

Pentium® III プロセッサのシリアル番号機能とその活用

Stephen Fischer -- インテル・コーポレーション、BMD-FM デザイン

James Mi -- インテル・コーポレーション、コンテンツ・グループ

Albert Teng -- インテル・コーポレーション、コンテンツ・グループ

インデックス・ワード: Pentium® III、インターネット、Java*、資産管理、情報管理

摘要

現在、家庭や企業で日々パーソナル・コンピュータを利用している人々の間では、インターネットが果たす役割がますます大きくなっています。このため、コンピュータになんらかの形式で固有の識別子を付けられるようにすることが重要になってきました。このような機能があれば、TCO 削減や電子商取引のためのシステム管理から情報管理まで、幅広いアプリケーションで数々のメリットが期待できます。

このようなニーズに応え、Pentium® III プロセッサでは、既存の命令セットにシリアル番号機能を組み込みました。シリアル番号機能は、製造過程でダイに書き込まれた情報を利用し、外部のソフトウェアで解読できる固有の識別子を生成します。

なお、インテルではユーザのプライバシーに配慮し、プロセッサ・シリアル番号機能をユーザが無効にできるようにしています。

はじめに

Intel®プロセッサ・シリアル番号(以下、ps#と略します)とは、Pentium® III プロセッサに新機能として導入された固有の数値識別子を指します。このシリアル番号は、外部のソフトウェアから解読できます。

プロセッサ・ベースのシリアル番号を利用すると、新しい種類のアプリケーションを簡単に実現できるようになります。また、認証レベルを高める手段として ps#を採用すれば、インターネット経由の電子トランザクションにも効果的です。企業では、ps#を利用すると、システムの設定や追跡が容易になるため、管理機能が向上します。取り扱いに注意すべきデータに関しては、ps#情報をデー

タのアクセス権に関連付けることにより、管理機能がより万全になります。

この記事では、プロセッサ・シリアル番号機能とは何かを示し、それが Pentium III プロセッサにどのように実装されているかを解説します。また、この機能によって実現できるアプリケーションの例を掲げた上で、オープン・ネットワーク環境で ps#によって CPU を識別し、ウェブ・サイト間の情報の相互相関を制限するアプリケーション・フレームワークを紹介します。

アーキテクチャとその実装

Pentium® III プロセッサに導入された ps#機能は、既存の CPUID の拡張命令によって情報を伝達します(参考資料 [1])。CPUID 命令の役割は、プロセッサの個別のパラメータを外部のソフトウェアへ返すことです。これらのパラメータには、製品ファミリ、モデル、ステッピング番号などの項目のほか、そのプロセッサで利用できる関数を示す機能フラグ・フィールドなど、機能に関する属性情報があります。したがって、ソフトウェアへ返すパラメータにプロセッサ・シリアル番号情報を加えることは、CPUID 命令の拡張として自然な流れでした。また、この命令はすべての特権レベル(PL0~PL3)で実行できるため、アプリケーション・レベルでも OS レベルでも実行できます。これにより、ps#機能の利用範囲を大幅に広げることができます。

ステッピング番号や機能フラグ・ビットなどのパラメータ情報は、EAX に対応する入力インデックス値を格納して CPUID 命令を実行すると、汎用レジスタ EAX、EBX、ECX、EDX で返されます。

```

Case EAX=0;
{
    EAX          = maximum index supported
    EBX:EDX:ECX = "GenuineIntel"
}
Case EAX=1;
{
    EAX          = Family:Model:Stepping
    EBX:ECX      = reserved
    EDX          = feature flags (New ps# feature flag bit 18)
}
Case EAX=2;
{
    EAX:EBX:ECX:EDX = cache and TLB parameters
}
Case EAX=3;
{
    EAX:EBX      = reserved
    ECX:EDX      = processor serial number data
}

```

図1: CPUID 命令の定義

ps#情報を取得するには、新しいインデックス(EAX=3)を使います。また、ps#機能の有無は、新しい機能フラグ、ビット 18 の設定によって確認できます。このため、プロセッサで ps#機能がサポートされ、かつ有効になっているかどうかを、外部のソフトウェアから判断できます。図 1 は、Pentium III プロセッサでサポートされている CPUID 命令の定義をまとめたものです。

プロセッサ・シリアル番号では、機能の有効と無効をユーザが切り替えられるようになっていました。これは、プロセッサ・シリアル番号が知られることによって、ユーザのプライバシーが侵害される恐れがあることを考慮したものです。この切り替えは、ユーザまたはシステム管理者が操作します。この機能を実装するため、「固定」プロパティを持つ新しい読み書き制御レジスタ・ディスエーブル・ビットを追加しました。CPUID 命令の実行中にプロセッサ内部のマイクロコードがこのビットを調べ、ps#情報を CPUID 命令レベルの機能に反映してもよいかどうかを確認します。このビットが「1」(ps#ディスエーブル)に設定されると、ソフトウェア的な手段では「0」に戻せません。プロセッサをハードウェア・リセットしないとディスエーブル・ビットをクリアできないため、ソフトウェアでディスエーブル操作を実行した後で、ソ

フトウェアによってユーザ設定が変更される心配がありません。また、ps#ディスエーブル・ビットには、最高の特権レベル(PL0)でしかアクセスできません。このようにアクセスが制限されているため、ユーザやシステム管理者が設定を変更するには、システム BIOS などの管理ソフトウェアや初期設定ソフトウェアが必要です。

Pentium III プロセッサが返す ps#情報は、ウエハ・ソートの段階で書き込まれたダイ上のポリシリコン・ヒューズ・ビットから得られます。CPUID 命令をサポートする基本マイクロコードは、これらの内部ヒューズ・ビットに書き込まれた論理値を読み取り、それらを連結して 64 ビット値を形成し、汎用レジスタ EDX と ECX によって返します。

基礎となるヒューズ技術は、ポリシリコンの配線上にチタンシリサイド層を組み合わせる新しいシリコン技法に基づいています(参考資料[2])。プログラムを書き込むには、高い電気的ストレスをかけ、チタンシリサイドを凝集させます。カレント・ミラー検知回路を使い、プログラムを書き込んだヒューズの抵抗を書き込んでいない基準ヒューズと比較測定し、論理値を返します。この技術によるプログラムの書き込み成功率はほぼ 100%で、熱動力や BT によるストレス状態でもこの信頼性は保たれま

す。各論理ヒューズ・ビットには、書き込みに成功した値の信頼性をさらに高めるための冗長ヒューズ・エレメントが組み込まれ、Pentium III プロセッサの製造過程で ps#用のシリアル値を生成し、書き込みプロセスを強化しています。

プロセッサ・シリアル番号機能の活用例

例 1: 管理機能の向上と TCO の削減

大企業の IT マネージャは、管理が行き届き、円滑に機能するコンピューティング・インフラストラクチャを維持するために、日々努力を続けています。Intel® Pentium® III プロセッサとその ps#機能は、IT 部門にとって、管理機能を高め、PC の TCO を削減する新しい手段となります。

従来、IT 部門は、ハードウェアを識別するために、ユーザ名、MAC アドレス、IP アドレス、GUID などの多様な方法を用いていました。しかし、これらの方法はいずれも、消去や変更のできない ps#に比べれば、一貫性と信頼性に欠けています。ps#を利用すれば、システムのユーザが変わったり、ネットワーク・カードを交換したり、システム・ソフトウェアや BIOS をリロードした場合でも、個々の PC を容易に識別できます。さらに、ps#によって、ソフトウェア資産管理に関する通知の信頼性も高まります。IT マネージャは、各システムでどのソフトウェアが実行されているか、誰がそのソフトウェアを使用しているのかを、確実に把握することができます。また、PC のハード・ディスクがクラッシュしたようなときも、ps#はヘルプ・デスクの担当者が問題を解決する手助けになります。

企業のシステム設定やソフトウェア・アップデートの際に ps#を利用すると、起動前でもインストール後でもリモートで確実に PC を識別できます。サポート担当者は、事前にプロセッサ・シリアル番号を把握していれば、データベースにその番号を入力し、PC がネットワークに接続された時点でソフトウェアが配信されるように事前にプログラムできます。これにより、個々の PC の設置場所まで出向いてサポートする回数が減り、設定プロセスも自動化されるため、時間の節約とサポート経費の削減になります。

マルチプロセッサ環境やクラスタリング環境で 1 個のプロセッサが故障した場合、どのプロセッサが故障したのかを特定するのは容易なことではありません。Windows

NT*オペレーティング・システムでは、論理プロセッサは識別できますが、物理プロセッサにはマッピングされません。しかし、ps#を使えば、IT 部門で故障箇所を正確に把握できるため、問題のプロセッサを回避して作業を続けることができます。これにより、負荷バランスとフォールト・トレランスが向上し、ユーザにとってのシステムの可用性が高まります。

例 2: e-Business の管理強化

インターネット・ベースの電子ビジネス(e-Business)によって、企業は互いに自由に情報をやりとりできるようになりますが、情報が意図した相手だけに届くように注意する必要性も高くなります。そのためには、ps#がうってつけの手段となります。

現在の技術では、個人や企業が自分のパーソナル・コンピュータや企業ネットワークにアクセスするユーザを認証するには、2つか3つの変数を使用しています。よく使われるのは、ログイン名とパスワードなど、「ユーザが知っている情報」を使う方法です。ハードウェア・キー(dongle)とスマートカードなど、「ユーザが所有する情報」を使う方法や、バイオメトリクスなど、「ユーザ自身に関する情報」を使う方法もあります。

Intel Pentium III プロセッサとその ps#技術が登場した今、認証手段に新しい「ユーザが所有する情報」が加わりました。ps#はパスワードと併用でき、機密扱いの企業情報を目的のプラットフォームだけに届けるアクセス・トークンとして機能します。たとえば、インターネット・ベースの旅行代理店のネットワークでは、システムのプロセッサ・シリアル番号を検証し、重要な価格情報を、認証を受けた旅行代理店のマシンだけに送信できます。ps#によって識別の信頼性が高まれば、企業のイントラネット情報を従業員のデスクトップへ流し、ps#で認証された社員が、各自の 401(k)制度、給与などの個人データにリアルタイムにアクセスできるようになります。また、企業がユーザにプレゼンテーションを提供する前に ps#認証の層を追加することにより、取り扱いに注意を要するビデオを同時プレゼンテーションで放送できます。

B to B の取り引きでは、企業のデジタル証明書や社内外の認証手段に ps#を組み合わされます。このため、ビジ

* 一般に、ブランド名または商品名は各社の商標または登録商標です。

ネス・パートナーは、自社の証明書を用意して許可されたプラットフォームからアクセスするだけで、プライベートな情報を収集できます。システムの ID を検証する方法については、この記事のプロセッサ・シリアル番号による「システムの検証」を参照してください。

例 3: 情報管理の向上

大量の情報が氾濫している現在では、情報の処理、保存、アクセスの主な手段として PC が活躍しており、いかに情報を管理するかが大きな課題となっています。ps#は、情報サービス・プロバイダが、侵入的な手段を取らずにエンド・ユーザに配信するデータやサービスをカスタマイズするための識別子となります。ps#は重要な情報や機密情報を管理・保護する手段としても優れており、データのバックアップと復元、リムーバブル記憶装置のデータ保護、ファイル・アクセスの管理、適切なユーザ間の文書のやりとりの確認など、幅広いアプリケーションを強化できます。

システムの検証

オープン・ネットワーク環境において、プロセッサのシリアル番号だけでシステムを確実に識別できるソフトウェア・システムを設計するのは容易ではありません。クライアント・システムが、ハッカーの操作する敵対的なシステムである可能性もあるため、クライアント・システムから返された ps#情報の真偽をサーバ側で判断するのは困難です。この問題を解決する基盤となるのが、プロセッサ・シリアル番号検証リファレンス・インプリメンテーション(RI)です。RI は、オープン・ネットワーク環境でクライアント・システムから ps#を収集し、ウェブ・サイト間の情報の相互相関を制限する手段として、インテルとソフトウェア・メーカ各社が共同で開発したものです。

不正防止

RI のソフトウェア・エージェントは、オープン・ネットワーク経由でダウンロードされるため、ハッカーの攻撃にさらされます。このような攻撃への対策として、ソフトウェア・エージェントは特別な不正防止技術を使って設計されており、これを不正防止ソフトウェア(TRS)エージェントと呼んでいます。TRS は、ハッカーによる攻撃の危険性を自動的に検出し、その攻撃から保護します。通常、非プロセッサ・ベースのハードウェア認証方法では、特権レベルの命令を使います。このような命令は、

アプリケーション・レベルでは直接利用できません。デバイス・ドライバなどのソフトウェア層を追加すると、余計にハッカーの攻撃にさらされやすくなります。一方、プロセッサ・シリアル番号命令は、アプリケーション・レベルで直接実行でき、特別なドライバも用意する必要がなく、不正防止機能を強化することができます。このような TRS エージェントは、各ソフトウェア・メーカから提供されています。API 関数の共通セットが定義され、各メーカに提供されているため、複数のメーカのエージェントを簡単に切り替えて使用できます。

このフレームワークは、不正行為を防止するため、クライアントの認証や検証をサーバで行うプロトコルを採用しています。エージェントの使用時間は短く、すぐに廃棄されるため、保護機能が強化されます。

プライバシーの保護

認証システムは、ウェブ・ベースのアプリケーションで重要な役割を果たしています。しかし、消費者のプライバシーを尊重せず、同意を得ずに消費者の個人情報を収集するユーザや企業が存在する可能性も否定できません。このため、インテルでは、プロセッサ・シリアル番号機能だけでなく、数種類のユーティリティ・ツールにも、消費者のプライバシーの問題に対処する手段を盛り込みました。さらに、RI の機能の中にも、ユーザのデータ保護を強化するものがあります。

- ps#を収集するソフトウェア・エージェントは、デジタル・コンテンツ(Internet Explorer[®]ではキャビネット・ファイル、Netscape Navigator[®]では Jar ファイル)でパッケージされ、ウェブ・サービス・プロバイダのデジタル署名を付けてクライアント・システムに配信されます。ウェブ・ブラウザはコンテンツを認識すると、そのソフトウェアにアクセス権を与えるかどうかをユーザに確認するため、ユーザの同意がなければ ps#は収集されません。
- ps#は、読み取られた後、サービス ID を使ってハッシングされ、別の固有の識別子に変換されます。ハッシングされた値は、クライアント・エージェントによってサーバに送信され、ユーザ・データベースに格納されます。

* 一般に、ブランド名または商品名は各社の商標または登録商標です。

- サービス ID は各サービス・プロバイダによって異なります。ハッシュ・アルゴリズムには伝達性がないため、異なるウェブ・サイトの間でユーザ・プロフィールを関連させることはできません。

ハッシュ・アルゴリズムは、一方方向の衝突回避アルゴリズムとして設計されているため、ハッシュされた値とサービス ID からプロセッサ・シリアル番号を推定できません。また、インテルではウェブ・サービス・プロバイダに対し、プライバシーに関する方針を消費者に明らかにすることを推奨しています。

パフォーマンスに関する考慮事項

RI のもう 1 つの重要な利点は、エージェントからのダウンロード時間が短いことです。ソフトウェア・エージェントのサイズが大きいと、ユーザはアクセスできるまで長時間待たされることとなります。RI で使われるエージェントは、ダウンロード時間を短縮するため、約 35KB のサイズに抑えられています。しかし、保護機能の水準はエージェントのサイズに比例するため、サイズが大きいほど保護機能も高くなります。このバランスを考慮し、小さなエージェントで、十分な時間、攻撃から保護できるようにしました。保護機能を補助するため、エージェントは動的に更新され、サーバにはタイムアウト機構が採用されています。

リファレンス・インプリメンテーションのアーキテクチャ

RI のフレームワークは、クライアント・モジュール、サーバ・モジュール、クライアントとサーバ間の通信用プロトコルで構成されています。

クライアント・モジュールは、クライアント登録用の非保護モジュールである登録エージェントと、クライアント検証用の TRS 保護モジュールである検証エージェントの 2 種類のクライアント・エージェントで構成されています。各エージェントを構成する Java*アプレットとネイティブ・コード DLL は、デジタル署名の付いたコンテナにパッケージされています。

サーバ・モジュールは、ウェブ・サイトへのアクセスを提供するほか、クライアント・セッションの管理とクライアント・システムの認証を担当しています。このほか、ウェブ・サーバは、ユーザ名とパスワードによる登録コードをバックエンド・データベースに格納します。

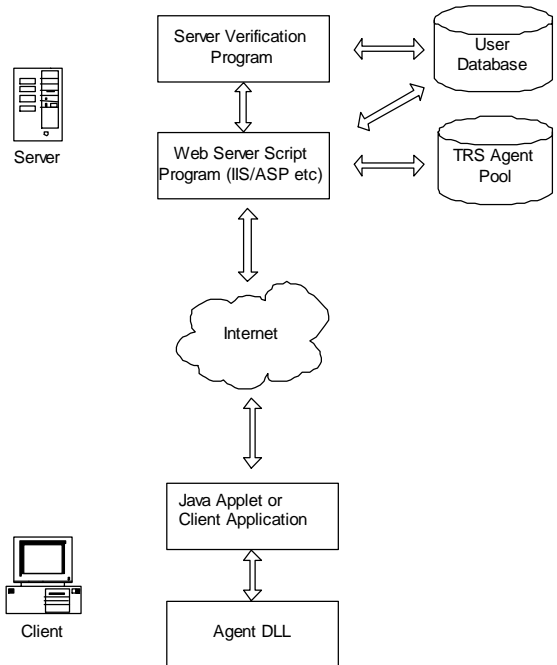


図2:リファレンス・インプリメンテーションのアーキテクチャ

システム検証の流れ

ユーザがウェブ・サーバにログオンすると、まず、サーバはユーザ ID とパスワードを要求し、登録エージェントをクライアント・システムにダウンロードします。登録エージェントは、ps#とサービス ID のハッシュである登録コードを計算してサーバに返し、サーバはそれを、ユーザ名とパスワードを使ってユーザ・データベースに格納します。次に、サーバは、エージェントのプールから無作為に選択した検証エージェントを送信します。どのエージェントにも不正防止機能があり、固有の非公開値が組み込まれています。この値は、あらかじめ指定した時間(通常は 10 分間)につき 1 回しか使用されません。エージェントはこの間、場数を踏んだハッカーの攻撃にも耐えられるように設計されています。検証エージェントは、クライアント・システムにダウンロードされると、検証コードを計算し、その検証コードをサーバに送信します。さらに保護機能を強化するため、このセッション中、指定時間内にクライアントから有効な検証コードが届かない場合には、サーバはタイムアウトします。

検証コードは、登録の場合と同様に、まず ps#とサービス ID をハッシュして計算されます。生成された登録コードは、検証エージェントに組み込まれている固有の非公開値を使ってさらにハッシュされます。これで検証コード

が生成され、ウェブ・サーバに送信されて検証を受けます。

サーバは、クライアントから有効な検証コードを受信すると、まず返された値を一時的に格納します。次に、サーバは、前に格納した登録コードを、クライアントに送信した検証エージェントに埋め込まれている固有の非公開値とハッシングし、認証コードを計算します。サーバは、この認証コードを検証コードと比較します。2つの値が一致すれば、クライアント・システムは認証され、ユーザは、コンテンツにアクセスしたり、希望するサービスを受けたりすることができます。

サポートされる環境

クライアント・エージェントは、Microsoft[®] Windows[®] プラットフォーム (Windows NT[®]、Windows[®] 98、Windows 95[®]) をサポートしています。RI は、Microsoft Internet Explorer[®] 4.0、Netscape Navigator[®] 4.x、AOL[®] の各ブラウザをサポートしています。また、単一プロセッサ、およびマルチプロセッサ構成のいずれのクライアント・システムにも対応します。マルチプロセッサ・システムの場合、利用できるプロセッサのうち、常に同じプロセッサから ps# が収集されます。同様のソフトウェアを開発して、Linux[®] などのクライアント・オペレーティング・システムをサポートすることもできます。サーバ環境については、リファレンス・インプリメンテーションは Windows NT と Unix[®] のいずれにも対応しています。

まとめ

Pentium[®] III プロセッサのシリアル番号機能は、この機能を備えたインテル製の各プロセッサに、アプリケーション・ソフトウェアで明確に取得できる固有の識別子を提供します。CPUID 命令を利用することにより、プロセッサの設計や将来のインプリメンテーションにほとんど影響を及ぼさずに、自然な方法でこの情報を提供できます。

この記事で述べたように、プロセッサ・ベースのシリアル番号機能は、さまざまな分野のアプリケーションで効果的な役割を果たします。特に注目される分野は、企業の資産管理、情報管理、e-Business の管理です。

Pentium III プロセッサの ps# 機能は、固有の識別子を生成するためのほかの方法とは異なり、ネットワーク・カー

* 一般に、ブランド名または商品名は各社の商標または登録商標です。

ドや IP アドレスなど、システム・ハードウェアやソフトウェアの設定変更による影響を受けません。この機能をプロセッサに組み込むと、次のような効果が期待できません。

- 固有の識別子によって不正防止機能が強化されるため、消費者やサービス・プロバイダ、コンテンツ・プロバイダの信頼が高まる。
- アプリケーション・レベルのソフトウェアで ps# を解読できる。通常のプロセッサ・ベースの方法では、特権レベルの命令 (PL0) を使うため、アプリケーション・レベルのコードでは利用できず、プラットフォーム間の互換性もない。

以上のように、Pentium III プロセッサの ps# 機能は、情報プロバイダや IT マネージャに大きな恩恵をもたらします。また、新しいアプリケーションがこの機能を利用するようになれば、その価値は消費者層へも広がります。

謝辞

プロセッサ・シリアル番号機能とアプリケーション・プログラムの発案、定義、改良に協力して頂いた方々に感謝します。とりわけ、Rob Sullivan、Ticky Thakkar、Natasha Oza、Vishesh Parikh、Jim Kolotorous、Susan Wojcicki、Peter Ruscito の各氏には、多大な貢献を賜りました。

参考資料

- [1] 『Pentium[®] Pro Family Developer's Manual, Volume 2: Programmers Reference Manual』、注文番号 000900-001、Intel Literature Sales, Mt. Prospect, IL、1996 年、pp.11-73 ~ 11-79
- [2] Mohsen Alavi、Mark Bohr、Jeff Hicks、Martin Denham、Allen Cassens、Dave Douglas、Min-Chun Tsai 著 『A PROM Element Based on Salicidic Agglomeration of Poly Fuses in a CMOS Logic Process』、1997 IEEE International Electron Devices Tech Digest、1997 年 12 月、pp.855 ~ 858

著者紹介

Stephen Fischer、インテル社フォルサム・デザイン・センターの設計エンジニア。Pentium[®] III プロセッサのマイクロコードおよび P6 マイクロアーキテクチャ関連の設計を担当。EISA チップ・セット、PCI バス/PCI チップ・セット、および Intel[®] MMX[®] テクノロジーの定義など、各種プログラムに参加した経験を持つ。1985 年にカリフォル

ニア州立大学サクラメント校でコンピュータ・サイエンスの理学士号を取得。現在、米国で6件の特許を保有。

電子メール・アドレス：sfischer@pcocd2.intel.com

James Mi、インテル社コンテンツ・グループのイネープリング・テクノロジー部門のマネージャ。アプリケーションのアーキテクチャと開発を担当。コンテンツ・グループ、TCAD、フラッシュ TD でマーケティング、ソフトウェア開発、ハードウェア開発の経験を持つ。1989年に中国フダン大学で物理学の理学士号、1991年にプリンストン大学で電気工学の理学修士号を取得。1992年にインテルに入社。米国で7件の特許を保有。

電子メール・アドレス：james.mi@intel.com

Albert Teng、ニュー・テクノロジー部門ディレクタ。企業/電子商取引ソリューション、セキュリティ、ナレッジ・マネジメント向けのクライアント・サーバ・アプリケーションを担当。マイクロコンピュータ研究所の技術管理職、インテル・チャイナのジェネラル・マネージャを歴任。1985年にインテルに入社するまでは、AT&T ベル研究所とイリノイ工科大学に勤務。1979年にオハイオ州立大学で博士号を取得。

電子メール・アドレス：albert.y.teng@intel.com