Vol.2015-OS-135 No.6
Vol.2015-EMB-39 No.6
2015/11/24

# 組込みコンピュータにおけるハイパーバイザを用いたリアルタイムシステム構成の提案

島田 工[1,a)] 矢代 武嗣[2,b)] 越塚 登[1,2,c)] 坂村 健[1,2,d)]

**概要**：組込みシステムでは高いリアルタイム性や信頼性が求められる．近年はそのような要求に加えて，より高度で複雑なアプリケーションを実装することが求められる．高度なアプリケーションを実装するために汎用 OS のみを用いるとリアルタイム性や信頼性を高く保つことが難しい．本稿では，ハイパーバイザを用いて複数の OS を独立した環境で動かすことを検討する．そのため，組込みシステムの要求を考慮したコンパクトなハイパーバイザを設計する．このようなハイパーバイザが少ない実装量で実現でき，様々なスケジューリング方式を採用できることを明らかにする．また，ハイパーバイザを用いたシステム全体の構成方法について検討する．

**キーワード**：組込みシステム，リアルタイムシステム，ハイパーバイザ，仮想化

## 1. はじめに

組込み機器に用いられるコンピュータシステムには長期間にわたって動作させるための信頼性や，機器そのものの性能維持のためにリアルタイム性が重要である．これらの要求を満たすため，組み込み機器の OS としてリアルタイム OS を用いることが多かった．近年，組込みプロセッサの性能向上や組込みアプリケーションそのものへの要求の複雑化に伴い，RTOS ではなくネットワーク接続などの多様な情報処理を得意とする汎用 OS を用いる事例が増えてきた．しかし，一般に汎用 OS はリアルタイム性が RTOS に劣る．また，複雑なアプリケーションは信頼性の低下を招くことが多い．そこで，組込みシステム上での仮想化技術が注目されている．仮想化技術を用いることで，RTOS と汎用 OS を同じハードウェア上で動作させ，RTOS にリアルタイムな処理をまかせ，汎用 OS に他の情報処理を任せることで，より高度なリアルタイムアプリケーションを構築できる．汎用 OS のみのシステムの場合，RTOS 上で組んだアプリケーションを移植する必要が出てくる．ハイパーバイザを用いることで RTOS を汎用 OS が併用できるようになれば，そのような移植は必要なくなる．また，仮想化環境により各システムの独立性を高めることができ，信頼性の向上にもつながる．

従来であれば，組込みシステム上での完全仮想化はコストが高く，OS にソースコードの変更を伴う準仮想化や [1]，仮想化に類似する方法 [2] で複数の OS を動作させるのが一般的であった．しかし，組込みシステム向けのプロセッサでハードウェアによる仮想化支援機構を搭載したものが利用可能になったことにより，より低いコストで完全仮想化を実現することが可能になった．すでに組込み向けプロセッサの仮想化支援機構を利用して，汎用プロセッサ向けにつくられたハイパーバイザをそれらのプロセッサに移植したものも利用可能である [3]．

しかし，汎用向けのハイパーバイザはそもそもつくられた目的が組込み向けでないことから多くの問題がある．具体的には，リアルタイム性を考慮していない．ほか，ハイパーバイザが汎用 OS に依存していることから，汎用 OS の脆弱性がハイパーバイザの脆弱性になりうるため信頼性が低い問題もある．もともと組込み向けにつくられたハイパーバイザも存在するが，リアルタイム性については多くの課題がある [4]．

本研究では，リアルタイムな組込みシステムに特化した軽量なハイパーバイザを構築した．リアルタイム性を保つため，OS へのハイパーバイザの干渉は可能な限り少なくなるよう設計した．仮想マシンのスケジューリングは，アプリケーションによりスケジューリング方針は大きく変わ

1 東京大学
The University of Tokyo
2 YRP ユビキタス・ネットワーキング研究所
YRP Ubiquitous Networking Laboratory
a) shimada@sakamura-lab.org
b) takeshi.yashiro@ubin.jp
c) koshizuka@sakamura-lab.org
d) ken@sakamura-lab.org

ⓒ 2015 Information Processing Society of Japan

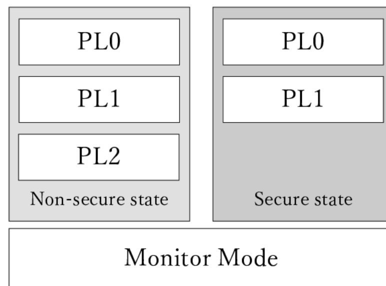


図 1　ARMv7-A におけるプロセッサモード

ると考えられるため，共通の API を定義し，ユーザーがこれを実装するという形式をとった．構築したハイパーバイザ上で複数の RTOS をいくつかののスケジューリング方式で動作させることができた．また，本研究で構築したハイパーバイザを用いて，どのような組込みシステムを構築するのかについての展望を述べる．

本稿の構成は以下の通りである．第 2 章では，組込みシステムにおける仮想化技術の背景を取り上げる．第 3 章では，それらを踏まえたハイパーバイザの設計方針について述べる．第 4 章では，ハイパーバイザの実装について述べる．第 5 章では，ベンチマークを用いてハイパーバイザの評価を行なう．第 6 章では，本研究でのハイパーバイザを用いた組込みシステムの構築方法の展望を述べる．第 7 章では，組込みシステムでの関連研究について述べる．最後の第 8 章では，まとめと今後の課題について述べる．

## 2. 背景

### 2.1 ARM アーキテクチャにおける仮想化

従来の組込みプロセッサ上でハイパーバイザを構築するには，OS のバイナリを実行時に動的に変換するか，OS のコードを予め書き換える準仮想化を用いる必要があった．近年，組込み向けのプロセッサである ARMv7-A アーキテクチャにおいて，Virtualization Extensions（VE）がオプションとして加わり，ARM アーキテクチャ上でより容易にハイパーバイザが構築できるようになった．

ARMv7-A での VE について簡単に説明する．詳細については文献 [5] より確認できる．VE は Popek と Goldberg による仮想化可能な要件 [6] を満たすために追加されたものであり，ゲスト OS 上のプロセッサの状態に対して変更を加えるようなセンシティブ命令をトラップできる．VE では従来の特権レベル（PL0 と PL1）よりも高い特権レベル（PL2）を持つの Hyp モードを追加している．Hyp モードは専用のシステムレジスタを持っており，それらは他の低い特権レベルのモードからは操作することができない．基本的には PL0 と PL1 のモード上で仮想マシンが動作し，PL2 でハイパーバイザが動作することになる．VE には Security Extension も含まれているため，プロセッサの状態はセキュア状態とノンセキュア状態に分かれるが，Hyp モードはノンセキュア状態のみに追加されている．プロセッサモードの全体図は図 1 のようになる．

VE には Large Physical Address Extension も含まれている．これにより，PL0 と PL1 内において従来の Stage-1 の MMU による変換されたアドレスに対して，それを別のアドレスに変換しアクセスさせる Stage-2 のメモリ管理ユニット（MMU）を利用できる．変換テーブルに基づき，Stage-1 の MMU により仮想アドレスは中間物理アドレス（IPA）に変換され，Stage-2 の MMU により IPA は物理アドレスに変換される．Stage-2 での変換テーブルを適切に設定することにより，PL1 と PL0 で動作するアプリケーションがアクセスできるメモリ領域を制限することができる．

ARMv7-A では Generic Interrupt Controller（GIC）が割り込みコントローラとして内蔵されている．GIC は Distributor と CPU interface に別れる．この内，CPU interface は VE により仮想 CPU interface が導入されている．これを利用することで PL2 モードからゲスト OS に対して仮想割込みを発生させることができる．Distributor に関しては仮想化されていない．そのため，割込みコントローラを仮想マシン上から扱うためには，Distributor へのアクセスをハイパーバイザにより制御する必要がある．

### 2.2 組込みシステムにおける仮想化

クラウド環境などの汎用システムでは，すでに仮想化技術は広く普及している．汎用システムで使われているハイパーバイザはすでに ARM の VE を利用した実装も存在する．

しかし，文献 [7] で述べられているように，組込みシステムと汎用システムでのハイパーバイザのユースケースは異なる．文献 [7] で述べられているユースケースは以下の通りである．

(1) 以前から使っているソフトウェアとより機能が多い OS を同時に使うため
(2) 汎用 OS と RTOS を同時に使うため
(3) 機能ごとの独立性を高め，安全性を高めるため

また，リアルタイムシステムにおいての問題についても知られている [4]．例えば，複数の異なる種類の OS を動かす場合，それぞれの OS でスケジューラを持つが，どのようなタスクをスケジュールして実行しているかはハイパーバイザが取得するのは容易ではない．様々な解決方法が提案されているが，どれもトレードオフが存在する．また，仮想化に限らず，マルチコア環境におけるリアルタイムシステムにおいては最適なスケジューリングアルゴリズムが存在しないことが証明されており，そのため，様々なスケジューリングが提案されている [8]．

ⓒ 2015 Information Processing Society of Japan

## 3. 設計

本研究では組込み向けのアーキテクチャであり，完全仮想化が実現可能である ARMv7-A を対象にハイパーバイザを構築する．組込みリアルタイムシステムに焦点を絞り，必要な機能とその実現方法を検討する．

### 3.1 ハイパーバイザのアーキテクチャ

ハイパーバイザのアーキテクチャとして Type-1 と Type-2 が知られている．Type-1 型のハイパーバイザとはハイパーバイザがハードウェア上で直接動作し，ゲスト OS がその上で動くというものである．Type-2 型はホスト OS 上でハイパーバイザが動作し，その上でゲスト OS が動作するものである．

Type-2 型の設計はホスト OS の機能を利用してハイパーバイザを実装できるため，様々な機能を容易に実装でき，ハードウェア間の移植性も高い．しかし，ホスト OS を介しての動作となるため各種動作へのオーバーヘッドが懸念され，スケジューリング方式もホスト OS のスケジューラの影響を受けうる．また，ホスト OS に機能を依存するためホスト OS を Trusted Computing Base（TCB）に含める必要がある．よって，本研究ではリアルタイム性を高く保つため，また TCB を小さくして信頼性を向上させるために Type-1 型のハイパーバイザを構築する．

### 3.2 スケジューリング方式

先述の通り，リアルタイムシステムにおける仮想化には様々な課題がある．本研究では，スケジューリング方式はアプリケーションに強く依存すると考え，特定のスケジューリング方法を実装するという方法を取らず，ユーザー側ができるだけ自由にスケジューリング方式を定められるような設計にした．これにより，ユーザーがスケジューリング方式に関する様々な解決策を採用できることが期待される．

### 3.3 デバイスの扱い

ARM アーキテクチャにおいて，外部デバイスへの操作は Memory Mapped IO（MMIO）として実現される．よって，仮想マシンからのデバイスへのアクセスは Stage-2 の変換テーブルの設定により制限することができる．複数の仮想マシンから同時にデバイスがアクセスできるようにするため，ハイパーバイザが仮想のデバイスドライバを持ち，仮想マシンに対して仮想のデバイスモデルを提供するという方法が汎用ハイパーバイザでは用いられている．しかし，このような方法はデバイスのアクセスに対してオーバーヘッドを生じるため，リアルタイム性に影響を与える可能性がある．また，ハイパーバイザにデバイスドライバと仮想デバイスモデルを新たに実装するコストも発生し，TCB の肥大化にもつながる．

本研究では，基本的に各デバイスは 1 つの仮想マシンからしかアクセスされないと考え，デバイスを直接操作することを許す．ただし，仮想マシンごとに決まったデバイスしかアクセスできないよう，Stage-2 での変換テーブルを適切に設定する．もし，デバイスが複数の仮想マシンから扱いたい場合は，仮想マシン間の通信を用い，必要なインターフェースを対象のデバイスにアクセスを許された仮想マシンが担当する．

## 4. ハイパーバイザの実装

本研究では，Type-1 型のハイパーバイザを新たに設計した．コード量を減らすため，libc などのライブラリは利用せず，汎用 OS にも依存しない．ハイパーバイザは ARMv7-A で VE を備えたプロセッサでの実装に特化した．実装言語は C 言語とアセンブラを用いている．

ハイパーバイザの起動のため，U-Boot[9] をブートローダーとして使っている．U-Boot により RAM を初期化し，ハイパーバイザとゲスト OS を特定のメモリに置く．その後，ハイパーバイザを Non-Secure PL2 モードで開始させる．U-Boot は起動時のみ用いられ，ハイパーバイザが開始された後はその機能は全く使われない．

ハイパーバイザは Non-Secure PL2 モードで動作し，ゲスト OS は Non-Secure PL1 と PL0 で動作する．Secure 状態は使わない．PL2 モードは独立したシステムレジスタを持つため，PL1 モード上での殆どの命令はトラップする必要がない．LPAE による Stage-2 の変換テーブルを利用することで，ゲスト OS が実際にアクセスできるメモリ領域を制限できる．Stage-2 の変換テーブルに関するシステムレジスタは PL2 モードからしかアクセスできないため，ゲスト OS は Stage-2 変換テーブルを変更することはできない．Stage-1 の変換テーブルはゲスト OS が自由に扱うことができる．

デバイスのアクセスに関して，ハイパーバイザではデバイスドライバを持たず，仮想マシンが専有するのは前述の通りである．ただし，割込みコントローラに関しては，すべての仮想マシンが使うため，仮想化したものを提供している．

スケジューリング方式をユーザーが自由に定められるようにするため，スケジューリングのための API を定めて，それをユーザーが実装するという方式をとる．API は RTEMS[10] のものを参考にした．API の一覧は表 1 にまとめた．

仮想マシン間の通信として共有メモリと仮想割込みを用いる．共有メモリは Stage-2 の変換テーブルによってページ単位で仮想マシンに割り当てられる．この共有メモリがどの IPA に割り当てられるかは，起動時に予め定めておく．仮想割込みは，仮想マシンからのハイパーコールを受けたハイパーバイザが GIC の仮想 CPU インターフェー

表 1　スケジューラ用 API

| 関数名 | 役割 |
|---|---|
| init | 初期化を行なう．起動時に一度だけ呼ばれる． |
| schedule | 次実行する仮想マシンを決定する．仮想マシンの状態が変化した場合に実行される． |
| block | 実行中の仮想マシンが他の仮想マシンに切り替わる場合実行される． |
| yield | 実行中の仮想マシンが実行を中止し，待ち状態に切り替わる時実行される． |
| allocate | 仮想マシンを初期化する時，スケジューリング用のパラメータを設定する． |
| enque | 仮想マシンを実行可能状態にする時実行される． |

表 2　ベンチマークプログラムの実行時間．括弧内はハイパーバイザの起動時間を含めた実行時間

| アプリケーション | ベアメタル環境 | ハイパーバイザ |
|---|---|---|
| 起動時間 | 22.56 msec | 15.77 msec (159.8 msec) |
| 相互通信 | 1.009 sec | 1.432 sec (1.575 sec) |

スを操作することで発生させる．この仮想割込みによって仮想マシン間の同期をとる．共有メモリ以外の領域は変換テーブルによって他の仮想マシンからはアクセスできないため，仮想割込みも，実際に発行するのはハイパーバイザであり，発生できる割込み ID は予め制限されている．よって， この機構が仮想マシンの独立性を損なうことはない．

## 5. 評価

本研究で実装したハイパーバイザを RTOS を用いたテストプログラムにより評価した．評価実験は RTOS として T-Kernel 2.0[11] を，評価ボードとして TWR-LS1021A（Cortex-A7 MPCore，1GB RAM）[12] を用いた．マルチコアの環境ではあるが，本研究では 1 つのコアのみを用い，シングルコアの環境として用いた．すべての実験の結果は表 2 にまとめた．

### 5.1 起動時間

起動時のオーバーヘッドを調べるため，T-Kernel の開発者のアプリケーションを起動させる *usermain* 関数が呼び出されるまでの時間を比較した．今回の実験では U-Boot の実行時間は含めていない．U-Boot がハイパーバイザを呼び出し *usermain* 関数を呼び出すまでの時間と，U-Boot から T-Kernel を呼び出し *usermain* を呼び出すまでの時間をそれぞれ 10 回計測し，その平均を比較した．その結果を表にまとめた．ハイパーバイザを含めた起動時間は 0.16 秒で，これは T-Kernel を直接起動させた場合の 7 倍の時間である．しかし，ハイパーバイザが T-Kernel を呼び出してから *usermain* 関数が呼び出されるまでの時間は T-Kernel を直接起動させた場合より短い．このため，遅延の原因はハイパーバイザの介在よりも，ハイパーバイザそのものの起動時間によるものと考えられる．

### 5.2 コミュニケーションのオーバーヘッド

仮想マシン間のオーバーヘッドを調べるため，2 つの T-Kernel をハイパーバイザ上で動作させ，互いにメッセージの受信と送信を繰り返すアプリケーションの実行時間を調べた．メッセージは互いに 1000 回送信する．ベアメタル環境のアプリケーションとして，1 つの T-Kernel 上でプロセスを 2 つ起動し，そのプロセス間で同様にメッセージの受信と送信を繰り返すものを用意した．ハイパーバイザでの実行時間は，ハイパーバイザの起動時間を含めても，ベアメタル環境の 1.5 倍ほどの実行時間である．オーバーヘッドの要因として，仮想マシン間の切り替えやメッセージ送信のために仮想割込みを発生させる必要があることなどが挙げられる．

表 3　ハイパーバイザのコード数

| C 言語 | 3856 行 |
|---|---|
| アセンブラ | 897 行 |
| 合計 | 4753 行 |

### 5.3 コードサイズ

今回実装したハイパーバイザのソースコードの行数を SLOCCount[13] を用い計測し，表 3 にまとめた．アプリケーション依存のコーディング部分を除いた合計のコードサイズはおよそ 5000 行である．汎用ハイパーバイザの 1 つである Xen[14] も同様に SLOCCount で計測した．バージョンは 4.5.1 を用いた．アーキテクチャ依存のコードが含まれていない xen/common のディレクトリ内だけでも 40587 行であり，これに比べると本研究のハイパーバイザは十分に小さいと考えられる．

### 5.4 スケジューリング

本研究では，固定優先度スケジューラと Earliest Deadline First（EDF）スケジューラ，ラウンドロビンスケジューラに関してハイパーバイザ上で実装した．固定優先度スケジューラは，予め定めた優先度に基づき仮想マシンをスケジューリングし，自分より高い優先度の仮想マシンがアクティブになるとプリエンプションが発生する．EDF スケジューラは各仮想マシンに実行周期を定めておき，これを元に締め切りを設定し，締め切りが早い方に高い優先度を割り当てた．ラウンドロビンスケジューラは，仮想マシンには特に優先度を定めず，仮想マシンの実行が wfi 命令などにより停止するか，予め定めておいた実行周期が来るご

とに別の仮想マシンに切り替えるものである．それぞれテストプログラムがすべてのスケジューラにおいて正常に動作することを確認した．

## 6. 今後の展望

本研究の評価として，現在は複数の RTOS を動かすことに成功している．今後は汎用 OS と RTOS を同時に動作させ評価実験を行なう予定である．また，現在の実装はシングルコアのものなので，マルチコアにおいてのスケジューリング API やその他の設計についての検討を行う必要がある．

### 6.1 Unikernel の利用

組込みシステム上で汎用 OS を動かすというのは負担が大きい．そこで，ハイパーバイザ上で動作させる OS として Unikernel[15] の利用を検討している．Unikernel はクラウド環境上で動作させる OS の形として考案されたものであり，単一のアプリケーションをハイパーバイザ上で動かすためのものである．

Unikernel では生成できるプロセスは 1 つのみであり，OS の機能も単一のアドレス空間で動作するように設計されている．そのため，従来の汎用 OS にある各種のオーバーヘッドが削減でき，メモリの消費量も少ない．また，利用する機能のみをリンクさせることっでバイナリサイズも小さくできる．

これらの特長は組込みアプリケーションでも有益なものである．Unikernel はクラウド環境でのアプリケーションとしての設計が主であるため，ネットワーク処理に長けており，RTOS の機能を補完することができると考えている．

Unikernel の実装は現在，汎用のハイパーバイザ上の仮想デバイスを前提として動いているため，デバイスドライバを新規に実装する必要がある．しかし，汎用 OS のデバイスドライバの実装を利用することで，ハードウェア上で直接 Unikernel を利用するというものがある [16]．同じような方式で本研究でのハイパーバイザ上でも Unikernel の動作が実現可能だと考えている．

本研究のハイパーバイザはリソースの消費量が少なくなるように設計されている．多くの RTOS もリソースの消費量が少なくなるように設計されている．これに加え，リソースの消費量が少ない Unikernel を用いることで高度なアプリケーションをよりリソースの限られた組込みプロセッサでも実現できると考えている．一般的に Unikernel の API は汎用 OS とは異なるため，アプリケーションを書き直す必要がある．しかし，汎用 OS も従来通り使うことができるため，パフォーマンスやリアルタイム性などを考慮し，必要な部分のみ Unikernel のアプリケーションとして移植し，そうでない部分は従来通り汎用 OS 上で動作させるということもできる．

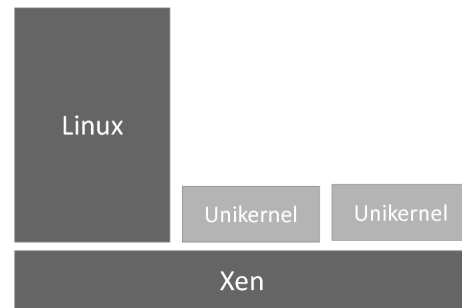


図 2　Jitsu におけるシステム構成

### 6.2 想定されるユースケース

本システムは仮想化支援機構が利用できる組込みプロセッサ上のシステムならば幅広く活用できると考えている．いくつか例を挙げる．まず，従来の組込み機器に対してネットワークからのインターフェースを提供したいという場合である．従来の組込み機器は多くは RTOS 上でアプリケーションが組まれていると考えられるが，ネットワーク接続に対応したものは少ない．本研究のハイパーバイザを用いることで，Unikernel によりそのようなインターフェースを提供し，実際の処理は従来と変わらず RTOS で実行すれば少ないコストでネットワーク接続の機能を追加できる．

また，小型飛行ロボットのようなシステムに対しても有効と考える．小型飛行ロボットはソフトウェアのリアルタイム性と信頼性がともに重要である．その一方で，ネットワーク経由で高度な処理を依頼することが多い．一部の画像処理のような高度な情報処理はネットワークの遅延を避けるために，そのロボット内で行なうということも考えられる．しかし，積載できる容量に厳しい制限があるため，ハードウェアをシンプルにしたいという要求がある．ハイパーバイザを用いれば，制御のための RTOS を動かすためのボードとネットワーク処理を行なう OS のハードウェアを分離することなく，信頼性を高めることができる．ロボット上に信頼性が低いユーザーの組んだアプリケーションを搭載しても，RTOS 側の独立性を保てば，ソフトウェア的に緊急停止する機能を実装することで現実の被害を軽減することができる．

## 7. 関連研究

従来，組込みプロセッサ上で複数の OS を動かすためにはハイパーバイザの場合，文献 [1] のような準仮想化方式を用いるものが多かった．しかし，準仮想化は OS そのものに変更を加える方式であるため，OS の信頼性を下げる．ハイパーバイザ以外の複数の OS を利用する手法として SafeG[2] などが挙げられるが，これも OS への変更を伴うものである．

**図 3** 従来のハイパーバイザにおけるシステム構成

**図 4** 本研究が目指すシステム構成

汎用プロセッサ上で動くハイパーバイザである KVM[3] や Xen[14] はすでに ARM の VE を利用した実装を利用できる．しかし，これらはもともと汎用プロセッサ向けにつくられているため，規模が大きくオーバーヘッドの大きさが懸念される．また，Linux に機能の一部を依存しているため，Linux の脆弱性がハイパーバイザの脆弱性に繋がりうる．

組込み向けにつくらたハイパーバイザでも，ARM VE に対応したものが存在する [17]．これはマイクロカーネル型の OS をベースにつくられたものであり，設計の指針が異なる．また，リアルタイムなスケジューリングについては検討されていない．

仮想マシン上の OS がデバイスに対して直接アクセスすることを許すものとして，BitVisor[18] や Quest-V[19] が挙げられる．前者は 1 つの OS のセキュリティを高めるためにつくられたもので，目的が異なる．後者は汎用プロセッサ向けにつくられているものの，目的の大部分を共有する．しかし，スケジューリング方式は単一のもので，ハイパーバイザの設計が異なる．

OS を複数搭載するのではなく，単一の OS 内で仮想化した環境を提供する手法としてコンテナ型仮想化が存在する [20]．コンテナ型仮想化を組込みシステムで用いたものとして，Cells[21] が存在する．コンテナ型仮想化は仮想マシンのサイズを抑えることができ，動作時のオーバーヘッドもハイパーバイザよりも少ないことが知られている．また，特殊なハードウェア機構も必要としないため，本研究の手法より広い範囲での適用が可能である．しかし，コンテナ型仮想化は仮想マシンの OS がホスト OS と同一のものに限定されてしまう．また，汎用 OS 上で実現されているため，TCB も大きくなる．ただし，コンテナ型仮想化はハイパーバイザ上での汎用 OS でも利用可能である．

組込みシステムに対して，Unikernel を用いた例として Jitsu[22] が存在する．これは Xen のツールをつくることで，Unikernel を効率よく起動させ Web サービスを提供するというものである．しかし，Xen は Linux に依存するハイパーバイザであり，TCB は大きくなる．

本研究のコンセプトと近いものとして文献 [23] が存在する．これは実際に実装されたものではないが，提案されたアーキテクチャは本研究が提案されたものに近い．この研究で提案されている Nonkernel が本研究でのハイパーバイザに相当し，仮想マシンが Unikernel に相当する．しかし，この研究で提案されているアーキテクチャはクラウド環境を想定しているものである．そのため，リアルタイムなスケジューリングは考慮されていない．

## 8. おわりに

本研究では，仮想化支援機構が利用できる組込みプロセッサを利用して，リアルタイムシステムに適したハイパーバイザの構築と，その運用方法の提案を行った．現在提案されている様々なリアルタイムシステム向けのスケジューリング方法を実現できるよう，ハイパーバイザは特定のスケジューリング方式を実装するのではなく，ユーザーが API を実装する形式にした．デバイスドライバも内部に持たず，仮想マシンが直接操作する形式をとった．結果，異なるスケジューリングポリシが実装できることが確認でき，オーバーヘッドや実装量も少なく済んだ．また，運用方法として，1 つのプロセスのみを動かすことに特化した Unikernel の利用を提案した．

本研究では，ハイパーバイザで動作させているのは RTOS のみである．また，シングルコア環境のみの対応となっている．マルチコア環境対応や他の種類の OS の動作は今後の課題である．そのうえで，他のハイパーバイザとの比較実験を行い，本研究で提案したアーキテクチャの有用性について検証する予定である．

### 参考文献

[1] Heiser, G. and Leslie, B.: The OKL4 Microvisor: Convergence Point of Microkernels and Hypervisors, *Proceedings of the First ACM Asia-pacific Workshop on Workshop on Systems*, APSys '10, New York, NY, USA, ACM, pp. 19–24 (online), DOI: 10.1145/1851276.1851282 (2010).

[2] 中嶋 健一郎，本田 晋也，手嶋 茂晴，高田広章：セキュリティ支援ハードウェアによるハイブリッド OS システムの高信頼化 (リアルタイムシステム)，情報処理学会研究報告.

EMB, 組込みシステム，Vol. 2008, No. 116, pp. 1–7（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110007099117⟩ (2008).

[3] Christoffer Dall and Nieh, J.: KVM/ARM: The Design and Implementation of the Linux ARM Hypervisor, *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, New York, NY, USA, ACM, pp. 333–348 (online), DOI: 10.1145/2541940.2541946 (2014).

[4] Zonghua Gu and Qingling Zhao: A State-of-the-Art Survey on Real-Time Issues in Embedded Systems Virtualization, *Journal of Software Engineering and Applications*, Vol. 05, No. 04, pp. 277–290 (online), DOI: 10.4236/jsea.2012.54033 (2012).

[5] ARM: *ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition* (2014).

[6] Popek, G. J. and Goldberg, R. P.: Formal Requirements for Virtualizable Third Generation Architectures, *Commun. ACM*, Vol. 17, No. 7, pp. 412–421 (online), DOI: 10.1145/361011.361073 (1974).

[7] Paulo Garcia, Gomes, T., Salgado, F., Monteiro, J. and Tavares, A.: Towards Hardware Embedded Virtualization Technology: Architectural Enhancements to an ARM SoC, *SIGBED Rev.*, Vol. 11, No. 2, pp. 45–47 (online), DOI: 10.1145/2668138.2668145 (2014).

[8] Robert I. Davis and Burns, A.: A Survey of Hard Realtime Scheduling for Multiprocessor Systems, *ACM Comput. Surv.*, Vol. 43, No. 4, pp. 35:1–35:44 (online), DOI: 10.1145/1978802.1978814 (2011).

[9] : WebHome < U-Boot < DENX, `http://www.denx.de/wiki/U-Boot`.

[10] Gedare Bloom and Sherrill, J.: Scheduling and Thread Management with RTEMS, *SIGBED Rev.*, Vol. 11, No. 1, pp. 20–25 (online), DOI: 10.1145/2597457.2597459 (2014).

[11] : T-Kernel 2.0, `http://www.tron.org/download/index.php?route=product/category&path=37`.

[12] : QorIQ LS1021A Tower System Module—Freescale, `http://www.freescale.com/tools/embedded-software-and-tools/hardware-development-tools/tower-development-boards/mcu-and-processor-modules/powerquicc-and-qoriq-modules/qoriq-ls1021a-tower-system-module:TWR-LS1021A`.

[13] : Counting source lines of code (sloc), `http://www.dwheeler.com/sloc/`.

[14] : Xen ARM with Virtualization Extensions whitepaper - Xen, `http://wiki.xenproject.org/wiki/Xen_ARM_with_Virtualization_Extensions_whitepaper`.

[15] Madhavapeddy, A., Mortier, R., Rotsos, C., Scott, D., Singh, B., Gazagnaire, T., Smith, S., Hand, S. and Crowcroft, J.: Unikernels: Library Operating Systems for the Cloud, *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '13, New York, NY, USA, ACM, pp. 461–472 (online), DOI: 10.1145/2451116.2451167 (2013).

[16] Cormack, J.: The rump kernel: A tool for driver development and a toolkit for applications, `https://www.netbsd.org/gallery/presentations/justin/2015_AsiaBSDCon/justincormack-abc2015.pdf`.

[17] : Genode - Genode Operating System Framework, `http://genode.org/`.

[18] Takahiro Shinagawa, Hideki Eiraku, Kouichi Tanimoto, Kazumasa Omote, Shoichi Hasegawa, Takashi Horie, Manabu Hirano, Kenichi Kourai, Yoshihiro Oyama, Eiji Kawai, Kenji Kono, Shigeru Chiba, Yasushi Shinjo and Kazuhiko Kato: BitVisor: A Thin Hypervisor for Enforcing I/O Device Security, *Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '09, New York, NY, USA, ACM, pp. 121–130 (online), DOI: 10.1145/1508293.1508311 (2009).

[19] Li, Y., West, R. and Missimer, E.: A Virtualized Separation Kernel for Mixed Criticality Systems, *Proceedings of the 10th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '14, New York, NY, USA, ACM, pp. 201–212 (online), DOI: 10.1145/2576195.2576206 (2014).

[20] Stephen Soltesz, Ptzl, H., Fiuczynski, M. E., Bavier, A. and Peterson, L.: Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors, *Proceedings of the 2Nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, New York, NY, USA, ACM, pp. 275–287 (online), DOI: 10.1145/1272996.1273025 (2007).

[21] Andrus, J., Dall, C., Hof, A. V., Laadan, O. and Nieh, J.: Cells: A Virtual Mobile Smartphone Architecture, *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, New York, NY, USA, ACM, pp. 173–187 (online), DOI: 10.1145/2043556.2043574 (2011).

[22] Madhavapeddy, A., Leonard, T., Skjegstad, M., Gazagnaire, T., Sheets, D., Scott, D., Mortier, R., Chaudhry, A., Singh, B., Ludlam, J., Crowcroft, J. and Leslie, I.: Jitsu: Just-In-Time Summoning of Unikernels, *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, Oakland, CA, USENIX Association, pp. 559–573 (online), available from ⟨https://www.usenix.org/conference/nsdi15/technical-sessions/presentation/madhavapeddy⟩ (2015).

[23] Muli Ben-Yehuda, Peleg, O., Agmon Ben-Yehuda, O., Smolyar, I. and Tsafrir, D.: The Nonkernel: A Kernel Designed for the Cloud, *Proceedings of the 4th Asia-Pacific Workshop on Systems*, APSys '13, New York, NY, USA, ACM, pp. 4:1–4:7 (online), DOI: 10.1145/2500727.2500737 (2013).

© 2015 Information Processing Society of Japan