Vol.2017-OS-140 No.10
2017/5/17

マルチコアにおけるリアルタイム性と CPU 使用効率を両立した組込みシステム向け VMM の開発

内匠真也[†1]　磯崎宏[†1]　橋本幹生[†1]　金井遵[†1]

概要：組込み機器においてマルチコア環境で複数の OS を動作させる場合，各 OS がコアを占有することでリアルタイム性を維持する方式が用いられる．しかし，OS 間で負荷の不均衡がある場合には CPU 使用効率が低下するため，本稿ではコアを共有しつつ，リアルタイム性を維持する VMM を提案する．本方式ではコアごとの動作を優先する OS(優先 OS)の設定に基づいて，優先 OS の動作中に発生した非優先 OS 向けの割込みをペンディングし，非優先 OS によるコアの横取りを防止する．コアごとに異なる優先 OS を設定するとともに，OS 側ではリアルタイムタスクを当該 OS の実行が優先されるコアに割り付けることで，複数の OS 上で動作するタスクのリアルタイム性を維持できる．評価では他 OS の負荷により割込み応答時間が遅延しないことを実機上で確認し，リアルタイム性の維持を確認した．

キーワード：VMM，仮想化，マルチコア，組込みシステム，LiSTEE™，TrustZone®

1. はじめに

FA(Factory Automation)や制御機器等の組込みシステムに対し，セキュリティと信頼性を向上する手段として，仮想化技術が注目されている [1]．仮想化技術では，1 つのシステム上に複数 OS の並行動作を可能とし，かつ，OS を管理する VMM(Virtual Machine Monitor)が各 OS を適切に分離・制御することから，複数の OS を実行しつつ OS 間のアクセス制御を実現する．たとえ，ソフトウェアのバグや悪意のある攻撃により特定の OS が意図しない挙動を行った場合であっても，VMM が適切に資源分離を行って他の OS への影響を遮断するため，各 OS のセキュリティや信頼性を向上できる．また，VMM を介して隔離した OS 間の情報共有により，OS を隔離しつつ，安全で高速な OS 間の通信機能を実現できる．

組込みシステムに対して仮想化技術を適用する場合の大きな課題の 1 つに，リアルタイム性と CPU 使用効率の向上の両立が挙げられる．リアルタイム性を維持するためには，実行時に計算，もしくは実行前に設定されたタスクの優先度に基づいてタスクをコアに割り当てる必要がある．一方，CPU 使用効率を向上させるためにはできる限り複数のタスクでコアを共有する必要がある．CPU 使用効率を向上させるために各 OS が CPU 内のコアを共有した場合，割込みにより起床した OS によるコアの横取りが発生し，システム全体では優先度を維持できない恐れがある．例えば，コアを割り当てられていない OS が割込みにより起動し，最優先のタスクを実行する OS からコアを横取りするといった事例が考えられる．そこで，リアルタイム性を必要とする組込みシステム向けの仮想化技術として，OS 間でコアの横取りが発生しないように，各 OS にコアを占有させる方式(以降この方式を OS パーティション方式と呼ぶ)が提案されている [2, 3] ．しかし，この方式では OS 間で負荷の不均衡がある場合に CPU 使用効率が低下するといった課題がある．

この課題を解決するために，我々はマルチコアが利用可能な環境下で，リアルタイム性と CPU 使用効率の両立を実現する LiSTEE™ for Real Time system (以下，LiSTEE RT と呼ぶ)を提案する．LiSTEE RT ではコアごとに設定された動作を優先する OS(以下，優先 OS と呼ぶ)に基づき，優先 OS の動作中に発生した非優先 OS 向けの割込みをペンディングする．この方式により，システム設計者はコアごとに異なる優先 OS を設定するだけで，複数の OS 上で動作するリアルタイムタスクのリアルタイム性を維持できる．また，実行可能な優先 OS のリアルタイムタスクが存在しない状態のコアにおいて，優先 OS に加え非優先 OS のノンリアルタイムタスクも実行することで，システム全体の CPU 使用効率が向上する．また，LiSTEE RT は OS 自体の改変を必要とせず，設計によっては VMM の改変も少ないため，保守性が高いという特徴も備える．

本稿では LiSTEE [4]をベースとして LiSTEE RT を構築し，仮想化の実現には ARM TrustZone® [5]を利用する．

本稿では 2 章で我々が達成すべき要件，3 章で関連研究について述べる．4 章では LiSTEE RT の設計，5 章で実装，6 章で評価結果について述べる．7 章では提案手法について考察を行い，8 章でまとめる．

2. 組込みシステム向け VMM の要件

複数の OS のスケジューリングする組込みシステム向け VMM に対し，本稿が達成を目指す要件についてまとめる． また，本稿では VMM を利用して，OS やタスクを設計・実装し，システム全体を構築する人をシステム設計者と定義する．

要件① リアルタイム性: 組込みシステムではタスクの実行遅延がシステムに深刻な被害を与える恐れがあるため，リアルタイム性は重要となる．割込みにより OS が起床し

†1 (株) 東芝

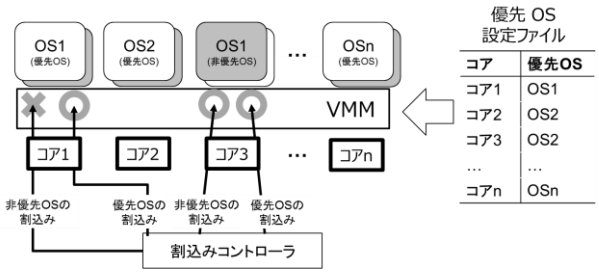

図 1 LiSTEE RT による割込み制御の概要

表 1 動作 OS に基づく割込みの制御

動作 OS	割込み発生元	割込みの入力
優先 OS	優先 OS	許可
優先 OS	非優先 OS	保留
非優先 OS	優先 OS	許可
非優先 OS	非優先 OS	許可

た場合, OS 間でコアの割当てが競合する．これはタスクの優先度を無視したコアの割当てを引き起こす危険性があり，リアルタイム性を阻害する要因となる．

要件② CPU 使用効率: 組込みシステムは省電力，小型化，低コスト化のため，計算リソースの最小化が求められる．一方で，IoT(Internet of Things)化に伴い，組込みシステムの処理は増大傾向にある．限られた計算リソースを有効活用するためには CPU の使用効率を高める必要がある．

要件③ 保守性: 複雑な VMM 機構や OS の変更を必要とする場合，システムの保守性が低下する．保守性が低下すると，バグの見逃しや適切なメンテナンスが困難となるため，システムに損害を与える恐れが高まる．保守性を維持するためには VMM の複雑な実装や OS の変更を避ける必要がある．

要件④ システム開発コスト: VMM の制約により OS やタスクに設定が必要な場合, システム全体の構築が複雑化し，開発コストが増大する．そのため, OS やタスクの設定を行うシステム設計者への負担を抑える必要がある．

3. 関連研究

組込みシステム向けに OS のリアルタイムスケジューリング方式の研究が行われている．

航空機向けソフトウェアの規格 ARINC 653 は OS またはタスク群のスケジューリング方法を定義しており [6]，この方式を搭載する組込みシステム向け VMM も存在する [7, 8]．ARINC 653 において，システム設計者はシステム周期とその周期中に対し OS への割当て時間を設定する．さらに，割り当て時間内に OS はタスクをスケジューリングし，実行する．システム設計者は実行したい処理のリアルタイム性に応じてシステム周期と各 OS の割当て時間を設定するため，システム全体ではリアルタイム性を維持できる．しかし，この方式は各 OS の最大処理量に合わせて割当て時間を決定する必要があるため，CPU の使用効率が低下しやすい．また, OS に時間が割り当てられるまで処理が遅延するため，ユーザの情報入力などの応答性が求められる非周期的な処理に向いていない．

汎用 OS・RTOS の並行動作を想定した VMM では RTOS を優先実行する方式が提案されている [9]．この方式を RTOS 優先方式と呼ぶ．また，LiSTEE もこの方式を採用しており，Secure World 上の OS の動作を優先する [4]. RTOS 優先方式では優先する OS(RTOS)を一つに定め, RTOS 動作時の他 OS への割込みはペンディングし，他 OS 動作時の RTOS の割込み発生時には RTOS にすぐさま遷移する．この方式では RTOS のリアルタイム性は維持できるが，他の OS は強制的にコアを横取りされるため，リアルタイム性を維持できない．すなわち，複数 OS 上のタスクのリアルタイム性を維持したい要求に適さない．

階層型フラットスケジューリング方式は OS が通知したタスクの優先度を基に VMM がスケジューリングを行う [10, 11]．つまり，OS はスケジューリング結果やタスクの状態を VMM に通知し，VMM は各 OS が通知した情報を基にスケジューリングを行う．結果として，VMM が全てのタスクを優先度に基づいてスケジューリングするため，リアルタイム性を維持できる．また，VMM がタスクに対して柔軟なコア割当てを行うため，CPU 使用効率の向上が期待できる．しかし，この方法では OS の改変が必要となり，スケジューリング機能の実装により VMM の実装が複雑になるため，保守性が低下する．また，VMM によるスケジューリングのオーバヘッドはリアルタイム性や CPU 使用効率に影響を与える恐れがある．

VMM の特定の OS にタスクのスケジューリングを集約する方式が提案されている [12]．スケジューリングを集約する OS をスケジューリングサーバと呼び，この提案方式をスケジューリングサーバ方式と呼ぶ．この方式ではスケジューリングサーバが疑似的に全てのタスクのスケジューリングを行い，タスクの優先度を無視した割込みを防止することで，リアルタイム性の維持を実現する．また，スケジューリングサーバがタスクにコアを柔軟に割り当てることで，CPU 使用効率を向上できる．一つの OS にスケジューリングを任せるため，既存の OS のスケジューリング機能を流用できる利点がある．しかし，階層型フラットスケジューリング方式と同様に, OS に改変が必要であり，実装が複雑になりやすく，OS のスケジューリング処理にオーバヘッドが発生する．

4. 提案手法

本章ではリアルタイム性と CPU 使用効率の両立を実現する LiSTEE RT の設計・機能について述べる．

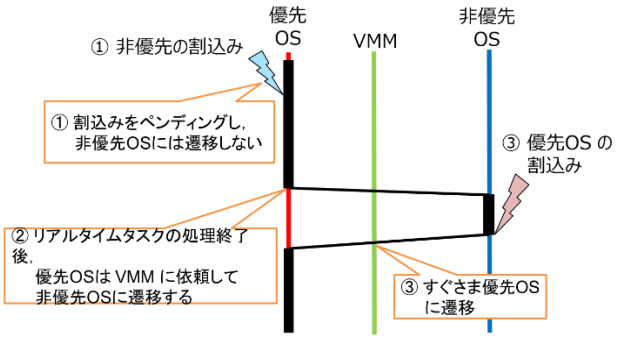

図 2 LiSTEE RT における OS の切替え

4.1 前提条件

本節では，提案手法を利用するための前提条件についてまとめ，以下に前提条件について述べる．

前提① コア数は 2 個以上: 我々はマルチコアを利用してリアルタイム性と CPU 使用効率の両立を実現する．そのため，システムは 2 個以上のコアの利用を想定する．

前提② VMM 上に動作する OS の事前決定: VMM 上に動作する OS の種別や数について，システム設計者は事前に決定する．

前提③ OS 間の通信における優先度の逆転が発生防止: 優先度の高いタスクが優先度の低いタスクの処理結果に依存する場合，優先度の逆転が生じる．OS 間で優先度の逆転が生じる場合，その解消は難しい．OS 間の通信において，システム設計者は処理依頼先のタスクの優先度を依頼元以上の優先度に設定することで，優先度の逆転を防止できる．

4.2 概要

LiSTEE RT はコアごとに設定された優先 OS に基づき，優先 OS のコアを横取りする割込みを防ぐことで，優先 OS 上に動作するタスクのリアルタイム性を維持する．優先 OS の割込みの制御方法は 4.3 節で述べる．

システム設計者はコアごとに異なる優先 OS を設定し，優先 OS 上にリアルタイムタスクを固定することで，複数 OS が持つリアルタイムタスクのリアルタイム性を維持できる．これを実現するための，タスクの動作コアを固定する CPU アフィニティ制御の詳細は 4.4 節で述べる．また，LiSTEE RT は 4.5 節で述べるコア間割込みを利用した通信機能により，OS 間通信のリアルタイム性を維持する．

システム設計者は処理中断可能なノンリアルタイムタスクが非優先 OS となるコアでも実行可能にすることで，組込みシステム全体の CPU 使用効率を向上できる．また，システム設計者は優先 OS に非優先 OS への遷移タスクを用意することで，当該タスクの実行時に非優先 OS は動作する．非優先 OS 切替え時に複数の非優先 OS が存在する場合，VMM はスケジューリング機能により次に実行する非優先 OS を選択する．適切なスケジューリング機能を実装

表 2 非優先 OS の遷移タスクの優先度と
優先 OS・非優先 OS 間のロードバランス

優先度	ノンリアルタイムタスクの ロードバランス	
	優先 0S	非優先 0S
アイドル	○	×
ノンリアルタイム	△	△
最低優先度 リアルタイム	×	○

することにより，非優先 OS 間の CPU 使用率と応答性を最適化できる．また，システム設計者は優先 OS 上に動作する非優先 OS への遷移タスクの優先度を設定することで，優先 OS のノンリアルタイムタスクと非優先 OS の CPU 使用率と応答性を最適化できる．優先 OS・非優先 OS 間の切替え制御について 4.6 節に述べ，VMM による非優先 OS のスケジューリングを 4.7 節に述べ，非優先 OS への遷移タスクの優先度設定による非優先 OS の CPU 使用率と応答性の調節を 4.8 節に述べる．

4.3 割込みの制御

LiSTEE RT の割込みの制御を図 1 に示す．LiSTEE RT ではシステム設計者が予め作成した優先 OS の設定ファイルに基づいて，コアの割当てを優先する OS を決定する．さらに，優先 OS のコアを横取りする割込みを防止するために，表 1 のように割込みを制御する．以下にその詳細について述べる．

- 優先 OS の動作時には優先 OS の割込みを許可するが，非優先 OS の割込みはペンディングする．
- 非優先 OS 動作時には非優先 OS の割込みを許可するとともに優先 OS の割込みも許可する．優先 OS の割込みが発生した場合，VMM は直ちに非優先 OS から優先 OS にコアの割当てを変更する．

上記割込み制御により特定 OS へのコア割当てを優先することで，その OS 上に動作するタスクのリアルタイム性を実現する．

OS への割込みを許可とペンディングか判断する方法は割込み発生ごとに VMM で判断する方法と OS 切替え時に事前の設定に基づいて HW で割込みの制御を変更する方法がある．

前者の場合，割込みを受け取った VMM は割込み発生元の OS を識別し，優先 OS の設定に基づき割込み入力の判定を行う．この方法では VMM が割込みの仮想化を行うため，OS 間で I/O 資源を共有できるが，割込み入力の判断のオーバヘッドによりリアルタイム性を損なう恐れがある．後者の場合，VMM が割込みコントローラの割込みマスク機能を使用して，OS 切替え時に HW レベルで割込みの入

力許可とペンディングを設定する．この方法では VMM による割込み入力の判断処理が行われないため，リアルタイム性を高く維持できるが，OS を識別した割込み入力の判断ができないため．OS 間で I/O 資源を共有できない．

組込みシステムではリアルタイム性を重視するため，HW により割込み入力の判断を行う方法を選択することが妥当と思われる．

4.4 OS 上でのアフィニティ制御

LiSTEE RT は割込みの制御により，優先 OS が使用中のコアの横取りを防止することで，優先 OS 上のタスクはリアルタイム性を維持する．そのため，システム設計者はコアごとに異なる優先 OS を設定するとともに，OS 側ではリアルタイムタスクを当該 OS の実行が優先されるコアに割り付けるように設定することで，複数の OS が持つリアルタイムタスクのリアルタイム性を維持できる．

システム設計者は処理中断可能なノンリアルタイムタスクを非優先コアに分散させることで，CPU 使用効率を向上する．各 OS のスケジューリング機能を利用することによりノンリアルタイムタスクに対する柔軟なコア割当てが期待できる．また，リアルタイム性を重視して，全てのノンリアルタイムタスクをリアルタイムタスクとは別のコアに固定する設計を選択することも可能である [13]．タスクの動作コアを指定する機能は多くの OS が持つ．例えば，Linux では，CPU Affinity 機能によりコアを固定でき，FreeRTOS ではタスク生成時に動作コアを固定できる．

4.5 OS 間の通信

コア間割込みを利用することにより，リアルタイム性を維持した OS 間の通信を実現する．LiSTEE RT は，優先 OS への割込み発生時にはすぐさま優先 OS が動作する．そのため，優先 OS 上に通信を受け取るタスクを動作させることで，コア間割込みにすぐさま反応し，他 OS の通信依頼をリアルタイムに処理できる．

4.6 OS の切替え制御

OS のコア割当てを優先するために LiSTEE RT は割込みによる OS の切替えを制御する．割込みを起点とした OS 切替え制御の様子を図 2 に示す，

処理①では優先 OS の動作時の非優先 OS への割込みをペンディングしている．そのため，優先 OS の状態を無視したコアの横取りは発生せず，優先 OS のリアルタイム性を維持できる．処理②では優先 OS は自ら VMM に依頼して，非優先 OS に遷移している．各コアの優先 OS が自ら遷移して非優先 OS のタスクを実行することで，CPU 使用効率の向上が可能となる．処理③では非優先 OS の動作時に優先 OS の割込みが発生しており，その割込みを VMM が受け取っている．VMM は優先 OS の処理が遅延しないように，すぐさま非優先 OS から優先 OS に切り替える．

優先 OS は非優先 OS にコアを横取りされず，動作が必要になったときにはすぐさま切り替わるため，リアルタイム性を維持できる．

4.7 VMM による非優先 OS のスケジューリング

LiSTEE RT では非優先 OS の実行も許可することで，CPU 使用効率を向上する．優先 OS から非優先 OS の切替え時，非優先 OS が複数存在する場合，次に実行する非優先 OS を選択するためのスケジューリング機能が VMM に必要となる．このスケジューリングは優先 OS が自ら非優先 OS の切替えを許可したときに行われるため，優先 OS の動作を妨げない．

スケジューリングは非優先 OS を対象に行うため，リアルタイム性を必要とせず，非優先 OS 間のロードバランスが必要となる．サーバ向けの既存のスケジューリング機能を実装することで，非優先 OS 間のロードバランスを最適化できるが，実装量の多いサーバ向けのスケジューリング機能は保守性の低下につながる恐れがある．そのため，ロードバランスと保守性のトレードオフを考慮したスケジューリング機能を選択・設計する必要がある．ちなみに，動作させる OS が 2 つの場合は実行させる非優先 OS が決定しているため，非優先 OS のスケジューリング機能を必要としない．

4.8 非優先 OS への遷移タスクの優先度設定による非優先 OS の CPU 使用率と応答性の調節

システム設計者は優先 OS に非優先 OS への遷移タスクをあらかじめ用意し，実行時に非優先 OS への遷移を可能にする．非優先 OS への遷移タスクについて優先度を設定することで，そのコア上の優先 OS・非優先 OS のノンリアルタイムタスクのロードバランスを調節できる．非優先 OS への遷移タスクの優先度とロードバランスの関係を表 2 に示し，以下に詳細を述べる．

非優先 OS への遷移タスクをアイドルタスクに設定: 優先 OS がアイドル状態になったときに非優先 OS に遷移する．これにより，優先 OS のノンリアルタイムタスクは高使用率で実行できるが，非優先 OS は優先 OS に実行可能なタスクが無くなるまで実行が遅延する．

非優先 OS への遷移タスクの優先度をノンリアルタイムに設定: 非優先 OS への遷移タスクをノンリアルタイムタスクと一緒にスケジューリングすることにより，非優先 OS は一般のタスクと同程度の実行時間を持つ．また，Linux の nice 値のような OS の機能により，ノンリアルタイムタスク内における非優先 OS への遷移タスクの優先度を設定できる場合，優先 OS と非優先 OS のロードバランスを調節できる．

非優先 OS への遷移タスクの優先度を最低優先度リアルタイムに設定: 動作可能なリアルタイムタスクが無くなったときに非優先 OS に遷移する．リアルタイムタスクが動作していない全ての時間を非優先 OS のノンリアルタイムタスクが使用できる．

5. 実装

LiSTEE は一つの OS を優先実行する方式を取っているため，複数の OS が持つリアルタイムタスクのリアルタイム性を維持できない．そこで，LiSTEE をベースとして LiSTEE RT を構築することで，この課題を解決する．

5.1 LiSTEE

LiSTEE は ARM のセキュリティ拡張機能 TrustZone を用いて仮想化を実現する．TrustZone では資源を Secure World と Non-Secure World に分割する．さらに，コアは Secure と Non-Secure の状態を持ち，コアが Non-Secure 状態のときには Secure World にアクセスできず，Secure 状態のときのみアクセスできる．これにより，Non-Secure World から Secure World の資源を保護できる. また, TrustZone は VMM が動作する Monitor モードを持ち，VMM は World 間の遷移，割込みの処理を行う．Monitor モードには SMC(Secure Monitor Call)命令により遷移可能であり，Non-Secure World は SMC 命令を実行することで，VMM を経由して Secure World と通信する．また，設定により，発生した割込みを Monitor モードが受け取るように設定できる．

LiSTEE の各 World には OS が動作し，資源とそれらを操作するタスクを管理する．LiSTEE では，Non-Secure World に Linux，Secure World に独自 OS (Secure OS)の動作が想定されており，Monitor モードに動作する LiSTEE Monitor がこれらの OS を調停する．LiSTEE Monitor は脆弱性の混入しやすい大規模実装の Linux から Secure OS の管理する資源を保護しつつ，OS 間で安全な通信機能を提供する．

5.2 割込み制御と OS 切替えの制御

ARM CPU には通常割込み(IRQ)と高速割込み(FIQ)が存在する．LiSTEE RT では Linux が IRQ を占有し，Secure OS が FIQ を占有する．つまり, IRQ をマスクすることで Linux の割込みのみをペンディングすることができ，FIQ をマスクすることで Secure OS の割込みのみをペンディングできる．また，IRQ・FIQ のマスクにより割込みコントローラが割込みのペンディングを行うため，VMM による割込み入力の判断処理を必要とせず，リアルタイム性をより高く維持できる．

今回の実装では 2 コア環境のみ対応し，コア 0 が Secure OS 優先，コア 1 が Linux 優先で固定する．LiSTEE Monitor は OS 切替え時に優先 OS の設定に基づいて，割込み(IRQ・FIQ)の入力先を制御する．実行中の OS は優先 OS・非優先 OS 問わず，その割込みをその OS に直接入力する．実行中ではない OS の割込みが発生した場合，非優先 OS の割込みの場合はペンディングし，優先 OS の割込みの場合は LiSTEE Monitor に入力されるように設定する．LiSTEE Monitor に割込みが入力された場合はすぐさま優先 OS に切り替え，受け取った割込みを優先 OS に入力する．

優先 OS が SMC を発行し，非優先 OS への遷移を依頼してきた場合，LiSTEE Monitor は非優先 OS を実行する．今回の実装では OS は 2 つしか動作しないため, 非優先 OS のスケジューリング機能は必要としない．

5.3 タスクの動作コアの固定

LiSTEE RT ではリアルタイム性を維持するために，各優先 OS 上に Secure OS と Linux のリアルタイムタスクを固定する必要がある．

Linux では CPU affinity 機能を利用してタスクの動作コアを固定する． Secure OS ではタスク設定ファイルが用意されており，この設定ファイルによりタスクの動作コアを固定できる．

6. 評価

LiSTEE RT のリアルタイム性と CPU 使用率の両立について評価を行う．評価環境には CortexA-9(2 コア，600MHz)を使用する. L1 キャッシュは 32KB, L2 キャッシュは 512KB, RAM は DDR3-SDRAM 512MB を持つ．また，Linux 3.4.5 を使用する．

6.1 割込み遅延時間

タスクは割込みを契機として動作するため，割込みの遅延時間はリアルタイム性に大きな影響を与える，本評価では LiSTEE RT 上の Linux の割込み遅延時間が Secure OS の負荷の影響を受けず，リアルタイム性を維持できることを確認する．さらに，LiSTEE RT 上の Linux，LiSTEE RT を適用していない Linux，従来の LiSTEE(RTOS 優先方式)上の Linux を比較する．また，今回の評価では LiSTEE RT，LiSTEE 共に文献 [4]の高速 OS 切替え手法は適用していない．

割込み発生からタスク起動までの割込み遅延時間を 1000 回計測する． Linux のコア 1(LiSTEE RT では Linux の動作が優先されるコア)に動作する割込み遅延時間計測タスクにより計測を行う．計測タスクはタイマにより起動用の割込みを設定し，実際にタスクが立ち上がった時間とタイマにセットした起動時間を引くことで遅延時間を計測する．LiSTEE RT，LiSTEE では Secure OS 上に周期 3ms，処理時間 300μs のビジーループを行うタスクを割込み遅延時間計測タスクと同じコアに動作させる．さらに，遅延時間

max	min	標準偏差	平均
19.84	8.32	0.77	10.35

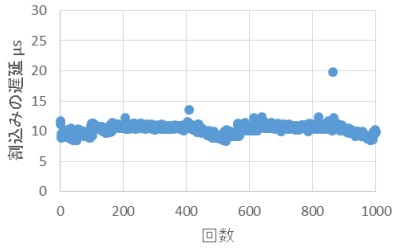

図 3 LiSTEE RT 上の Linux の割込み遅延時間

max	min	標準偏差	平均
25.92	7.6	0.748	9.463

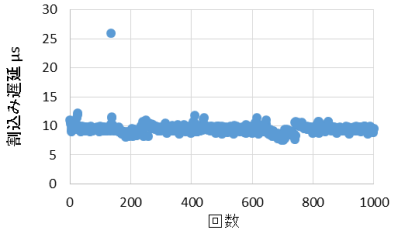

図 4 Linux の割込み遅延時間

max	min	標準偏差	平均
350.88	9.36	128.46	75.53

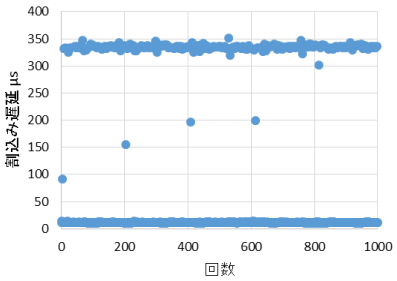

図 5 LiSTEE 上の Linux の割込み遅延時間

計測ごとに非優先 OS(Secure OS)へ遷移し，遷移後 Secure OS は待機する．

LiSTEE RT 適用時の評価結果を図 3 に，LiSTEE 適用時の評価結果を図 4，LiSTEE RT を適用していない Linux の評価結果を図 5 に示す．図 3 と図 4，どちらの分布でも 20～25μs 付近に外れ値が出現しているが，LiSTEE RT を適用していない Linux にも出現しているため，Linux の問題として今回は検討対象外とする．

LiSTEE では Secure OS が Linux からコアを横取りするため，図 5 では Secure OS の周期タスク処理 300μs 分割込みが遅延している．一方，LiSTEE RT 適用時には Secure OS の影響を受けておらず，図 3 では 300μs を超える遅延は発生していない．

表 3 評価に使用するタスク群の設定

	動作 OS	動作コア	周期	処理時間	CPU 使用時間
タスク 1	Linux	コア 1	1ms	600us	60%
タスク 2	Linux	コア 1	100ms	10ms	10%
タスク 3	Secure OS	コア 1	1000ms	200ms	20%
タスク 4	Linux	コア 0	1000ms	400ms	40%
タスク 5	Secure OS	コア 0	100ms	50ms	50%

LiSTEE RT 適用時と LiSTEE RT を適用していない Linux を比較した場合，標準偏差の差は小さいが，LiSTEE RT の割込みの平均が約 0.9μs 大きくなっている．本評価では LiSTEE RT は Secure OS 内で Linux の割込みを待機しているため， OS の遷移分だけ遅延時間が大きくなったと考えられる．OS の遷移処理は毎回同じ命令群を実行し，処理時間は固定になると考えられるため，最悪実行時間は推測可能と思われる．最悪実行時間が見積り可能であれば，リアルタイム性に大きな影響を与えないと思われる．

6.2 リアルタイム性と CPU 使用効率の両立

高 CPU 使用効率を達成しつつ，デッドラインミスが発生しないことを確認する．評価では 10%の CPU は OS が利用すると仮定し，CPU 使用効率が 100%となるように 90%の CPU 使用効率が必要な周期タスク群を実行し，LiSTEE RT においてデッドラインミスが発生しないことを確認する．デッドラインミスはリアルタイム性の欠如によって発生するため，CPU を限界まで使用すると思われるタスク群に対してデッドラインミスが発生しないということは時間制約を守るリアルタイム性の高さと高 CPU 使用効率の両立を確認できる．また，比較対象として，OS パーティション方式でも同タスク群を実行する．

LiSTEE RT 上に表 3 のタスク群を 2 分間動作させる．タスク群は代表的なリアルタイムスケジューリング方式 Rate-Monotonic Scheduling に基づき，周期の短いタスクに高い優先度を付けており，タスク群は動作時にビジーループを行う．LiSTEE RT の FA 適用を想定し，FA 制御タスクに用いられる PLC(Programmable Logic Controller)の最大周期 1ms [13]のタスクを含めた．コア 0，コア 1 ともに両 OS のタスクが動作しており，CPU 使用効率の合計はどちらも 90%となる．

比較対象の OS パーティション方式では表 3 のタスク群について，Secure OS のタスクはコア 0（Secure OS の占有するコア）に，Linux のタスクはコア 1（Linux の占有するコア）に動作させる．この場合，コア 1 の予想 CPU 使用時

間の合計が 130%となり，スケジューリング不可能なため，デッドラインミスが発生すると考えられる．

上記評価を行った結果，LiSTEE RT ではデッドラインミスは発生せず，OS パーティション方式ではデッドラインミスが発生した．OS パーティション方式では，「[sched_delayed] sched: RT throttling activated」というエラーメッセージが表示され，デッドラインミスが発生した．

以上の結果より，高 CPU 利用率環境における LiSTEE RT のリアルタイム性の維持を確認した．一方，OS パーティション方式ではデッドラインミスが発生し，リアルタイム性と CPU 使用効率の両立を達成できなかった．

7. 考察

7.1 リアルタイム性

6.1 節において LiSTEE RT が非優先 OS のコアの横取りを防止し，優先 OS のリアルタイム性を維持できることを確認した．また，LiSTEE RT 導入による割込みの遅延は 0.9us 固定であり，このオーバヘッドを予め想定することで，リアルタイム性を維持できる．

一方，キャッシュ・メモリバス資源の競合によるストールや VMM の OS 切り替えによりロックを持ったタスクが休止する Lock-Holder Preemption (LHP) [14]が生じることにより，処理が大きく遅延し，リアルタイム性を損なう危険性がある．上記課題に対して，文献 [15]， [16]のような対策技術を併用できる．

また，I/O 資源と割込みを優先 OS と非優先 OS 間で共有する場合，VMM が非優先 OS の割込みをペンディングするための判断処理により，リアルタイム性を損なう恐れがある．そこで，本論文の実装では，各 OS が占有した IRQ・FIQ を制御し，VMM での判断処理を行わずに HW により非優先 OS の割込みをペンディングすることで，この問題を回避した．

7.2 CPU 使用効率

6.2 節において，LiSTEE RT による高 CPU 使用効率とマルチコア資源の有効活用について確認した．ただし，今回は 2OS の動作を想定し，VMM に非優先 OS のスケジューリング機能を実装していない．非優先 OS 間の CPU 使用率と応答性，VMM への実装量に考慮したスケジューリング機能について検討する必要がある．

7.3 保守性

LiSTEE RT は OS の改変を必要とせず，HW で割込み入力の判断を行う場合，VMM の実装は優先 OS の設定に基づいて割込みコントローラの設定を変更するだけのため，実装量は少ない．ただし，VMM で割込みの入力判断を行う場合や複雑な非優先 OS のスケジューリング機能を実装する場合，実装量の増加について注意する必要がある．

表 4 各方式における要求到達度の比較

方式	リアルタイム性	CPU 使用効率	保守性
LiSTEE RT	○	○	○
ARINC 653 方式	◎	×	○
OS パーティション方式	◎	×	○
RTOS 優先方式	×	○	○
階層型フラットスケジューリング方式	○	○	×
スケジューリングサーバ方式	○	○	×

7.4 システム開発コスト

LiSTEE RT は開発者にタスクの動作コアを設定する作業や各コアに優先 OS を設定する作業を課すため，以下の節でそれらについて考察する．

7.4.1 LiSTEE RT 適用による開発者への負担

LiSTEE RT を利用する際には，リアルタイム性に基づいてタスクを振り分け，リアルタイムタスクのみをコアに固定する作業が必要となる．一般的な組込みシステムでは，タスクの機能は決まっているため，リアルタイムタスクとノンリアルタイムタスクに容易に振り分けることができる．そのため，上述の作業は加わるものの，開発の大きな妨げにはならないと考えられる．

7.4.2 コア数によるシステム設計時の制約

LiSTEE RT は，1 つのコアに，優先 OS を 1 つのみ割り当てる．すなわち，LiSTEE RT でリアルタイム性を維持できる OS の最大個数は，コア数と同じとなる．例えば，ある OS のリアルタイムタスクのスケジューリングに 2 コア以上を必要とする場合，複数のコアに対して同一の OS を優先 OS としなければならない．このとき，リアルタイム性を維持できる OS の最大個数は減少する．

開発者はコア数，動作させる OS，OS 上のリアルタイムタスクが必要とするコア数を考慮して，LiSTEE RT を利用する必要がある．

7.5 関連研究との比較

3 章で述べた関連研究に対し，2 章の各要件の到達度を示した結果を表 4 に示す．システム開発コストについては全ての方式で要件を到達しているため，表 4 からは省いている．ARINC 653 スケジューリング方式と OS パーティシ

ョン方式は CPU 使用効率低下の課題を持つが，リアルタイム性を高く維持できる．これらの方式では OS が切り替わらないため，LHP は発生しない． OS パーティション方式ではコアのキャッシュを占有できるため，キャッシュ・バスの競合を回避できる可能性がある．また，ARINC 653 スケジューリング方式では OS の割当て時間内には他 OS は動作しないため，キャッシュ・バスの競合は発生しない．

LiSTEE RT は全ての要件を満たしている．一方，階層型フラットスケジューリング方式とスケジューリングサーバ方式はリアルタイム性の維持と CPU 使用効率の両立のためにソフトウェアに大きく頼った解決を行っているため，保守性が大きく低下する．ただし，これらの方式は理論的にリアルタイム性を維持できる OS の数を制限しない特徴がある．

8. おわりに

本稿ではリアルタイム性と CPU 使用効率を両立する LiSTEE RT について述べた．LiSTEE RT はシステム設計者がコアごとに設定した割込みを優先する OS に基づき，割込みと OS 切替えを制御する．評価ではリアルタイム性と CPU 使用効率の両立について確認した．今後はキャッシュ・メモリバス資源の競合，LHP の課題解決に向けてより詳細な評価を行う．また，3 コア以上のマルチコアに対応し，複数 OS 動作に対応できるように実装を行う．

9. 参照文献

[1] Z. Gu , Q. Zhao, “A State-of-the-Art Survey on Real-Time Issues in Embedded Systems Virtualization,” Journal of Software Engineering and Applications，Vol. 5，No. 4, 2012.

[2] Y. Li, M. Danish , R. West, “Quest-V: A Virtualized Multikernel for High-Confidence Systems,” Technical Report, Boston University,, Boston, 2011.

[3] J. SERRA, J. RODRIGUES, T. ALMEIDA , A. AND MENDES, “Multi-criticality Hypervisor for Automotive Domain.,” In Inforum Conference, 2014.

[4] 金井遵 , 磯崎宏, “高速な OS 切替え機構を有する組込み機器向けセキュアモニタ LiSTEE,” 情報処理学会 研究会報告 2013-OS-126(19),pp.1-8(2013), 2013.

[5] ARMLtd, “ARM Security Technology Building a Secure System using TrustZone Technology,” PRD29-genc-009492c, 2009.

[6] Airlines Electronic Engineering Committee, “ARINC 653−Avionics Application Software Standard Interface,” 2003.

[7] M. Masmano, I. Ripoll, A. Crespo and J. J. Metge, "XtratuM: A Hypervisor for Safety Critical Embedded Systems," Proceedings of Real-Time Linux Workshop, Dresden, 28-30, Dresden, September 2009.

[8] S. S. Xi, J. Wilson, C. Y. Lu , C. Gill, “Realizing Compositional Scheduling through Virtualization,” Technical Report, University of Pennsylvania, Philadelphia, 2011.

[9] 本田晋也, 均. 鈴木, 樋口正雄 , 福井昭也, “車載システム向けハードウェア仮想化支援機能による RTOS 一体型仮想マシンモニタ,” 情報処理学会研究会報告 2017-OS-139(9).pp.1-7(2017), 2017.

[10] C. Augier, “Real-Time Scheduling in a Virtual Machine Environment,” Proceedings of Junior Researcher Workshop on Real-Time Computing (JRWRTC), Nancy, March 2007.

[11] A. Lackorzynski, A. Warg, M. Voelp , H. Haertig, “Flattening Hierarchical Scheduling,” Proceedings of the tenth ACM international conference on Embedded software , pp. 93‑102, New York, NY, USA, 2012.

[12] D. Sangorrin, S. Honda , H. Takada, “Integrated Scheduling for a Reliable Dual-OS Monitor,” Information and Media Technologies，Vol. 7，No. 2 ， pp. 627-638, June 15, 2012.

[13] 山田真大, 小林良岳, 本田晋也 , 高田広章, “マルチコアにおけるコア隔離機構 Timer Shield による Linux のリアルタイム性向上,” コンピュータ ソフトウェア，Vol. 31，No. 4，pp. 77-96, 2014.

[14] V. Uhlig, J. LeVasseur, E. Skoglund , U. Dannowski, “Towards scalable multiprocessor virtual machines,” In Proceedings of the 3rd Virtual Machine Research and Technology Symposium,pp 43-56, 2004.

[15] 北原祐, 本田晋也 , 高田広章, “組込みマルチコアシステムにおけるリアルタイム性保証機構の評価,” 研究報告組込みシステム 2017-EMB-44(42),pp.1-6(2017) , 2017.

[16] 山崎修平 , 河野健二, “Lock-Holder Preemption に対する Pause Loop Exiting の限界に関する調査,” 研究会報告,2016-OS-138(12),pp.1--7, 2016.