

リアルタイム仮想化ソフトウェア基盤におけるタイマ割り込み通知機構

金城 聖[†] 永島 力[†] 元濱 努^{††} 片山 吉章^{††} 毛利 公一^{†††}

[†] 立命館大学大学院理工学研究科 ^{††} 三菱電機株式会社情報技術総合研究所

^{†††} 立命館大学情報理工学部

近年、携帯電話や情報家電、車載システムなどの組込みシステムでは、リアルタイム性と信頼性の保証とともに、高度な情報処理機能への要求が高まっている。このような要求への解決策として仮想計算機モニタ (VMM) を利用したリアルタイム OS (RT-OS) と高機能 OS の共存によるアプローチがある。VMM では、タイマデバイスが仮想化されるためその精度が低下する可能性がある。そこで、既存の VMM のタイマ管理について実験を行った。その結果、プロセッサ数以上のゲスト OS が存在する場合、タイマ割り込み周期に揺らぎが発生することがわかった。タイマ割り込み周期に揺らぎが発生すると、ゲスト OS としての RT-OS は正確に時間を計測することができず、リアルタイム性を保証するのが困難になる。そこで、本論文では、タイマ割り込み周期の揺らぎを解消する手法として、実計算機上の未使用のタイマデバイスを利用し、仮想計算機上の RT-OS に対して安定したタイマ割り込みを通知する機構について述べる。

Management of timer interrupt in Real-Time Virtual Machine Monitor

Akira KANASIRO[†] Chikara NAGASHIMA[†]

Tsutomu MOTOHAMA^{††} Yoshiaki KATAYAMA^{††} Koichi MOURI^{†††}

[†] Graduate School of Science and Engineering, Ritsumeikan University

^{††} Mitsubishi Electric Corporation

^{†††} College of Information Science and Engineering, Ritsumeikan University

Recently, in addition to guarantee of response performance, requirements for high functionality have become high in embedded systems like cell phones, electric household appliances and car-mounted systems. Some researches tried to fill this needs by coexistence of operating systems (OS) on a virtual machine monitor (VMM). However real-time OS cannot be usually run on existing VMMs because it is difficult that they send timer interrupts stably to guest OS. In this paper, we describe a management method of timer interrupts for real-time OS on VMM by using additional timer device.

1 はじめに

近年、携帯電話や家電製品、車載システムなどの組込みシステムでは、ハードウェアの高性能化とともに多機能化が進んでおり、リアルタイム性や信頼性に加え、GUI やマルチメディア処理といった情報処理機能が必要になっている。それらの組込みシステムでは、 μ ITRON や VxWorks などに代表されるリアルタイムオペレーティングシステム (以下、RT-OS と記す) が利用され、機器制御を主要処

理とした最小限の機能提供によってリアルタイム性と信頼性を保証している。また、高度な情報処理機能を提供するシステムでは、Windows や Linux などの高機能 OS が利用され、これらの OS は、複雑な情報処理を主要処理としており、複雑なシステム構成によって実現されている。

リアルタイム性、信頼性の保証と高度な情報処理機能という 2 つの異なる特性をもつ要求を同時に実現する方法として、2 種類の方法があげられる。一つは、単一の OS にすべての機能を実装する方法である。この方法では、RT-OS

を高機能化するか、高機能 OS をリアルタイム化することとなり、その設計・実装は困難であり、開発コストも増加する。2 番目の方法として、複数 OS の同時動作を可能とする OS 共存手法を使う方法がある。この方法は、おのこの異なる特性を持つ OS を共存動作させ、システム全体として複数の OS の動作を管理することで異なる要求を満たす方法である。この方法では、既存の RT-OS や高機能 OS といった既存の資産の利用が容易になるため、開発コストの増加を防ぐことが可能である。以上から、我々は、近年の高機能化への要求を満たしつつ、RT-OS のリアルタイム性を保証する基盤技術として、OS 共存手法を利用する。

本論文では、OS 共存手法を利用したリアルタイム性と高度な情報処理機能を実現するシステムを提案するにあたり、まず既存の OS 共存手法について調査を行った。OS 共存手法には、実計算機上で複数の OS を動作させる非仮想化方式と仮想化方式がある。仮想化方式については、ホスト OS 型とハイパーバイザ型の 2 つに分けられる。それぞれの特徴については、2 章で詳しく述べる。本論文で提案する手法では、複数 OS の動作が容易に実現でき、計算機資源の管理も柔軟に行えるハイパーバイザ方式を利用する。

組込みシステムの特性として、高い信頼性、リアルタイム性、リソース制約に従うことへの要求があげられる。ハイパーバイザ方式を利用した場合の課題として、重要となるのがリアルタイム性の保証である。ハイパーバイザ方式でリアルタイム性を保証する上で重要となるのが、ハイパーバイザ上で動作する OS (以下、ゲスト OS と記す) のタイマ管理である。OS のタイマ管理は、タイマデバイスからのタイマ割り込みを利用して実現されている。しかし、ゲスト OS は、タイマデバイスからのタイマ割り込みがハイパーバイザに占有されているため、タイマ割り込みを利用できない。この問題を解決するためにハイパーバイザは、ゲスト OS に対して、仮想タイマ割り込みという仕組みを用意している。しかし、仮想タイマ割り込みは、他のゲスト OS の動作やハイパーバイザの動作の影響を受けるため、ゲスト OS 内の時間の精度が低くなってしまう。そこで、ゲスト OS 内の時間の精度を向上させるタイマ割り込み通知手法を検討するために、まず既存のハイパーバイザである Xen [1] の調査を行い、既存手法のタイマ性能評価実験を行った。

以下、本論文では、2 章で OS 共存手法について述べ、3 章でリアルタイム仮想化ソフトウェア基盤について述べる。また 4 章と 5 章で Xen のタイマ管理手法の調査と性能評価実験について述べる。6 章で、より精度の高いタイマ割り込み通知手法について述べ、7 章で提案手法の考察を行う。8 章で関連研究と提案手法の比較を行い、おわりにで本論文をまとめる。

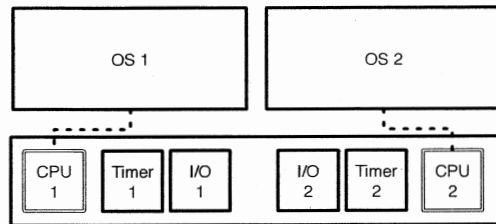


図 1 非仮想化方式

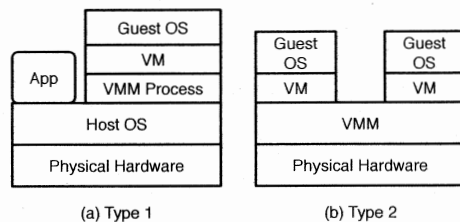


図 2 仮想化方式

2 OS 共存手法の比較

既存の OS 共存手法には、非仮想化方式と仮想化方式のホスト OS 型、ハイパーバイザ型がある。以下、両方式の特徴について述べる。

2.1 非仮想化方式

非仮想化方式は、近年登場したマルチコアプロセッサを利用し、図 1 に示すように、プロセッサコア毎に異なる OS を動作させることを可能にする方式である。I/O デバイスの管理については、使用するデバイスを OS 毎に分割し、固定割当てを行うことで実現している。そのため I/O 処理は単体の OS を動作させた場合とほぼ同等の速度で実行することが可能となる。また、非仮想化方式には、プロセッサの共有を可能にする方式もある。

2.2 仮想化方式

仮想化方式には、OS が動作する環境を仮想計算機 (以下、VM と記す) として提供することで OS の共存を実現している。仮想化方式は、さらに、VM の構成によって 2 種類に分けられる。以下では、それぞれをホスト OS 型 (Type 1)、ハイパーバイザ型 (Type 2) と称して説明する。

ホスト OS 型 この方式は、ホスト OS 上のアプリケーションで VM を構成し、その上で別の OS を動作させる方式である。図 2(a) に示すように、ゲスト OS からは、ホスト OS の存在が隠蔽されている。この方式では、ゲスト OS に対して提供している仮想ハード

ウェアは、エミュレーションによって実現しており、ゲスト OS から実計算機上で動作しているのと同様に認識される。この方式の代表例としては、VMware Server や Virtual PC などがある。

ハイパーバイザ型 この方式は、仮想計算機モニタ（以下、VMM と記す）を利用した方式である。VMM とは、図 2(b) に示すように、計算機資源を隠蔽し、ゲスト OS に対して VM を提供するソフトウェアで、ハードウェア上で動作する。ホスト OS 型では、実計算機資源をホスト OS で管理しているが、ハイパーバイザ型では VMM が管理している。この方式の例としては、Xen や VMware ESX Server などがある。

2.3 方式の比較と考察

非仮想化方式は、使用するハードウェアを OS 毎に分割し固定割当てを行っているため、オーバーヘッドが小さいという利点がある。しかし、そのために利用できるハードウェアが限られたり、OS のデバイス管理機構に変更を加える必要があるなどの導入の困難さが問題としてある。また、OS が特権モードで動作するため、悪意の有無にかかわらず、他の OS の動作に影響を与えてしまう問題がある。

仮想化方式のホスト OS 型は、非仮想化方式と比較して、OS が動作する環境を VM として提供するため、ゲスト OS が利用可能なハードウェア数の制限が緩和される。しかし、ゲスト OS に対して、仮想化したハードウェアを提供しているため、I/O 処理では、仮想化によるオーバーヘッドが発生する。また、ホスト OS 上で、様々なプロセスやアプリケーションと並行して、ゲスト OS が動作する。そのため、ホスト OS にかかる負荷によって、ゲスト OS のパフォーマンスが低下してしまう問題がある。

仮想化方式のハイパーバイザ型では、計算機資源の管理を VMM が行うため、ゲスト OS 同士は独立して動作する。そのため、ゲスト OS 同士の動作によって与える影響は小さい。しかし、計算機が仮想化されているため、I/O 処理にはオーバーヘッドが発生する。

組込みシステムの特性として、信頼性、リアルタイム性、リソース制約に従うことへの要求を実現することが求められる。そのため、OS 共存手法を利用したシステムには、安全に計算機資源が利用できること、OS の動作が他の OS の動作に影響を与えないこと、計算機資源の管理が柔軟に行えることが求められる。このことから、本稿で提案するシステムでは、計算機資源を VM という単位で仮想化し、VM 上で動作するゲスト OS の資源利用と動作の干渉を低減することができ、ゲスト OS と異なる階層で実計算機資源の管理が可能であるハイパーバイザ方式を利用する。

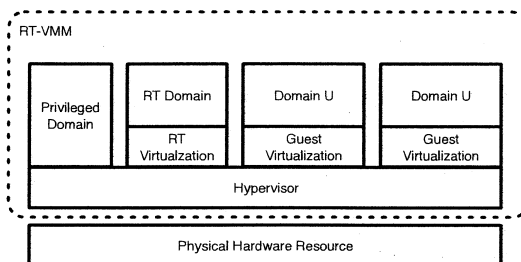


図3 リアルタイム仮想化ソフトウェア基盤の構成

3 リアルタイム仮想化ソフトウェア基盤 (RT-VMM)

3.1 概要

RT-VMM は、RT-OS と高機能 OS を共存動作を可能とするシステムである。図3に示すように、RT-VMM では、ゲスト OS が動作する VM をドメインという単位で管理される。特に RT-OS が動作するドメインを RT ドメイン (RT Domain) という専用のドメインで管理し、RT ドメインに対してリアルタイム性を保証することに特化した仮想計算機環境 (RT Virtualization) を提供する。高機能 OS は、ゲスト OS 用の仮想計算機環境 (Guest Virtualization) が提供されるゲストドメイン (Domain U) 上で動作する。RT-VMM の構成は、RT-ドメインとゲストドメイン、各ドメインの管理を行うインタフェースを提供する特権ドメイン (Privileged Domain)、さらに、ドメインに対して仮想化された計算機資源を提供し、実計算機資源 (Physical Hardware Resource) を隠蔽する RT ハイパーバイザ (RT Hypervisor) から構成される。

RT-VMM を実現する上で、以下の点について考慮する必要がある。

厳密な時間管理 RT-OS では、タスクの起床や切替えの契機に時間を利用する。この時間は、タイマ割込みによって計測されている。仮想化技術を利用した場合、時間の計測に利用されるタイマ割込みは仮想化され、精度が低下する。RT ドメインを実現するには、タイマ割込みの精度の向上が重要となる。

I/O 処理のオーバーヘッド RT ドメインは仮想計算機であるため、RT ドメインから入出力要求は、一度 RT ハイパーバイザを経由して処理される。実計算機の資源は、他のドメインからも利用されるため、入出力処理は排他的なものとなる。RT ドメインを実現するためには、入出力処理の優先度制御など、計算機資源の管理について検討する必要がある。

システム全体のフットプリント 組込みシステムでは、リ

ソース制約に従うことが求められる。RT-VMM では、複数のドメインと RT ハイパーバイザによって構成される。そのためシステム全体のフットプリントが、従来のシステムと比較して大きくなってしまふ。そのため、フットプリントを考慮したシステムの構築が重要となる。

本論文では、上記の要求の内、リアルタイムタスクの起床、切替えて利用されるタイマ割り込み管理に着目して述べる。

3.2 設計・実装方針

本システムは、既存の VMM である Xen に改良を加えることで実現する。Xen は IA-32, PowerPC, IA-64 にて開発が活発に行われている。本システム的设计・実装は、IA-32 版の Xen を利用して行う。本システムは、組込みシステムでの動作を想定しているため、IA-32 に依存しない設計を行う必要がある。IA-32 版 Xen では、完全仮想化と準仮想化の 2 種類の方法が用意されている。前者の方法は、実計算機と同様の構成を VM として提供し、ゲスト OS に変更を加えることなく動作を可能とする。この方法は、CPU 依存の仮想化支援機構を利用して実現している。後者の方法は、ゲスト OS に改良を加え、入出力や特権命令の処理に関して、VMM と専用の通信機構を利用することで、エミュレーションによるオーバーヘッドを低く抑えてゲスト OS の動作を可能とする。この方法は、専用の機構を利用することなく実現されており、IA-32 以外の Xen でも利用されている。組込みシステム向けのプロセッサでの動作を想定した場合、完全仮想化と比較して、準仮想化の方がアーキテクチャに対する依存性が低い。そのため、本システムでは、準仮想化を利用した設計を行っていく。

4 Xen のタイマ割り込み管理

Xen は、サーバシステムの統合や二重化などのシステムの可用性を向上することを目指して開発されたオープンソースの VMM である。そのため、タイマ割り込み管理は、Xen 上で動作するゲスト OS に対して公平性が保たれるように設計されており、RT-OS の動作を考慮していない。そこで、タイマ割り込みの精度を向上させるために、Xen のタイマ管理について調査を行った。

Xen が受信するタイマ割り込みには、プラットフォームタイマ割り込みと時限処理の実行に利用するローカル APIC(以下、LAPIC と記す) タイマ割り込みがある。

プラットフォームタイマ割り込み プラットフォームタイマ割り込みには、Programmable Interval Timer(PIT), High Precision Event Timer(HPET) などのタイマデバイスが利用される。これらのデバイスは起動時に設定された一定の周期でタイマ割り込みを発生する。

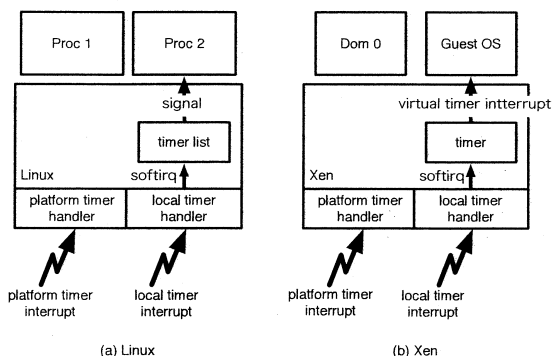


図4 タイマ割り込み管理

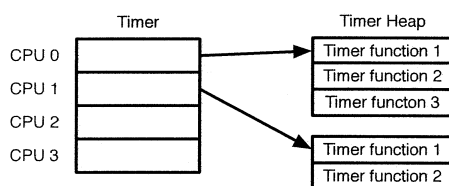


図5 時限タイマ

LAPIC タイマ割り込み LAPIC タイマ割り込みには、各プロセッサコア上に搭載されている LAPIC タイマが利用される。LAPIC タイマ割り込みは、遅延処理を実現するソフト割り込みの生成に利用される。

Xen のタイマ割り込み管理は、Linux のタイマ割り込み管理をもとに実装されている。図4に示すように、Xen には、Linux と同様に、2種類のタイマ割り込みを処理するハンドラが存在する。プラットフォームタイマ割り込みハンドラ(platform timer handler)の動作は、Linux と Xen の両者に大きな違いはなく、起動時からのプラットフォームタイマ割り込みの回数を計測することでシステム時間の計測を行っている。

LAPIC タイマ割り込みは、LAPIC タイマ割り込みハンドラ(local timer handler)で処理され、ハンドラ内部でソフト割り込み(softirq)を生成する。以後の処理は、すべてソフト割り込みによって実現されている。Linux と Xen のタイマ割り込み管理において両者の違いが見られるのは、このソフト割り込み以降の処理である。図4(a)に示すように、Linux ではソフト割り込みをタイマリスト(timer list)と呼ばれる時限処理を実行するのに利用される。Xen の場合、時限タイマ(timer)の実行に利用されている。

時限タイマ 時限タイマは、Xen の時限処理が登録されて

いるデータ構造である。ヒープ構造によって構成されている。時限タイマの実行は、ソフト割込みハンドラ内で行われる。図5に示すように、プロセッサコアごとに、時限タイマテーブルが用意されている。時限タイマテーブルに登録されている処理は、そのタイマテーブルを所有しているプロセッサ上で実行される。Xen もしくはゲスト OS が時限処理を時限タイマに登録する際に、ヒープ構造を再構成し、次に LAPIC タイマ割込みを起こす時間を LAPIC に設定する。

Linux の場合、タイマリストに登録されている処理には、プロセスにシグナルを送る処理やスケジューリングに関わる処理が登録されている。Xen の場合は、ゲスト OS に仮想タイマ割込みを通知する処理やドメインスケジューリングに関する処理が登録されている。

仮想タイマ割込み ゲスト OS の時刻は、ゲスト OS の責任で計測される。時刻の計測のためにはタイマ割込みを受信する必要がある。Xen では、ゲスト OS が受信するタイマ割込みを仮想タイマ割込みとして実現している。図4(b)に示すように、ゲスト OS への仮想タイマ割込みの通知に時限タイマを利用しており、仮想タイマ割込みの周期は 10ms に設定されている。

5 実験

Xen 上で動作させた RT-OS と実機で動作させた RT-OS でタイマ割込み周期の計測を行った。

5.1 実験方法

本実験は、RT-OS として μ ITRON 4.0 仕様 [2] に準拠した TOPPERS/JSP 1.4.2 [3](以下、TOPPERS と記す) を IA-32 に移植したもの (TOPPERS/x86) と Xen の準仮想化に対応したもの (TOPPERS/xen) を利用して行った。TOPPERS/x86 では、PIT によるタイマ割込みを利用し、1ms の周期でタイマ割込みを生成するように設定している。TOPPERS/xen では、仮想タイマ割込みを利用する。実験内容は、以下に示す。

実験 1 実計算機上で単一のプロセッサコアで動作させた TOPPERS/x86 のタイマ割込み周期の計測。

実験 2 Xen 上でドメインスケジューラに Credit スケジューラを利用し、一つのコアにドメイン 0 と TOPPERS/xen が動作するドメインを割り付けた場合のタイマ割込み周期の計測。

実験 3 Xen 上でドメインスケジューラに Credit スケジューラを利用し、ドメイン 0 と TOPPERS/xen が動

表 1 実験環境

CPU	Core 2 Quad 2666Mhz
VMM	Xen 3.2.0
Dom0	Fedora 7(GUI 環境)

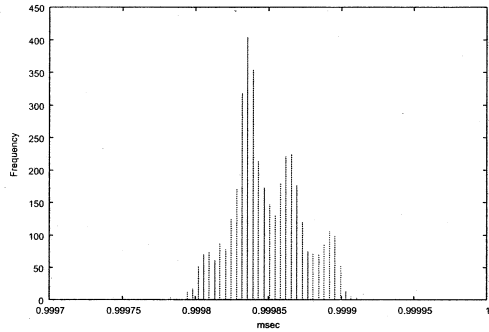


図 6 実験 1 の結果

作するドメインのそれぞれに異なるプロセッサコアを割り付けた場合のタイマ割込み周期の計測。

実験 4 Xen 上でドメインスケジューラに SEDF スケジューラを利用し、一つのコアにドメイン 0 と TOPPERS/xen が動作するドメインを割り付けた場合のタイマ割込み周期の計測。

実験 5 Xen 上でドメインスケジューラに SEDF スケジューラを利用し、ドメイン 0 と TOPPERS/xen が動作するドメインのそれぞれに異なるプロセッサコアを割り付けた場合のタイマ割込み周期の計測。

上記の 5 つの実験では、プロセッサのタイムスタンプカウンタを利用し、4000 回の割込み周期を計測した。実験 2 から実験 5 に関しては、Xen 上で動作で、ドメインスケジューリングによる実行ドメインの切替えが発生する場合と発生しない場合、またドメインスケジューリングアルゴリズムに Credit スケジューラを使用した場合と SEDF スケジューラを使用した場合に分けて行っている。これは、ドメインスケジューリングによるタイマ割込み周期への影響を観測するためである。実験に利用した計算機環境については、表 1 に示す。

5.2 結果

実験 1 によって得られたタイマ割込み周期分布を図 6 に示す。図 6 からは、グラフの x 軸の範囲の関係で、タイマ割込みの周期が分散しているように見える。しかし、0.99980ms から 0.99990ms の間に密集しており、実際には

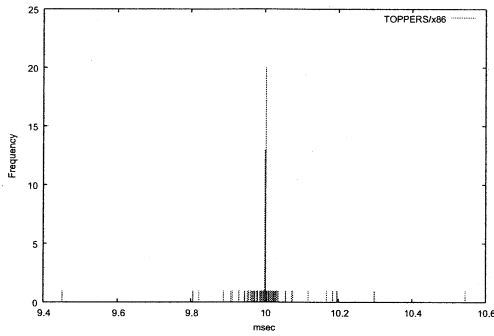


図7 実験2の結果

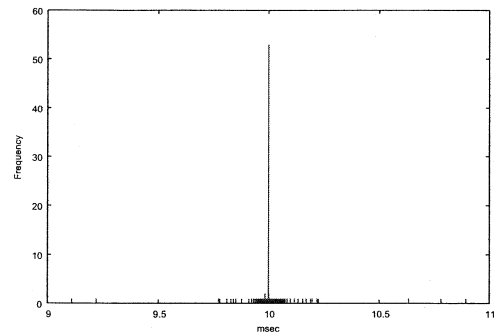


図9 実験4の結果

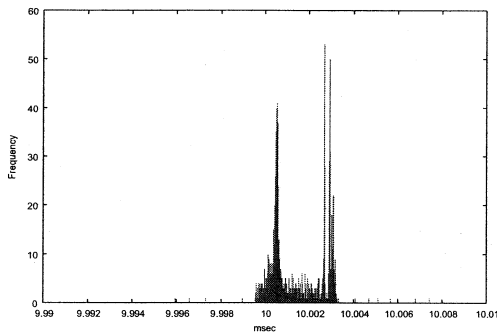


図8 実験3の結果

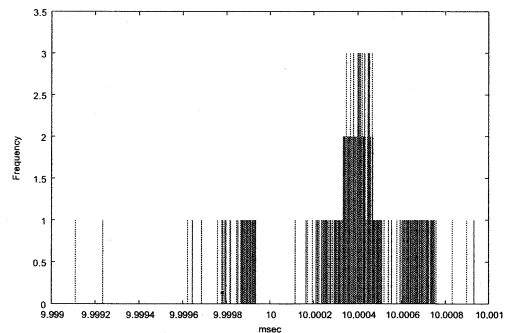


図10 実験5の結果

ほぼ一定の周期でタイマ割込みが発生している。

実験2によって得られたタイマ割込みの周期分布を図7に、実験3で得られたものを図8に示す。図7からは、10ms 付近の密集率が低く、周期が全体に分散していることがわかる。図8からは、10ms 付近の密集率が高く、約10ms の周期で発生するタイマ割込みの頻度が多いことがわかる。

実験4によって得られたタイマ割込みの周期分布を図9に、実験5で得られたものを図10に示す。図9からは、10ms 付近の密集率が低く、周期が全体に分散していることがわかる。図10からは、10.004ms 付近が最も密集率が高いが、頻度は少なく、実験3の結果と比べて、周期は分散していることがわかる。

5.3 考察

計測環境において、TOPPERS/x86 でのタイマ割込み周期は 1ms、TOPPERS/xen でのタイマ割込み周期は 10ms と設定の違いがある。それを考慮に入れた場合、TOPPERS/x86 が受信するタイマ割込みの周期は、ほとんど一定である。図8と図10から、ドメインスケジューラは、

SEDF よりも Credit スケジューラの方が高い性能を出している。また、ドメインの切替えが発生する場合は、両スケジューラともタイマ割込みの周期は、広範囲に分散している。この原因として、ドメインに対して異なるプロセッサを割り付けている場合は、時限タイマにドメインスケジューリングに関わる処理が登録されないため、安定した周期でタイマ割込みを受信している。しかし、ドメインのスケジューリングが発生する場合は、プロセッサの割り当てから外れてしまうことがあるため、タイマ割込み周期にジッタが発生する。また、時限タイマの実行にソフト割込みは、優先度の高い処理が発生した場合、処理の遅延を許可しているため、その際に発生する遅延もジッタ発生の原因として考えられる。

6 提案手法

6.1 概要

本論文で提案する手法では、RT ドメインに対して通知する仮想タイマ割込み周期の分散を防止するために、未使用タイマデバイスを利用する。

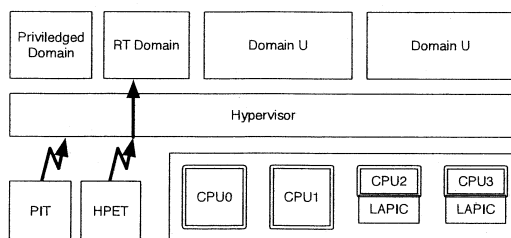


図 11 システム構成

図 11 に示すように、未使用タイマデバイスが生成するタイマ割り込みを RT ドメインに通知することで、周期の分散を解消することが可能となる。ゲストドメインに対しては、Xen が標準で提供する LAPIC 由来の仮想タイマ割り込みを利用する。

6.2 機 構

本手法を実現するためには、以下に示す機構を Xen に追加実装する必要がある。図 12 にシステムの構成を示す。

- RT ドメイン用タイマ初期化機構
RT ドメインに対して割り付けられているプロセッサコアにタイマ割り込みを通知するようにタイマデバイスを初期化する。
- RT ドメイン用タイマ割り込みハンドラ
初期化機構によって設定されたタイマデバイスからのタイマ割り込みを処理する。このハンドラ内で現在稼働中の RT ドメインに対してタイマ割り込みを通知する機構を呼び出す。
- RT ドメイン用タイマ割り込み通知機構
ハイパーバイザから RT ドメイン上のゲスト OS に対してタイマ割り込みを通知する。

RT ドメイン用タイマ割り込みの発生から RT ドメインに通知されるまでの処理の流れは次のようになる。

1. RT ドメイン用に初期化されたタイマがタイマ割り込みを生成する。
2. RT ドメイン用タイマ割り込みハンドラが起動し、タイマ割り込み受信を確認する。
3. その後に RT ドメイン用タイマ割り込み通知機構を経由して RT ドメインにタイマ割り込みを通知する。

6.3 評価方法

本手法の評価方法として、次の場合におけるタイマ割り込み周期の計測を検討している。

- 実機上 RT-OS と RT ドメイン上 RT-OS の比較

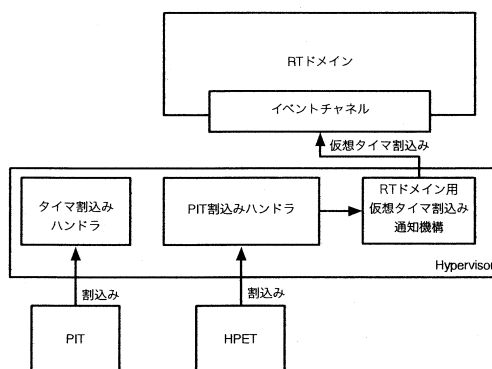


図 12 ソフトウェア構成

実機と RT ドメインの比較によって、ハイパーバイザを仲介することによる遅延を計測する。

- RT ドメイン上 RT-OS と Xen 標準のゲストドメイン上で動作する RT-OS の比較
専用タイマデバイスを利用する RT ドメインと Xen 標準のソフト割り込みを利用したゲストドメインにおけるタイマ割り込み周期のジッタを計測する。
- ゲスト OS 起動下での上記 2 パターンの計測
RT ドメイン以外のゲストドメイン上でゲスト OS が動作しているときの RT ドメインの負荷耐性を評価する。

7 考察と検討事項

7.1 IA-32 アーキテクチャへの依存性の考察

提案手法で利用している PIT や HPET などのプラットフォームタイマは、一般的な IA-32 アーキテクチャの計算機に搭載されているものである。組込み向けのシステムでは、これらのタイマは搭載されていない。本節では、専用のタイマを利用する手法の組込みシステムにおける有用性について考察する。

組込みシステムでは、ARM や SH といったプロセッサが頻繁に利用される。これらの組込み向けプロセッサは、現在マルチコア化が急速に進んでいる。ARM ベースの MPCore プロセッサや、非対称型マルチコアプロセッサである MP211 [4] では、プロセッサ上に複数のタイマを搭載している。これらのタイマを利用すれば、提案手法のように未使用のタイマを利用する方法は有用であるといえる。

7.2 複数の RT ドメイン動作に関する検討

複数の RT ドメインの動作への要求がある場合について検討する。提案手法は、Xen をベースに設計を行っているため、タイマ管理に関しても Xen の仕様によって制限

されてしまう。提案手法のモデルでは、一つの RT ドメインに対して専用のタイマを割り当てることで、タイマ性能の信頼性を高めている。しかし、複数の RT ドメインが動作する場合、提案手法では、RT ドメインのスケーラビリティがタイマの搭載数によって制限されてしまう。この制限を解消するためには、Xen に依存している部分を変更する必要がある。プロセッサコア上にある LAPIC タイマは現在ハイパーバイザ内の時限処理に利用されている。この時限処理をプラットフォームタイマによって実現するように設計をしておし、LAPIC タイマを解放すれば、スケーラビリティについて、プロセッサコア数分は保証される。

8 関連研究

本論文では、複数のゲスト OS が動作する環境で RT-OS 動作を実現する方法として、RT-VMM を提案した。既存研究の中にも、RT-OS の動作を想定し、複数の OS 共存させて動作可能とすること目的とした研究が盛んである。

三菱電機のマイクロクラスタリング OS [5] は、従来の SMP 環境での OS のブートプロセスに変更を加え、異なるプロセッサコア上で異なる RT-OS と高機能 OS のカーネルが実行されるように設計されており、各 OS は特権モードで動作する。提案システムでは、仮想化技術を利用しているため、ゲスト OS は特権モードを利用しないため、他のゲスト OS の動作に与える影響を緩和することが可能である。

OKL4 [6] は、L4 マイクロカーネルに対してハイパーバイザの機能を追加した OS である。ゲスト OS は、L4 マイクロカーネル上に構築された VM 上で動作する。OKL4 の特徴としては、リアルタイム性の保証をゲスト OS 上の RT-OS ではなく、L4 マイクロカーネルで保証する点である。リアルタイム性が必要となるアプリケーションは、L4 マイクロカーネル上で動作させ、機能性は VM 上のゲスト OS で実現する方式をとっている。提案システムでは、RT-OS もゲスト OS として動作する。そのため、資産流用性は高い。また、OKL4 のようにリアルタイムタスクの実行・終了といった OS の機能を実装する必要が無いため、ハイパーバイザがシンプルな構成で実現可能である。VirtualLogix 社の VirtualLogix VLX [7] は、組み込みシステム向けのリアルタイム VMM である。RT-OS のリアルタイム性を維持しつつ、Linux や Windows などの高機能 OS の動作を可能にする。VLX の特徴は、ゲスト OS を優先度ベースでスケジューリングし、RT-OS を再優先度で動作させる。また、非 RT-OS からの I/O 要求を RT-OS のデバイスドライバをフロントエンドドライバとして処理することで、ゲスト OS の動作を実現している。提案システムでは、I/O 要求の処理は、RT ドメインと異なる特権ドメインで処理を行っている。そのため、VLX と比べて RT ドメインの特権は低いものとなり、システム全体に与える

影響が小さい。

9 おわりに

近年の組み込みシステムにおける高度な情報処理機能に対する要求が増大している。本論文では、この新たな要求と従来の要求を実現するために、OS 共存手法として仮想化技術を利用した RT-VMM を提案した。RT-VMM では、RT-OS と高機能 OS の共存によって、リアルタイム性・信頼性・機能性を実現する。

RT-VMM の実現において、リアルタイム性の保証を行うには、厳密な時間管理が求められ、そのために安定した周期でタイマ割り込みを通知する必要がある。この問題を解決するにあたり、既存の VMM である Xen のタイマ周期について実験をおこなった。実験結果からは複数のゲスト OS がプロセッサを共有する場合、タイマ割り込みの周期に揺らぎが発生することがわかった。本論文では安定した周期でタイマ割り込みをゲスト OS に通知するために、未使用のタイマデバイスを利用したタイマ割り込み管理手法について述べた。また、本手法の他のアーキテクチャにおける有用性について考察を行った。今後の課題としては、上記手法の実装・評価と検討事項で述べた複数の RT ドメイン動作時の時間管理の信頼性向上の検討がある。

参考文献

- [1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, Andrew Warfield: "Xen and the Art of Virtualization," In Proceedings of the 19th ACM Symposium on Operating Systems Principles 2003, pp. 164–177 (2003).
- [2] ITRON Project, *muITRON 4.0* 仕様, <http://www.ertl.jp/ITRON/home-j.html>
- [3] TOPPERS プロジェクト, TOPPERS/JSP カーネル, <http://www.toppers.jp/>
- [4] NEC エレクトロニクス, MP211, <http://www.necel.com/>
- [5] 遠藤 幸典, 菅井 尚人, 山口 義一, 近藤 弘都: "シングルチップマルチプロセッサ上のハイブリッド OS 環境の実現—システムアーキテクチャ—," 第 66 回情報処理学会全国大会講演論文集, Vol.1, pp. 9–10 (2004).
- [6] Open Kernel Labs, OKL4, <http://www.ok-labs.com/products/okl4>.
- [7] VirtualLogix, Inc., VirtualLogix VLX, <http://www.virtuallogix.com/>