Windows のメモリ管理の進歩

2007 年 10 月 12 日

機械翻訳に対する免責事項

本書は、英語で作成されたホワイト ペーパーを、機械翻訳システムにより日本語に翻訳したものであり、日本語をお使いになるお客様が英語原文を理解する上での便宜として、参考用に提供するものです。

Microsoft は和訳の正確さについて、一切保証するものではありません。また、本和訳文書の使用によって、直接または間接的に起こりうる一切の問題について、Microsoft はいかなる責任も負わないものとします。

要約

このペーパーでは、Windows Vista® とWindows Server® 2008 でのメモリ管理の進歩に関する情報を提供します。Microsoft がオペレーティング システム内部に実装した変更点について説明し、これらの進歩を活用するためのガイドラインを、アプリケーション開発者、ドライバ作成者、およびハードウェア メーカーに提供します。

このペーパーは、Mark Russinovich および David Solomon 共著の『Windows Internals』で説明されているような、Windows のメモリ管理の基礎を理解していることを前提としています。

この情報は、下記のオペレーティング システムに適用されます。

 Windows Server 2008
 Windows Vista

このペーパーの最新バージョンは、次の Web サイトに掲載されています。
 <http://www.microsoft.com/whdc/system/cec/MemMgt.mspx>

**フィードバック :** このペーパーが有用かどうか、フィードバックをお送りください。コメントは、下記の URL から送信できます。
 <http://connect.microsoft.com/Survey/Survey.aspx?SurveyID=4925&SiteID=221>

ここで説明しているリファレンスやリソースの一覧は、このペーパーの最後に掲載されています。

目次

[はじめに 4](#_Toc187723148)

[メモリ マネージャについて 4](#_Toc187723149)

[仮想アドレス スペース 5](#_Toc187723150)

[カーネル仮想アドレス スペースの動的な割り当て 5](#_Toc187723151)

[x86 アーキテクチャの詳細 6](#_Toc187723152)

[64 ビット アーキテクチャの詳細 7](#_Toc187723153)

[x86 アーキテクチャでのカーネル モード スタック ジャンプ 7](#_Toc187723154)

[余剰プール メモリの使用 8](#_Toc187723155)

[セキュリティ : Address Space Layout Randomization 9](#_Toc187723156)

[イメージ読み込みアドレスに対する ASLR の影響 9](#_Toc187723157)

[ASLR の利点 10](#_Toc187723158)

[動的ベースのイメージを作成する方法 11](#_Toc187723159)

[I/O 帯域幅 11](#_Toc187723160)

[Microsoft SuperFetch 12](#_Toc187723161)

[ページ ファイルへの書き込み 12](#_Toc187723162)

[メモリ マネージャとキャッシュ マネージャの連携 13](#_Toc187723163)

[プリフェッチ スタイルのクラスタリング 14](#_Toc187723164)

[大きなファイルの管理 15](#_Toc187723165)

[休止状態とスタンバイ 16](#_Toc187723166)

[高度なビデオ モデル 16](#_Toc187723167)

[NUMA のサポート 17](#_Toc187723168)

[リソースの割り当て 17](#_Toc187723169)

[既定のノードおよびアフィニティ 18](#_Toc187723170)

[割り込みの関係 19](#_Toc187723171)

[アプリケーション用の NUMA 対応システム関数 19](#_Toc187723172)

[ドライバ用の NUMA 対応システム関数 19](#_Toc187723173)

[ページング 20](#_Toc187723174)

[スケーラビリティ 20](#_Toc187723175)

[効率と並列処理 20](#_Toc187723176)

[ページ フレーム番号と PFN データベース 20](#_Toc187723177)

[大容量ページ 21](#_Toc187723178)

[キャッシュの境界で整列されたプール割り当て 21](#_Toc187723179)

[仮想マシン 22](#_Toc187723180)

[負荷分散 22](#_Toc187723181)

[その他の最適化 23](#_Toc187723182)

[システムの整合性 23](#_Toc187723183)

[ハードウェア エラーの診断 23](#_Toc187723184)

[コードの整合性およびドライバ署名 24](#_Toc187723185)

[バグ チェック中のデータの保存 24](#_Toc187723186)

[行うべきこと 24](#_Toc187723187)

[ハードウェア メーカーの場合 24](#_Toc187723188)

[ドライバ開発者の場合 24](#_Toc187723189)

[アプリケーション開発者の場合 25](#_Toc187723190)

[システム管理者の場合 25](#_Toc187723191)

[リソース 25](#_Toc187723192)

免責事項

本書には、今後変更される可能性のある予備情報も含まれます。

本書に記載されている情報は本書の発行時点における Microsoft の見解を述べたものです。市場ニーズの変化に対応する必要があるため、本書は記載された内容の実現に関する Microsoft の確約とはみなされないものとします。また本書に記載された情報の正確さについて、保証するものではありません。

本書は情報の提供のみを目的としており、明示または黙示に関わらず、本書について Microsoft はいかなる保証をするものでもありません。

適用される著作権法に従うことはお客様の責任です。本書の一部または全部を、電子的、機械的、複写、録音、その他いかなる手段およびいかなる形式によっても、またいかなる目的のためにも、Microsoft の書面による許可なく複製、転送、または検索システム等へ格納等することは禁じられています。ただし、これによってお客様に著作権法上認められる権利を制限するものでもありません。

Microsoft は本書に記載されている内容に対して、特許権、特許出願、商標権、著作権、またはその他の知的所有権を有する場合があります。本書は Microsoft の書面による明示的な許諾がある場合を除き、これらの特許権、商標権、著作権またはその他の知的所有権に関する権利をお客様に許諾するものではありません。

別途記載のない限り、本書中に記載されたサンプルの会社名、団体名、製品名、ドメイン名、氏名、電子メール アドレス、ロゴ、人、場所、出来事等は架空のものであり、実在の会社、団体、製品、ドメイン名、電子メール アドレス、ロゴ、人、場所または出来事との関連を示唆するものではありません。

© 2007 Microsoft Corporation. All rights reserved.

Microsoft、MSDN、SuperFetch、Visual Studio、Windows、Windows Server、Windows Vista は、米国 Microsoft Corporation およびその関連会社の米国およびその他の国における登録商標または商標です。

本書中で使用されている実在の会社名や製品名には、各権利者の保有する商標が含まれることがあります。

# はじめに

Microsoft は、Windows Vista® と Windows Server® 2008 でのメモリ管理に対し、大幅な強化機能を実装しました。これらの変更により、機能が追加され、下記の領域でパフォーマンスが向上します。

* 仮想アドレス (VA) スペースのより効率的な使用。
* セキュリティの強化。
* I/O 帯域幅のより効率的な利用。
* より高速の休止状態/スタンバイおよび再開。
* Windows Vista の高度なビデオ モデルのサポート。
* NonUniform Memory Access (NUMA) アーキテクチャのサポート。
* サーバー ハードウェアとアプリケーションのより良いスケーラビリティ。
* より優れたシステムの整合性。

メモリ管理の変更点の多くは、アプリケーションやドライバに対し透過的です。したがって、既存のコードは、変更なしで動作します。ただし、一部の変更点を活用するには、このペーパーで説明するように、開発者は、アプリケーションを変更するか再リンクする必要があります。

# メモリ マネージャについて

メモリ マネージャは、オペレーティング システムの物理メモリと仮想メモリの割り当ておよび管理を処理します。メモリ マネージャが提供する最も重要なサービスは、次のとおりです。

* ページ メモリ プール、非ページ メモリ プール、システム キャッシュなど、主なシステム リソースの管理。
* 物理メモリへの VA スペースのマッピング。
* ページング。
* 各プロセスのアドレス スペースの、互いのスペースやオペレーティング システム自体からの保護。

図 1 が示すように、メモリ マネージャは、I/O マネージャやキャッシュ マネージャと連携し、必要なデータにプロセスが迅速にアクセスできるようにします。



図 1. メモリ マネージャ、I/O マネージャ、およびキャッシュ マネージャ

図 1 が示すように、ファイル システムは、アプリケーションから I/O 要求を受信し、I/O マネージャまたはキャッシュ マネージャを呼び出して、要求を処理します。I/O マネージャは、デバイスとアプリケーション間の相互作用を処理します。I/O マネージャとアプリケーションはどちらも、メモリ マネージャを呼び出し、ドライバのためにファイルをマップし、内部使用用にメモリを割り当てます。メモリ マネージャは、マップされたファイルに対する後続のすべてのアクセスでの要件に合わせて、ページ フォールトを処理します。キャッシュ マネージャは、高速 I/O とキャッシュされた I/O の両方に対し、ファイル システムとメモリ マネージャ間のインターフェイスを提供します。キャッシュ マネージャは、カーネル VA スペースの一部を割り当て、キャッシュされた I/O アクセス パターンに基づき、ファイルのビューをマップします。

# 仮想アドレス スペース

Window Vista では、VA スペースをより効率的に使用し、管理を簡略化し、より多数のプロセッサやより大きなメモリ構成に対しスケーラビリティを向上させるため、重要な変更点を実装します。これらの変更により、レジストリ サイズ、構成、および Stock Keeping Unit (SKU) に基づく違いが大幅になくなります。VA スペースの主な変更点は、次のとおりです。

* カーネル仮想アドレス スペースの動的な割り当て
* x86 アーキテクチャでのカーネル モード スタック ジャンプ
* 余剰プール メモリの使用

## カーネル仮想アドレス スペースの動的な割り当て

Windows Vista およびそれ以降の Windows リリースでは、カーネル VA スペースは動的に割り当てられます。ページ メモリ プールや非ページ メモリ プールなど、重要なシステム リソースのサイズと場所は修正されなくなりましたが、その代わり、運用要件に従い動的に調整されます。その結果、システム チューニングは自動で行われます。管理者は、リソースのアンバランスを防ぐため、システムを手動で再構成する必要は、通常ありません。

図 2 は、カーネル VA スペース割り当ての変更の影響を示します。



図 2. カーネル仮想アドレス スペース

図 2 が示すように、以前の Windows リリースでは、カーネル VA スペースは静的に割り当てられていました。リソースは、固定のサイズおよび場所に割り当てられました。静的割り当てでは、メモリ プール、システム ページ テーブル、および他のシステム リソースのサイズに、人為的な制限がありました。メモリ プールなど、一部のリソースのサイズは、レジストリ キーまたは SKU によって設定されました。

Windows Vista およびそれ以降の Windows リリースでは、カーネル VA スペースは、ユーザー アドレス スペースと同様に、動的に割り当てられます。カーネル VA スペースは、アーキテクチャで利用可能な仮想メモリのサイズによってのみ制限されます。個々のリソースのサイズと場所は、現在のシステム必要条件に従って変更できます。したがって、システム リソースは、VA スペース内の固定の場所には存在しなくなりました。Windows Vista は、リソース サイズを決定するため以前の Windows バージョンがブート時に使用していた、レジストリ キーの値を無視します。動的な割り当ての結果、すべてのリソースは、すべてのリクエスタで利用可能です。

システムがブート時に事前に割り当てる代わりに、必要な VA スペースを動的に割り当てることができるので、特別なプールや Driver Verifier などの機能は、再起動しなくても有効にできます。Windows Vista およびそれ以降の Windows リリースでは、そのような機能は、使用されていない場合、メモリをまったく必要としません。したがって、機能は、使用されていない場合、”解放” されています。一方、以前の Windows バージョンでは、そのような機能が有効な場合、使用されていなくても、カーネル VA スペースがブート時に割り当てられました。

### x86 アーキテクチャの詳細

32 ビット Windows では、完全なカーネル VA スペースは、システムが使用しているすべてのリソースによって共有されます。個々のリソースは、非ページ プールを除き、事前に設定されたサイズ制限がありません。非ページ プールは、物理メモリの 75% に制限されています。標準構成でブートされた 32 ビット システムには、完全な 2 GB のカーネル VA スペースがあり、各リソースが共有します。管理者は、bcdedit コマンドを使用して、ユーザー VA スペースのサイズを増やすことができます。それにより、カーネル VA スペースのサイズが、同じ分だけ減ります。

システム リソースの割り当ては現在、ブート時ではなく動的に決定されるので、3 GB のユーザー VA スペースと物理アドレス拡張 (PAE) の構成でブートされる 32 ビット システムは、64 GB のメモリ全体を現在使用できます。以前の 32 ビット Windows バージョンでは、3 GB のユーザー VA スペースと PAE の構成でブートされたとき、16 GB のメモリ制限がありました。

開発者と管理者は、カーネル モード デバッガを使用して、x86 アーキテクチャでのカーネル アドレス スペース使用に関する情報を取得できます。フラグ値が 0x21 の **!vm** カーネル モード デバッガ拡張機能は、プロセス固有の情報なしで、カーネル VA スペースの使用に関する情報を表示します。このコマンドは、アドレス スペースの断片化の検査に役立つ場合があります。32 ビット システムではカーネル VA スペースが大幅に少ないので、このコマンドは、64 ビット システムより 32 ビット システムに大きな影響を与えます。

32 ビット アーキテクチャではまれに、著しい断片化により、カーネル VA スペースがなくなる場合があります。32 ビット バージョンの Windows Vista SP1 と Windows Server 2008 は、そのようなシステムでのリソース サイズを管理者が制限できる、一連のレジストリ キーをサポートします。これらのキーは、64 ビット システムでは無視されます。詳細については、MSDN® で「Memory Management Registry Keys」を参照してください。

### 64 ビット アーキテクチャの詳細

64 ビット Windows では、コンピュータ内の物理メモリの量にかかわらず、非ページ プールを除き、各リソースで、1 TB のシステム キャッシュと共に、128 GB が利用可能です。非ページ プールは、Windows Server 2008 では物理メモリの 75%、Windows Vista および以前の Windows リリースでは 40% に制限されています。

例として、512 MB の RAM を持つ 64 ビット システムを検討します。Windows XP では、そのようなシステムのページ プールは比較的小さく、1 GB 前後です。ただし、1 TB の RAM を持つシステムでは、ページ プールははるかに大きく、最大 128 GB です。Windows Vista では、ページ プールは、物理メモリのサイズにかかわらず、512 MB のシステムと 1 TB のシステムの両方において、128 GB です。管理者は、大量のページ プールを使用するアプリケーションを実行するため、システムを再構成したり、レジストリ設定を変更したりする必要がなくなりました。

## x86 アーキテクチャでのカーネル モード スタック ジャンプ

Windows Vista は、カーネルへの再帰呼び出しでカーネル モード スタックのスペースをより効率的に使用することにより、x86 アーキテクチャのカーネル VA スペースの可用性をさらに向上させます。スレッドが Windows API システム呼び出しを実行し、32 ビット カーネルに切り替わる場合、Windows Vista メモリ マネージャは、最初のスタックに連結している新しい 16 KB スタックを生成して、カーネル モード スタック領域を提供します。スレッドがシステム呼び出しから解放されると、メモリ マネージャは、スタックを切り離し削除します。“カーネル モード スタック ジャンプ” と呼ばれるこの機能は、スレッドで必要となる VA スペースの量を減らし、したがって、スレッドのメモリ フットプリントのサイズが減ります。カーネル モード スタック ジャンプは、各ターミナル サーバー クライアントでのより効率的なメモリ使用につながり、したがって、各コンピュータで、通常 2 倍から 4 倍の数のクライアントを実行できます。

図 3 は、x86 アーキテクチャのカーネル モード スタック領域の使用において、以前の Windows バージョンと Windows Vista がどう異なるかを示します。



図 3. 32 ビット Windows Vista でのカーネル モード スタック ジャンプ

以前の Windows バージョンでは、システム呼び出しを実行するスレッドは、拡張された 64 KB の仮想スタックを受け取ります。ただし、4 KB のガード ページを含む 12 KB のみが、各システム呼び出しで使用可能です。スレッドは、引き続きスタック領域を使用するかどうかにかかわらず、終了するまで、このスタックを保持します。そのようなスレッドでは、スレッドがカーネル モード コードを実行していない場合、追加の領域は使用されません。

Windows Vista の 32 ビット バージョンでは、カーネルへの切り替えにつながるコールバックを Win32k.sys が発行するごとに、スレッドは、16 KB のスタックを追加で受け取ります。入れ子になっている各システム コールバックが完了すると、メモリ マネージャは、関連付けられたスタックを削除します。

追加のカーネル モード スタック領域を必要とするドライバは、**KeExpandKernelStackAndCallout** 関数の使用により、この機能を活用できます。この機能により、ドライバは、スタック領域のサイズを指定できます。このスタックを使用して、システムは特定のコールバック関数を呼び出し、スタックの最大サイズは、Ntddk.h で MAXIMUM\_EXPANSION\_SIZE の値として定義されています。現在のスタックに十分な領域がない場合、必要に応じてシステムは、1 つ以上の追加のスタックを割り当て、指定したドライバ コールバックが戻ると、スタックを削除します。**KeExpandKernelStackAndCallout** は、64 ビット アーキテクチャ用の Windows Server 2003、およびすべてのアーキテクチャ用の Windows Vista でサポートされています。

## 余剰プール メモリの使用

カーネル モード コンポーネントが 1 ページより大きいメモリを割り当てると、メモリ マネージャは現在、その割り当ての終端と次のページ境界の間のプール メモリを使用して、他のメモリ要求を満たします。

ドライバは、任意の割り当ての終端を越えて、メモリにアクセスしてはなりません。Driver Verifier は、長年このエラーをチェックしてきました。したがって、適切にテストされた既存のドライバでは、問題は生じないはずです。しかし、ドライバ作成者は、この変更点に注意する必要があります。ドライバは、割り当てられた領域の終端を越えたメモリを使用すると、別のコンポーネントに割り当てられたメモリや、そのドライバ自体に割り当てられたメモリの内容を破損させる可能性があります。

図 4 は、Windows Vista でのプール メモリの割り当てが、以前の Windows リリースとどう異なるかを示します。



図 4. 余剰プールの割り当て

図 4 では、ドライバは N バイトを割り当てています。N は、1 ページより大きく、ページ サイズの整数倍より小さな値です。以前の Windows リリースでは、その割り当ての終端と次のページ境界の間のメモリは、使用されず余ります。しかし、Windows Vista およびそれ以降のリリースでは、メモリ マネージャは、そのようなメモリを他のリクエスタに割り当てることができます。

# セキュリティ : Address Space Layout Randomization

システム セキュリティを向上させ、バッファ オーバーランの悪用で生じる可能性のある障害を制限するため、Windows Vista は、DLL や実行可能ファイルのイメージを、読み込みを行うごとに、異なるアドレスに読み込みます。ASLR (Address Space Layout Randomization) と呼ばれるこの技法は、悪意のあるアプリケーションが特定のシステム API の場所を予測するのを、はるかに困難にします。

開発者は、**/DYNAMICBASE** リンカ オプションを使用して、DLL と実行可能ファイルを ASLR がサポートするよう、明示的に指定する必要があります。このオプションは、フラグをイメージ ヘッダーに設定し、ランダムに選択したアドレスにイメージを読み込むことができることを示します。ソフトウェア メーカーに対しては、自社製品で ASLR をサポートするよう強くお勧めします。

## イメージ読み込みアドレスに対する ASLR の影響

Windows Vista では、スタートアップ時にメモリ マネージャは、ユーザー モード アドレス スペースの最上部にある、256 個の 64 KB 境界のアドレスのいずれかから、イメージ読み込みバイアスをランダムに選択します。イメージ読み込みバイアスは、ASLR をサポートする DLL (動的ベースの読み込み用フラグ付きの DLL) をメモリ マネージャが読み込む領域の開始アドレスです。同じイメージ読み込みバイアスが、システム上のすべてのユーザー モード プロセスに適用されます。その結果、各プロセスは、動的ベースの DLL を共有できます。

以前の Windows バージョンでは、メモリ マネージャは、DLL ヘッダーで指定された読み込みアドレスを使用して、すべてのプロセスで毎回同じ場所に DLL を読み込もうとしていました。これは、アドレス スペースの競合が、プロセス内で起きないことを前提としています。その結果、ハッカーは、Microsoft またはサード パーティの製品でバッファ オーバーランのバグを見つけたとき、これらのイメージの内部にある特定関数の既知のリンク アドレスを使用して、攻撃を開始できました。

図 5 は、仮説的な 32 ビット Windows Vista システム内の読み込みアドレスに対する、ASLR の影響を示しています。



図 5. 読み込みアドレスに対する ASLR の影響

図 5 で、メモリ マネージャは、最初の動的ベースの DLL を、ランダムに選択したイメージ読み込みバイアスに読み込みます。その後、アドレス スペースの上位に向かって、残りの動的ベースの DLL にアドレスを割り当てます。**/DYNAMICBASE** を使用してリンクされる各 DLL は、それらを使用するすべてのプロセスで、同じアドレスに読み込まれます。したがって、それらのプロセスで、コード共有が可能になります。その際、十分な VA スペースが利用可能で、VA スペースの競合が各プロセス内で起きないことが前提となります。実行可能ファイル イメージの場合、メモリ マネージャは、同様なパターンに従って、イメージ ヘッダーに格納されているベース読み込みアドレスに近い、ランダムに選択した 64 KB 境界のアドレスを選択します。各プロセスは、DLL を共有するのと同じ方法で、実行可能ファイルを共有できます。

DLL または実行可能ファイルのイメージが、それを使用しているすべてのプロセスでアンロードされると、メモリ マネージャは、次にそれが必要となった場合、同じアドレスにそれを読み込むとは限りません。その代わり、メモリ マネージャは、再びランダムに読み込みアドレスを選択します。

Windows Vista は、以前の Windows バージョンと同じ方法で、ASLR をサポートしない DLL を読み込みます。具体的には、システムは、イメージ ヘッダーで指定されたアドレスに、イメージを読み込もうとします。別の DLL がその場所に既に読み込まれているため競合が生じる場合、システムは、プロセスで利用可能な最高位のアドレスから探し始め、アドレス スペースを下降して、イメージを読み込むことができる場所を見つけます。追加のプロセスが同じ DLL を読み込むとき、システムは、それらのプロセスで、同じアドレスにイメージを読み込みます。後続のプロセスで競合が生じた場合、システムは、プリロードされた DLL を再配置します。これにより、DLL を読み込むにつれ、”ドミノ現象” が生じる可能性があります。

## ASLR の利点

ASLR は、読み込みアドレスを予測できなくすることにより、セキュリティを向上させます。さらに、ASLR は、以前の Windows バージョンより、狭い範囲の VA に各 DLL を配置します。これにより、競合を避け、ページ テーブル領域を節約し、より隣接した VA スペースをアプリケーション用に残すことができます。動的ベースの DLL は、それらを使用する各プロセスで、引き続き共有されます。

ASLR は、Windows Vista と Windows Server 2008 のユーザー モード実行可能ファイルと Microsoft 提供の DLL に適用され、Windows Server 2008 のカーネル、ハードウェア アブストラクション レイヤ (HAL)、およびドライバ コンポーネントに拡張されます。ドライバは、自動的に動的に再配置されますが、ユーザー実行可能ファイルや DLL は、**/DYNAMICBASE** リンカ オプションを使用して、明示的に指定する必要があります。

ASLR は、実行不可の保護と連携して、バッファ オーバーフローなど、アプリケーションの脆弱性を利用しようとするマルウェアに対する、強固なセキュリティを提供します。ASLR は、それだけでは、バッファ オーバーランに対し、比較的弱いセキュリティ対策です。システム ルーチンのアドレスを判定できないハッカーは、コードを注入し、スタックから実行できます。実行不可の保護も、同様に弱い手段です。これは、コードの実行を防ぎますが、ハッカーがシステム関数の既知のアドレスを使用するのを防ぐことができません。ただし、連携して使うと、ASLR および実行不可は、非常に強力です。ハッカーは、悪意のあるコードをスタックから実行することも、DLL の既知のアドレスから実行することもできません。

一般に、ASLR は、カーネル メモリ管理レベルで実装されるので、パフォーマンスに影響を及ぼしません。32 ビット システムでは、パフォーマンスが若干向上する場合があります。これは、コードの共有と、アドレス スペースのより効率的な使用によるものです。ただし、イメージの数とサイズによっては、多くのランダム イメージと共に読み込まれる高密度の 32 ビット システムで、パフォーマンスが低下する場合があります。

## 動的ベースのイメージを作成する方法

既定で、ASLR は、すべてのシステム DLL と EXE に適用されます。これらはすべて、**/DYNAMICBASE** を使用してリンクされます。ソフトウェア メーカーに対しては、ASLR をサポートするよう強くお勧めします。動的ベースのイメージを作成するには、開発者は、/DYNAMICBASE オプションを使用して、アプリケーションを再リンクする必要があります。このオプションは、バージョン 8.00.50727.161 以降の Microsoft Linker で利用でき、Microsoft® Visual Studio® 2005 SP1 でサポートされています。このスイッチは、以前の Windows バージョンでは無視されます。したがって、下位互換性の問題は生じません。

ASLR と実行不可の両方を使用するには、開発者は、**/DYNAMICBASE** だけでなく、**/NXCOMPAT** フラグを使用する必要があります。

ASLR と実行不可保護の詳細については、MSDN で 「Windows Vista ISV Security」を参照してください。

# I/O 帯域幅

プロセッサ速度とメモリ サイズは、この数年で、数倍増加しました。ただし、I/O 速度やディスク速度は、固体ドライブなど最近の機能向上により傾向は変わりつつありますが、おそらく 2 倍がいいところです。I/O サブシステムが単に追いつかないので、プロセッサやメモリの向上点の多くは、十分利用されていません。I/O 速度の向上は、Windows Vista の重要な課題であり、下記の領域での変更を伴いました。

* Microsoft SuperFetch™
* ページ ファイルへの書き込み
* メモリ マネージャとキャッシュ マネージャの連携
* プリフェッチ スタイルのクラスタリング
* 大きなファイルの管理
* 休止状態とスタンバイ

## Microsoft SuperFetch

SuperFetch は、データ使用のパターンを分析し、それらのパターンによりデータをメモリにプリロードする、Windows Vista の適応型ページ プリフェッチ技法です。

エンド ユーザーが作業に取り掛かると、メモリ マネージャは、活動に関する情報をログに記録します。ユーザー モード SuperFetch サービスは、ログを分析し、使用のパターンを見つけ出し、システム全体においてよく使用されるページの優先度を高くします。その後、サービスは、メモリ マネージャを呼び出し、定期的な使用に基づき、適切なときに、それらのページをプリロードします。メモリ マネージャは、プロセッサが他の作業で使用されておらず、ディスクが使用中でない場合、低優先度でプリロードします。

たとえば、金曜正午に給料支払処理を実行するため、定期的に使用されるシステムを考えてみます。メモリ マネージャは、給料支払アプリケーションで必要なページに関する情報をログに記録します。また、SuperFetch サービスは、これらのページが定期的に使用されているので、正午ごろの読み込みに対して優先度を高くするよう認識します。その結果、仮想メモリがより効率的に使用され、エンド ユーザーのパフォーマンスが向上します。

SuperFetch の詳細については、「Windows PC Accelerators」を参照してください。SuperFetch は、Windows Server 2008 ではサポートされません。

## ページ ファイルへの書き込み

Windows Vista には、ページ ファイルへの書き込みをより高速でより効率的にする、多くの拡張機能が含まれています。Windows Vista で、メモリ マネージャは、より多くのデータを各操作で書き込み、ページを近隣のページと揃え、完全にゼロであるページは書き込みません。

以前の Windows バージョンでは、書き込み操作のサイズは、64 KB に制限されていました。その結果、メモリ マネージャは、単一の操作でページ ファイルに、最大 64 KB を書き込むことができました。Windows Vista では、この 64 KB の制限がなくなりました。したがって、メモリ マネージャは、はるかに大きなクラスタをページ ファイルに書き込むことができます。書き込み操作は現在、通常は 1 MB です。

以前のアルゴリズムは、データがディスクにできるだけ速く書き込まれるよう設計されていました。しかし、Windows Vista では、目標は、速いだけでなくスマートに書き込むことです。データの読み書き用にディスク ヘッドを移動させるのは、比較的時間のかかる操作です。したがって、そのような操作の回数を減らすと、I/O 帯域幅の向上に役立ちます。メモリ マネージャは、変更されたページをバッキング ストアに書き込む準備をする際、近隣のページも検査して、それらがワーキング セットからマップ解除またはトリミングされたかどうかにかかわらず、そのすべてを書き込む必要があるかどうか判断します。したがって、書き込み操作には、近隣のページのクラスタが含まれています。その一部はワーキング セット内にまだ存在し、他のページは存在しません。このように書き込みをクラスタ化することにより、メモリ マネージャは、ディスク上のデータがより隣接するようにして、同じデータに対する後続のページイン要求を高速にするのに役立ちます。

さらに、メモリ マネージャは現在、読み取りや書き込みを行う前に、ゼロで埋まっているページをチェックします。内部トレースによると、場合によっては、書き込み操作の 7% から 8% が、すべてゼロのページに関連していたことを示しています。そのようなページを書き込むため (読み取る可能性もあります)、貴重な I/O 帯域幅を使用する代わりに、メモリ マネージャは単に、ゼロ ページの一覧にページを追加して、ページ テーブルで “demand-zero” としてページをマークします。対応する VA が後でアクセスされた場合、メモリ マネージャは、ディスクからゼロ ページに読み取る代わりに、メモリ内のゼロ ページを使用します。

Windows Server 2008 と Windows Vista SP1 は、適切な物理メモリを使用してシステム上の変更ページの書き込みを遅らせることにより、追加の I/O 帯域幅を保持します。変更された (“ダーティ”) ページの決定済みしきい値を使用する代わりに、これらの Windows リリースは、利用可能な物理メモリの量やディスクの現在の電源状態を考慮するアルゴリズムに基づき、変更ページを書き込みます。このアプローチを用いると、必要なディスク I/O 操作が減ります。また、ラップトップのディスクに電力を供給する頻度を減らすことにより、ラップトップ バッテリの寿命を延ばすことができます。

## メモリ マネージャとキャッシュ マネージャの連携

Windows Vista は、書き込み操作において、メモリ マネージャとキャッシュ マネージャのより緊密な連携をサポートします。メモリ マネージャとキャッシュ マネージャは、ワーカー スレッドを使用して、変更されたページをバッキング ストアに書き込みます。メモリ マネージャは、そのような 2 つのスレッドを使用します。1 つは、変更されたページの書き込みスレッドで、もう 1 つは、マップされたページの書き込みスレッドです。キャッシュ マネージャは、複数の遅延書き込みスレッドを使用します。遅延書き込みスレッドは、バッキング ストアに書き込まれるシステム キャッシュから、ダーティ ページを定期的にキューに入れます。Windows Vista は、これらのスレッドの動作を調整して、シーク操作を最適化し、待機時間を減らし、I/O 操作の重複を最小にします。追加の拡張機能は、並列操作を提供し、可能な場合は常に、ロックの使用を避けます。

変更ページの書き込みスレッドは、変更ページの数、利用可能なページの数、および既存のキャッシュ ページの再利用の組み合わせに基づき、必要に応じて、メモリからページング ファイルに、ダーティ ページを書き込みます。

マップされたページの書き込みスレッドは、マップ ファイルからバッキング ストアに、ダーティ ページを一定間隔で書き込みます。Windows Vista およびそれ以降の Windows リリースでは、マップされたページの書き込みスレッドは、ダーティ ページの一覧を一定間隔でスイープし、すべてのページをバッキング ストアにフラッシュします。空きページの数が少ない場合、システムは、より短い間隔を使用して、スイープを加速します。以前の Windows リリースでは、マップされたページの書き込みスレッドは、絶対値である 5 分間隔で、すべてをフラッシュしました。Windows Vista は、以前の Windows リリースより、ダーティ ページを早く書き込み、書き込み操作は通常、より少ないデータで済みます。

キャッシュ マネージャの遅延書き込みスレッドは、バッキング ストアに書き込まれるシステム キャッシュから、ダーティ ページを定期的にキューに入れます。遅延書き込みスレッドは、マップされたページの書き込みスレッドと連携し、整合性のとれた方法で、書き込み要求をファイル システムに発行します。具体的に言うと、システムは、変更ページ一覧内の順序に基づいてダーティ ページを書き込む代わりに、可能な限り、最初のページから最後のページへ線形順序で、ページをバッキング ストアに書き込みます。このアプローチは、大きなスパース ファイルの場合、非常に効率的です。

長さが 1 MB から 200 MB に拡張されたディスク ファイルを考えてみます。新しいページの一部にはデータが含まれており、他のページには含まれていません。以前の Windows バージョンでは、たとえば 180 MB の位置にある、ファイルの末尾近くのページが、まず書き込まれる場合があります。ファイル システムは、1 MB から 180 MB のディスク ブロックを、ゼロで埋めます。これは、間にあるブロックに追加の変更ページのデータが書き込まれる前に、システムがクラッシュした場合のファイル破損を防ぐためです。次に、ファイル システムは、独立で機能しているマップされたページの書き込みスレッドと遅延書き込みスレッドから、要求が到着した任意の順序で、追加のブロックを書き込みます。Windows Vista では、マップされたページの書き込みスレッドと遅延書き込みスレッドが連携して、すべての変更ページを、1 MB から 180 MB の順に書き込みます。いずれにせよ、これらのページはすべて、最終的には書き込む必要があります。線形順序でページを書き込むと、ディスク上の読み取り/書き込みヘッドが、より効率的に移動します。また、変更されたページが既に存在する、間にあるブロックをゼロで埋めるという、時間のかかる操作を避けることができます。

また、メモリ マネージャは現在、複数の非同期フラッシュもサポートし、パフォーマンスを上げます。書き込みスレッドは、重複する複数の書き込み要求をすべてのデータに対して発行し、そのような要求がすべて完了するのを待機します。以前の Windows バージョンでは、スレッドは、要求を一度に 1 つずつ、連続的に発行していました。非同期要求の使用により、書き込み操作は、可能な場合、並列で実行できます。その間、書き込みスレッドは、個々の書き込み要求の I/O 速度に束縛されるのではなく、プロセッサとメモリの速度で引き続き動作します。

## プリフェッチ スタイルのクラスタリング

メモリ マネージャは、ページの大きなクラスタをプリフェッチして、ページ フォールトを満たし、システム キャッシュを事前設定します。プリフェッチ操作は、仮想メモリ内のワーキング セットにではなく、システムのページ キャッシュにデータを直接読み込みます。したがって、プリフェッチされたデータは、VA スペースを消費せず、フェッチ操作のサイズは、利用可能な VA スペースのサイズに制限されません。プリフェッチされたページは、スタンバイ一覧に置かれ、ページ テーブル エントリ (PTE) において移行中とマークされます。プリフェッチされたページが後で参照された場合、メモリ マネージャは、ワーキング セットにそれを追加します。ただし、ページが参照されなかった場合、それを解放するのに、システム リソースは不要です。

プリフェッチされたクラスタ内の任意のページが既にメモリ内にある場合、メモリ マネージャは、それらを再び読み取りません。代わりに、図 6 で示すように、メモリ マネージャはダミー ページを使用して、それらを表します。



図 6. プリフェッチ スタイルのクラスタリング

図 6 で、物理ページ自体は必ずしも隣接していませんが、ページ A、Y、Z、および B に対応するファイル オフセットと VA は、論理的に隣接しています。ページ A と B は非常駐のため、メモリ マネージャはそれらを読み取る必要があります。ページ Y と Z は、メモリ内に既に常駐しているため、読み取る必要はありません。(実際、それらは、バッキング ストアから最後に読み取られた時点から、既に変更された可能性があります。その場合、それらの内容を上書きしてしまう重大なエラーになります。) ただし、単一の操作でページ A と B を読み取れば、最初にページ A を読み取り、次にページ B を読み取るより効率的です。したがって、メモリ マネージャは、バッキング ストアから 4 つのページ (A、Y、Z、および B) をすべて読み取る、単一の読み取り要求を発行します。そのような読み取り要求には、使用可能なメモリの量や現在のシステム使用量などに基づき、一度に読み取ることが可能な最大限のページが含まれています。

メモリ マネージャは、要求を記述するメモリ記述子リスト (MDL) を構築する際、ページ A と B への有効なポインタを提供します。ただし、ページ Y と Z に対するエントリは、システム全体での単一のダミー ページ X をポイントします。メモリ マネージャは、X を参照可能にしないので、バッキング ストアにある古い可能性のあるデータで、ダミー ページ X を埋めることができます。ただし、コンポーネントは、MDL の Y と Z のオフセットにアクセスする場合、Y と Z の代わりに、ダミー ページ X を参照します。

メモリ マネージャは、任意の数の破棄されたページを単一のダミー ページとして表すことができ、そのページは、同じ MDL に複数回埋め込まれている場合があります。また、異なるドライバに対して使用されている複数の同時実行の MDL に埋め込まれている場合もあります。したがって、破棄されたページを表す場所の内容は、いつでも変わる可能性があります。

プリフェッチ スタイルのクラスタリングは、MDL 内のポインタを直接参照してデータを読み取る、これら少数のドライバの操作に影響する場合があります。ドライバ作成者は、MDL によって記述されるページの順序や内容に関し、仮定を行ってはなりません。また、MDL によって参照されるどの場所のデータの値にも依存してはなりません。一般に、大半のドライバは、MDL 内のメモリの場所を直接参照してデータを得ることはありません。したがって、この制限は、少数のドライバにのみ影響します。

MDL がマップするページ内のデータ値に基づき、解読の実行やチェックサムの計算を行うドライバは、システムが提供する MDL のポインタを参照して、データにアクセスしてはなりません。その代わり、正しい操作を保証するには、このようなドライバは、ドライバが I/O マネージャから受け取ったシステム提供の MDL に基づき、一時的な MDL を作成する必要があります。ドライバに関するヒント「その MDL の中にはいったい何があるのでしょう?」では、そのようなドライバが従う必要のある手順を概説しています。

直接の I/O 書き込み操作用のバッファを MDL が記述している場合、I/O 要求を発行したアプリケーションは、同じページのビューをそのアドレス スペースにマップした可能性もあります。その場合、アプリケーションは、ドライバがデータを参照すると同時に、そのデータを変更する可能性があります。ドライバは、内容をダブルバッファしてデータのスナップショットを参照する一時的な MDL を作成することにより、この状況を適切に処理する必要があります。

基礎 MDL ページ内のデータにアクセスせずに、通常の I/O 操作の一部として MDL を使用するドライバは、一時的な MDL を作成する必要はありません。内部的に、メモリ マネージャは、常駐のページをすべて追跡し、各ページがどのようにマップされたか記録します。ドライバが MDL をシステム サービス ルーチンに渡して I/O を実行するとき、メモリ マネージャは、正しいデータが使用されることを保証します。

プリフェッチ スタイルのクラスタリングは元々、Windows XP で導入され、数か所で使用されました。Windows Vista は、オペレーティング システムの全体でプリフェッチ スタイルのクラスタリングを使用し、より大きなパフォーマンスの利点を提供します。

## 大きなファイルの管理

Windows Vista は、大きなスパース ファイルでの操作において、より優れた I/O パフォーマンスを提供します。Windows Vista メモリ マネージャは、ファイルがまたがるディスク ブロックを記述するため、Adelson-Velsky/Landis (AVL) ツリーを使用して、ファイルの範囲を記述します。AVL ツリーは、自動的にバランスをとるバイナリ ツリーです。このツリーは、以前の Windows リリースが使用していたリンク リストより、効率的な操作を提供します。リンク リストは、ファイル内のすべてのセクションを、リニアに参照する必要がありました。

AVL ツリーは、大きなファイルのマップ、フラッシュ、および削除を行うためファイル オフセットを使用するシステム関数を、大幅に加速します。その結果、バックアップは現在、Windows XP の約 2 倍高速になっています。

## 休止状態とスタンバイ

Windows Vista およびそれ以降の Windows リリースでは、休止状態とスタンバイは、以前の Windows バージョンより高速で効率的です。システムは現在、下記の 2 つの手順で、休止状態とスタンバイを実行します。

1. 物理メモリの内容を、ディスク上の休止状態ファイルにコピーします。デバイス スタックはすべてアクティブです。

2. 休止状態パス上のものを除き、デバイス スタックをすべてシャットダウンします。変更されたデータのみ、休止状態ファイルにコピーします。

休止状態とスタンバイは現在、フォールト トレランスなシステムで用いられているのと同じメモリ管理ミラーリング技法を使用しています。デバイス スタックがすべてアクティブな間、メモリ マネージャは、物理メモリの内容をディスクにコピーします。したがって、コピー操作は、より大きな I/O サイズ、scatter/gather 直接メモリ アクセス (DMA)、および他の高度で効率的な I/O 技法を活用して、休止状態用にデータを保存できます。そのような技法には、プリフェッチのサポートが含まれています。その結果、迅速な再開に必要なページは、必要に応じてメモリに読み込まれ、休止状態ファイルに含まれます。

初期コピー操作の完了後、システムは、休止状態パス上のものを除き、デバイス スタックをすべてシャットダウンします。その後、システムは、最初のコピー操作後に変更されたデータのみ、休止状態ファイルにコピーします。以前の Windows リリースと異なり、休止状態は、ページ キャッシュを削除しなくなりました。代わりに、システムは、何が使用されているかに基づき、インテリジェントにキャッシュ データを休止状態ファイルに書き込みます。また、データをクラスタ化し、各書き込み要求が、以前の Windows リリースの約 2 倍になるようにします。その結果、休止状態とスタンバイは、はるかに速くなり、ユーザーは、それらがどのように異なるか理解する必要がなくなりました。同等なシステム スナップショットの場合、休止状態に要する全体的な時間は、以前の Windows リリースの約 2 倍高速です。また、休止状態ファイルは、およそ半分のサイズです。

# 高度なビデオ モデル

Windows Vista およびそれ以降の Windows リリースの新しいビデオ アーキテクチャは、最新のグラフィックス処理装置 (GPU) や仮想メモリを十分に活用して、より現実的なシェーディング、テクスチャリング、フェーディング、およびゲームやシミュレーション用の他のビデオ特殊効果を提供します。

ビデオ アーキテクチャをサポートするため、メモリ マネージャは、Rotate Virtual Address Descriptor (VAD) と呼ばれる新しいマッピング種類を提供します。Rotate VAD により、ビデオ ドライバは、ユーザー ビューを、標準アプリケーション メモリから、Cached、Noncached、または Write-combined の AGP (Accelerated Graphics Port) やビデオ RAM に、ページ単位で迅速に切り替えることができ、すべてのキャッシュ属性をフルサポートします。このように、ビデオ アーキテクチャは、より高いパフォーマンス用に GPU を使用して、データを直接転送でき、不要なページをオンデマンドで入出力して回転できます。図 7 は、標準物理メモリまたはグラフィックス メモリ内のページに、VA をどのようにマップできるかを示します。



図 7. Rotate Virtual Address Descriptor

図 7 で、ユーザーの VA スペース内の場所に対するページ テーブル エントリは、ビデオ RAM や AGP 内のページで支持されているページ、またはファイル内のページで支持されているページを参照できます。ビューを切り替えるには、ビデオ ドライバは、新しいアドレスを提供するだけで済みます。このテクノロジにより、ビデオ ドライバは、GPU を直接転送に使用でき、以前のビデオ モデルと比べ、パフォーマンスを 100 倍に向上できます。

# NUMA のサポート

Windows Vista は、以前の Windows のどのリリースより、NUMA アーキテクチャの機能をより完全に使用します。NUMA サポートの基本理念は、メモリ マネージャとオペレーティング システムにインテリジェンスをできるだけ組み込み、アプリケーションが個々のコンピュータ ハードウェアの詳細から分離されるようにすることです。

Windows Vista と Windows Server 2008 の NUMA サポートでは、下記の領域で変更が行われています。

* リソースの割り当て
* 既定のノードおよびアフィニティ
* 割り込みのアフィニティ
* アプリケーション用の NUMA 対応システム関数
* ドライバ用の NUMA 対応システム関数
* ページング

## リソースの割り当て

スタートアップ時に、Windows Vista は、NUMA ノード アクセス コスト (各ノードから他のノードへの距離) のグラフを構築します。システムは、このグラフを使用して、操作中のページなど、リソースを取得する最適なノードを決定します。適切なリソースが最適なノードで利用できない場合、システムは、グラフを調べ、次善の選択肢を決定します。

アプリケーションは、オプションで最適なプロセッサを指定できますが、指定しなくとも、特定のコンピュータのアーキテクチャに関する知識は不要です。Windows は、可能な場合には常に、アプリケーションが最適なプロセッサで動作することを保証します。また、アプリケーションが割り当てる任意のメモリが、最適なプロセッサのノード、つまり最適なノードから提供されることも保証します。最適なノードが利用できない場合、またはそのリソースがなくなった場合、システムは、次善の選択肢を使用します。

以前の Windows リリースは、現在のプロセッサのノードからメモリを割り当てていました。また、そのリソースがなくなった場合、任意のノードから割り当てていました。したがって、以前の Windows リリースでは、最適でないノードで短時間動作したプロセスは、そのノードに永続的なメモリ割り当てを持っており、その結果、より遅く非効率な操作が生じていました。

Windows Vista の既定値では、システムは、プロセスが他のノードで動作中の場合でも、最適なノード上のメモリを割り当てます。メモリが最適なノードで利用できない場合、システムは、利用可能なメモリがたまたま存在するノードではなく、最適なノードの最も近くにあるノードを選択します。全体として、Windows Vista の既定値は、プロセスとそのリソースが同じノードまたは最適なノードに存在する可能性を上げることによって、よりインテリジェントなシステム全体のリソース割り当てにつながります。

同じ既定値が、カーネル モード ドライバに適用されます。カーネル モード ドライバは、呼び出しスレッド、システム スレッド、または任意のスレッドのプロセス コンテキストで動作できます。I/O 要求でしばしば生じるように、ドライバが呼び出しスレッドのコンテキストで動作中にメモリを割り当てる場合、システムは既定値として、最適なノードをスレッドのプロセス用に使用します。**DriverEntry**、*AddDriver*、*EvtDriverDeviceAdd*、および関連するスタートアップ関数でよく生じるように、ドライバがシステム スレッドのプロセスで動作している場合、システムは、システム プロセスの最適なノードを使用します。ドライバが任意のスレッド コンテキストで動作している場合、システムは、最適なノードをそのプロセス用に使用します。ドライバは、このペーパーの「ドライバ用の NUMA 対応システム関数」で後述する **MmXxxx** システム関数を使用することにより、これらの既定値を無視できます。たとえば、ドライバは、そのデバイスがそのノードに割り込む場合、特定のノード上のメモリを割り当てる場合があります。

システムには、すべてのノードの物理メモリを含んでいる、単一の非ページ メモリ プールがあります。最初の非ページ プールは、VA の連続的な範囲にマップされています。コンポーネントが非ページ プールを要求すると、メモリ マネージャは、プールへのインデックスとして、スレッドの最適なノードを使用します。その結果、メモリは、最適なノードから割り当てられます。ページ メモリ プールは、Windows Server 2003 で NUMA 対応になりました。

内部的に、システム PTE とシステム キャッシュは現在、各ノードで均等に割り当てられています。以前は、そのようなメモリは、ブート ノードに割り当てられ、まれに、そのノードの空きページがなくなることがありました。メモリ マネージャ自体の内部ルック アサイド リストも、同様に NUMA 対応です。

## 既定のノードおよびアフィニティ

前のセクションで記したように、Windows Vista は、メモリ割り当て用の既定のノードとして、最適なプロセッサを含んでいるノードを使用します。Windows XP および以前の Windows リリースでは、既定値は、スレッドが動作中のプロセッサを含んでいるノードです。

アプリケーションは、下記のいずれか (優先順) に基づき、NUMA アフィニティを指定できます。

* VAD。**VirtualAllocExNuma** または **MapViewOfFileExNuma** を使用。
* セクション。**CreateFileMappingNuma** を使用。
* スレッド。**SetThreadAffinityMask** または **SetThreadIdealProcessor** を使用。
* プロセス。**SetProcessAffinityMask** を使用。

システムは、可能な場合は常に、アプリケーションが指定したアフィニティを使用します。前のセクションで説明したように、Windows はそのような要求をすべて満たそうとしますが、すべての要求が、要求したノードから完全に満たされる保証はありません。適切なリソースが要求したノードで利用できない場合、システムは、適切なリソースのある最適なノードを使用します。このアプローチは、メモリが最適なノードで利用可能になるのを無制限に待つのではなく、要求を迅速に満たします。

## 割り込みの関係

メッセージがシグナルする割り込み (MSI または MSI-X) をサポートする PCI デバイス用のドライバは、デバイスが生成する MSI メッセージごとに、割り込みアフィニティを指定できます。割り込みアフィニティとは、デバイスの割り込みサービス ルーチン (ISR) が動作するプロセッサのセットです。この機能は、特に Receive-Side Scaling (RSS) をサポートするネットワーク インターフェイス カード (NIC) で、パフォーマンスを大幅に上げることができます。

ドライバは、割り込みを接続するとき、特定の MSI メッセージに対しアフィニティを指定できます。また、ドライバは、その INF の DDInstall.HW セクション内の **Interrupt Management\Affinity Policy** レジストリ キーに値を設定することにより、既定のアフィニティとアフィニティ ポリシーも設定できます。管理者は、これらの値をレジストリでも設定できます。

詳細については、WinHEC プレゼンテーション 「NUMA I/O Optimizations」、ホワイト ペーパー「Interrupt Architecture Enhancements in Windows」、および WDK の「Interrupt Affinity and Priority」 を参照してください。NIC 固有の情報については、WDK の「NDIS MSI-X」を参照してください。

## アプリケーション用の NUMA 対応システム関数

Windows Vista は、アプリケーション用に、下記の新しい NUMA 対応システム関数をサポートします。

* **VirtualAllocExNuma** は、仮想メモリの範囲を予約またはコミットして、特定のノード上のメモリを要求します。
* **CreateFileMappingNuma** は、ファイル マッピング オブジェクトを作成するか開き、特定のノード上のメモリを要求します。
* **MapViewOfFileExNuma** は、ファイル マッピング オブジェクトのビューをマップし、特定のノード上のメモリを要求します。
* **AllocateUserPhysicalPagesNuma** は、特定のノードから物理メモリを割り当てます。
* **QueryWorkingSetEx** は、特定の VA が現在割り当てられているノードの取得に使用できます。

これらの関数は、同様な名前の既存の関数の NUMA 対応バージョンです。これらの関数の詳細については、MSDN Web サイトを参照してください。

## ドライバ用の NUMA 対応システム関数

ドライバは、下記の 2 つの新しいシステム関数を使用して、特定のノード上のメモリ割り当てに対しアフィニティを指定できます。

* **MmAllocateContiguousMemorySpecifyCacheNode**
* **MmAllocatePagesForMdlEx**

**MmAllocateContiguousMemorySpecifyCacheNode** は、NUMA アーキテクチャをサポートするコンピュータ上の特定のノードからドライバがメモリを要求できる点を除き、既存の **MmAllocateContiguousMemorySpecifyCache** 関数に似ています。

**MmAllocatePagesForMdlEx** は、**MmAllocatePagesForMdl** に似ています。ただし、ドライバは、現在のスレッドの最適なノード上でのみページをオプションで要求でき、割り当て時にページをゼロで埋める過程をスキップでき、ページのマッピングに使用されるキャッシュの種類を指定できます。

## ページング

Windows Server 2008 は、ページング用に、NUMA 拡張機能をさらに組み込みます。Server 2008 は、アプリケーションの最適なノードにページをプリフェッチし、ソフト ページ フォールトが生じたとき、最適なノードにページを移行します。ソフト ページ フォールトは、要求されたページをメモリの他の場所でシステムが見つけることができる場合に生じます。一方、ハード ページ フォールトでは、ディスクからページを読み取る必要があります。

# スケーラビリティ

Windows がより大型で強力なコンピュータで実行されるにつれ、Microsoft は、引き続きシステムの機能を拡張して、より多く、より高速なプロセッサと RAM をサービスするためスケールアップします。

## 効率と並列処理

メモリ マネージャの内部で多くの点が向上した結果、メモリ割り当ては現在、より少数の I/O 操作とより少数のロックで、最適なスループットを達成できます。

内部的に、メモリ マネージャは現在、リンク リストの代わりにビットマップを使用して、非ページ プール内の空きページを追跡します。リンク リストと異なり、ビットマップはロックなしで検索でき、それにより、関連するロックの競合を 50% 以上削減できます。さらに、ビットマップは、隣接した空きページの自動的な結合を提供します。また、Windows Server 2008 は、ビットマップを使用して、システム PTE を記述します。

大きな共有セットは現在、ハッシュされず、直接マップされます。ハッシュ テーブル内のエントリ数が非常に多い場合、ハッシュ テーブルのサイズを動的に変更できない限り、通常、競合が頻繁に生じます。ただし、サイズの変更を大きなセットに対して実行するには、CPU の能力がかなり必要です。したがって、こうした状況では、直接のマッピングがハッシュより効率的です。

Windows Server 2008 では、物理的に隣接したメモリの割り当てが、非常に強化されました。メモリ マネージャは現在、通常、ワーキング セットをトリミングしたり、I/O 操作を実行したりせずに、ページを動的に置換するので、隣接したメモリを割り当てる要求は、成功する可能性がかなり増しています。さらに、より多くの種類のページ (特に、カーネル スタックやファイル システム メタデータ ページなど) が現在、置換の対象候補となっています。その結果、より多くの隣接したメモリを、どのタイミングでも通常利用できます。さらに、そのような割り当てを得るためのコストも、大幅に減っています。

## ページ フレーム番号と PFN データベース

ページ フレーム番号 (PFN) データベースには、コンピュータ内のすべての物理メモリに関する情報が含まれています。Windows Vista SP1 と Windows Server 2008 の 64 ビット エディションでは、ページ フレーム番号は、64 ビット長で、大量のメモリと NUMA アーキテクチャをサポートします。NUMA アーキテクチャでは、物理アドレス スペースにおいて、メモリがまばらに配置される場合があります。

以前の Windows リリースでは、新しいページが必要な場合は常に、メモリ マネージャは、PFN スピンロックを取得し、PFN データベースを通じて連結された適切な一覧から新しいページを削除しました。Windows Vista は、代わりに、各 NUMA ノードとページ カラーに対し、直ちに利用可能なゼロ ページと空きページの短い一覧を維持します。(ページ カラーは、各ページでキャッシュ行が競合する可能性を減らすため、メモリ マネージャが使用する技法です。)多くの場合 (特に demand-zero フォールトや copy-on-write フォールトの場合)、システムは現在、PFN ロックを取得しないでも、単一のページを取得できます。スピンロック取得の回数を減らすと、他のプロセッサでスピンが生じる可能性が減るので、並列処理が向上します。

## 大容量ページ

Windows Server 2003 は、ユーザー モード アプリケーション用に、大容量ページを導入しました。Windows Vista と Windows Server 2008 は、大容量ページをより広範に内部で使用しており、大容量ページの拡張サポートを提供します。Windows Vista と Windows Server 2008 は、大容量ページを下記の用途で使用します。

* 初期非ページ プール
* PFN データベース
* ユーザー アプリケーションとドライバ イメージ
* ページ ファイルで支持されている共有メモリ
* ユーザー モード **VirtualAlloc** 割り当て
* ドライバの I/O 領域のマッピング

ユーザー モード アプリケーションは、MEM\_LARGE\_PAGES フラグ付きで **VirtualAlloc** 関数を使用することにより、x86 ベースのシステムで最大 4 MB のページを割り当てることができます。表 1 は、Windows ハードウェア プラットフォームでサポートされる大容量ページのサイズの一覧です。

表 1. 大容量ページのサイズ

|  |  |
| --- | --- |
| アーキテクチャ | 大容量ページのサイズ |
| x86 | 4 MB |
| x86 (PAE 有効) | 2 MB |
| x64 | 2 MB |
| Itanium | 16 MB |

アプリケーションは、現在の大容量ページのサイズを判断するため、**GetLargePageMinimum** を呼び出すことができます。

Windows Vista のメモリ マネージャは、大容量ページの範囲を、以前の Windows リリースより迅速に割り当てます。範囲全体が隣接している必要はなくなりました。したがって、大容量ページを割り当てる試みは、成功する可能性が高くなり、ページ スラッシングが生じる可能性は低くなります。たとえば、アプリケーションが 10 MB の大容量ページを要求する場合、Windows Vista およびそれ以降の Windows リリースは、10 MB の物理的に隣接したメモリを探す代わりに、それぞれが 2 MB の、5 つの大容量ページを割り当てることができます (大容量ページが各ハードウェア プラットフォームで 2 MB の場合)。

Windows Vista のメモリ マネージャは、割り当てられたメモリがどの NUMA ノードに属するかも追跡します。また、スレッドを適切なノードにディスパッチして、それらをゼロで埋めることにより、大容量ページを並列にゼロで埋めることができます。

## キャッシュの境界で整列されたプール割り当て

Windows Vista は、キャッシュの境界で整列されたプール割り当てのサポートを実装しています。ドライバは、キャッシュの境界で整列されたメモリを要求するため、**ExAllocatePoolXxx** 関数で下記のフラグを指定できます。

* **NonPagedPoolCacheAligned**
* **PagedPoolCacheAligned**

これらのフラグは、以前の Windows リリースで定義されていましたが、無視されました。

## 仮想マシン

効率やスケーラビリティは、優れたサーバー パフォーマンスに対してのみ重要なのではなく、Windows が仮想化システム内でゲスト オペレーティング システムとして動作するのにも重要です。Windows Vista には、仮想マシンのシナリオでパフォーマンスを上げる、複数の変更点が組み込まれています。

変換ルック アサイド バッファ (TLB) は、VA から物理アドレスへの変換をキャッシュし、プロセッサがこの情報に迅速にアクセスできるようにします。アドレスが TLB に存在しない場合、プロセッサは通常、複数のメモリ参照を作成する必要があり、それにはかなり時間がかかります。その結果、全体的なシステム パフォーマンスは、TLB のヒット率が下がるにつれ、減少します。

アドレスが TLB に存在する可能性を上げる 1 つの方法は、TLB をあまり頻繁にフラッシュしないことです。ページが無効になるごとに、そのエントリをバッファからフラッシュする必要があります。ページは、マップされていない場合、解放された場合、ワーキング セットからトリミングされた場合、copy-on-write 操作で変更された場合などに、無効となります。また、エントリは、ページの保護属性やキャッシュ属性が変更された場合にも、フラッシュする必要があります。

すべてのプロセッサで変換バッファ全体をフラッシュするのは、オペレーティング システムのオーバーヘッドがかなり必要となる、負荷の比較的高い操作です。さらに、バッファをフラッシュした後、再び事前設定する必要があります。Windows Vista は、めったにバッファ全体をフラッシュしません。その結果、仮想マシンは、非常に効率的に機能できます。

仮想化システムが複数のゲスト オペレーティング システムをホストする場合、ゲストのサイズは、ハイパーバイザのパフォーマンスを制約し、スケーラビリティを制限する可能性があります。使用するメモリ フットプリントを少なくし、より良いゲスト システムであるため、Windows Server 2008 は、予測的に割り当てた不要なメモリを解放します。特に、システムは、最初の非ページ プールが使用されていない場合、そこからメモリを再生して利用します。

## 負荷分散

Windows Vista は、負荷分散に役立つ、下記の新しいイベントをエクスポートします。

* **LowCommitConditionNotification**
* **HighCommitConditionNotification**
* **MaximumCommitConditionNotification**

ペレーティング システムのコミット チャージが、現在のコミット制限に比べて低い場合、**LowCommitConditionNotification** イベントが設定されます。言い換えれば、メモリ使用量が少なく、多くのスペースが割り当て用に利用可能です。

オペレーティング システムのコミット チャージが、現在のコミット制限に比べて高い場合、**HighCommitConditionNotification** イベントが設定されます。言い換えれば、メモリ使用量が多く、利用可能なスペースはわずかしかありません。十分なディスク領域を利用できる場合、システムは、管理者が設定した制限までページ ファイルのサイズを自動的に増やすことにより、より多くのメモリを取得します。短期的なオプションは、現在のシステム負荷を軽くすることです。長期的なソリューションは、ページ ファイルの最小サイズを増やすか、RAM を追加することです。

オペレーティング システムのコミット チャージが最大コミット制限に近い場合、**MaximumCommitConditionNotification** イベントが設定されます。言い換えれば、メモリ使用量が非常に多く、利用可能なスペースはわずかしかありません。また、システムは、管理者が設定した現在の制限のため、ページング ファイルのサイズを増やすことができません。十分なディスク領域を利用できる場合、システム管理者は、コンピュータを再起動しないでも、ページング ファイルのサイズや数を、常に増やすことができます。他の選択肢は、ページ ファイルの最小または最大サイズを増やすか、RAM を追加するか、負荷を軽くすることです。

これらのイベントは、Windows Server 2003 で追加されたプール通知イベントを補完します。ドライバおよび他のカーネル モード コンポーネントは、これらのイベントに対して登録できます。メモリ関連の通知イベントの詳細については、WDK の「Standard Event Objects」を参照してください。

## その他の最適化

その他のメモリ マネージャの最適化は、下記の領域に関係しています。

* **VirtualAlloc** と AWE (Address Windowing Extensions) の割り当て。
* **VirtualProtect** 関数。
* 64 ビット システムでの Windows on Windows (WOW)。

Windows は、変更内容をユーザー アドレス スペースに同期させるため、プロセスごとのアドレス スペース ロックを取得します。以前の Windows バージョンでは、このロックは、排他的アクセスのみサポートしていましたが、Windows Vista では、このロックは、共有アクセスと排他的アクセスの両方をサポートします。その結果、**VirtualAlloc** や **VirtualQuery** など、多くの操作が現在、並列で動作できます。全体として、**VirtualAlloc** 内の変更により、AWE 割り当てに必要な時間が短縮され、シナリオによっては 2500% 以上短縮されます。

**VirtualProtect** 関数は、仮想メモリ内のページの領域に対するアクセス保護を変更します。ページのアクセス保護属性が変化した場合、プロセッサは、対応する TLB エントリをフラッシュする必要があります。Windows Server 2008 は、複数の単一要求を個々のプロセッサに発行するのではなく、単一のフラッシュ要求を、TLB にエントリが含まれている可能性のあるすべてのプロセッサに発行します。その結果、**VirtualProtect** は、大きな領域に対するアクセス保護を、以前の Windows バージョンより 60 倍速く変更できます。

64 ビット アーキテクチャで、Windows Vista は、プール メモリの代わりに demand-zeroed メモリを使用して、32 ビット バイナリ エミュレーション用にページ テーブル ビットマップを割り当てます。この変更により、32 ビット バイナリは、必要なシステム メモリ フットプリントが減り、実行する I/O 操作の回数が減るので、はるかに効率的に動作できます。

# システムの整合性

ユーザーは、オンライン クラッシュ解析 (OCA) を通じて、システム クラッシュに関するデータを Microsoft にアップロードできます。このデータは、一般的なシステム クラッシュの原因について有用な情報を提供し、システム破損の可能性を検出して処理する、複数のシステム拡張機能の実現につながりました。Windows Vista と Windows Server 2008 には、下記の領域でシステムの整合性を向上させるための改良点が組み込まれています。

* ハードウェア エラーの診断
* コードの整合性およびドライバ署名
* バグ チェック中のデータの保存

## ハードウェア エラーの診断

このペーパーの「ページ ファイルへの書き込み」で前述したように、Windows は、ゼロ ページの一覧を維持します。DMA 転送エラーや単一ビットのエラーなど、ハードウェア エラーが起きると、ページが最初にゼロで埋められた後、メモリが破損する可能性があります。したがって、Windows Vista は、これらのページが実際にゼロであることを確認するため、一覧をチェックします。システムはエラーを検出すると、エラーが生じた物理アドレスとエラーの性質を、イベント ログに記録します。この情報は、ハードウェア障害によって生じる単一ビット エラーを、ピンポイントで特定するのに役立ちます。

そのようなエラーが頻繁に起きるコンピュータは、アプリケーションのハングやクラッシュが起きやすく、それらは多くの場合、追跡するのが非常に困難です。OCA のデータによると、そのような障害は、以前予測されていたより、はるかによく生じることが示されています。ハードウェア メーカーは、エラー訂正符号 (ECC) メモリの使用により、診断を支援して、そのようなエラーを防くことができます。

## コードの整合性およびドライバ署名

メモリ マネージャは、コードの整合性についてイメージを検証する、シンプルで高速な技法を実装しています。この機能は、x64 ベース システムのカーネル モード ドライバに、必須のコード署名を強制します。

コード署名の詳細については、「Windows Vista を実行している x64 ベースのシステム上のカーネル モジュールのデジタル署名」、「カーネル モードのコード署名の手順」、および「Windows ドライバの署名要件の概要」を参照してください。

Windows Vista は、システム全体のドライバとセッション ドライバの両方に対して、修正プログラムのホット適用をサポートします。したがって、修正プログラムは、ユーザーのシステムを再起動しないでもインストールできます。これにより、ユーザーは、セキュリティ更新プログラムがリリースされたらすぐに、再起動を待つ必要なく利用できます。

## バグ チェック中のデータの保存

Windows Vista は、特定の非破壊バグ チェックが生じたとき、以前の Windows バージョンより多くのデータを保持します。たとえば、システムがカーネル モード コンポーネントの一部をページインしているとき、バグ チェックが生じた場合、必要な情報がコンポーネントで欠けているので、システムは動作を続行できません。ただし、既にシステム キャッシュ内にあるデータは、影響を受けません。データ損失を防ぐため、メモリ マネージャは、変更されたデータをすべて、システム キャッシュからそのバッキング ストア (通常はディスク ファイル) に書き込んでから、バグ チェックを発行します。重大なのは、カーネル モード コードやデータ内のページに対する障害のみです。ユーザー プロセス コードやデータ内のページに障害が起きても、アプリケーションで例外が生じるだけです。

システム データをさらに保護するため、Windows Vista は、システム キャッシュのビューを、読み取り専用とマークする機能をサポートします。レジストリは、この機能を使用して、不注意なドライバの破損からそのビューを保護します。したがって、レジストリ データは、アクティブに変更されている場合を除き、読み取り専用です。

ドライバ作成者は、新しい **.pagein** デバッガ コマンドを使用して、ディスクにページ アウトされたカーネル モード メモリ アドレスの内容を表示できます。このコマンドの詳細については、Debugging Tools for Windows を参照してください。

# 行うべきこと

このペーパーで説明した大半のメモリ管理強化機能は、内部的なものであり、管理者、ソフトウェア開発者、およびハードウェア メーカーにとっては透過的です。ただし、いくつかの変更点では、最大限の利点や向上したユーザー エクスペリエンスを得るため、認識やアクションが必要です。

ハードウェア メーカー、ドライバ開発者、アプリケーション開発者、およびシステム管理者にとっての最も重要な影響は、次のとおりです。

## ハードウェア メーカーの場合

* ECC を使用してください。

## ドライバ開発者の場合

* ドライバが割り当てた範囲を越えて、メモリにアクセスしないでください。Driver Verifier を使用して、このエラーを検出してください。
* MDL の内容に直接アクセスするドライバで、ダミー ページを正しく処理してください。
* 必要に応じて **KeExpandKernelStackAndCallout** を使用し、追加のカーネル スタック領域を得てください。
* NUMA アーキテクチャ対応であり、デバイスにとって重要な場合に割り込みアフィニティを指定するドライバにおいて、新しい NUMA 対応システム関数を使用してください。
* 操作中に解放できるメモリをドライバが割り当てる場合、システム負荷に関する通知用の新しいイベントを使用してください。
* 特に x64 アーキテクチャにおいて、Windows Vista のドライバ署名要件に注意してください。
* **.pagein** デバッガ コマンドを使用して、ページ アウトされたカーネル モード データを検査してください。

## アプリケーション開発者の場合

* **/DYNAMICBASE** オプションと **/NXCOMPAT** オプション付きで再リンクして、Windows Vista の実行不可保護と共に、ASLR を有効にしてください。
* 現在、メモリ割り当て用の既定の NUMA ノードは、現在のノードではなく最適なノードである点に注意してください。
* 新しい NUMA 対応システム関数を使用して、メモリ割り当てを制御し、NUMA アーキテクチャ上のページの場所を照会してください。

## システム管理者の場合

* 動的なカーネル VA 割り当てを理解して、システム チューニングを変更できるようにしてください。または、まったくチューニングを行わないようにします。
* デバッガで !vm 21 コマンドを使用して、32 ビット システムでのカーネル VA スペース使用の詳細を検査してください。
* ゼロ ページの破損エラーがないかどうか、システム イベント ログをチェックしてください。可能な場合は常に、クラッシュ データを OCA にアップロードしてください。

# リソース

#### MSDN:

Windows Vista ISV Security

<http://msdn2.microsoft.com/en-us/library/bb430720.aspx>

Memory Management Registry Keys

<http://msdn2.microsoft.com/en-us/library/bb870880.aspx>

Windows Driver Kit:

<http://msdn2.microsoft.com/en-us/library/aa972908.aspx>

**Kernel-Mode Driver Architecture Design Guide**Memory Management
Interrupt Affinity and Priority
Standard Event Objects

**Kernel-Mode Driver Architecture Reference**Standard Driver Routines
Driver Support Routines

**Driver Development Tools**Boot Options for Driver Testing and Debugging

**Network Design Guide**
NDIS MSI-X

#### Windows Hardware and Driver Central:

Driver Signing Requirements for Windows [home page]

<http://www.microsoft.com/whdc/winlogo/drvsign/drvsign.mspx>

**Digital Signatures for Kernel Modules on Systems Running
 Windows Vista
Summary of Windows Kernel-Mode Driver Signing Requirements**

Windows PC Accelerators: Performance Technology for Windows Vista

<http://www.microsoft.com/whdc/system/sysperf/accelerator.mspx>

What Is Really in That MDL?

<http://www.microsoft.com/whdc/driver/tips/mdl.mspx>

Interrupt Architecture Enhancements in Windows

<http://www.microsoft.com/whdc/system/bus/PCI/MSI.mspx>

NUMA I/O Optimizations

<http://download.microsoft.com/download/a/f/d/afdfd50d-6eb9-425e-84e1-b4085a80e34e/SVR-T332_WH07.pptx>

#### Microsoft TechNet:

Inside the Windows Vista Kernel: Part 3 (April 2007)

<http://www.microsoft.com/technet/technetmag/issues/2007/04/VistaKernel/>

#### Book:

Windows Internals, Fourth Edition,
Russinovich, Mark, and David A. Solomon. Redmond, WA: Microsoft Press, 2005

<http://www.microsoft.com/MSPress/books/6710.aspx>