

複数 OS 混載環境における高精度時刻同期方式の検討

山本遼介[†] 出原章雄[†] 桐村昌行[†]

三菱電機株式会社 情報技術総合研究所[†]

1. はじめに

近年、産業向けシステムや車載機器といった組込みシステムにおいて、制御系処理と情報系処理を混在させる需要がある。制御系処理と情報系処理を混在させるために、ハイパーバイザを用いてリアルタイム OS(RTOS)と汎用 OS(GPOS)を同時に動作させるシステムが普及している。

ただし、上記のような複数 OS 混載環境の開発では、システム全体の動作分析が困難である。システム全体の動作分析には、各ゲスト OS の挙動を紐づけるためにシステム共通時刻が必要である。しかし、RTOS+GPOS のハイパーバイザ環境では、各ゲスト OS にタイマを固定的に割り当て、各ゲスト OS が異なる時刻を有する場合がある。さらに、RTOS の動作を解析するためには、マイクロ秒単位の時刻を必要とし、各ゲスト OS から確認する共通時刻の差も 1 μ s 以下が求められる。

そこで、ゲスト OS 間における高精度な時刻同期方式を検討した。本稿では、検討した方式と適用例について述べる。

2. 想定システム

想定システムの構成図を図 1 に示す。本環境では、ハイパーバイザ上で RTOS と GPOS が動作し、各ゲスト OS に CPU コアを固定的に割り当てている。CPU コアを固定的に割り当てている理由は、RTOS のリアルタイム性を損なわないためである。また、各ゲスト OS が共通の時刻を有し、この時刻をロギングアプリが取得することで、共通時刻を利用することができる。

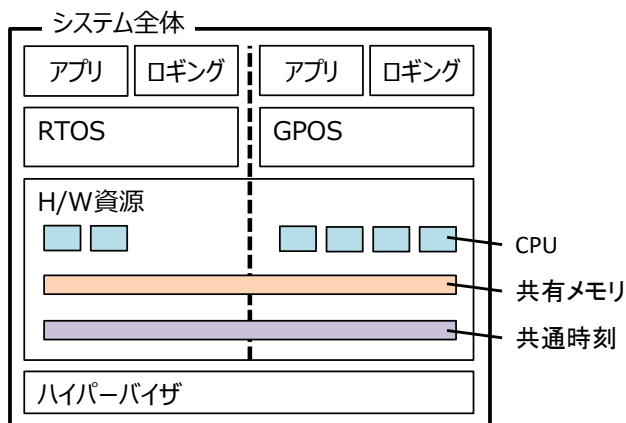


図 1 システム構成図

3. 前提条件

前章で述べた環境において、RTOS と GPOS での共通時刻の条件として、次を定義する。

- ① ゲスト OS 間の時刻差は 1 μ s 以内
- ② 別 CPU コアから共通時刻にアクセス可能
- ③ ユーザランドから共通時刻にアクセス可能

RTOS では、マイクロ秒単位での動作が求められる。そのため、システム全体の解析において、各ゲスト OS での時刻差は、1 μ s 以内であることが求められる (条件①)。

2 章で述べた通り、各ゲスト OS は割り当てられた CPU コア上で動作する。そのため、別コアからであっても共通時刻にアクセス可能であることが求められる(条件②)。

各ゲスト OS の時刻取得処理に差が生じた場合、各ゲスト OS での動作ログにおいて時刻整合性が保たれない。特に、GPOS 側のアプリでログ取得を行う場合、アプリがカーネルランドを経由して時刻を取得すると、RTOS 側と比較し、時間を要してしまう。そのため、ユーザランドからアクセス可能であることが求められる(条件③)。

4. 検討方式

前章で述べた条件を元に方式を検討する。各ゲスト OS で共通の時刻を用いるためには、「各ゲスト OS が保有するタイマの時刻を同期する」方式と「各ゲスト OS が同一タイマを参照し、時刻を同期する」方式がある。

前者は、時刻を同期するにあたり、各ゲスト OS のタイマの差分が必要である。しかし、マイクロ秒単位の精度で差分を取得することは困難である。例えば、各ゲスト OS 間で割り込みを用いて時刻を同期させる方法を考える。H/W やアーキテクチャに依存するが、割り込み発生から割り込みハンドラ到達までの時間が、数 μ s 程度かつ一定でない場合がある。そのため、異なるタイマの時刻を同期させることは困難である。

後者は、同一のタイマを用いるため、各ゲスト OS で時刻差は生じない。よって、今回は、各ゲスト OS が同一タイマを参照し、同期させる方式を採用する。

以上より、本方式は、前章条件①～③を満たすタイマをゲスト OS で共有することで、時刻を同期させる。

5. 検討方式の実現方法

5.1 適用システム

4章で述べた方式について、実際の適用例を検討する。適用システムは、表1のとおりである。ハイパーバイザは市販ハイパーバイザを採用し、改変不可能であることを想定する。

表1 適用システム

GPOS	Windows 10
RTOS	市販 RTOS
ハイパーバイザ	市販ハイパーバイザ
CPU	Intel Core i7 2.6GHz

5.2 タイマの選定

Intel アーキテクチャに搭載される代表的なタイマの一覧を表2に示す。表2より、表1の環境であればTSCを採用することが考えられる。TSCの分解能は、CPUの内部クロックと同等の分解能であるため、1MHz以上の分解能で動作可能である。また、TSCはCPUコアごとにカウンタが異なるが、各カウンタの差分を吸収可能であるため、同一の時刻として利用可能である(詳細は後述)。最後に、TSCはrdtsc命令を用いることで時刻を取得可能であり、rdtsc命令はユーザランドから実行可能である。

以上より、表1に示す環境であれば、TSCを採用するべきである。

表2 タイマー一覧

名称	①分解能	②別コアで共通利用可能	③ユーザランドからアクセス
PIT	○1.193182MHz	○	×(I/Oポート)
RTC	×32.768KHz	○	×(I/Oポート)
HPET	○機種依存(100MHzを保証)	○	×(MMIO)
ACPI PM Timer	○3.579545MHz	○	×(MMIO/IO)
Local APIC Timer	○機種依存(CPUと同等)	×	×(MMIO)
TSC	○機種依存(CPUと同等)	○	○(rdtsc命令)

5.3 利用方法

TSCを各ゲストOSで利用するためには、TSC_ADJUST MSRの値を取得し、各コアで取得した値から減算することで、同期時刻にアクセス可能である。TSCのカウント値は、次の式で求められる[1]。

TSCのカウント値 = ARTのカウント値 *クロックレートの補正 + オフセット

TSC_ADJUST MSRは、ART(Always Running Timer)からのオフセットを示す。そのため、TSCのカウント値からTSC_ADJUST MSRの値を減算することで、コア間共通の値を取得することが可能である。

6. 考察

6.1 ハイパーバイザの制約

ハイパーバイザがTSC-Offset機能を利用していた場合、RTOSとGPOSで確認できる共通時刻に差が生じる。TSC-Offset機能は、ハイパーバイザがゲストOSのTSCのカウント値に対してオフセットを付与する機能である。また、この機能によるオフセット値は、ハイパーバイザが意図的に公開しない限り、ゲストOSから確認することはできない。そのため、TSCによる時刻同期が困難となる。

TSCに限らず、タイマデバイスを仮想化している場合、ハイパーバイザがタイマのカウント値を仮想化している可能性がある。そのため、ハイパーバイザ環境では、「時刻の仮想化の有無」、「実タイマと仮想タイマの差」を確認することが必要である。

6.2 タイマの制約

ゲストOSを単体再起動させる場合、再起動による影響を受けないタイマを選定する必要がある。再起動時、ゲストOSはデバイスの初期化を行う。その際、各ゲストOSが共有しているタイマにアクセス可能であれば、共有タイマを初期化する可能性がある。その場合、他方のゲストOSにおいて共通時刻の整合性が保たれない。

再起動に影響を受けないタイマは、「コアごとに存在する」や「時刻ソースのカウントがリセットされない」といった特性を持つ。TSCの場合、両方とも該当し、ゲストOS単体再起動の影響を受けない。

なお、上記のような特性を持たないタイマを共有する場合は、共有方法を検討する必要がある。例えば、ゲストOSのタイマドライバを改変する方法や、ハイパーバイザでタイマの初期化を防ぐ方法が考えられる。

7. おわりに

本稿では、RTOS+GPOS構成のハイパーバイザ環境におけるゲストOS間での共通時刻を持たせる方式について検討した。本手法では、1MHz以上の分解能かつ、別コアからアクセス可能かつ、ユーザランドからアクセス可能なタイマをゲストOSが共有することで、RTOS+GPOS構成であっても有用な共通時刻を利用することが可能である。

本手法は、ハイパーバイザやタイマの機能に依存する課題があった。今後は、ハイパーバイザやタイマに依存しない時刻同期方法について検討する。

参考文献

- [1] Intel Corporation: “Intel® 64 and IA-32 Architectures Software Developer’s Manual” (2019年10月)