

汎用 OS にリアルタイム性を付加するアプローチ:

# Windows のリアルタイム拡張「RTX」と組み込み機器への応用

<http://monoist.atmarkit.co.jp/mn/articles/1106/24/news001.html>

Windows OS にリアルタイム機能を付与する「RTX(Real Time Extension)」をご存じだろうか。本特集では、Windows をリアルタイムシステム化するアプローチの1つとして RTX を紹介する。

2011年06月24日 10時34分 更新

[石黒一敏(東京エレクトロン デバイス エンベデッド・ソリューション部), @IT MONOist]

## 1. はじめに

---

ご存じの通り、マシン制御を伴うような組み込みシステムでは“リアルタイム性”が求められます。一般的に、こうしたシステムは RTOS(Real Time OS) が用いられますが、その一方で、汎用 OS にリアルタイム性を付加するアプローチも存在します。

今回の特集では、IntervalZero 社が開発した“Windows にリアルタイム機能を付与する”製品「RTX(Real Time Extension)」にフォーカスし、汎用 OS にリアルタイム性を付加するアプローチと RTX の特徴について紹介します。

RTX の最新バージョンである「RTX 2011」は、マイクロソフトの組み込み OS「Windows Embedded Standard 7 SP1」に対応し、最大 32CPU までのマルチコア対応が可能となっています。ちなみに、開発元の IntervalZero 社は、今から 30 年以上前の 1980 年、米国ボストンに設立された Windows 関連テクノロジーを得意とするベンダーで、設立当初の VenturCom、その後の Ardence を経て、現在の社名 IntervalZero となっています。

## 2. Windows をリアルタイムシステム化するアプローチ

---

“Windows がリアルタイム性を保障できないこと”はマイクロソフト自身も認めることです。Windows 環境でリアルタイム性を必要とする場合は、他の RTOS との併用、あるいは Windows にリアルタイム機能を付加する製品を使用しなければなりません。

さて、ここであらためて“リアルタイムシステム”について考えてみましょう。

リアルタイムシステムとは、別の言い方をすると「**応答時間が保障されたシステム**」といえます。ある限られた時間内に必ず処理を実行できることがリアルタイムシステムの要件です。もちろん、Windows 自身もそれなりに高速ですが、どんなに高性能な CPU を使っても最大で数十ミリ秒の遅延(レイテンシ)が発生します。相手が人間であればこの値は十分短いため、“Windows はソフトリアルタイム性についての機能を満たしている”と考えてもよいでしょう。

ただし、マシン制御のように 1000 分の 1 秒、つまり 1 ミリ秒以内には処理を行わなければならないような“ハードリアルタイム処理”を行うには、次のようなアプローチを取る必要があります。

### (1) RTOS システムとの併用

専用 RTOS として設計された「VxWorks」や「QNX」のような OS を別途用意してリアルタイム処理部分を担当させる方法があります。そして、Windows の得意とする GUI と組み合わせてシステムを構築します。当然、Windows と RTOS 用に別々のハードウェアが必要となり、また 2 つの OS 間の通信制御も必要となります。システム間をシリアルポート、TCP/IP 経由で通信するような構成もありますし、RTOS 側を PCI のオプションボードとして構成し、PC の拡張スロットに接続してデュアルポート RAM による高速の共有メモリでの通信を行う構成もあります。

### (2) ハイパーバイザー方式

最近の CPU が持つ仮想化支援機能 (Intel は Intel VT、AMD では AMD-V) を使って、異なる複数のシステムを 1 つのハードウェアに実装してしまう方法です。ハイパーバイザーは起動時に CPU の各コア、メモリなどのハードウェアリソースを分割し、分割された環境が相互に影響を与えないように初期化します。そして、一方の環境では Windows、別の環境では RTOS をそれぞれ起動して走らせることができます。従って、ソフト的な構成は RTOS との併用と同じです。RTOS が x86 アーキテクチャで動作していたのであれば、ほぼそのままの形で 2 つのシステムを 1 つのハードウェアに統合できます。そして、1 つのハードウェアでありながら、例えば RTOS 側はそのまま動作させつつ、Windows 側だけをリセットして再起動させるといったことが可能です。難点としては、Windows と RTOS に加えてハイパーバイザーのライセンス費用が掛かること、また開発環境も従来通り Windows と RTOS を複数管理する必要があることです。

### (3) Windows リアルタイム拡張

そして、Windows 自身にリアルタイム機能を付与する“Windows のリアルタイム拡張”です。こちらは、Windows に何らかの手を加えてリアルタイム機能を追加する方法です。例えば、リアルタイム OS である Windows CE を利用し、Windows へ専用スケジューラを導入して Windows とともに動作させます。Windows CE 側に高いプライオリティを持たせることでリアルタイム性を確保し、リアルタイム処理のすき間を縫って Windows を動作させます。Windows CE のタスクの 1 つとして通常の Windows が動作しているようなイメージです。この構成の場合、CPU コアが

1つであるとリアルタイム側の負荷に Windows が大きく影響を受けてしまいますが、最近のマルチコアCPUであれば、それぞれの処理を各CPUコアに分散させることで、効率の良い処理を行わせることができます。なお、本稿で紹介する RTX もこのように Windows をリアルタイム拡張しますが、RTX の場合は「HAL (Hardware Abstraction Layer)」上に RTX サブシステムを導入するというアプローチを取っています。

ここでは3つのアプローチについて紹介しましたが、注意しておきたいのは、いずれの方法を用いても“**従来の Windows アプリケーションがリアルタイム動作するようになるわけではない**”ということです。リアルタイム機能は、追加されたリアルタイム側のプログラムによって実現されるものです。Windows システムを使用するのは、Windows の GUI による操作性、開発の容易性を活用するのが目的です。

最近では、制御機器であってもマルチタッチ機能を活用した“スマートフォン・ライク”な操作パネルがトレンドになりつつあります。これらを考えると、操作部に Windows を使っていくのは正しい流れといえるでしょう。

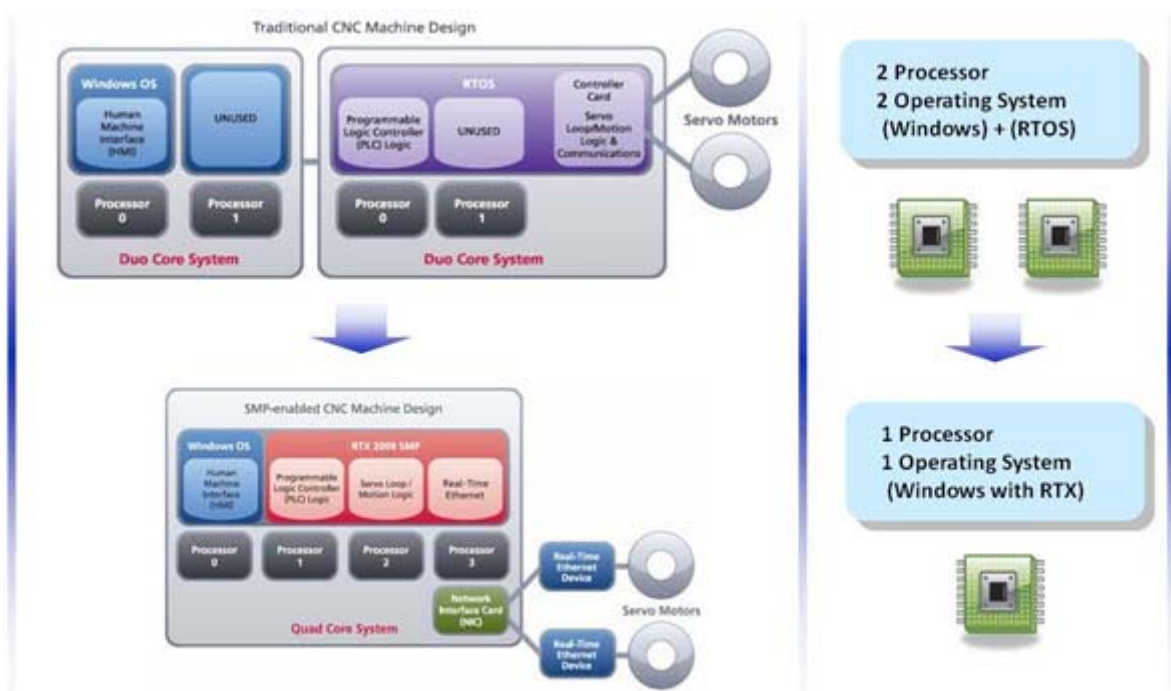


図1 Windows のリアルタイム拡張で、よりシンプルなシステムへ

さらに、もし Windows ベースの PC だけでリアルタイム制御が可能であればコストダウンなど多くのメリットが享受できます。図1のように複数必要であったハードウェア/ソフトウェアのシステムは、よりシンプルなシステムへと簡略化されます。長期間生産される組み込み機器においては専用部品をできるだけ使わないことが BOM(部材)コストを抑えるポイントです。

### 3. RTX のアーキテクチャ

RTX は、HAL 上に RTX サブシステムを導入する手法を用いています。HAL の拡張 I/F を使って RTX サブシステムが Windows へ入り込むような形態となっており、RTX は Windows システムの内部に導入されていることになります。

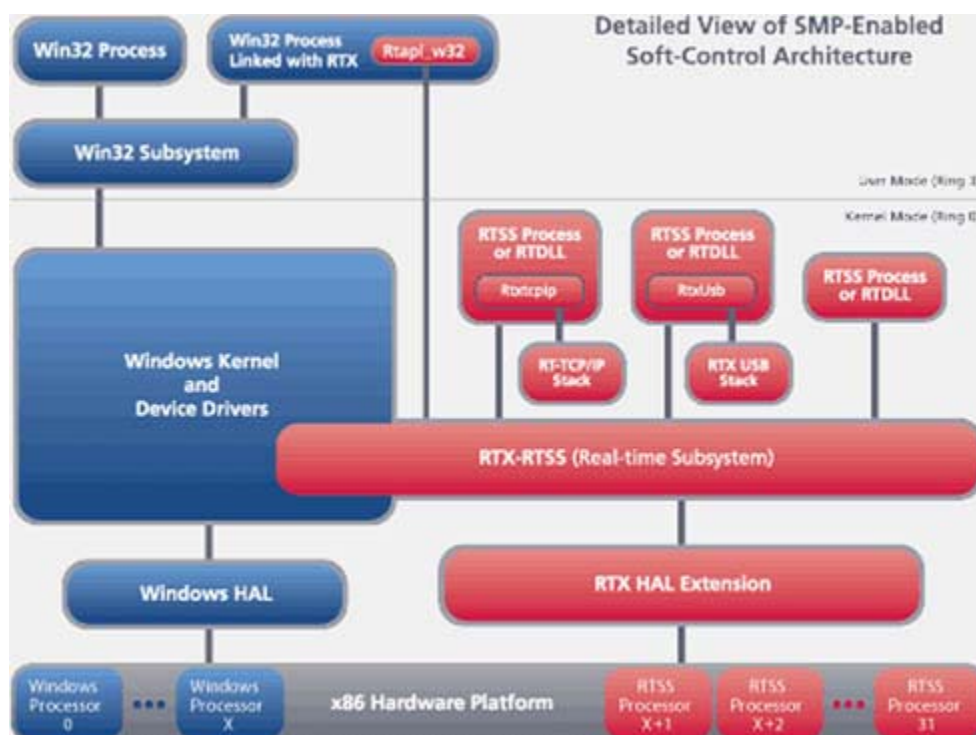


図 2 RTX のアーキテクチャ

ここで考慮すべきは、Windows のメモリ管理です。Windows は仮想メモリで動作しており、物理メモリが足りなくなるとそのページを HDD へスワップアウトさせることによって、物理メモリよりも大きな仮想メモリ空間を提供します。当然、動作すべきプログラムがメモリ上に存在しなければ、HDD からのスワップインが発生します。これには相応の時間がかかり、Windows がリアルタイム性を保障できない要因の 1 つとなっています (ちなみに、Windows CE はリアルタイム OS ですが、こちらもデマンドページング: “必要になるまでそのコードを読み出さない” を無効にしておかなければリアルタイム性は発揮できません)。

すなわち、リアルタイム処理にかかわるプログラムは、物理メモリ上に必ず実体が存在する必要があります。この問題に対して RTX は、Windows の「非ページメモリ」を利用することで対応しています。非ページメモリとは、Windows が管理するスワップアウトされない、つまりプログラムが物理メモリ上に存在していることが保障されているメモリ領域です。Windows システムにおいても即時に応答しなければならない割り込みハンドラ、カーネルなどはここに配置されています。

RTX サブシステムは、ドライバとしてこの非ページ領域にロードされ、また RTX のアプリケーションも同領域にロードされて動作します。この部分は OS のカーネルレイヤーであり、x86 アーキテクチャでいう「Ring0」となります (図 2 の中央赤色の部分)。

RTX サブシステムは、約 0.5M バイトとコンパクトに作られており、実質的にシステム全体を管理するスケジューラとなります。基本的に RTX アプリケーションをリアルタイム動作させ、RTX のアプリケーションがアイドル状態になったときだけ、Windows 側を動作させるように機能します。

ちなみに、非ページメモリには上限があり、Windows OS の種別、搭載メモリによって上限は変化します。例えば、Windows XP の場合は 256M バイトが上限です。RTX アプリと Windows アプリは共有メモリによってプロセス間通信とデータの受け渡しを行うことができますが、これらのバッファも非ページメモリ上に確保されるため、大量のデータを扱う場合、非ページメモリが不足するという問題が発生します。このようなケースにおいて、IntevalZero 社から 1 つのアイデアが提供されています。それは、Windows が使用するメモリを制限することで、“Windows が使用していないメモリを巨大な共有メモリとして活用してしまおう”というものです。Windows XP においては Boot.ini、Windows Vista / Windows 7 においては BCD ストアのオプションで、Windows が使用するメモリを制限できます。例えば、1.5G バイトのメモリが搭載されたシステムにおいて、Windows 側が 1.0G バイトしか使用しないように指定すれば、残りの 512M バイトを Windows 管理外の単なるメモリとして自由に使うことができます。

#### 4. HAL 拡張アプローチのメリット

---

Windows の HAL (Hardware Abstraction Layer) は、ハードウェア抽象化層を意味する、マルチプラットフォームに対応するためのレイヤーです。

かつて、Windows NT が登場したとき、Windows は DEC Alpha、MIPS、PowerPC など x86 アーキテクチャ以外の RISC CPU 上でも動作していました。プラットフォームの相違を HAL によって吸収することで、さまざまなプラットフォーム上で共通仕様の OS を動作させようとする仕組みです。その後、Intel の CPU が、ほぼ Windows のベース CPU となり、“Windows = Intel プラットフォーム”という図式が出来上がりました。

ところが、2011 年春、ラスベガスで開催された CES において、マイクロソフトは“開発中の次期 Windows OS「Windows 8」は ARM プロセッサもサポートする”と発表し、実際に動作デモも行いました。この展開に世間はあっと驚いたわけですが、もともと HAL によってマルチプラットフォーム対応できる Windows としてみれば、それほど大変ではなかったものと想像できます。

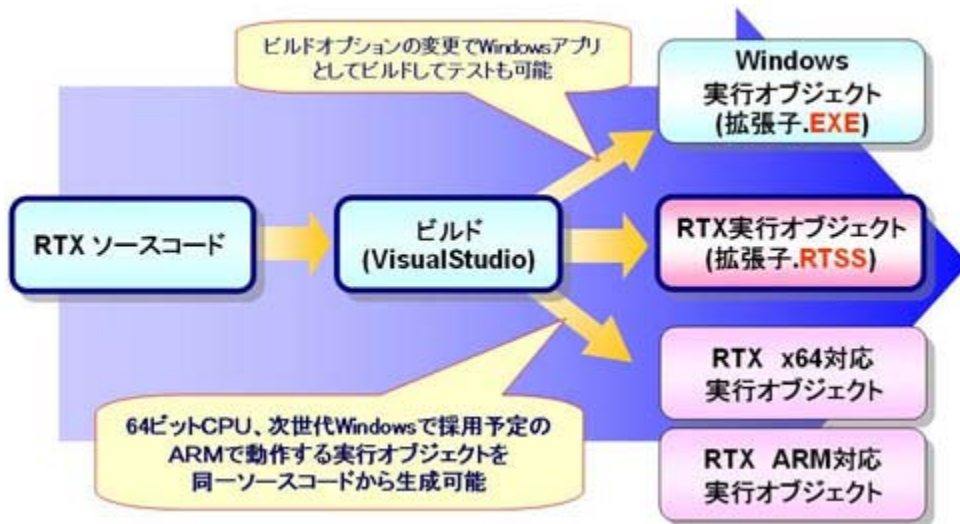


図3 RTXソースコードは、ビルドオプションでいろいろな実行形式に対応

実は、HAL上に位置するRTXも同じように、HALの恩恵を受けることができます。CPUアーキテクチャが変更されても、RTXサブシステムとRTXアプリケーションは新しいCPU用に再コンパイルすれば容易に動作可能です。また、リリース予定の64ビット版RTX、そして、将来WindowsがARM対応したときのARM版RTXにおいて、RTXはソースコードレベルで互換性があります。つまり、プラットフォームが変わっても一度作成したソースコードをそのまま流用できるということです。

その他、リアルタイム性はありませんが、RTXアプリケーションをWindows上で動作するファイル(\*.exe)としてビルドし、テストすることも可能です。

## 5. マルチコア対応

前述の通り、最新のRTX 2011は、最大32コアのシステムまでサポート可能となっています。この機能に対して、多くのユーザーは「そんなに多くのコアを何に使うの?」という反応を示します。実際、RTXを使ったモーションコントロールの事例でも「デュアルコアCPUで十分!」との評価が多数です。

実は、その答えは意外な分野にありました。最初にRTXの最大32コアサポートを応用したのはマルチメディア系のソリューションです。

デジタルオーディオミキサーへの応用として、従来DSPを使っていたデジタルオーディオのピッチコントロール、ミキシングコントロールなどを各CPUコアでオーディオチャンネルごとに処理するというものです。これまで高価なDSPボードを何枚も並べて実現していた20chのデジタルミキサーは、Xeon(6コア)×4プロセッサのボード1枚とRTXで置き換えることが可能となりました。

これは、先のマルチプラットフォーム対応と組み合わせることで、スケーラビリティメリットを發揮します。例えば、デジタルミキサーにおいて、より高性能な応用であれば 64 ビット版 RTX、コンシューマ向けの数チャンネルのミキサーであれば ARM プロセッサを使用した安価なシステムを選択できます。さらに、これらのシステム構築において各チャンネルを制御する RTX アプリケーションは、全く同一のソースコードで管理できるわけです。

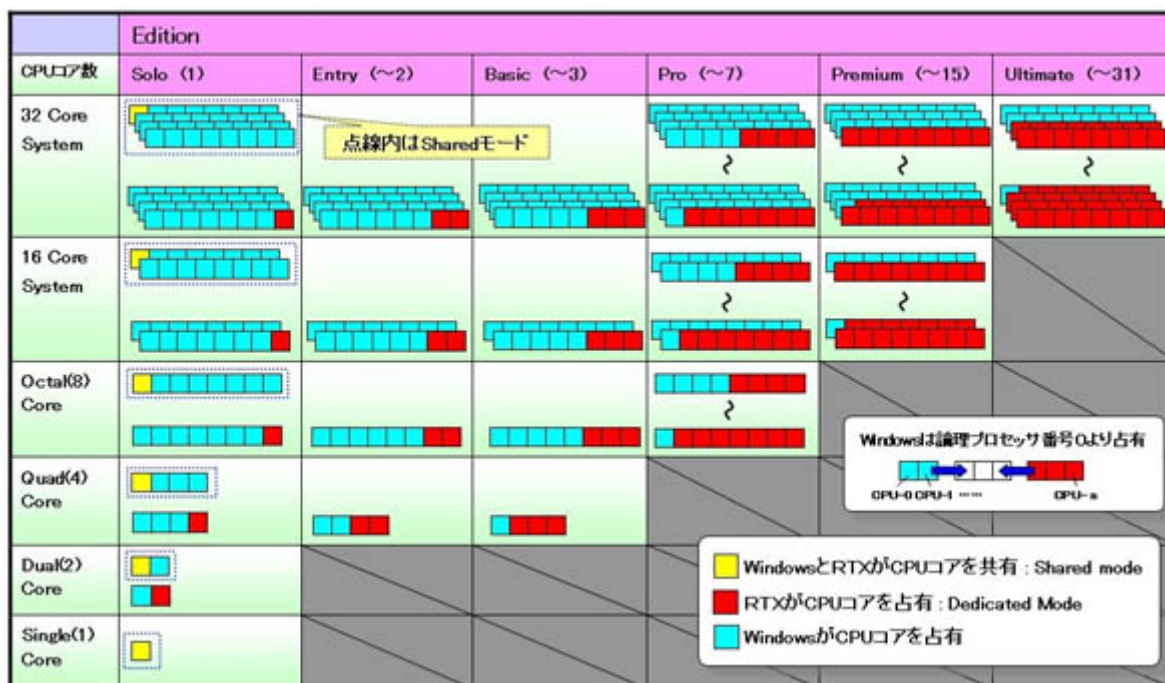


図 4 RTX のエディションとコアの関連

現在、RTX には 6 種類のエディションが用意されており、RTX が使用する CPU コアの数に応じて柔軟に対応可能となっています。今後、急速にマルチコア化が進む中、RTX にはいろいろな活用方法が準備されています。さらに、RTX のマルチコア対応は SMP (Symmetric Multi Processing) を完全サポートしており、コアの数が増えても各 RTX アプリケーションのオーバーヘッドが増加しないという特徴があります。

## 6. RTX の開発

RTX での開発 / デバッグは「Visual Studio」を使用します。RTX アプリケーションについては、ドライバのデバッグと同じく Visual Studio に「WinDbg」を併用するなどの手法が用いられますが、実際はリアルタイム動作しているアプリケーションをブレークさせても有用な情報を得られることは少なく、付属のツール類を使用してタスクの状態、使用されているリソースなどをチェックする手法が効率的です。また、既に最大 32CPU までのマルチコアに対応した RTX ですが、これらのマルチコア対応された複数のスレッドを効率的にデバッグできるような対応も計画されています。

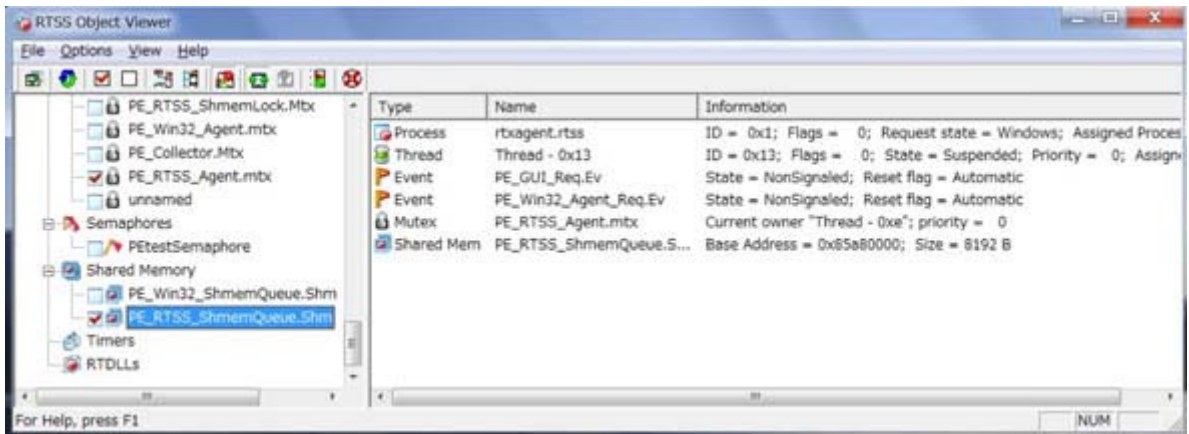


図 5 RTX アプリの状態を確認できるツール「RTSS Object Viewer」

Visual Studio は「Visual Studio 2010」まで対応しており、RTX の SDK を Visual Studio のアドインとしてインストールする形態を取ります。これで RTX アプリケーション、DLL などを新規プロジェクトとして選択するだけで RTX のひな型が準備され、開発を進めることができます。使用する API も従来の Win32API をそのまま流用できることも RTX を導入するメリットの 1 つです。

RTX の開発言語は、基本的に C / C++ ですが、.NET Framework のマネージドコードである C# からも RTX を呼び出せるようなライブラリが用意されています。C# を使って、Windows 側のアプリケーションを開発し、RTX と連携して動作させることが可能です。

また、マルチコアの並列処理を考える場合、単純にコンパイラを変更するだけで大きな効果が得られることも覚えておくといよいでしょう。RTX は、代表的な Intel コンパイラにも対応していますので、性能面を追求する場合、これらを試してみる価値はあると思われます。

## 7. RTX の実装

RTX を動作させるためには、Windows 上に RTX サブシステムをインストールする必要があります。このモジュール群を「RTX ランタイム」と呼び、RTX は Windows NT から最新の Windows 7 SP1 まで 1 種類のパッケージで対応しています。

組み込み用の「Windows Embedded Standard 2009 (XP Embedded)」、あるいは「Windows Embedded Standard 7」で動作させることも問題ありません。RTX サブシステムは、自動起動設定を行っておけば Windows と同時にサービスを起動できるので、電源を投入後 Windows ロゴが表示されるタイミングとほぼ同時に使用を開始することができます。

## 8. RTX の特徴: ブルースクリーンと RTX

RTX にはいろいろな特徴がありますが、その中でも注目すべきは Windows がブルースクリーンになっても動作を継続できることです。

「なぜ、ブルースクリーン後でも動作するのか？」という質問はよく聞かれます。これは、

1. **ブルースクリーンは、いわゆるクラッシュとは異なる予期された動作であること**
2. **RTX サブシステムと RTX アプリケーションは、非ページメモリに常駐していること**

が理由です。

ブルースクリーンが発生すると(何も操作できなくなるので)、「もうダメだ……」と思ってしまいがちですが、これは Windows システムがシステムクラッシュの危険を事前に検知して停止している状態です。つまり、このまま処理を続けるとメモリ、ファイルを破壊してしまうなどの危険性があるので止まっているだけで、実際 RTX が走っている非ページメモリ自体は健在であることがほとんどです。残念ながら、Windows 側のユーザー I/F は停止してしまっているため、通常の操作は継続できませんが、“フェイルセーフ”という観点からは十分です。

## 9. RTX の特徴:何でもできてしまう RTX アプリケーション

---

RTX を使うと物理メモリ、I/O などのシステムリソースへアクセスすることが簡単にできてしまいます。取り扱いを間違えれば、Windows カーネルさえも破壊してしまう危険性がありますが、その反面、この機能は非常に魅力的ともいえます。

実際、RTX のユーザーから「CPU コアの温度を知りたい」という要望がありました。最近の CPU は、各コア上に温度センサーを備えており、Windows 側は ACPI (Advanced Configuration and Power Interface) にて、常に温度の監視を行っています。万一、CPU ファンが停止して急激に温度が上昇した場合、Windows はクリティカルシャットダウンしてシステムを緊急停止します。

高価なダイヤモンドを研磨するような工作機械を考えた場合、機械がいきなり止まって何十万円もする素材がダメになってしまえば、たとえ 1 カ月に 1 回の頻度であったとしても大問題です。温度検知とリアルタイム性能とは関係ありませんが、温度が上昇して Windows 側がシャットダウンしてしまうような兆候を検知し、対処することが RTX 側で必要となりました。

Intel CPU の場合、CPU のコア温度である DTS (デジタル温度センサー) 値は MSR レジスタのインデックス「0x19c」で読み出され、RDMSR 命令を実行した CPU コアのセンサー値が得られます。RDMSR は特権命令であるため通常のユーザーモードのアプリケーションからは実行させることはできません。しかし、RTX の場合は、次のように直接 RTX アプリケーションの中にインラインアセンブラで記述し、そのまま実行させることができます。

```

//-----
// Read Digital Thermal Sensor
// Input: none
// Output: 0..127 - DTS value
//         < 0 - invalid DTS value

short read_DTS() {
_asm{
    push    ecx
    push    edx
    mov     ecx, 0x19c // MSR index
    rdmsr                    // Read MSR
    shr     eax, 16
    and     ax, 0x807f // strip DTS value and valid flag
    xor     ax, 0x8000 // MSB is 1 if invalid
    pop     edx
    pop     ecx
}
}

```

このファンクションで得られる DTS 値は、「TJunction」と呼ばれる各 CPU シリーズによって異なる基準温度からの差分となります。テストした環境の CPU では、“TJunction = 100 ”なので差分を計算して表示させると次のような結果となります。

```

#define TJUNCTION 100 // TJunction of "Intel Core2Duo T5500"

void _cdecl wmain( int argc, wchar_t **argv, wchar_t **envp )
{
    int t;
    RtWprintf(L"RTX demo - Getting CPU temperature\n");
    t = read_DTS();
    if (t >= 0) {
        t = TJUNCTION - t;
        RtWprintf(L" Current CPU temp = %d C\n", t);
    } else {
        RtWprintf(L"Invalid DTS\n");
    }
    ExitProcess(0);
}

```



図 6 RtxServer によるメッセージの表示

このように RTX アプリケーション内で出力されたメッセージは、「RtxServer」と呼ばれるコンソールに出力されます。RTX アプリケーション内でも、Windows と同じように「Printf()」関数が使えますが、Windows 側のダイアログに結果を表示させるとリアルタイム性がその時点で失われます。従って、RTX のシステムにおいては非ページメモリ内に確保されたバッファにメッセージはいったん出力され、それを RtxServer にて表示させることでリアルタイム性に影響しない工夫がなされています。

## 10. RTX の性能と今後

RTX を Core 2 Duo (2.66GHz) のシステムで動作させたときの各種状態遷移の時間を示します。ほとんどの遷移は、1 マイクロセカンド前後で完了しています。いままでハードリアルタイムの指標であった 1 ミリセカンドのさらに 1000 分の 1 です。

RTX性能実測値 (μ sec)	
(Core2 Duo 2.66GHz/ Windows XP Pro RTX Dedicated mode)	
タスクへのイベントセット	0.14 - 1.24
イベントセットから待ちタスクへの遷移	0.21 - 0.45
ミューテックス開放から待ちタスクへの遷移	0.26 - 0.64
セマフォ開放から待ちタスクへの遷移	0.24 - 1.10
同一プライオリティ間のタスク遷移	0.13 - 0.58
タスクのプライオリティ変更	0.23 - 0.60
割り込みタスク応答時間	0.00 - 1.00

図 7 RTX の性能実測値

多くのユーザーは、1 ミリセカンドで応答してくれば十分で、この数値は過剰であるといえます。しかし、これは従来のシステムでは 1 ミリセカンド程度のリアルタイム性が限界だったので、それに合うだけのシステム設計しか行わなかったということではないでしょうか？

マイクロセカンドオーダーのリアルタイム性能が、PC と RTX のような製品との組み合わせで得られるとすればどうでしょうか。既に、DSP との置き換えは始まっています。将来はさらに新しい応用分野、アイデアが生まれてくるはずです。なお、今回紹介した RTX は、今後、64 ビット化、次世代 Windows の ARM プロセッサへの対応、より高性能なマルチプロセッサシステムである NUMA への対応などが予定されています。