかもしれない。このケースではゲスト OS の整合性は再起動することにより回復する。

我々のアプローチでは、もし Linux カーネルが攻撃されると、空間的アイソレーションを用いない場合、攻撃者は他の OS カーネルへ侵入可能となる。他のカーネルを攻撃するために、攻撃者は Linux カーネルのアドレススペースにコードを挿入する必要がある。最近の攻撃はカーネルルートキットを使用する傾向がある。カーネルルートキットは自らを検出が困難になるように潜ませようとするため、従来のセキュリティツールえはそれらを検出できない。例えばいくつかのルートキットはプロセス管理に使用されるカーネルデータ構造体を修正するかもしれないが、我々のアプローチを用いることで、モニタリングサービスがルートキットが修正しようとするデータ構造体の一貫性ををチェックし、ルートキットを顕在化することで、セキュリティツールにルートキットを検出させることを可能とする。現在では、モニタリングサービスはソースコードからデータ構造に関する仕様を自動的に抽出し、カーネル内のデータ構造に関して、整合性をチェックするために利用するインバリアントを学習により発見する。本研究では、以上を実現するためのツールも開発し、Toppers 上で動作するモニタリングサービスを自動生成するためのシステムを構築した。また、このシステムを利用することにより、Linux カーネル内のルートキットが検出可能であることを示した。

また、モニタリングサービスと Linux カーネルの間の同期メカニズムに関しても研究をおこなった。Linux カーネルとモニタリングサービスの間の共有データ構造体を排除するために、Linux カーネルを修正することは不可能なので、本アプローチではモニタリングサービスが Linux のデータ構造をアクセスする場合に楽観的な同期を用いる。

もちろん、モニタリングサービスは Linux カーネルから保護されるべきであり、モニタリングサービスを保護するためのメカニズムは様々なものが存在する。例えば、モニタリングサービスを実行するためにコプロセッサや特別なデバイスを使用することも可能である。組込みシステムを構築するためのマルチコアプロセッサは各 CPU コアにコアローカルメモリを搭載していることが多い。コアローカルメモリはそれを所有する CPU コアによってのみアクセスされる．本アプローチでは一つの CPU コアはモニタリングサービスの実行に専念することを仮定する。その結果、Linux カーネルはそのコアローカルメモリにアクセスすることができないので、モニタリングサービスを保護することが可能となる。モニタリングサービスのサイズが大きくなった場合は、前節で説明したセキュアページャを用いることにより保護が可能となる。

2-D ロギングサービス

現在の組込みシステムは単体として用いられるのではなく、ネットワークに接続され、大規模システムの一部として機能することが多い。ハードウェアの高性能化やアプリケーションへの機能要求の増大により、ソフトウェアのコード量は増大し、これらの要因によりシステム障害はより複雑化している。形式検証手法を利用した場合でも、数百万行ある全てのコードに対して検証をおこなうことは困難である。また、十分なテストをおこなったとしても、障害を発生する恐れやセキュリティホールなどが残る可能性が高い。こうした現状に対し、本研究では、実時間でアプリケーションの振る舞いの異常を検出するロギングサービスを提案した。ロギングサービスは出荷時にテストや検証により予知できなかった異常を運用時に検出し、即座に管理者に通知するシステムである(図 4-2-9)。

本システムは、稼働時にシステム内で発生する様々な異常を検出するためのシステムとして DEOS に貢献する。例えば、異常を検出してハードウエア仮想化層状に複数の OS を動作させることにより、監視用 OS を介して異常を発生した OS を再起動することが可能となる。以下に、 本提案で構築したロギングサービスの特徴を述べる。

(1) 低オーバーヘッド

組込みシステムではコスト制約もありハードウェア資源が十分でないケースが多い。 本研究では、運用時ロギングと解析のオーバーヘッドが 1.2%で異常検出の付加価値が提供できることを示した。低オーバーヘッドの実現のために、本研究ではシステムコールなどのイベントログではなく、プロセス毎の資源利用率のログを用いた。プロセスが呼び出すシステムコールのトレースを使用する場合は、プロセスが呼び出したシステムコールの全ての実行ログが必要であるが、資源利用率を用いる場合は、周期毎にプロセスの資源利用量を統計値として取得するコストのみがオーバヘッドとなるので、ログの取得・解析にかかるコストが小さい。また、検出精度を上げるために、プロセス毎の資源利用率を用いた。

(2) ブラックボックスアプローチ

組込みシステムでは、様々な製品やサービスを考慮しないといけないので、使用するプログラミング言語も一つではなく、様々な言語が使用されることを想定する必要がある。本提案では、特定の言語に依存しない資源情報を入力情報として用いた。資源使用率を考慮して異常を検出する際、CPU のみの資源では、例えば攻撃時に生じるメモリ量の変化やネットワークの使用率の変化が重要な場合にプロセスの振る舞いを解析することが困難となる。プロセスの動作を詳細にモデル化するためには、3 つの資源(CPU、メモリ、ネットワーク)を対象とした分析が必要であると考えた。また、資源利用率は確率的な状態遷移によりモデル化できると考え、HMM（Hidden Markov Model）を用いた。これらにより、アプリケーションには一切手を入れずに精度の高いモデルを作成することができた。

(3) オンラインでの検出

組込みシステムは、 B2B のサーバマシンと異なりコンシューマーが利用する機器がターゲットである。通常マシンの状況を熟知した管理者が存在し、異常発生時の問題解析や通知をおこなうことを前提とすることができない。そのため、組込み機器がネットワークに接続されている場合は，ネットワークを介して異常をオペレータが監視しているサーバ側のノードに通知するものとした。また、被害が拡大しないよう、異常が検知されたプロセスの資源利用量を制限することによりシステムの負荷が予測不可能に増大しないように制御することを可能とした。

(4) 未知の異常の検出

実際の製品の開発では、出荷前に想定できる範囲内でのテストをおこなうため、既知の問題は出荷前に対処する。そのため、運用時に発生するのは、想定外の攻撃やタイミングに関わる問題などである。こうした想定外の問題を効率よく検出するためにロギングサービスでは、正常状態を学習し、稼働時は、正常状態と異なる状態を検出した場合に、異常として報告する。これにより、運用時に未知の異常を検出することを可能とした。

以下に検証結果を示す。ロギング時のシステム全体にかかるオーバーヘッドの計測では、 ロギングシステムを動作しない状態、学習中、運用中時に、動作させたプログラムの実行に必要とした時間を計測し比較をおこなったところ、学習時のオーバーヘッドは 1.1%，運用時のオーバーヘッドは 1.2%と、非常に低い結果が得られた。

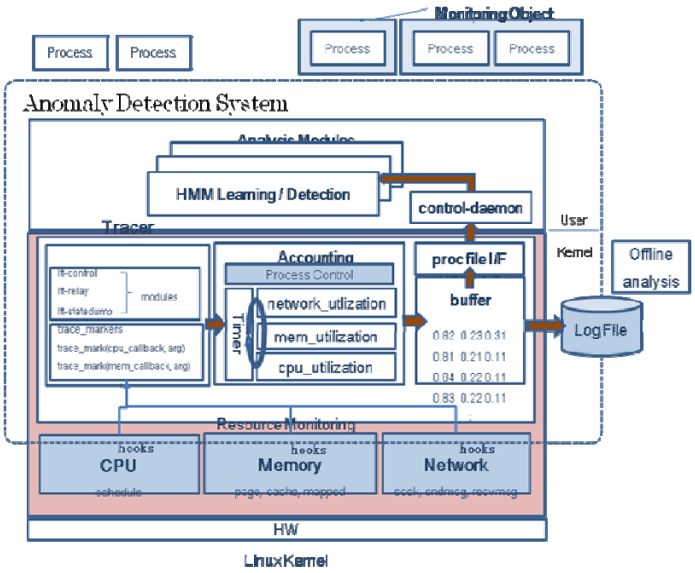

図 4-2-9 ロギングシステム

また、異常検出精度に関する検証では、サーバクライアントシステムを構築し、クライアントからサーバに対して正常なアクセスと攻撃が含まれたアクセスをおこない、正常に攻撃が検出されることを示した（図 4-2-10，図 4-2-11）。

比較対象としては、SQL インジェクションとバッファオーバーランの検出について、本手法（HMM）による方式、 閾値、 移動平均、増加率を用いた各手法を利用した場合の異常検出精度を比較した。結果より、HMM を利用した場合の検出精度は、誤認識はあるものの、検出精度が高いことがわかった。

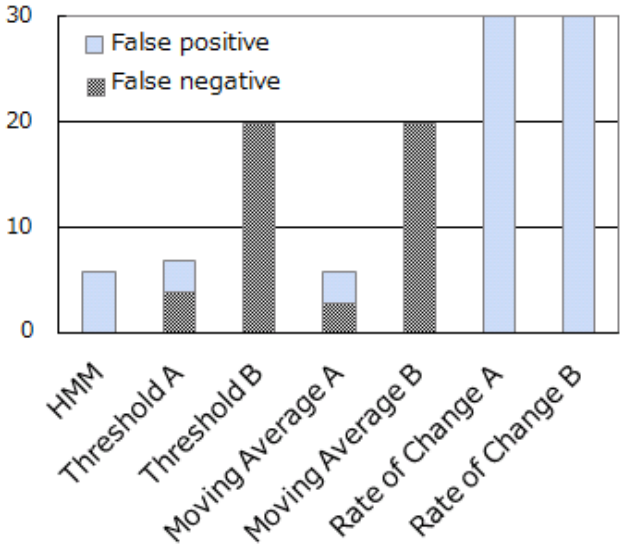

図4-2-10　SQLインジェクションの検出

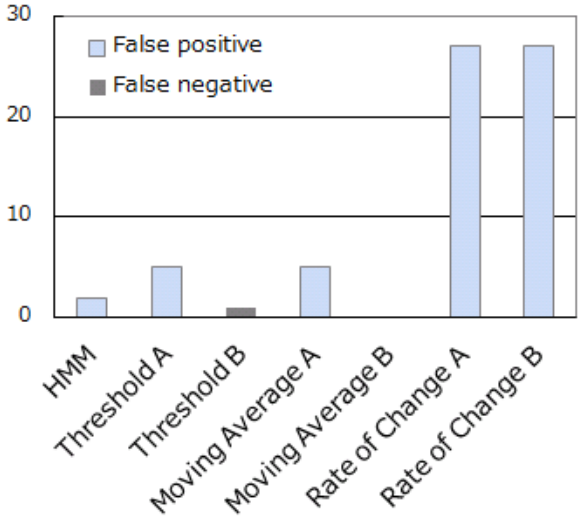

図4-2-11 バッファオーバーランの検出

2-D. VM サブコアチームへの貢献

中島チームの VM サブコアチームの最大の貢献は、OS カーネルのセキュリティの保護の重要性を主張し、D-RE に D-System Monitor を導入した点である。D-System Monitor の導入により、中島チームが開発した OS カーネルのインテグリティを保証するモニタリングサービスと河野チームが開発する様々なセキュリティソリューションを D-RE に統合することを可能とした。

また、LAAS レポートの D-Visor/D-System Monitor に関する報告の中で、仮想化層のデザイントレードオフに関する記述とケーススタディにおける SPUMONE の記述を担当した。

3. 成果の現状

早稲田大学チームが開発した SPUMONE は、SH4 のマルチコアボードである RP1 上で動作し、マルチコアプロセッサ上で仮想化層が稼働することを実証している。現状では、仮想コアマイグレーション機能を提供することにより、各 OS のワークロードに応じて動的に仮想コアの移動を実装している。また，SMP Linux 利用時に大きな問題となるロックホルダープリエンプションに関しても解決し、高い応答性とスループットを両立することを保証している。

また、アイソレーションシステムを利用することにより、SPUMONE が提供する高い性能と、ゲスト OS の

変更容易性を犠牲にすることなく、RTOS と Linux 間の時間アイソレーションの実現、D-System Monitor を Linux から保護するための機能が提供可能であることを実証した。
最後に、モニタリングサービスはゲスト OS のソースコードから自動的にインバリアントを生成するため、OS に依存せずに汎用的に利用することができる。特に、従来の同様のシステムと比べて、OS のバージョンが変更されるたびにルールを記述し直す必要がなくなる。また、RTOS 等の従来のインテグリティ管理システムが考慮していなかった OS に対しても適用が可能であるため広く組込みシステムにおける利用が可能となる。

4. 領域への貢献、研究開発の価値

早稲田大学グループによる領域への貢献として、主に以下の 2 点がある。

1） D-Visor と D-System Monitor の役割を明確にし、組込み機器のようなリソース制約があり、ハードウエアによる仮想化支援が存在しない場合でも、ゲスト OS の性能やリアルタイム性を低下させずに D-System Monitor を保護することが可能であることを示した。これにより、D-RE が組込みシステムでも利用可能であり、より広い分野で DEOS プロセス／アーキテクチャが利用可能であることを示した。

2） D-System Monitor の1つとしてインテグリティ管理をおこなうモニタリングサービスを提供した。モニタリングサービスはソースコードが存在すればどのような OS に対してもインテグリティ管理をおこなうことが可能である。そのため、Linux だけではなく、様々な RTOS に対してもインテグリティ管理をおこなうことが可能である。

早稲田大学グループによる研究開発自体の価値としては、以下の 2 点が重要である。

1) 組込みシステム向け D-Visor である SPUMONE を開発した。SPUMONE はマルチコアプロセッサ上で動作し、軽量で高性能の仮想化層である。 従来の組込みシステム向け仮想化層は、ゲスト OS の変更量が多く、バージョンアップが多い現在の Linux のような OS を使うためには不適切であった。しかし、SPUMONE は新規に提案するアーキテクチャを採用することにより、性能を低下させずに、ゲスト OS の変更量を最小限にして仮想化層を利用可能とすることが可能であることを示した。

2) SPUMONE は性能を低下させない代償として、D-System Monitor をゲスト OS から保護することが出来ず、信頼性を低下させてしまう。しかし、早稲田チームが開発したセキュアページャはゲスト OS の性能を低下させずに、D-System Monitor を保護することを可能とする。既存の方式では、ハードウエアによる仮想化支援がない場合に D-System Monitor の保護をおこなうことは大幅な性能低下を引き起こしていたので、本プロジェクトが提案する方式の新規性は高いと考えられる。

(2)研究成果の今後期待される効果

早稲田大学チームの研究成果は、今後，マルチコア化が進む組込みシステムにおいて、基盤システムの信頼性を向上するために極めて重要である。本プロジェクトでは、提案する仮想化層が組込みシステムで実際に利用されるマルチコアプロセッサ上で実現可能であることを示し、提案手法が十分に実用的であることを示した。
しかし、本提案は組込みシステムだけではなく、大規模なクラウドコンピューティング環境を実現するためにも非常に有効である. 従来のクラウドコンピューティング向けの仮想化層は複雑で巨大であるため、単純に物理コアを仮想化し、消費電力を低下させるために使用することは困難であった。しかし、SPUMONE を利用することにより物理コアのみを仮想化することにより、使用していないコアを簡単に ON/OFF することが可能となる。
また、セキュアページャーはクラウドコンピューティング環境におけるセキュリティを安価に強化する仕組みとしても有効である。特に、本プロジェクトで開発したモニタリングサービスは

様々な OS に対しても利用可能なため、Linux 以外のオープンソースとして提供される BSDOS 等の OS でも利用可能となる。
　以上より、本提案は DEOS アーキテクチャの利用範囲を広げるだけではなく、DEOS と独立して利用することも可能である。

4．2　組込みシステムに適したアイソレーション機能の研究（筑波大学　追川グループ）
(1)研究実施内容及び成果

1.研究開発のねらい

　筑波大学グループは、図 4-1-1 の DEOS プロセスにおける筑波大学グループ担当箇所に示すように、DEOS プロセスの障害対応サイクルにおける予兆検知、障害の原因究明および迅速対応、そして開発時の検証に寄与する研究開発を行った。筑波大学グループで主に研究開発した D-System Monitor ランタイム API は、他チーム、他グループによって開発された複数の D-System Monitor 機能に対して共通基盤となる実行環境を提供し、機能の統合および単一システム上での利用を可能にするものである。D-System Monitor 実行のための共通基盤を提供することで、機能開発者は異なる D-Visor での実現方法に捕らわれず、機能の開発に集中することができ、開発された機能は複数の D-Visor 上で使用可能になるため、迅速対応に寄与することができる。また、複数の D-System Monitor 機能が単一システム上で利用可能になることで、相乗効果から原因究明にも寄与しうる。筑波大グループではまた、東京大学前田チームで開発されたモデル検査器を用い、仮想化層検証技術の開発を行った。これは、DEOS プロセスの開発時における検証に寄与するものである。またこの仮想化層の検証技術の研究開発から、仮想化層においてハードウェアへの操作が正しく行われることを検証により保証することにより、ハードウェアへの操作が正しく行われなくなった場合の障害の予兆検知として利用可能なことを明らかにした。

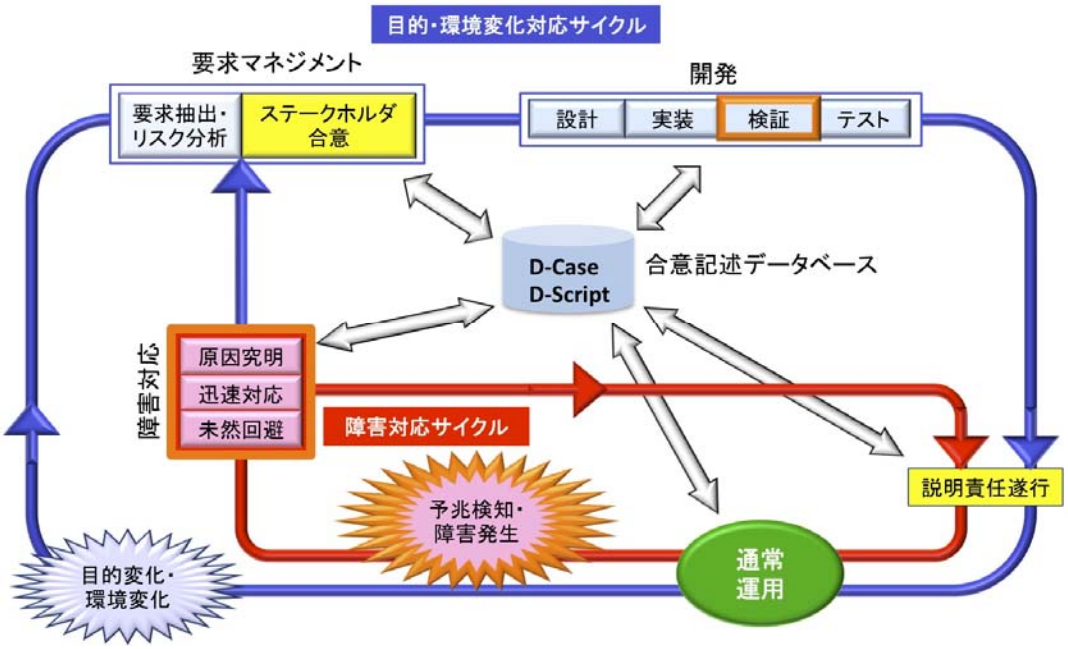

図 4-1-1　DEOS プロセスにおける筑波大学グループ担当箇所

D-System Monitor および D-Visor は、図 4-1-2 に示す通り、DEOS Runtime Environment (D-RE) の構成要素となっている。D-Visor は複数 OS の実行環境を提供する。D-Visor のアイソレーション機能により OS と D-System Monitor、D-Box の実行環境は切り離され、D-System Monitor による OS の外部からの監視を可能にする。

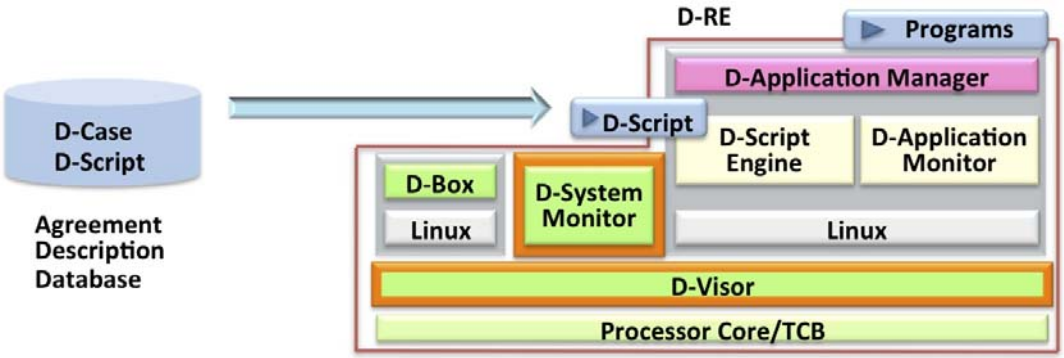

図 4-1-2 DEOS Runtime Environment (D-RE) における筑波大学グループ担当箇所

さらに筑波大学グループは、VM サブコアチームが発足した平成 21 および 22 年度にリーダー、平成 23 年度はサブリーダーとして研究開発を推進した。VM サブコアチームにおいて、White Paper V.2, LAAS レポートを取りまとめ、D-System Monitor ランタイム API を策定するための研究開発を行った。また、VM サブコアチームの成果を統合するデモシステムの開発を行った。統合デモにより、VM サブコアチームの成果が DEOS プロセスに対応し動いていることを示すことが出来る。

次節では、D-System Monitor ランタイム API、仮想化層検証技術、VM サブコアチームにおける研究開発に関して説明する。

2. 研究開発実施方法

2. 1 D-System Monitor 機能統合のためのランタイム API

D-System Monitor は、主にシステムのセキュリティを高めることを目的として、監視対象となる OS を OS 外部から監視する機能を提供するものである。DEOS Runtime Environment (D-RE)の構成するコンポーネントの一つであり、D-Visor 上で OS とは別に動作する。OS カーネル内部に悪意を持ったソフトウェアであるマルウェアが侵入すると OS が乗っ取られてしまう。乗っ取られた OS は自身ではマルウェアを検知することができないため、OS 内部からの監視は意味をなさない。しかし、OS 外部からカーネルの内部構造や振る舞い、入出力データを検査することで、マルウェアの存在を暴くことは可能である。そのため、D-System Monitor は D-Visor 上で OS とは別に動作し、OS を外部から監視する。

D-System Monitor は異なる脅威に対応するために、異なる検査を行う複数の実装が出てくることになる。D-System Monitor を使用するシステムは、そのシステムが立ち向かう必要のある脅威に対応した検査機能の実装を選択し、使用できることが求められる。従って、複数の検査機能が同時に使用可能でなければならない。さらに、システムの要求により、使用する D-Visor も変わってくるが、D-Visor ごとに D-System Monitor を実装し直すことは避けなければならない。そこで、筑波大グループでは D-Visor からは独立し、目的の異なる複数の検査機能を同時に動作可能にする D-System Monitor の実行環境を定義する API として D-System Monitor ランタイム API を策定した。D-System Monitor の検査機能は、この API を用いて監視対象 OS を検査するようにすることで、1 つの検査機能は複数の D-Visor で利用可能になり、また複数の検査機能を同時に使用可能になる。

図 4-2-1 に D-System Monitor ランタイム API を用いた D-System Monitor からなるシステムの構成を示す。監視機能は D-System Monitor ランタイム API を用いて実現され、ランタイム API を通して監視対象 OS の内部構造体や仮想デバイスの監視を行う。ランタイム API は、D-Visor の機能を用いて、その API が提供する機能を実現する。D-System Monitor ランタイム API を用いることで、D-System Monitor は複数の監視機能を容易に提供することができる。また、新たな攻撃に対応するために開発した監視機能を追加することも可能であるし、不要になった監視機能の削除も可能である。図 4-2-1 では、監視機能はそれぞれ監視対象 OS の異なる部分を監視しているかのように描かれているが、必ずしもその必要はなく、同じ部分を異なった方法で監視することも可能であり、また 1 つの監視機能が複数の監視対象 OS の内部構造体や仮想デバイスを監視することも可能である。

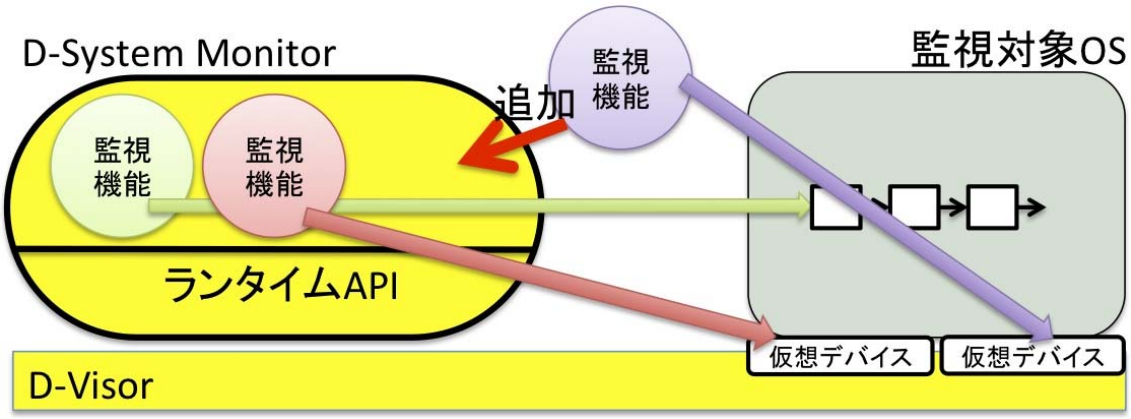

図 4-2-1 D-System Monitor ランタイム API

D-System Monitor ランタイム API を使用することのメリットとして、D-System Monitor における検査機能の迅速な開発と、複数の検査機能を用いることによる相乗効果による原因究明があげられる。検査機能の迅速な開発は、常に進化する攻撃方法に対応するために重要である。検査機能の開発者は攻撃方法やマルウェアの動作に精通していても D-Visor の内部構造に精通していることは考えにくく、その実現方法を調査するには時間がかかる。D-System Monitor ランタイム API を検査機能から分離して提供することにより、検査機能の開発者はマルウェアの発見する機能の実現に集中することができ、また開発した検査機能は全ての D-Visor でそのまま使用可能になる。従って、D-System Monitor における検査機能の迅速な開発が可能になる。

D-System Monitor ランタイム API を使用するもう一つのメリットは、複数の検査機能を用いることによる相乗効果による原因究明である。検査機能はマルウェアを発見するため、ある特定の方法を適用するために開発される。例えばキーロガーであれば、キー入力を保存し攻撃者に対しその内容を送る必要があるため、大量のキー入力に対してはディスクやネットワークへの出力が発生することが考えられるため、人工的な入力を与えそのような振る舞いを検出する方法が考案されている。一方、モニタリングシステムのように、汎用的にマルウェアの疑いがあるプロセスとして自身の存在を隠蔽するようなプロセスを検出する方法もある。振る舞いの検査だけではマルウェアの存在に確証を得ることが出来なくとも、モニタリングシステムと併用することで、振る舞いの検査による人工的な入力に対し、活発に動作する自身を隠蔽するプロセスをモニタリングシステムが見つけることが出来れば、マルウェアに侵入されていることがわかる。このように複数の検査機能を用いることにより、個々の方法では発見できないマルウェアを複数の方法を用いることで発見可能になる相乗効果が期待でき、原因究明に寄与することが出来る。

2.2 D-Visor のための仮想化層検証技術

D-Visor は、D-RE が必要とするシステムの論理分割を行い System Container を実現するため、アイソレーション機能を提供する。D-Visor のための仮想化層検証技術は、DEOS プロジェクトで研究開発されているモデルチェッカ(DEOS モデルチェッカ)を用いた静的検証により、D-Visor のアイソレーション機能を検証する。そのため、ページフォルト時におけるシャドウページテーブルのアップデート処理、シリアルラインデバイスに対する操作に着目し、ソースコードに仕様記述を追加し、それぞれの妥当性を静的に検証した。これらにより、D-Visor およびハードウェアデバイスの空間的アイソレーションが検証できることを示した。

シャドウページテーブルとは VMM で管理する、ゲスト OS のページテーブル(ゲストページテーブル)のコピーであり、プロセッサが仮想から物理へのアドレス変換のために使用するのはシャドウページテーブルである。ゲスト OS は自身のメモリ領域内に確保したゲストページテーブルを操作し、VMM はシャドウページテーブルを操作する。2 つのページテーブルの一貫性を保つための 1 つの契機となるのが、ページフォルトである。ページフォルト発生時に、VMM はゲストページテーブルにおけるページフォルトが発生したアドレスのエントリを参照し、シャドウページテーブルの対応するエントリにコピーする(シャドウページテーブルのアップデート)。この時、シャドウページテーブルに含まれる VMM のエントリを破壊してしまうと、アイソレーションが行われず、システム全体の障害要因となってしまう。また、アップデートす

るエントリのページテーブル内のインデックスも、有効な範囲内である必要がある。これらの 2 点について仕様記述を行い、操作の妥当性を検証した。また、VMM のエントリを破壊してしまう可能性のあるソースコードでは、モデル検査器が不正であると報告することも確認した。

```
page_addr_t cr2 = read_cr2();
/*@
 $assumes ((cr2.l2 >= 0) && (cr2.l1 >= 0) && (cr2.offset >= 0) &&
 (cr2.l1 < NUM_PTE) && (cr2.offset < NUM_PAGE_OFFSET));
 */
if (cr2.l2 < VMM_TOP_PDE)              // protects VMM from an update.
    update_guest_paging_at(&cr2);
else
    panic("CR2 out of range");
```

図 4-2-2 VMM におけるシャドウページテーブルアップデートの妥当性チェック例

2 つめの検証例は、ゲスト OS がデバイスへの操作を正しい順序で行っていることを VMM がチェックし、そのチェックからデバイスへの操作順序が不正である場合に、デバイスへの操作を中断し、ゲスト OS への通知が行われることを保証するものである。これにより、不正な操作によりデバイスが異常状態になることからのアイソレーションが可能になる。このチェックが確実なものであることを検証により保証することで、特にゲスト OS がパススルーによりデバイスを操作する場合に、ゲスト OS のバグまたは悪意のある操作によりデバイスを使用不可能な状態にすることを防ぐことができる。このための仕様記述を、シリアルラインの入出力に対して行い、確実にゲスト OS への通知が行われることを検証した。

2．3 VM サブコアチームにおける研究開発

筑波大学グループは、VM サブコアチームが発足した平成 21 および 22 年度にリーダー、平成 23 年度はサブリーダーとして、VM サブコアチームの研究開発を推進した。筑波大学グループは、VM サブコアチームにおいて、White Paper V.2, LAAS レポートを取りまとめた。また、D-System Monitor ランタイム API を策定するための研究開発を行い、その成果をもとに VM サブコアチームの成果を統合するデモシステムの開発を行った。

White Paper V.2 では、オープンシステムディペンダビリティを実現する要素技術として各チームで開発されている技術が記述された。筑波大学グループは、VM サブコアチームの提供する技術をフレームワーク（現在の DEOS Runtime Environment (D-RE)）において仮想化技術を提供する D-Visor および OS の動作を外部から OS とは独立に監視する D-System Monitor の定義、そして D-Visor および D-System Monitor が実現する技術について 5.5 章の「仮想化技術とその応用」に取りまとめた。White Paper V.3 では、DEOS プロセスを中心とする形態にまとめ直されたため、V.2 の VM サブコアチームに関連する要素技術の内容は 5.4 章の「セキュリティ対応」に引き継がれた。

LAAS レポートでは、筑波大学グループは Executive Summary, D-System Monitor Runtime API, a Verification Technique for Virtualization Layers について執筆を行い、その他 VM サブコアチームの著作を取りまとめ、体裁を整え、校正を行い、英文校閲に対応し、報告書を完成させた。さらに、LAAS からの質問に対する回答の取りまとめも行った。LAAS レポートに関連し、2010 シンポジウム「オープンシステムディペンダビリティに向けて」においては、ワークショップにてレポートの内容をもとに発表を行い、質疑応答に対応した。LAAS レポートは、最終的に LAAS より

This part of the work and the content of the corresponding document is excellent.

との良い評価を得ることができた。

D-System Monitor ランタイム API を策定するための研究開発は、D-System Monitor の監視機能および D-Visor の研究開発を行っているのが複数のチームに渡るため、VM サブコアチームで検討を行った。ランタイム API が監視対象 OS の内部構造および振る舞いの両方の監視に対応できるようにするため、内部構造を監視する早稲田グループのモニタリングシステム、振る舞いを監視する河野チームの FoxyKBD, RootkitLibra を主な題材として API を検討した。また、振る舞いを監視するためには仮想デバイスが必要となるが、複数種類の D-Visor でのサポートを考慮した検討を行った。VM サブコアチームでの検討結果から、D-System Monitor ランタイム API の大枠を決定し、筑波大グループでの D-System

Monitor ランタイム API の実装で詳細を決定した。

VM サブコアチームの統合した成果を ET2011 にてデモンストレーションするため、デモシステムのシナリオや構成について VM サブコアチームで検討を行った。統合デモでは、VM サブコアチームの成果が DEOS プロセスに対応し動いていることを示すことが重要である。D-System Monitor および D-Visor は、DEOS プロセスにおける通常運用から予兆検知・障害発生を経て障害対応を行う障害対応サイクルに寄与するものであるため、通常運用時および障害発生の両方で VM サブコアチームの成果が有効であることを示すことができるシナリオとした。図 4-2-3 にそのシナリオを示す。通常運用時では D-System Monitor によるオーバヘッドを最小に留め、また監視対象 OS の構成を変更することがないように、軽量な監視機能として早稲田グループのモニタリングシステムのみを動作させ、予兆検知を行うものとした。予兆検知・障害発生時には、原因究明・迅速対応が可能なことを示すため、D-System Monitor に複数の監視機能を導入するものとした。2 通りの状態を用意し、これらにより通常運用から障害対応への移行を表すことで、D-System Monitor および D-Visor が DEOS プロセスに対応し動いていることを示す。

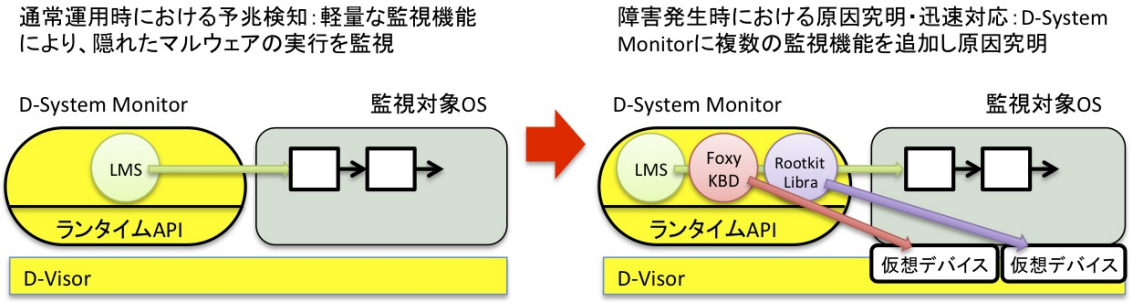

図 4-2-3 D-System Monitor の通常運用から障害対応への移行

上記の VM サブコアチームの成果を統合したデモンストレーションは、筑波大グループにおいて D-System Monitor ランタイム API の実装を用いて構築した。

3. 成果の現状

筑波大学グループで開発したシステムは、VM サブコアチームの他のチームが開発したシステムと統合され、最終成果報告会では VM サブコアチームの成果を統合したデモンストレーションを行う。上述した通り、VM サブコアチームの成果を統合したデモンストレーションは、筑波大グループにおいて D-System Monitor ランタイム API の実装を用いて構築した。D-System Monitor の監視機能である早稲田グループのモニタリングシステム、河野チームの FoxyKBD, RootkitLibra を除き、D-System Monitor ランタイム API、x86 システムで動作するメモリ保護機能を含むアイソレーション機能を提供する D-Visor、本 D-Visor 上で動作する Linux が、VM サブコアチーム統合デモにおける筑波大学グループの貢献である。

D-Visor のための仮想化層検証技術では、仮想化層による空間的アイソレーションが検証できることを示し、またゲスト OS によるデバイスへの操作についても、検証により仮想化層で正しい操作が行われることを保証できることを示した。デバイスに対し正しい動作を行い、常に正常な状態に保つようにすることは、高い安全性が要求されるシステムでは重要であり、その成果については、特許を出願済みである（特願 2011-140264）。

筑波大学グループでは、さらに以下の 3 点について実用性を高めるための取り組みを行った。

1）評価用パッケージの作成

開発したソフトウェアを簡単に評価できるように、DEOS センターと共同で評価用パッケージを作成している。パッケージは、x86 システムを対象とし、現在 Linux ディストリビューションとして標準的に用いられている Ubuntu 環境を前提としている。パッケージは、筑波大学グループで開発した、D-Visor、本 D-Visor 上で動作する Linux カーネル、D-System Monitor（ランタイム API を用いて実装された早稲田グループおよび河野チームの監視機能を含む）、ソースコードからの構築方法やインストール方法、使用方法が記述されたドキュメントからなる。ドキュメントの記述に沿って作業をすすめることで、容易に評価環境を構築できるようになっている。

2）組み込みシステム向けの実装

筑波大学グループで開発したシステムは、通常運用時では少ないメモリ量で動作するように実装されているため、組み込み用途の小さなシステムでも D-Visor 上で監視対象 OS とあわせて D-System Monitor が動作可能であり、またハードディスクがないシステム構成でも動作可能である。一般的な仮想化ソフトウェアはサーバ用途で開発されているため、メモリ資源は十分にあるものとして設計、実装されている。一方、組み込みシステムは必要な資源は少なければ少ないほどコスト削減につながるため、少ないメモリ量で動作することは重要な要因である。ハードディスクの存在を前提としなくて良いことも、様々なハードウェア構成がとられる組み込みシステムでは、有用である。

3）移植性

筑波大学グループで D-System Monitor ランタイム API を実装するにあたり、仮想デバイス機能として virtio を用いることにした。Virtio は、ゲスト OS 側で用いるデバイスドライバが Linux カーネルのメインツリーに取り込まれているため、Linux の仮想デバイスとして標準的に用いられるようになってきている。D-Visor は利用者の要求に合わせ、必要な機能を提供するものを選択する必要があるが、D-System Monitor ランタイム API を移植する必要がある場合も、virtio 機能をサポートするものであれば、その移植は容易であり、より多くの利用者の獲得につながる。

4. 領域への貢献、研究開発の価値

筑波大学グループによる領域への貢献として、主に以下の 2 点がある。

1）D-System Monitor ランタイム API による監視機能の統合

D-System Monitor ランタイム API は、DEOS Runtime Environment (D-RE) における D-System Monitor に複数の監視機能の統合を可能にする。領域では、D-System Monitor として、複数の監視機能が開発されてきたが、それぞれ独立に開発されたため、それぞれ独自の方法で D-Visor を変更し、監視に必要な監視対象 OS にアクセスしていた。D-System Monitor ランタイム API は、監視対象 OS へのアクセス方法を統一するため、新たな攻撃方法に対応した監視機能の開発を迅速に行えるようになる。また、複数の監視機能が使用できるようになるため、原因究明が強化され得る。D-System Monitor ランタイム API により、VM サブコアチームの他のチームが開発したシステムと統合され、最終成果報告会では VM サブコアチームの成果を統合したデモンストレーションを行うことができるようになった。

2）VM サブコアチームにおける研究開発の推進

筑波大学グループは、VM サブコアチームが発足した平成 21 および 22 年度にリーダー、平成 23 年度はサブリーダーとして、VM サブコアチームの研究開発を推進した。筑波大学グループは、VM サブコアチームにおいて、White Paper V.2, LAAS レポートを取りまとめた。また、D-System Monitor ランタイム API を策定するための研究開発を行い、その成果をもとに VM サブコアチームの成果を統合するデモシステムの開発を行った。

筑波大学グループによる研究開発自体の価値としては、以下の 2 点が重要である。

1）組み込みシステム向けアイソレーション機能を提供する仮想化技術

筑波大学グループでは、組み込みシステムをターゲットに、x86 プロセッサの特徴を生かし機能を単純化し実装をシンプルにすることで、少ないメモリ量で動作する仮想化層を開発した。仮想化層はメモリ保護を行いまた使用するプロセッサを分離することで、仮想化層上で動作するゲスト OS 間の空間的アイソレーションを実現する。一方、デバイスやソフトウェアからの割り込みはゲスト OS への直接通知が可能であり、またデバイスへの直接アクセスを行えるようにすることで、組み込みシステム向けのオーバヘッドの少ない仮想化層が実現できることを示した。さらに、直接通知をする場合でも仮想化層を割り込み可能な状態で実行でき、リアルタイム性を高めることができることを明らかにした。

2）組み込み向け仮想化層の検証技術

仮想化層はプロセッサやデバイスなどのハードウェアに依存した部分がその実装の多くを占め、抽象化された動的な資源管理など、これまで検証の対象となってきた機能は、特に組み込み向けの仮想化層には含まれない。一方、ハードウェアが常に正常な状態であるように操作を行っているかは組み込み向け仮想化層でもその妥当性が検証されるべき性質である。仮想化層のソフトウェアに記述されるハードウェアへの操作はビット操作で表され、ソフトウェアには状態が保存されないため、そのままでは静的検証を適用できない。そのため、ビット操作を抽象化した構造体への操作として表し、状態をトレースすることで静的検証が適用可能になることを示した。また、ハードウェアをモデル化し、その状態をトレースすることで、ハードウェアが正常な状態に保たれる範囲で操作を行っているか、検証可能であることを明らかにした。

⑵研究成果の今後期待される効果

筑波大学グループの行った研究開発成果に対し、今後期待される効果としては、主に以下の 2 点がある。

1）D-System Monitor ランタイム API
D-System Monitor ランタイム API を用いることにより、D-Visor の内部構造に精通する必要はなくなるため、D-System Monitor における監視機能の研究開発の加速が期待される。また、監視機能の作成方法の変化も考えられる。これまでは、検査を行うための機能がある程度の規模のソフトウェアとして実現されてきたが、細分化され部品化された機能を組み合わせ、D-Script により必要に応じた監視を行う方法も考えられる。

2）仮想化層の検証技術
仮想化層の検証技術のうち、ハードウェアが正常な状態に保たれる範囲で操作を行っているか、検証する技術は、説明責任遂行のためのエビデンスと原因究明に利用できることが期待される。デバイスへのある操作（単一の操作に限らず、複数デバイスに渡る操作の連続でも構わない）が致命的な障害につながる場合、複合的な要因で上位レイヤにおけるチェックをすり抜けた場合でも、仮想化層において最後のチェックを行うことができる。障害を防げなかった場合でも、ハードウェアがどのような状態にあったかを記録することで、原因究明に利用できることが考えられる。

§５ 成果物等

1. HP(URL: https://repos.dcl.info.waseda.ac.jp/spumone/trac/wiki/ドキュメント)にて公開した.

2. DEOS センターに開発したソフトウエアをパッケージにして納入する予定である.

⑵規格

⑶知財出願

①国内出願（1 件）
1. 仮想化レイヤにモデル検査を適用することにより、ゲスト OS によるデバイス操作を強制する方法、追川修一、筑波大学、2011 年 6 月 24 日、特願 2011-140264.

②海外出願（1 件）

③その他の知的財産権
他に記載するべき知的財産権があればご記入下さい。（実用新案 意匠 プログラム著作権 等）

⑷原著論文発表 （国内（和文）誌 1 件、国際（欧文）誌 47 件）

1. Yuki Kinebuchi, Hidenari Koshimae and Tatsuo Nakajima, Constructing Machine Emulator on Portable Microkernel, The 22nd Annual ACM Symposium on Applied Computing Seoul, pp.1197-1198, 韓国, 2007年3月

2. Yu Murata Wataru Kanda Tatsuo Nakajima, ″A Study on Asymmetric Operating Systems on Symmetric Multiprocessors″, In Proceedings of the 2007 IFIP International Conference on Embedded and Ubiquitous Computing, 2007(Best Paper Award), pp.182-195, 台湾, 2007年12月

3. Yuki Kinebuchi, Midori Sugaya, Shuichi Oikawa, Tatsuo Nakajima, ``Task Grain Scheduling for Hypervisor-Based Embedded System'', In Proceeding of the 10th IEEE International Conference on High Performance Computing and Communications (HPCC-08), pp.190-197, 中国, 2008年9月

4. Sun Lei, Tatsuo Nakjaima, ``A Lightweight Kernel Objects Monitoring Infrastructure for Embedded Systems'', In Proceedings of The 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2008), pp.55-60, 台湾, 2008年8月

5. Hiroo Ishikawa, Alexandre Courbot, Tatsuo Nakajima, ``A Framework for Self-healing Device Drivers'', Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems, pp. 277-286, イタリア, 2008年10月

6. Lei Sun, Tatsuo Nakajima, ``A Lightweight Detection and Recovery Infrastructure of Kernel Objects for Embedded Systems'', In Proceedings of The 2008 International Conference On Embedded and Ubiquitous Computing(EUC 2008), pp. 136-143, 中国, 2008年12月

7. Wataru Kanda, Yuki Kinebuchi, Yu Yumura, Tatsuo Nakajima, ``SPUMONE: LightWeight CPU Virtualization Layer for Embedded Systems'', In Proceedings of The 2008 International Conference On Embedded and Ubiquitous Computing(EUC 2008), pp.144-151, 中国, 2008年12月

8. Shingo Aoyagi, Shuichi Oikawa: IXIV VMM: A VMM on 2-Level Ring Architecture. In Proceedings of the 8th IEEE International Conference on Computer and Information Technology, pp. 533-538, 2008.

9. Tomohiro Katori, Lei Sun, Dennis Nilsson, Tatsuo Nakajima, ``Building a Self-Healing Embedded System in a Multi-OS Environment'', In the proceedings of the 24th Annual ACM Symposium on Applied Computing, pp.293-29, アメリカ, 2009年3月

10. Midori Sugaya, Yuki Ohno, Andrej van der Zee and Tatsuo Nakajima, ″A Lightweight Anomaly Detection System for Information Appliances″, 12th IEEE International Symposium on Object/component/service-oriented Real-time

distributed Computing (ISORC 2009), pp.257-266, 日本, 2009年3月

11. Lei Sun, Dennis K. Nilsson, Tomohiro Katori and Tatsuo Nakajima, "Online Self-Healing Support for Embedded Systems", 12th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2009), pp.283-287, 日本, 2009年3月

12. Ki-duk Kwon, Midori Sugaya and Tatsuo Nakajima, Analysis of Embedded Kernel using Kernel Analysis System, The 6th International Conference on Embedded Software and Systems, pp. 417-422, 中国, 2009年5月

13. Yuki Kinebuchi, Kazuo Makijima, Takushi Morita, Midori Sugaya, Tatsuo Nakajima, "CONSTRUCTING MULTI-OS PLATFORM WITH MINIMAL ENGINEERING COST", International Embedded Systems Symposium 2009, pp.195-206, ドイツ, 2009年9月

14. Andrej van der Zee, Alexandre Courbot, Tatsuo Nakajima, "mBrace: Action-based Performance Monitoring of Multi-Tier Web Applications", In Proceedings of The 2009 International Conference On Embedded and Ubiquitous Computing(EUC 2009), pp.,166-173, 2009年8月, DOI: http://doi.ieeecomputersociety.org/10.1109/CSE.2009.219

15. Wataru Kanda, Yu Murata and Tatsuo Nakajima, "SIGMA System: A Multi-OS Environment for Embedded Systems", Journal of Signal Processing Systems, Vol.59, No.1, pp.33-43, 2010年

16. Midori Sugaya, Yuki Ohno, and Tatsuo Nakajima, Lightweight Anomaly Detection System with HMM Resource Modeling, International Journal of Security and Its Applications, pp35-54, Vol. 3, No. 3, 2009年7月

17. Yuki Kinebuchi, Kazuo Makijima, Takushi Morita, Midori Sugaya, Tatsuo Nakajima, "CONSTRUCTING MULTI-OS PLATFORM WITH MINIMAL ENGINEERING COST", International Embedded Systems Symposium 2009, pp.195-206, 2009年9月, DOI: 10.1007/978-3-642-04284-3_18

18. Lei Sun, Yuki Kinebuchi, Tomohiro Katori, Tatsuo Nakajima, Runtime Self-Diagnosis and Self-Recovery Infrastructure for Embedded Systems, International Conference on Self-Adaptive and Self-Organizing Systems, pp.284-285, 2009.年9月
19. DOI: http://doi.ieeecomputersociety.org/10.1109/SASO.2009.21

20. Lei Sun, Hiromasa Shimada, Tatsuo Nakajima, Reusable Integrity Management Services for Embedded Systems,The 2009 IEEE International Conference on Service-Oriented Computing and Applications (SOCA'09), pp.65-72, 2009年12月

21. Ki Duk Kwon, Midori Sugaya, Tatsuo Nakajima, "KTAS: Analysis of Timer Latency

for Embedded Linux Kernel”, International Journal of Advenced Science and Technology(IJAST), Vol. 18, 2010.

22. Yuki Kinebuchi, Kazuo Makijima, Takushi Morita, Alexandre Courbot, and Tatsuo Nakajima, ″Composition Kernel: A Software Solution for Constructing a Multi-OS Embedded System,″ EURASIP Journal on Embedded Systems, vol. 2010, Article ID 458085, 8 pages, 2010. DOI:10.1155/2010/458085.

23. 菅谷 みどり 中島 達夫, ``情報家電向け Linux の CPU 資源制限システムの設計と実装'', 電子情報通信学会論文誌 D, Volume J93-D No.10 , Oct. pp.2001-2010, (2010)

24. Hiromasa Shimada, Alexandre Courbot, Yuki Kinebuchi, Tatsuo Nakajima, ″A Lightweight Monitoring Service for Multi-Core Embedded Systems″, In Proceedings of the 13th Symposium on Object-Oriented Real-Time Distributed Computing, pp.202-210, 2010, DOI: 10.1109/ISORC.2010.12

25. Yuki Ohno, Sayaka Akioka, Midori Sugaya, Tatsuo Nakajima, ″A Library-Based Tool to Improve CPU Assignment for Multicore Processor-Based Pervasive Servers,″ In Proceedings of IEEE 16th International Conference on Embedded and Real-Time Computing Systems and Applications, pp.114-123, IEEE Press, (2010), DOI: 10.1109/RTCSA.2010.21

26. Tatsuo Nakajima Yuki Kinebuchi Alexandro. Courbot Hiromasa Shimada T-H. Lin Hitoshi Mitake, ″Composition Kernel: A Multi-core Processor Virtualization Layer for Rich Functional Smart Products″, In Proceeding of The 8th IFIP Software Technologies for Future Embedded and Ubiquitous Systems(SEUS2010), pp. 227-238, LNCS 6399 Springer, (2010), DOI: 10.1007/978-3-642-16256-5_22.

27. Tatsuo Nakajima, Yuki Kinebuchi, Hiromasa Shimada, Alexandre Courbot, Tsung-Han Lin, Temporal and Spatial Isolation in a Virtualization Layer for Multi-core Processor based Information Appliances, 16th Asia and South Pacific Design Automation Conference, pp.645-652, 2011,

28. Sun Lei, Tatsuo Nakajima, “Online Self-Diagnosis Self-Recovery Infrastructure for Embedded Systems”, International Journal of Security and Its Applications, Vol. 4, No. 4, 2010.

29. Akihiro Suzuki, Shuichi Oikawa: SIVARM: a Virtual Machine Monitor for the ARM Architecture, In Proceedings of the IEEE Region 10 International Conference (TENCON 2010), pp. 1088~1093, 2010.

30. Akihiro Suzuki, Shuichi Oikawa: Implementation of SIVARM: A Simple VMM for the ARM Architecture, In Proceedings of the 1st International Conference on Networking and Computing (IC-NC 2010), pp. 290~291, 2010.

31. Shuichi Oikawa, Jin Kawasaki: Simultaneous Logging and Replay for Recording Evidences of

System Failures. In Proceedings of the 8th IFIP Software Technologies for Future Embedded and Ubiquitous Systems (SEUS 2010), Springer-Verlag LNCS 6399, pp. 143-154, 2010. (DOI: 10.1007/978-3-642-16256-5_15)

32. Akihiro Suzuki, Shuichi Oikawa: Implementation of Virtual Machine Monitor for ARM Architecture. In Proceedings of the 10th IEEE International Conference on Computer and Information Technology, pp. 2244~2249, 2010.

33. Jin Kawasaki, Shuichi Oikawa: Co-Locating Virtual Machine Logging and Replay for Recording System Failures. In Proceedings of the 10th IEEE International Conference on Computer and Information Technology, pp.1352-1357, 2010. (DOI: 10.1109/CIT.2010.242)

34. Hitoshi Mitake, Yuki Kinebuchi, Alexandre Courbot, Tatsuo Nakajima, Coexisting Real-Time OS and General Purpose OS on an Embedded Virtualization Layer for a Multicore Processor, In Proceedings of the 26th ACM Symposium On Applied Computing, pp.629-630, 2011.

35. Hiromasa Shimada, Yuki Kinebuchi, Tsung-Han Lin, Alexandre Courbot, Tatsuo Nakajima, Design Issues in Composition Kernels for Highly Functional Embedded Systems, In Proceedings of the 26th ACM Symposium On Applied Computing, pp.338-345, 2011.

36. Tsung-Han Lin, Yuki Kinebuchi, Alexandre Courbot, Hiromasa Shimada, Hitoshi Mitake, Chen-Yi Lee, Tatsuo Nakajima, Hardware-assisted Reliability Enhancement for Embedded Multicore Virtualization Design, In Proceedings of the 14th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing, pp.241-249, 2011. DOI: 10.1109/ISORC.2011.37

37. Yuki Kinebuchi, Hitoshi Mitake, Yohei Yasukawa, Takushi Morita, Alexandre Courbot, Tatsuo Nakajima. A Study on Real-time Responsiveness on Virtualization Based Multi-OS Embedded Systems. In Proceedings of the 1st International Conference on Pervasive and Embedded Computing and Communication Systems, pp.369-378, 2011.

38. Hitoshi Mitake, Yuki Kinebuchi, lexandre Courbot, Tatsuo Nakajima.Towards co-existing of Linux and real-time OSes, Ottawa Linux Symposium 2011. (In Press).

39. Hiromasa Shimada, Tatsuo Nakajima, ``External Integrity Checking with Invariants'', In Proceedings of IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications, Vol.2, pp.122-125, 2011.

40. Ning Li, Yuki Kinebuchi, Tatsuo Nakajima, ``Enhancing Security of Embedded Linux on a Multi-core Processor'', In Proceedings of IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications,

Vol.2, pp.117-121, 2011.

41. Tsung-Han Lin, Yuki Kinebuchi, Hiromasa SHimada, Hiroshi Mitake, Chen-Yi Lee, Tatsuo Nakajima, ``Hardware-Assisted Reliability Enhancement for Embedded Multi-core Virtualization Design'', In Proceedings of IEEE 17th International Conference on Embedded and Real-Time Computing Systems and Applications, Vol.2, pp.101-105, 2011.

42. Tatsuo Nakajima, Tetsuo Yamabe, Mizuki Sakamoto, ``Proactive Ambient Social Media for Supporting Human Decision Making''. International Conference on Ubiquitous Intelligence and Computing 2011, pp.25-39, LNCS 6905, 2011.

43. Shuichi Oikawa, Jin Kawasaki: Simultaneous Virtual-Machine Logging and Replay, IPSJ Journal of Information Processing, Vol. 19, pp. 400~410, August 2011.

44. Megumi Ito, Shuichi Oikawa: Making a Virtual Machine Monitor Interruptible, IPSJ Journal of Information Processing, Vol. 19, pp. 411~420, August 2011.

45. Akihiro Suzuki, Shuichi Oikawa: Implementing a Simple Trap and Emulate VMM for the ARM Architecture, In Proceedings of the 17th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2011), pp. 371~379, 2011.

46. Hitoshi Mitake, Yuki Kinebuchi,Tsung-Han Lin and Tatsuo Nakajima, Solving the Lock Holder Preemption Problem in a Multicore processor-based Virtualization Layer for Embedded Systems, 2nd International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS), 2012.

47. Tsung-Han Lin, Hitoshi Mitake, Yuki Kinebuchi and Tatsuo Nakajima, GLOBAL SCHEDULING FOR THE EMBEDDED VIRTUALIZATION SYSTEM IN THE MULTI-CORE PLATFORM, 2nd International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS), 2012.

48. Mizuki Sakamoto, Tatsuo Nakajima, Tetsuo Yamabe, Todorka Alexandrova, Harmonizing Virtual Forms into Traditional Artifacts to Increase Their Values, 3rd International Symposium on Ambient Intelligence, 2012.

(5)その他の著作物(総説、書籍など)

1. 中島　達夫, 組込みオペレーティングシステム概論, 映像情報メディア学会, ２００９年, ５月号, 633-637, 2009

2. 仮想化を利用したディペンダブル OS の構築,追川修一,ソフトウェアデザイン, pp. 98-100, 2010 年 2 月.

3. 電子情報技術産業協会　組込みマルチコアハンドブック技術・応用編 第4.1章 マルチコアプロセッサのための基盤ソフトウエア，pp.40-55，社団法人電子情報技術産業協会 マイクロプロセッサ専門委員会，2010年2月

⑹国際学会発表及び主要な国内学会発表

① 招待講演　　（国内会議 0件、国際会議 10件）

1. Tatsuo Nakajima, “Software Infrastructure for Next Generation Computing Environments”, Colloquium, Rutgers University, 米国, 2007年12月

2. 中島　達夫,「ディペンダブル組込みOSの挑戦」STARCフォーラム/シンポジウム2008，日本，2008年7月

3. Yutaka Ishikawa, Hajime Fujita, Toshiyuki Maeda,Midori Sugaya, Mitsuhisa Sato, Toshihiro Hanawa, Yuki Kinebuchi, Tatsuo Nakajima, Jin Nakazawa, and Hideyuki Tokuda ″Towards an Open Dependable Operating System″, 12th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2009) 日本，2009年3月（招待論文）

4. Tatsuo Nakajima, An Operating System for Future Multi/many-cores Information Appliances, The Second International Workshop on Wireless Sensor Networks and Embedded Systems Research, 韓国, 2009年6月（招待論文）

5. 中島　達夫，ヒューマンファクターを考慮した次世代情報家電サービス，CEATEC Japan，幕張，2009年10月

6. 中島　達夫，マルチコアプロセッサのための基盤ソフトウエア，CEATEC Japan，幕張，2009年10月

7. Tatsuo Nakajima, Yuki Kinebuchi, “A Composition Kernel for Multicore Dependable Embedded Systems”, Carnegie Mellon University, 2010年3月

8. 中島　達夫,「ーマルチコア、マルチOSをフレキシブルにつなぐー 組込みマルチコアプロセッサのための仮想化技術」，第9回 SuperH™オープンフォーラム，８月27日

9. Tatsuo Nakajima, Temporal and Spatial Isolation in a Virtualization Layer for Multi-core Processor based Information Appliances, ASPDAC 2011, 2011年１月（招待論文）

10. Tatsuo Nakajima, “Composition Kernel: A Multi-core Processor Virtualization Layer for Rich Functional Smart Products,” Alcatel Lucent Technical Academy, 2011年２月

② 口頭発表　　（国内会議 0件、国際会議 14件）

1. Yuki Kinebuchi, Hidenari Koshimae, Shuichi Oikawa , Tatsuo Nakajima, Dynamic

Translator-based Virtualization, The 5th IFIP Workshop on Software Technologies for Future Embedded & Ubiquitous Systems (SEUS 2007), ギリシャ, 2007 年 5 月

2. Lei Sun, Hiroo Ishikawa, and Tatsuo Nakajima, Kernel Data Structure-based Runtime Monitoring, In Proceedings of the First International Workshop on Dependable Ubiquiotus Nodes, 日本, 2007 年 11 月

3. Hiroo Ishikawa and Tatsuo Nakajima, Micro-Reboot Support for Multi-Server Operating Systems, In Proceedings of the First International Workshop on Dependable Ubiquitous Nodes, 日本, 2007 年 11 月

4. Dennis K. Nilsson, Lei Sun, Tatsuo Nakajima, ``A Framework for Self-Verification of Firmware Updates over the Air in ECUs'', 3rd IEEE Workshop on Automotive Networking and Applications, アメリカ, 2008 年 12 月

5. Tatsuo Nakajima, et. al., ``An Operating System Architecture for Future Information Appliances'', In Proceedings of IFIP International Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, イタリア, 2008 年 10 月

6. Kiduk Kwon, Midori Sugaya, and Tatsuo Nakajima, ″Analysis of High Resolution Timer Latency using Kernel Latency Analysis System in Embedded System'', First International Workshop on Software Technologies for Future Dependable Distributed Systems (STFSSD 2009) , 日本, 2009 年 3 月

7. Yuki Kinebuchi, Wataru Kanda, Yu Yumura, Kazuo Makijima, and Tatsuo Nakajima, ″A Hardware Abstraction Layer for Integrating Real-Time and General-Purpose with Minimal Kernel Modification″, First International Workshop on Software Technologies for Future Dependable Distributed Systems (STFSSD 2009) , 日本, 2009 年 3 月

8. Yuki Ohno, Midori Sugaya, Andrej van der Zee, and Tatsuo Nakajima, ″Anomaly Detection System using Resource Pattern Learning″, First International Workshop on Software Technologies for Future Dependable Distributed Systems (STFSSD 2009) , 日本, 2009 年 3 月

9. Shuichi Oikawa: Implementing the Dependable Operating SystemArchitecture on the IA-32 Architecture. In Proceedings of the 1st International WorkshoponSoftware Technologies for Future Dependable Distributed Systems (STFSSD2009), 2009.

10. Tatsuo Nakajima, Yuki Kinebuchi, Alexandre Courbot, Hiromasa Shimada, Tsung-Han Lin, and Hitoshi Mitake, Composition Kernel: A Multi-Core Processor Virtualization Layer for Highly Functional Embedded Systems, The 16th IEEE Pacific Rim International Symposium on Dependable Computing, pp. 223-224, 2010 年 12 月

11. Yuki Kinebuchi, Tatsuo Nakajima, Core-Local Memory Assisted Protection, The 16th IEEE Pacific Rim International Symposium on Dependable Computing, pp. 233-234, 2010 年 12 月

12. Shuichi Oikawa: Enforcing Dependable Operations by Model Checking a Virtualization Layer, In Proceedings of the 1st International Workshop on Open Systems Dependability, 41st IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 254-256, 2011.

13. Tatsuo Nakajima, Tetsuo Yamabe, Todorka Alexandrova and Mizuki Sakamoto, Digital Physical Hybrid Design: Harmonizing the Real World and the Virtual World, 1st International Workshop on Large-Scale Cyber-Physical Systems (LCPS 2011), 2011年12月

14. Hitoshi Mitake, Hiromasa Shimada, Tsung-Han Lin, Ning Li, Yuki Kinebuchi, Chen-Yi Li, Daisuke Yamaguchi, Takumi Yajima, Tatsuo Nakajima, Light-Weighted Virtualization Layer for Multicore Processor-Based Embedded Systems. Second International Workshop on Runtime Environments/Systems, Layering, and Virtualized Environments(RESoLVE 2012), 2012年3月

③ ポスター発表　　(国内会議 0件、国際会議 12件)

1. 発表者(所属)、タイトル、学会名、場所、月日

1. Yuki Kinebuchi and Tatsuo Nakajima, Design and Implementation of Cooperative Priority-based Scheduling, Poster Presentation, ACM EuroSys 2007, ポルトガル, 2007年3月

2. Lei Sun, Hiroo Ishikawa, Tatsuo Nakajima, Specification-based Runtime Kernel Data Structure Monitoring, In SOSP 2007, Poster Presentation. アメリカ, 2007年10月

3. Midori Sugaya, Yuki Kinebuchi, Shuichi Oikawa, Tatsuo Nakajima, VPE: Virtual Periodic Execution for Embedded System, In RTAS 2008, WIP Presentation. アメリカ, 2008年12月

4. Yuki Kinebuchi, Wataru Kanda, Yu Yumura, Tatsuo Nakajima, “Software Infrastructure for Future Many-Core Embedded Systems”, International Conerence on Eurosys 2008, Poster Presentation, イギリス, 2008年4月

5. Lei Sun, Hiroo Ishikawa, Tatsuo Nakajima, A Kernel Data Structure-based Monitoring Infrastructure for Embedded System, International Conerence on Eurosys 2008, Poster Presentation, イギリス, 2008年4月

6. Akihiro Suzuki, Shuichi Oikawa: Development of Mesovirtualization for the ARM Architecture. 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2010), Work-in-Progress, 2010.

7. Yuki Kinebuchi, Alexandre Courbot, Hiromasa Shimada, Tatsuo Nakajima, A Composition Kernel: Towards Multicore Embedded Systems, Poster Paper, Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2010), 2010年3月

8. Hiromasa Shimada, Yuki Kinebuchi, Alexandre Courbot, Tatsuo Nakajima, Reliability and Security Issues in Multiple OS Environments for Future Embedded Systems, Poster Paper, Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2010), 2010 年３月

9. Hitoshi Mitake, Yuki Kinebuchi, Alexandre Courbot, Tatsuo Nakajima, Handling Lock-Holder Preemption in Real-Time Virtualization Layer for Multicore Processors, RTCSA 2010, 2010 年８月.

10. Tatsuo Nakajima, Yuki Kinebuchi, Hiromasa Shimada, and Tsung-Han Lin A Composition Kernel: A Software Infrastructure for Building Complex Embedded Systems, Demo, PRDC 2010, 2010 年 12 月

11. Tatsuo Nakajima, Yuki Kinebuchi, Hiromasa Shimada, and Tsung-Han Lin A Composition Kernel: A Software Infrastructure for Building Complex Embeded Systems, Demo, RTCSA 2011, 2011 年 8 月

12. Shogo Saito, Shuichi Oikawa: Utilization of a SIMD unit in the OS Kernel, International Workshop on Advances in Networking and Computing, 2nd International Conference on Networking and Computing (IC-NC 2011), 2011 年 12 月

13. Daisuke Yamaguchi, Takumi Yajima, Chen-Yi Lee, Yuki Kinebuchi and Tatsuo Nakajima. SPATIAL ISOLATION ON REALTIME HYPERVISOR USING CORE-LOCAL MEMORY. 2nd International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS), 2012. 2012 年 2 月

(7)受賞・報道等

① 受賞

＊ IFIP International Conference on Embedded and Ubiquitous Computing, 2007 最優秀論文賞

②マスコミ(新聞・TV等)報道

③その他

(8)成果展開事例

①実用化に向けての展開

- 開発したハードウエア仮想化層に関しては，研究室 HP(URL: https://repos.dcl.info.waseda.ac.jp/spumone/trac/wiki/ドキュメント)にて公開した.
- ソースコードとドキュメントをCDにパッケージ化してDEOSセンターに来年3月までに納入する.
- JEITA マイクロプロセッサ専門委員会にてマルチコア向け仮想化技術の可能性に関する有効性を提言，JEITA が出版するマルチコアハンドブック技術応用編において開発した技術の解説をおこない，組込みシステムにおける仮想化技術の重要性を啓蒙した.

① 社会還元的な展開活動

- 本研究で得られた仮想化技術，モニタリング技術に基づき，民間企業 2 社(守秘義務有り)と共同研究中.
- 開発したハードウエア仮想化層に関しては，研究室 HP(URL: https://repos.dcl.info.waseda.ac.jp/spumone/trac/wiki/ドキュメント)にて公開した.
- JEITA マイクロプロセッサ専門委員会にてマルチコア向け仮想化技術の可能性に関する有効性を提言，JEITA が出版するマルチコアハンドブック技術応用編において開発した技術の解説をおこなった.
- 昨年の STARTC シンポジウムでの講演や10月の CEATEC での講演を通して DEOS プロジェクトにおける仮想化技術に関する講演をおこなった. 特に今後の組込みシステムではディペンダビリティを考慮することが重要であることを紹介した.
- PRDC2010, RTCSA2011 等における国際会議でのデモンストレーション
- エンベデッドテクノロジー(2006、2007，2008，2009におけるデモンストレーション
- 国内電機メーカ5社における開発した技術に関して解説をおこなった.

§6 研究期間中の主なワークショップ、シンポジウム、アウトリーチ等の活動

年月日	名称	場所	参加人数	概要
2009/3/18-20	12th IEEE International Symposium on Object/component/service-oriented Real-time distributed Computing (ISORC 2009)	東京	約100名	分散組込みリアルタイムシステムに関する国際会議，ディペンダビリティは会議のメインテーマの1つ，DEOS の成果を招待論文として企画する. プロシーディングは IEEE-CS により出版
2009/3/17-18	First International Workshop on Software Technologies for Future Dependable Distributed Systems (STFSSD 2009)	東京	約50名	ワークショップ内で DEOS セッションをワークショップ内に企画し，海外への成果の宣伝をおこなう. プロシーディングは IEEE-CS により出版

§7 結び

本プロジェクトが開発したディペンダビリティの向上技術は，今後大きな問題となるスマートフォンのセキュリティ問題に関する解法を提供している. また，将来のクラウド環境のセキュリティを向上するための基本的なメカニズムとしても利用することが可能である. 以上の意味で，ディペンダビリティの向上に大きな貢献をおこなったと考える.

また，本プロジェクトから SPUMONE の開発に携わった4名の学生が博士を取得し，内外の著名な企業において活躍している. また，現在，4名の博士課程の学生が本プロジェクトに関係する研究をおこない，優れた成果をあげている. 以上より，システム分野における人材育成にも大きく貢献していると考えられる.