Hewlett Packard Enterprise

HP-UX メモリ管理 ホワイト・ペーパー バージョン 1.4

《 2000 年時点 》

—目次—

ご注意

本書の目的

物理メモリと仮想メモリの概要

物理メモリの役割

仮想メモリの抽象化

プロセッサーのメモリ関連部分

仮想メモリの構造

仮想メモリから物理メモリへのマッピング

ページ可用性の維持

スワップ・スペース管理

デマンド・ページングの概要

メモリ内のプロセス構造のセットアップ方法

ご注意

本書に記載された内容は、予告なしに変更されることがあります。

本書に記載された内容は、製品や特定の目的に対する適合性を保証するものではなく、ヒューレット・パッカード社はそれらに関する一切の責任を負いません。

また、本書の記載の誤り、あるいは、本書の配布、内容、利用によって生じる偶発的、結果的損害に対しても当社は一切の責任を負いません。

保証：HPE の製品および交換部品に対する個々の保証条項については、HPE 営業担当までお問い合わせください。

米国政府の制限的権利：米国政府機関による使用、複製、開示は、subparagraph © (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 for DOD agencies, and subparagraphs © (1) and © (2) of the Commercial Computer Software Restricted Rights clause at FAR 52.227-19 for other agencies の規定により制限を受けます。

初版：1997 年 4 月（HP-UX リリース 10.30） 第 2 版：2000 年 9 月（HP-UX リリース 11.11）

本書の目的

- 物理メモリと仮想メモリの概要の解説
- 仮想メモリに関連する各種の構造とその目的の説明
- 物理メモリと仮想メモリとの間の相互マッピング方法の説明
- メモリ・ページの作成方法と、実行中のプロセス／スレッドでメモリ・ページを使用可能にする方法の説明
- スワップ・スペースの管理方法の解説
- メモリ内でプロセス構造をセットアップする方法の説明
- メモリ・ページの割り当て、解放、および回復の方法と理由の解説

物理メモリと仮想メモリの概要

HP-UX のメモリ管理システムは、スレッドとプロセスに対してメモリ・リソースを安全で効率的に使用できるように設計されています。このシステムの機能は次のとおりです。

- 各プロセスに対して完全なアドレス空間を提供し、別のプロセスすべてからそのアドレス空間を保護する。
- 物理メモリよりも大きなサイズのプログラムを実行できる。
- 物理メモリに配置するスレッドとプロセスを決め、スレッドとプロセスをメモリに出し入れする。
- スレッドまたはプロセスの仮想アドレス空間のうち、物理メモリ内にない部分を管理し、物理空間に配置すべきアドレス空間部分を決める。
- プロセス間での効率的なメモリ共有を可能にする。

あらゆるプロセス（実行中のプログラム）またはプロセス内の実行スレッドのデータと命令は、実行時に物理メモリに存在し、CPU から使用可能になっている必要があります。

プロセスを実行するために、カーネルがプロセスごとの仮想アドレス空間を作成し、セットアップします。この仮想アドレス空間の一部が物理メモリにマッピングされます。仮想メモリによって、ユーザー・プロセスの合計サイズが物理メモリを超えることができます。HP-UX では「デマンド・ページング（demand paging）」が採用されているため、必要な場合のみ（つまり「オン・デマンド（on demand）」で）仮想ページをメイン・メモリにロードし、最近使用されていないプロセスのアドレス空間部分を追い出すことで、スレッドとプロセスを実行できます。

「メモリ管理（memory management）」という言葉は、物理メモリと仮想メモリを管理し、システムのリソースをユーザー・プロセスとシステム・プロセスで効率的に共有できるようにする規則を意味します。

このシステムでは、ページアウトと非アクティブ化を組み合わせて物理メモリを管理します。ページングでは、最近参照されていないページがメイン・メモリからディスクに時々書き込まれます。ページとは、一連のアクセス属性をもつ仮想アドレスにマッピングできる物理メモリの最小単位です。負荷の高いシステムでは、参照されていないページがメモリの大部分を占める場合があります。

非アクティブ化は、空き物理メモリ・プールをシステムで十分に維持できない場合に行われます。プロセス全体が非アクティブ化されると、そのプロセスに関連のあるページが参照されないため、二次記憶領域に書き出すことができます。非アクティブ化されたプロセスは実行できず、そのデータを参照できません。

二次記憶領域は、物理メモリを補完します。メモリ管理システムは使用可能メモリをモニターし、使用可能メモリが少ない場合、プロセスまたはスレッドをスワップ・デバイスと呼ばれる二次記憶領域に書き出します。プロセスの実行に必要になったデータがスワップ・デバイスから物理メモリに読み込まれます。

ページ

ページは、データとコードを保存するために割り当てることができる連続した物理メモリ・ブロックの最小単位です。また、ページはメモリ保護の最小単位でもあります。すべての HP-UX システムのページ・サイズは 4KB です。

PA-RISC システムでは、物理メモリの各ページが PPN（Physical Page Number : 物理ページ番号）でアドレス指定されます。PPN は、物理アドレスをソフトウェア上で物理ページ番号に「変換」したものです。特別な場合を除き、ページへのアクセス（およびページに含まれるデータへのアクセス）は仮想アドレスによって行われます（仮想変換をオフ（D ビットと I ビットをオフ）にしなければならない場合、ページはその絶対アドレスによってアクセスされます）。

仮想アドレス

プログラムのコンパイル時に、コンパイラがコードの仮想アドレスを生成します。仮想アドレスは、メモリ内の位置を表します。コンパイルされたコードを実行するために、仮想アドレスを物理アドレス（メモリ内の物理ページの位置）にマッピングする必要があります。ユーザー・プログラムでは、仮想アドレスのみを使用します。

カーネルとハードウェアが、CPU がメモリにプロセスを配置できるようにこれらの仮想アドレスと物理アドレスのマッピングを調整します。これを「アドレス変換（address translation）」と呼びます。

PA-RISC アーキテクチャはセグメントに区分されています。つまり、完全な仮想アドレスはスペース ID（SID）とそのスペース内のオフセットから構成されます。

このオフセットは 32 ビットまたは 64 ビット幅です。以前の PA-RISC プロセッサー（PA-RISC 2.0 以前）は、32 ビットのオフセットのみサポートしています。

これに対して、カーネルでは、完全なスペースとオフセットが扱われます。

カーネルにとって、PA-RISC プロセッサー上で実行されるすべてのプロセスは、スペースとオフセットの両方から構成されるグローバル仮想アドレス（GVA：Global Virtual Address）をもつ単一のグローバル仮想アドレス空間を共有します（GVA は、64 ビット（幅）モードで実行される PA-RISC 2.0 プロセッサー上では 96 ビットであり、以前のプロセッサー上ではそれよりも小さくなっています）。このグローバル仮想アドレス空間もカーネルで共有されます。

どのプロセスでも、任意のグローバル仮想アドレスの作成と読み取りまたは書き込みの試行が可能ですが、カーネルはページ単位のアクセス制御機構を利用してプロセス間の余分な干渉を防止します。

仮想ページが物理メモリに「ページング（paging）」される場合、物理メモリ・アロケータによって空き物理ページがその仮想ページに割り当てられます。これらのページは、その使用履歴に応じてメモリ全体にランダムに分散されることがあります。仮想ページのロード先をプロセッサーに通知するためには、変換が必要です。仮想アドレスを物理アドレスに変換するプロセスを仮想アドレス変換と呼びます。

潜在的に仮想アドレス空間が物理アドレス空間よりもはるかに大きい場合があります。仮想メモリ・システムでは、CPU が使用可能な物理メモリよりもはるかに大きなプログラムを実行できるため、より多くのプログラムを同時に実行できます。

デマンド・ページング

プロセスを実行するために、データやテキストなどのすべての構造をセットアップする必要があります。ただし、プロセスが「要求する（demand）」までページはメモリにロードされません。このため、デマンド・ページングという言葉が使われます。デマンド・ページングでは、プロセスで実行する必要がある時点でプロセスのさまざまな部分が物理メモリにロードされます。一度にメモリで必要とされるのは、プロセス全体ではなくプロセスのワーキング・セットのみです。実際にページにアクセスするときに、変換が行われます。

物理メモリの役割

メモリは、データを記憶するための「入れ物（container）」です。高速データ記憶のための汎用的な保管場所が CPU の近くにあり、これらはランダム・アクセス・メモリ（RAM）または「メイン・メモリ（main memory）」と呼ばれています。CPU がプロセスを実行するためには、プロセスで参照されるコードとデータが RAM に存在していなければなりません。RAM は、すべてのプロセスで共有されます。

システム内のメイン・メモリが多いほど、頻繁にページングや非アクティブ化を行うことなく、システムでより多くのデータにアクセスでき、より多くの（または、より大きな）プロセスを保持して実行することができます。なお、メモリ常駐リソース（ページ・テーブルなど）もメイン・メモリ内のスペースを消費します。このため、アプリケーションで使用可能なスペースは減少します。

システムの起動時に HP-UX がディスクから RAM にロードされ、システムがシャットダウンするまでメモリに常駐します。

ユーザー・プログラムとコマンドもディスクから RAM にロードされますが、このような小さな部分は必要な時点でロードされます。プログラムが終了すると、プロセスで使用されたメモリがオペレーティング・システムによって解放されます。

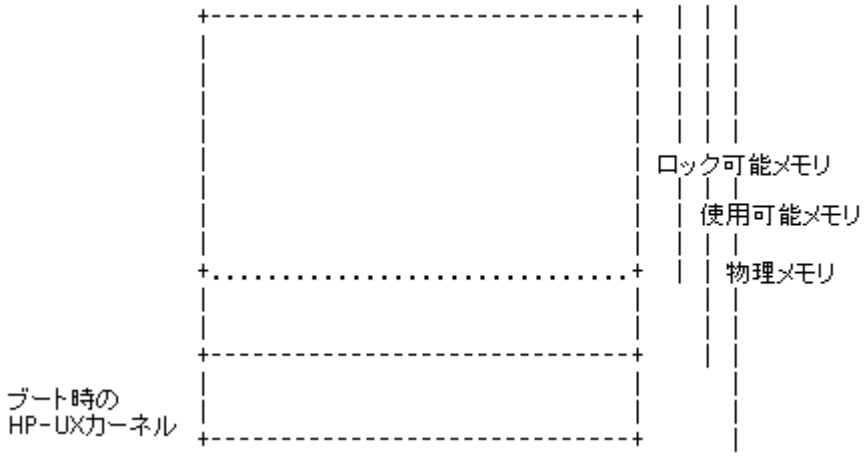
ディスク・アクセスは、RAM アクセスに比較すると遅くなります。過度なディスク・アクセスは待ち時間の増加やスループットの低下を引き起こすことがあるため、ディスク・アクセスがシステムのボトルネックになることがあります。これを防止するには、ある程度のバッファリングを行う必要があります。バッファリング、ページング、および非アクティブ化の各アルゴリズムによって、ディスク・アクセスが最適化され、現在実行されているプログラムのデータとコードを RAM からディスクに戻すタイミングが決められます。ユーザー・プログラムまたはシステム・プログラムによってデータがディスクに書き込まれる際、そのデータはプログラムの RAM から直接書き込まれるか（たとえば、"raw"デバイスへの書き込み）、バッファ・キャッシュにバッファリングされて、比較的大きなチャンク（かたまり）でディスクに書き込まれます。また、プログラムがファイルとデータベース構造をディスクから RAM に読み込むことがあります。システムをシャットダウンする前に sync コマンドを発行すると、バッファ・キャッシュで変更されたバッファすべてがディスクにフラッシュされます（書き出されます）。

各プロセッサーには、レジスタとキャッシュも搭載されています。これらは、メイン・メモリよりも高速です。実際のプログラム実行はレジスタ内で行われ、レジスタがキャッシュおよび別のレジスタからデータを取得します。キャッシュには、メイン・メモリの一部の現在の作業コピーが入っています。メモリ管理についての以降の説明では、ほとんどの場合、キャッシュとレジスタを完全に無視し、メイン・メモリから直接データと命令にアクセスするものとして扱います。

注記：キャッシュとレジスタは実際にはメモリの一種ですが、本書では後述の「プロセッサーのメモリ関連部分」で取り上げます。

ここからは、しばらく「メイン・メモリ」についてのみ説明します。

図 1 プロセスで使用可能な物理メモリ



使用可能メモリ

メイン・メモリのうち、カーネル用に予約されていない部分を、使用可能メモリと呼びます。使用可能メモリは、ユーザー・プロセスの実行のためにシステムで使用されます。

すべての物理メモリがユーザー・プロセスで使用可能だとは限りません。カーネル・テキストと初期化済みデータが約 10MB の RAM を占有します。さらに、カーネル bss（未初期化データ）と（特に）カーネル・ブート時に割り当てられるさまざまな構造によって、さらにメモリが使用されます。これらの構造の多くは、非常に大きい場合があります。一部の構造のサイズはカーネル調整パラメータによって決められますが、多くの構造のサイズはシステム内の物理メモリ量を基準に決められます。たとえば、このような構造が、物理メモリの 4096 バイト・ページそれぞれで 96 バイトのエントリを 1 つ持つ場合があります。

HP-UX カーネルは、システム初期化時にデータ構造すべてを割り当てるのではなく、正常な動作時にシステムの必要に応じて一部のカーネル構造を動的に割り当て、解除します。この割り当ては、使用可能なメモリ・プールから行われます。このため、任意の時点で、使用可能メモリの一部をカーネルが使用し、残りのメモリをユーザー・プログラムで使用できます。

物理アドレス空間はハードウェアで使用されるアドレスの範囲全体であり（32 ビット（ナロー・モード）カーネルでは 4GB）、メモリ・アドレス空間、プロセッサー依存コード（PDC: processor-dependent code）アドレス空間、および I/O アドレス空間に分割されます。次の図は、計算に使用できる大量のメモリを示しています。メモリ・アドレス空間はシステム・アドレス空間の 15/16 を占めますが、PDC と I/O に割り当てられるアドレス空間は比較的小さい範囲のアドレスです。

図 2 システム・アドレス空間の主要セクション（32 ビット・カーネルの場合）

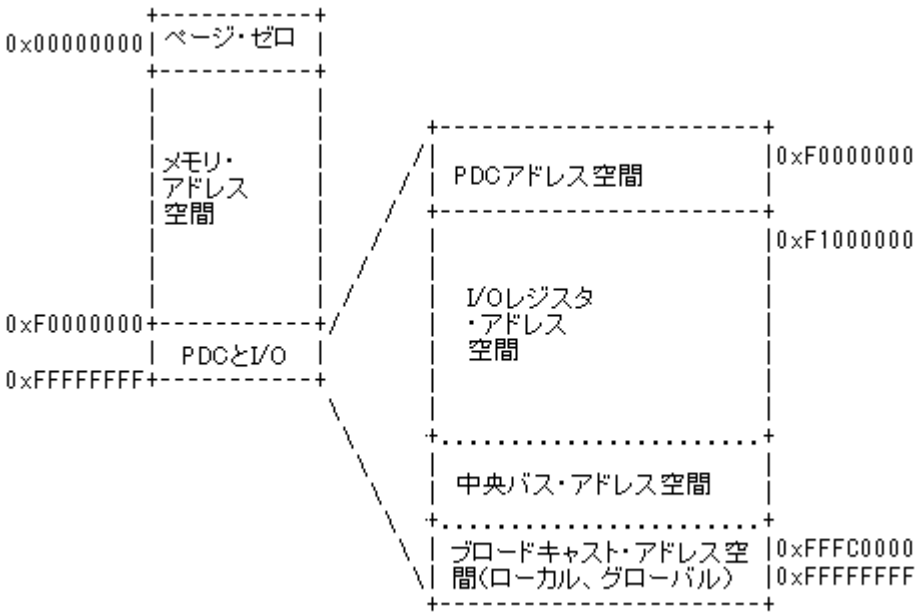
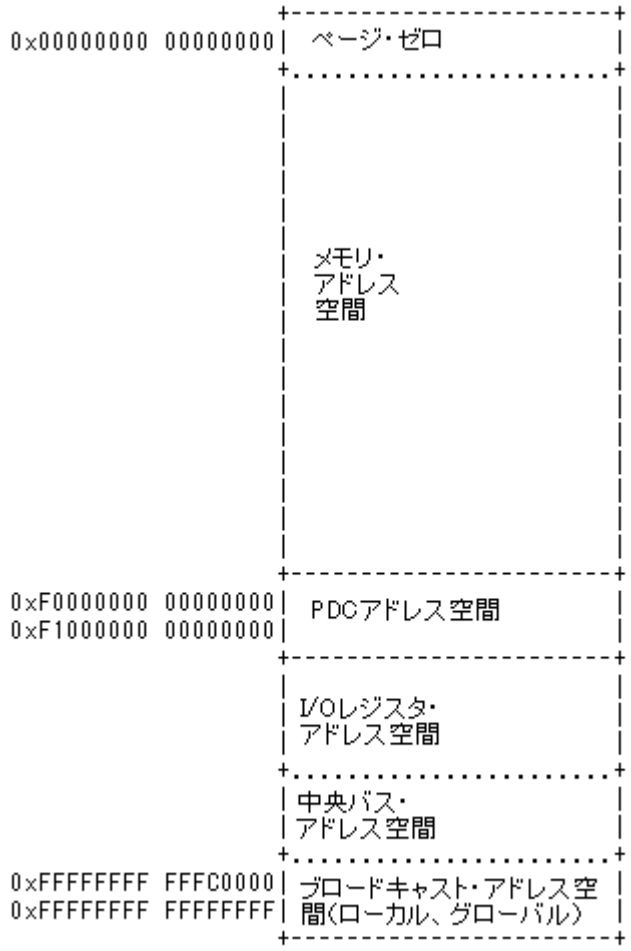


図 3 システム・アドレス空間の主要セクション（64 ビット・カーネルの場合）



ロック可能メモリ

プロセスが存続している間、システム・コール（mlock、plock、shmctl など）によってメモリ内に保持されるページを、ロック済みメモリと呼びます。ロック済みメモリをページングすることはできず、ロック済みメモリを持つプロセスを非アクティブ化することもできません。一般に、ロック済みメモリには、頻繁にアクセスされるプログラムやデータ構造（アプリケーション・コードの重要な部分など）が存在します。これらをメモリに常駐することで、アプリケーションのパフォーマンスが向上します。

ロック可能なメモリ量を追跡する変数が lockable_mem です。

使用可能メモリは物理メモリの一部であり、カーネルとそのデータ構造にとって必要な量を差し引いたものです。
lockable_mem の初期値は、起動後にシステム上で使用可能なメモリからシステム・パラメータ unlockable_mem の値を差し引いた値です。

ロック可能メモリの値は、次の要因によって決まります。

- インタフェース・カードの数と調整パラメータの値によって変わるカーネルのサイズ。
- システムによって異なる使用可能メモリ。

- システム・パラメータ unlockable_mem は、カーネル調整パラメータであり、unlockable_mem の値が変わると、lockable_mem の初期値も変わる。

HP-UX では、ユーザーがロックできる使用可能メモリの量に明確な限度はありませんが、ロックできないメモリ量については制限があります。

メモリを使用する他のカーネル・リソース（動的バッファ・キャッシュなど）によって、次の変化が起こることがあります。

- メモリが使用されると、ロックできるメモリ量が減る。
- メモリが解放されると、ロックできるメモリ量が増える。

ロック済みのメモリ量が増えるにつれて、ますます少なくなる使用可能メモリ・プールをめぐって既存のプロセス間で競合が起こります。この残りのメモリ・プール内のページ数がページングしきい値（lotsfree）を下回ると、システムで一般に使用される適度なメモリを維持するために、vhand をスケジュールすることでページング・メカニズムが有効になります。

プロセスの進行に十分なスペースを確保するよう、注意が必要です。十分なスペースがない場合、適度な空きメモリを確保するために、システムが頻繁にページングと非アクティブ化の処理を実行しなければならなくなります。

二次記憶領域

システムのメイン・メモリが不足すると、データが二次記憶領域に移されます。アクティブなプロセス用のメモリを確保するために、このデータは通常、システム・バスまたはネットワークを介してアクセス可能なディスクに保存されます。

元来スワップとは、プロセス全体をメイン・メモリと二次記憶領域との間で移動させる物理メモリ管理方式（以前の UNIX）を意味します。最近の仮想メモリ・システムではプロセス全体がスワップされることはなく、ページング方式が採用されています。この方式では、データと命令の個々のページを必要に応じて二次記憶領域からページインしたり、別の用途でメモリを解放するために再びページアウトすることができます。ページ方式を補助するのが非アクティブ化方式です。非アクティブ化方式では、システムのメモリ不足が深刻になった場合にプロセス全体を追い出すことができます。このように元来の方式とは異なりますが、旧称のなごりを残す形で、ページアウトされたデータの保存専用の二次記憶領域は、現在でも「スワップ・スペース（swap space）」と呼ばれています。

デバイス・スワップは、ディスク全体またはディスクの LVM（1）論理ボリュームの形をとることができます。スワップ用の空きスペースを確保するようにファイル・システムを構成できます。これをファイル・システム・スワップと呼びます。さらに多くのスワップ・スペースが必要な場合は、デバイス・スワップまたはファイル・システム・スワップのいずれかとして、実行中のシステムにスワップ・スペースを動的に追加できます。ディスク・スペースまたはファイル・システム内のディレクトリをスワップ用に割り当てる場合は、swapon コマンドを使用します。

（1）論理ボリューム・マネージャ（LVM）は、従来のディスク・パーティションよりも柔軟にディスク記憶領域リソースを処理できる基本ソフトウェア、およびコマンド群で構成されています。

仮想メモリの抽象化

コンピュータの使用可能 RAM の容量は限定されていますが、各 32 ビット HP-UX プロセスには、1GB のクワドラント（4 つに分割されたもの）が 4 つ割り当てられており、それぞれ計 4GB の仮想アドレス空間を保有しています（64 ビット HP-UX プロセスにはさらに大きな仮想アドレス空間が割り当てられますが、64 ビットでアドレス指定可能な仮想アドレスの全範囲[16 エクサバイト、つまり 16×1018]を実際に使用することはできません。64 ビット HP-UX プロセスでも、仮想アドレス空間は同じサイズの 4 つのクワドラントに分割されます）。これを仮想メモリと呼びます。

仮想メモリは、コンピュータ上で各プロセスを実行するスペースを十分に確保するためのソフトウェア構造です。また、仮想メモリ領域を割り当てるハードウェア（外部記憶装置）は欠かせません。

PA-RISC の仮想スペース

ソフトウェアのコンパイル時と実行時には、物理メモリよりも何倍も大きなメモリ空間をプログラマに提供する仮想アドレスが生成されます。

HP-UX は、共用アドレス空間（SAS：Shared Address Space）を使用するオペレーティング・システムです。特定の仮想アドレス（スペース ID を含む）がすべてのプロセスについて同じメモリ・ページを参照するため、プロセスの状況が変化しても変換は変わりません。

したがって、スペース ID（セグメント）とオフセット（通常は、単に「仮想アドレス（virtual address）」と呼ばれます）に使用できるビット数によって、カーネルとすべてのプロセスで使用可能な仮想アドレス空間全体の最大許容サイズが決まります。

PA-RISC の進化に伴い、スペースとオフセットに使用できるビット数が増えています。PA-RISC 2.0 では、スペース ID は 32 ビット（HP-UX 11.11 で実際に使用されるのは 18 ビット）であり、オフセットは実質 42 ビットです（ただし、64 ビット・フィールドに保存されます）。（PA-RISC 1.1 システムとナロー（32 ビット）モードで実行される PA-RISC 2.0 では、オフセットがこれよりも小さくなります。）

注: ただし単一プロセスでは、アクセス可能な仮想アドレス空間に大きな制限があります。たとえば、32 ビット SHARE_MAGIC の実行可能テキストは 1GB に制限され、データの上限も 1GB です。また、共用仮想アドレス空間の総量は、理論的にアドレス指定可能な量よりもはるかに少なく制限されています。メモリ・ウィンドウを使用しない場合、ワイド・モード（64 ビット）システム上の共用空間の総量は約 8TB（つまり、2 つの 64 ビットのクワドラント）に制限されています。

物理アドレス

物理アドレスは、4096 バイトのデータを表すメモリ内のページを指し示します。物理アドレスには、このページへのオフセットも含まれます。このため、完全な物理アドレスは、PPN とページ・オフセットで構成されます。PPN は、ページが配置される物理アドレスの上位 20 ビットまたは 52 ビットです。これらのビットは 12 ビットのページ・オフセットと連結して、32 ビットまたは 64 ビットの物理アドレスを形成します。

図 4 32 ビット物理アドレスのビット・レイアウト

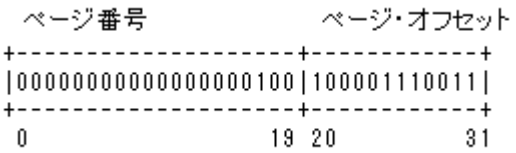


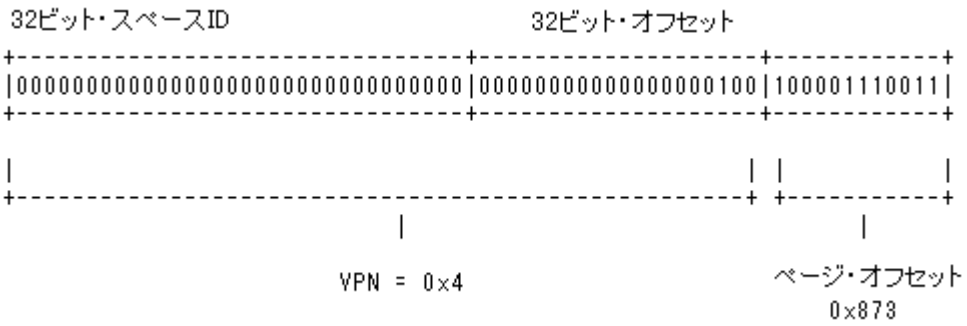
図 5 64 ビット物理アドレスのビット・レイアウト

仮想アドレスから物理アドレスへの変換を処理するために、仮想アドレスも仮想ページ番号（VPN : Virtual Page Number）とページ・オフセットで構成されているとみなす必要があります。ページ・サイズが 4096 バイトなので、オフセットの下位 12 ビットがページへのオフセットだとみなされます。スペース ID とオフセットの上位ビットが VPN です。

任意のアドレスについて、最下位 12 ビットを破棄することでページ番号を判断することができます。残りは、仮想アドレスの VPN または物理アドレスの PPN です。

次の図は、32 ビット仮想アドレス 0x4873 のビット・レイアウトを示しています。

図 6 32 ビット仮想アドレスのビット・レイアウト



VPN のアドレスを変換して、割り当てられたページ番号を取得し、該当するページ・オフセットを参照する必要があります。

プロセッサーのメモリ関連部分

図 7 主要コンポーネントを示すプロセッサー・アーキテクチャ

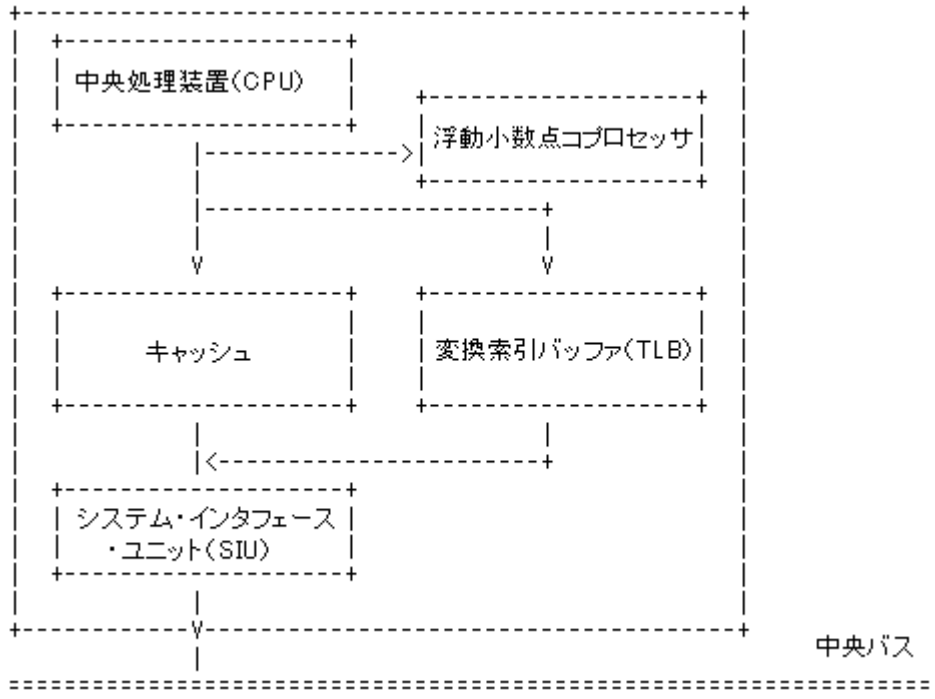


図 7 と表 1 に、主要なプロセッサー・コンポーネントを挙げています。その中でもレジスタ、変換索引バッファ（TLB：Translation Lookaside Buffer）、およびキャッシュがメモリ管理に不可欠であり、表 1 に続いてこれらのコンポーネントについて詳しく説明します。

表 1 プロセッサーのアーキテクチャ、コンポーネント、および目的

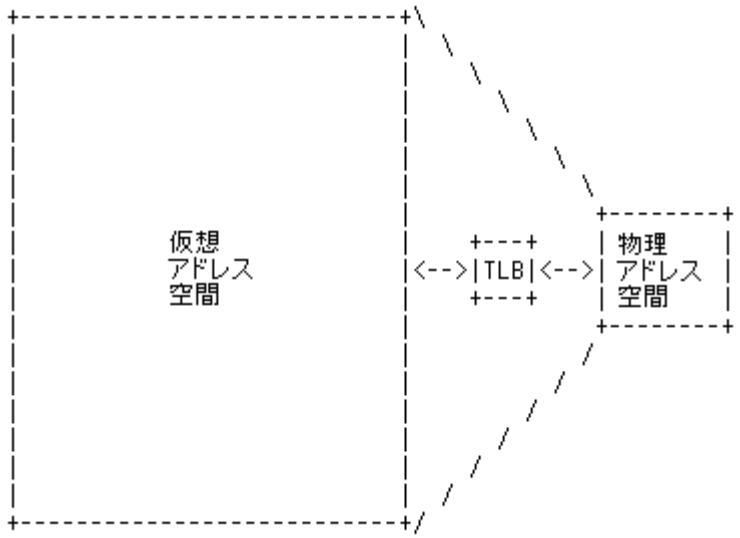
コンポーネント	目的
中央処理装置（CPU）	メモリからプログラムとデータを読み取り、プログラム命令を実行する主要コンポーネント。CPU には、次のコンポーネントが搭載されている。 ● データを保持する高速メモリとしてのレジスタ。レジスタは、計算、割り込み処理、保護機構、および仮想メモリ管理のために命令によって操作される。この章の最後で、レジスタについてより詳細に解説している。 ● 命令を解釈（デコード）して、適切な CPU ハードウェアをアクティブ化する制御信号を生成することで、CPU 動作の調整と同期化を行う制御ハードウェア（命令／フェッチ・ユニットとも呼ばれる）。 ● 実際の算術演算、論理演算、およびシフト演算を行う実行ハードウェア。実行ハードウェアは各種の特定のタスクを実行できるが、最も一般的な実行ハードウェアは算術論理

	演算装置（ALU：Arithmetic and Logic Unit）とシフト・マージ演算装置（SMU：Shift Merge Unit）。
命令／データ・キャッシュ	このキャッシュは、データと命令に高速にアクセスするために CPU で使用される高速メモリの部分。最後にアクセスされたデータがこのキャッシュに保持される。
変換索引バッファ（TLB）	このプロセッサー・コンポーネントの次の機能により、CPU が仮想アドレス空間を通じてデータにアクセスできる。 ● 仮想アドレスを物理アドレスに変換する機能。 ● アクセス権をチェックする機能。この機能により、要求しているプロセスに正しい権限がある場合のみ、命令、データ、または I/O へのアクセスが許可される。
浮動小数点コプロセッサー	CPU に代わって特別なタスクを実行する補助プロセッサー。
システム・インタフェース・ユニット	中心的な（ネイティブ）バスと CPU との通信を可能にするバス回路。

変換索引バッファ（TLB）

変換索引バッファ（TLB：Translation Lookaside Buffer）は、仮想アドレスを物理アドレスに変換します。

図 8 TLB の役割

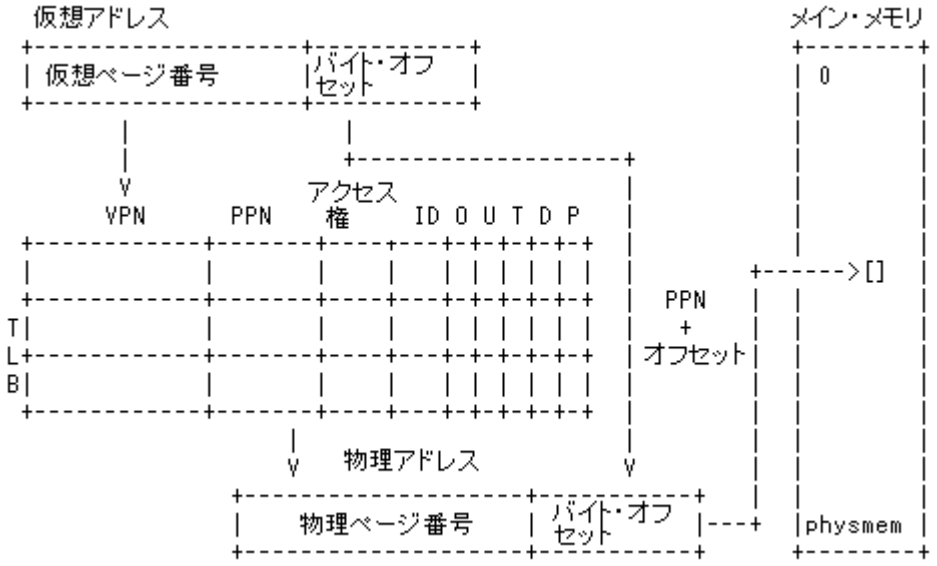


アドレス変換はメモリ階層の最上位から処理されます。まず最速コンポーネント（プロセッサー上の TLB など）をヒットし、次にページ・ディレクトリ・テーブル（メイン・メモリ内の PDIR）に移動し、最後に二次記憶領域に移ります。

TLB によるアドレス変換

TLB は仮想ページ番号（VPN）を見つけるために変換を検索し、物理メモリの参照に使用される物理ページ番号（PPN）を取得します。

図 9 TLB はアドレス変換用のキャッシュ



物理メモリのすべてのページの変換を保持できる十分な大きさの TLB が理想的ですが、このような TLB は非常に高価です。その代わり、TLB には、メモリ内のページ・ディレクトリ・テーブル（PDIR : Page Directory Table）からのエントリのサブセットが保持されます。最後に利用された変換のコピーが TLB によってキャッシングされるため、PDIR を調べる速度が向上します。

TLB の目的は仮想アドレスを物理アドレスに変換することなので、仮想モードでのメモリ・アクセス時のみ TLB が検索されます。この条件は、PSW（Processor Status Word）の D ビット（または、命令アクセスの I ビット）で示されます。

TLB の構成と種類

モデルに応じて次のいずれかの方法で、プロセッサー上で TLB を構成できます。

- 統合 TLB ― データと命令の両方の変換を保持する単一の TLB。
- 個別のデータ TLB（DTLB）と命令 TLB（ITLB） ― データまたは命令のいずれかのみの変換を保持する、プロセッサー内のデュアル TLB ユニット。

個別の DTLB と ITLB を装備することの利点は、データと命令の局所性のさまざまな特性とアクセス種類（データに対する頻繁なランダム・アクセスや、比較的連続的な命令の単一使用）を考慮できることです。

ブロック TLB

TLB のサイズは制限されているため、なるべく少ないエントリを使用して最大限のメモリを変換することが望まれます。PA-RISC 2.0 プロセッサーでは可変ページ・サイズを使用でき、適切な場合はラージ・ページ・サイズを使用するようにメモリが構成されます。特に、ブート時にカーネル用に最初に割り当てられるメモリは、そのメモリに適した最大限のページ・サイズでマッピング

されます（可能であれば他のメモリはラージ・ページ・サイズでマッピングされますが、メモリの少ないシステムでは特に、これが不可能になるというトレードオフ関係があります）。

PA-RISC 2.0 より前の PA-RISC プロセッサーでは、汎用可変ページ・サイズがサポートされていません。 その代わり、ブロック TLB を装備することができます。ブロック TLB は非常に小さいですが、そのエントリで 4KB の複数ページ（つまり、複数の HPDE）をマッピングできます。ブロック TLB のエントリは、常駐するカーネル・メモリの参照に使用されます（参照されるメモリをページ・アウトすることはできません）。一般にブロック TLB はグラフィックスに使用されます。その理由は、グラフィックスのデータが大きなチャンクでアクセスされるからです。また、ブロック TLB は、その他の静的領域（カーネルのテキストやデータなど）のマッピングにも使用できます。

TLB エントリ

TLB は仮想アドレスを物理アドレスに変換するため、各エントリには仮想ページ番号（VPN）と物理ページ番号（PPN）の両方が含まれます。また、アクセス権、アクセス識別子（ID）、および 5 つのフラグも含まれます。

表 2 TLB フラグ（PA 2.x アーキテクチャ）

フラグ	名前	意味
O	Ordered（順序付き）	ロード／格納のためのデータへのアクセスは、強度に応じてランク付け（強い順序付け、順序付け、弱い順序付け）される（モデルと定義については、PA-RISC 2.0 の仕様を参照）。
U	Uncacheable（キャッシング不可）	メモリ・アドレス空間からのページへのデータ参照をキャッシュに移動できるかどうかを決める。通常は、I/O アドレス空間にマッピングされるページへのデータ参照、またはキャッシュに移動してはならないメモリ・アドレス空間について、1 に設定される。
T(1)	Page Reference Trap（ページ参照トラップ）	このフラグが設定されている場合、このページにアクセスすると、ハードウェアまたはソフトウェアのトラップ・ハンドラによって参照トラップが処理される。
D	Dirty（ダーティ）	このフラグが設定されている場合、このビットは、メモリ内の関連ページがディスク上の同じページと異なることを示す。このページを無効にする前にフラッシュする必要がある。
B	Break（ブレーク）	このビットによって、このページへの書き込みが可能な命令に対してトラップが発生。
P	Prediction method for branching（分岐予測方法）	パフォーマンス・チューニングで使用（オプション）。

PA 1.x アーキテクチャでは、E ビット（「有効」ビット）は、TLB エントリがメモリ内の物理ページの現在の属性を反映することを示します。

ページ・テーブルまたは PDIR

このオペレーティング・システムでは、メモリ内の現在の仮想ページすべてを追跡するページ・ディレクトリ（PDIR : Page Directory）というテーブルがメモリ内で維持されます。ページが一部の仮想アドレス空間内でマッピングされる際、PDIR 内のエントリに割り当てられます。PDIR は、メモリ内の物理ページに仮想アドレスをリンクします。

PDIR は、ハッシュ・ページ・ディレクトリ・エントリ（HPDE : Hashd Page Directory Entries）と呼ばれるソフトウェア構造のメモリ常駐テーブルとして実装されます。HPDE には、仮想アドレスと物理アドレスが入っています。プロセッサーが索引のない物理ページを TLB 内で検索する必要がある場合、仮想アドレスで PDIR を検索して、一致するアドレスを見つけることができます。

PDIR テーブルは、衝突チェーンを持つハッシュテーブルです。仮想アドレスはハッシュテーブル内のエントリのいずれかへのハッシュに使用され、仮想アドレスが一致するチェーン・エントリが見つかるまで対応するチェーンが検索されます。

ページ・テーブルは純粋なソフトウェア構造ではないことに注意してください。TLB ミスを処理するハードウェアを備えるシステムでは、TLB ミス・フォールトの解決時に TLB に挿入する適切な変換の検索を試みるハードウェアによって、このページ・テーブルが調べられます。

ページ・フォールト

TLB 内に変換がないと、トラップが生じます。プロセッサーがその変換を PDIR で見つけることができると、TLB にその変換を挿入し、実行を継続できます。見つけられない場合、ページ・フォールトが発生します。

ページ・フォールトとは、プロセスで必要とされるアドレスがメイン・メモリ内にない場合に発生するトラップです。この事態は、PDIR ミスとも呼ばれます。PDIR ミスは、ページが空きリスト、ページ・キャッシュ、またはディスクのいずれかに存在することを示します。この場合、メモリ管理システムは、スワップ・デバイス上またはファイル・システム内で要求されたページを検索し、メイン・メモリにロードする必要があります。

これに対して、PDIR ヒットは、TLB 内に仮想アドレスの変換が存在することを示します。

ハッシュ・ページ・ディレクトリ（hpde および hpde2_0）構造

各 PDE には、仮想アドレスから物理アドレスへの変換に関する情報と、仮想メモリの各ページの管理に必要なその他の情報が含まれます。

PA-RISC 1.1 システムと PA-RISC 2.0 システムでは、異なる HPDE 構造が使用されていますが、フィールドの名前と目的はほとんど同じです。次の表では、PA-RISC 1.1 の HPDE（struct hpde）と PA-RISC 2.0 の HPDE（struct hpde2_0）の両方の構造要素を示しています。

表 3 ハッシュ・ページ・ディレクトリ struct hpde および struct hpde2_0

要素	PA-RISC のバージョン	意味
pde_valid	PA-RISC 1.1	有効な PDE エントリを示すためにカーネルが設定するフラグ。
pde_invalid	PA-RISC 2.0	無効な PDE を示すためにカーネルが設定するフラグ。
pde_vpage	両方	仮想ページ - 4096 バイトごとに分割される仮想オフセット。
pde_space	両方	完全な仮想スペース ID を含む。
pde_rtrap	両方	データ参照トラップ対応ビット。設定されている場合、ページにアクセスすると、ページ参照トラップ割り込みが発生。
pde_dirty	両方	メモリ内とディスク上でページが異なる場合にマークされるダーティ・ビット。
pde_dbrk	両方	TLB で使用されるデータ・ブレーク。
pde_ar	両方	TLB で使用されるアクセス権。 (1)
pde_uncache	両方	非キャッシュ・ビット。
pde_order	PA-RISC 2.0	強い順序付けビット。
pde_br_predict	PA-RISC 2.0	分岐予測ビット。
pde_ref_trickle	両方	参照用の漸増ビット。htbl を直接検索できるハードウェアを備えるシステム上で pde_ref で使用。
pde_block_mapped	両方	ページがブロック TLB でマッピングされ、エイリアス（別名）指定できないことを示すブロック・マッピング・フラグ。
pde_executed	両方	簡易キャッシュ・フラッシュ・アルゴリズムで使用され、ページがテキストとして参照されることを示す。 (2)
pde_ref	両方	特定の割り込みを受信した場合にカーネルが設定する参照ビット。ページが最近使用されたかどうかを通知するために vhand で使用。
pde_accessed	両方	簡易キャッシュ・フラッシュ・アルゴリズムで使用され、ページがデータ・キャッシュに存在できることを示す。

pde_modified	両方	スワップ・デバイスに最後に書き込まれてからページが変更されたかどうかを示す、高水準仮想メモリ・ルーチンへのインジケータ。
pde_uip	両方	トラップ処理コードで使用されるロック・フラグ。
pde_protid	両方	TLB で使用される保護 ID。
pde_os	PA-RISC 2.0	使用中のエントリ。
pde_alias	両方	仮想エイリアス・フィールド。設定されている場合、PDE が、スパースな（疎な）PDIR のメンバーとしてではなく、カーネル・メモリの他の場所から割り当てられている。
pde_wx_demote	PA-RISC 2.0（64 ビット・カーネルのみ）	ユーザー・スペース FIC（フラッシュ・インストラクション・キャッシュ：Flush Instruction Cache）。
pde_phys	PA-RISC 1.1	物理ページ番号。ページ・サイズ（4096 バイト）ごとに分割された物理メモリ・アドレス。
pde_phys_u	PA-RISC 2.0	物理ページ番号。最上位 25 ビット。
pde_phys	PA-RISC 2.0	物理ページ番号。ページ・サイズごとに分割された最下位 27 ビット・アドレス。
var_page	PA-RISC 2.0	ページ・サイズ。
pde_next	両方	次のエントリへのポインタ（リストの最後はヌル）。

（1）アクセス権の詳細は、『PA-RISC 2.0 Architectural reference』の第 3 章「Addressing and Access Control」を参照してください。このフィールドをプログラムで操作する方法の詳細は、マンページ mmap(2)および mprotect(2)を参照してください。

（2）簡易キャッシュ・フラッシュは、キャッシュをフラッシュするかどうかをカーネルに通知するためのパフォーマンス拡張機能です。

命令／データ・キャッシュ

キャッシュは、最近アクセスされた命令とデータを保存するための、プロセッサー・モジュール上の高速連想型メモリです。プロセッサーがデータに即時にアクセスするか、またはデータを求めて（より遅い）メイン・メモリに移る必要があるかを、キャッシュがプロセッサーに通知します。

メイン・メモリから CPU に移動するキャッシュ可能なデータは、キャッシュを通過します。逆に、キャッシュは、CPU がメイン・メモリとデータをやり取りする手段として機能します。キャッシュにより、最後に要求されたデータと命令のコピーが維持されるため、CPU がデータにアクセスする時間が短縮されます。

ほとんどのメモリ・アクセスは、以前にアクセスされたアドレスに非常に近いか同じアドレスへのアクセスなので、キャッシュによってシステム・パフォーマンスが向上します。CPU がアドレスを要求するたびにデータ・ブロックがキャッシュにロードされるという形で、この特性が活用されます。データ・ブロックはキャッシュのサイズ、連想性、およびワークロードに依存しますが、ほとんどの場合（パフォーマンスの測定に応じて）、キャッシュはユーザーが次回に希望するデータを保持するため、ユーザーはそのデータを参照できます。

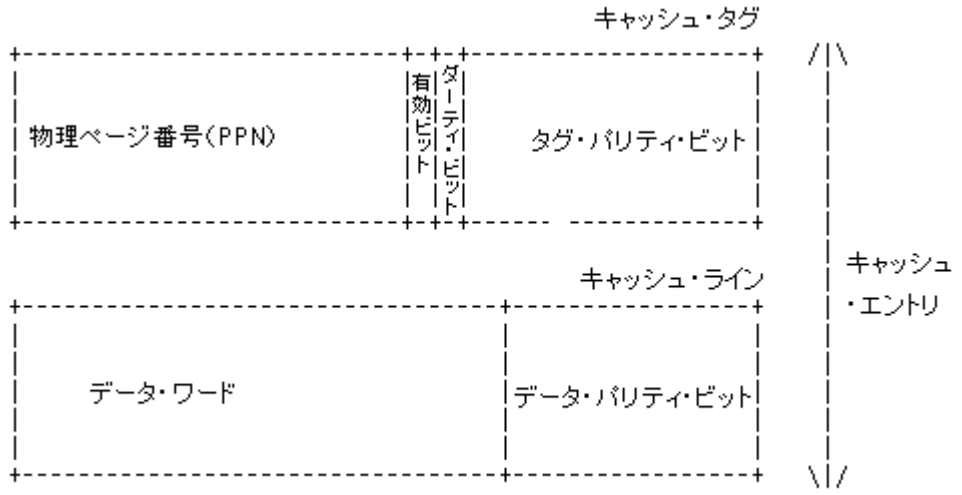
キャッシュの構成

一部のモデルには、統合キャッシュまたは命令とデータ用の個別のキャッシュが PA-RISC プロセッサーに搭載されています（局所性に優れ、高速になります）。マルチプロセッシング・システムでは各プロセッサーに固有のキャッシュが搭載され、キャッシュ・コントローラにより整合性が維持されます。

キャッシュ・メモリ自体は、次のように構成されます。

- キャッシュとメイン・メモリの間で受け渡されるデータと同じサイズ単位として定義された、キャッシュ・ラインという同じサイズのブロックの量。キャッシュ・ラインの長さは 16、32、または 64 バイトであり、そのサイズの倍数で整列しています。
- キャッシュ・ラインの内容を説明し、求められているデータが存在するかどうかを判断するための、各キャッシュに 1 つのキャッシュ・タグ。このタグには、次の情報が含まれます。
    - データが存在するメイン・メモリ内のページを識別する物理ページ番号（PPN）。
    - フラグ・ビット。設定されている場合、有効なフラグが、キャッシュ・ラインに有効なデータが含まれていることを示します。キャッシュ・ラインの内容が CPU により変更されている（つまり、メイン・メモリではなくキャッシュに最新のデータが含まれている）場合、ダーティ・ビットが設定されます。ダーティ・ビットが設定されていない場合、フラグは「クリーン」だといいます。これは、キャッシュ・ラインの内容が変更されていないことを意味します。その他の実装固有のフラグも存在することがあります。
    - キャッシュ・タグとキャッシュ・ラインの両方に、ラインが正しいことを確認するチェックサムに使用される関連パリティ・ビットが含まれます。

図 10 キャッシュ・タグとキャッシュ・ラインから構成される各キャッシュ・エントリ



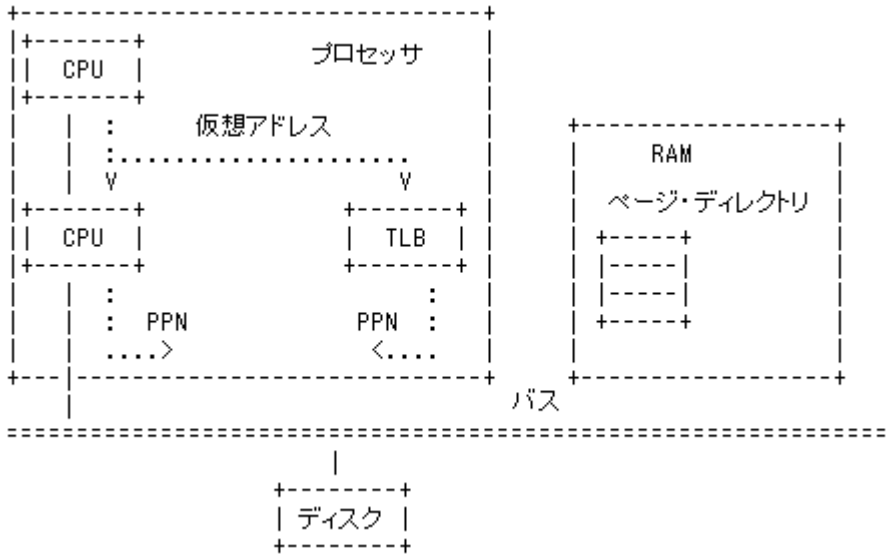
CPU によるキャッシュと TLB の使用

プロセスの実行時、プロセスのコード（テキスト）とデータが参照のためにプロセッサー・レジスタに保存されます。データまたはコードがレジスタ内に存在しない場合は、CPU が必要なデータの仮想アドレスを TLB とキャッシュ・コントローラに提供します。実装に応じて、キャッシュの直接マッピング、連想化、または完全な連想化が可能です。最近の PA の実装では、直接連想型キャッシュと完全連想型 TLB が使用されています。キャッシュには事実上索引が付けられているため、仮想アドレスを同時に TLB とキャッシュに送信できます。

複数の仮想ページから 1 つの物理ページを参照することはできません。また、1 つの仮想アドレスを 2 つの異なる物理アドレスに変換することもできません。つまり、PA-RISC ではハードウェア・アドレス・エイリアシングがサポートされていません。ただし、HP-UX では、EXEC_MAGIC 実行可能ファイルでのみテキストのソフトウェア・アドレス・エイリアシングが実装されています。

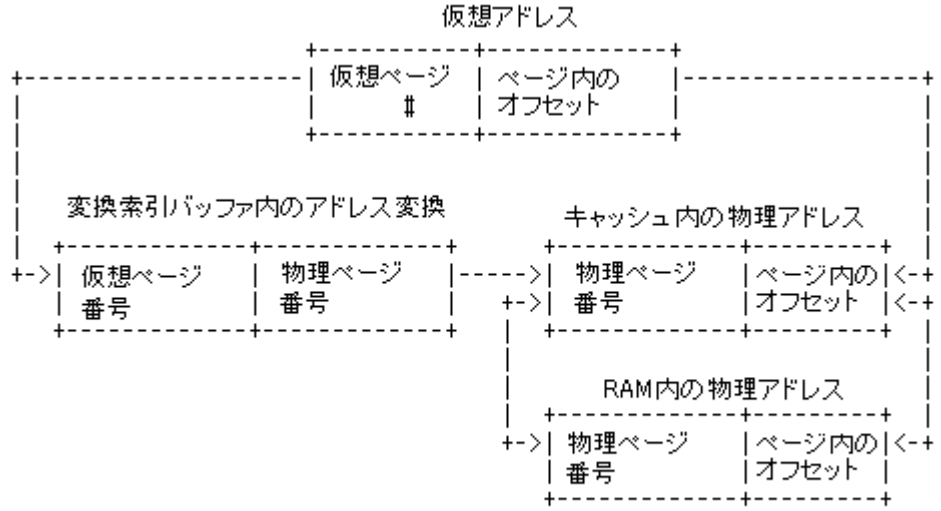
キャッシュ・コントローラは、仮想アドレスの下位ビットを使用して、直接マッピングされたキャッシュに索引を付けます。キャッシュの各索引により、物理ページ番号（PPN）を含むキャッシュ・タグとデータのキャッシュ・ラインがわかります。キャッシュ・コントローラが特定のキャッシュ場所でエントリを見つけると、キャッシュ・タグ内の PPN と TLB によって返された PPN を調べ、それが正しいものかどうかを確認するためにキャッシュ・ラインをチェックします。というのも、メイン・メモリのさまざまな場所からのブロックを、特定のキャッシュ場所に正しくマップできるためです。データがキャッシュ内に存在しないにもかかわらずページが変換される場合に生じるデータ・キャッシュ・ミスは、キャッシュ・コントローラによって完全に処理されます。TLB ミスは、TLB 内でページが変換されなかった場合に発生します。変換が PDIR 内にもない場合、ページ・フォールト処理コードを使用して変換がフォールトインされます。変換が RAM 内にない場合、データとコードをディスクからページングしなければならないことがあります。この場合、ディスクとメモリとの間の処理を実行する必要があります。

図 11 キャッシュからの PPN と TLB からの PPN の比較



次の図は、仮想アドレスの要素と物理アドレスの要素のマッピングを詳細なレベルで示しています。

図 12 仮想アドレスの変換



TLB のヒットとミス

プロセッサーは次の順序で、アドレスが「ヒットまたはミス（hit or miss）」のどちらなのかを検証します。

- TLB が検索されます。つまり、プロセッサーが発行した仮想アドレスとバイト・オフセットそれぞれで TLB 内のエントリが指し示されます。
- エントリが有効であれば、そのエントリは TLB ヒットと呼ばれます。TLB には、キャッシュ内でアクセスできる有効な物理ページ番号（PPN）が含まれます。
- エントリが無効かまたは TLB に物理ページ番号が含まれない場合、TLB ミスが発生します。このミスを処理する必要があります。一部のシステムでは、ハードウェア・ウォーカーが PDIR を検索し、ページが見つからない場合、TLB を更新します。ハードウェア TLB ハンドラを搭載していないシステムでは、ソフトウェア割り込みが発生します。ソフトウェア割り込みによってこのフォールトの解消と TLB の更新が行われ、アクセスが続行可能になります。

アクセス制御とページ保護における TLB の役割

TLB は仮想アドレス変換を補助するだけでなく、プロセッサーに代わってセキュリティ機能を果たします。これは、アクセスを制御し、ユーザー・プロセスが権限のあるデータのみを参照できるようにするという形で行われます。

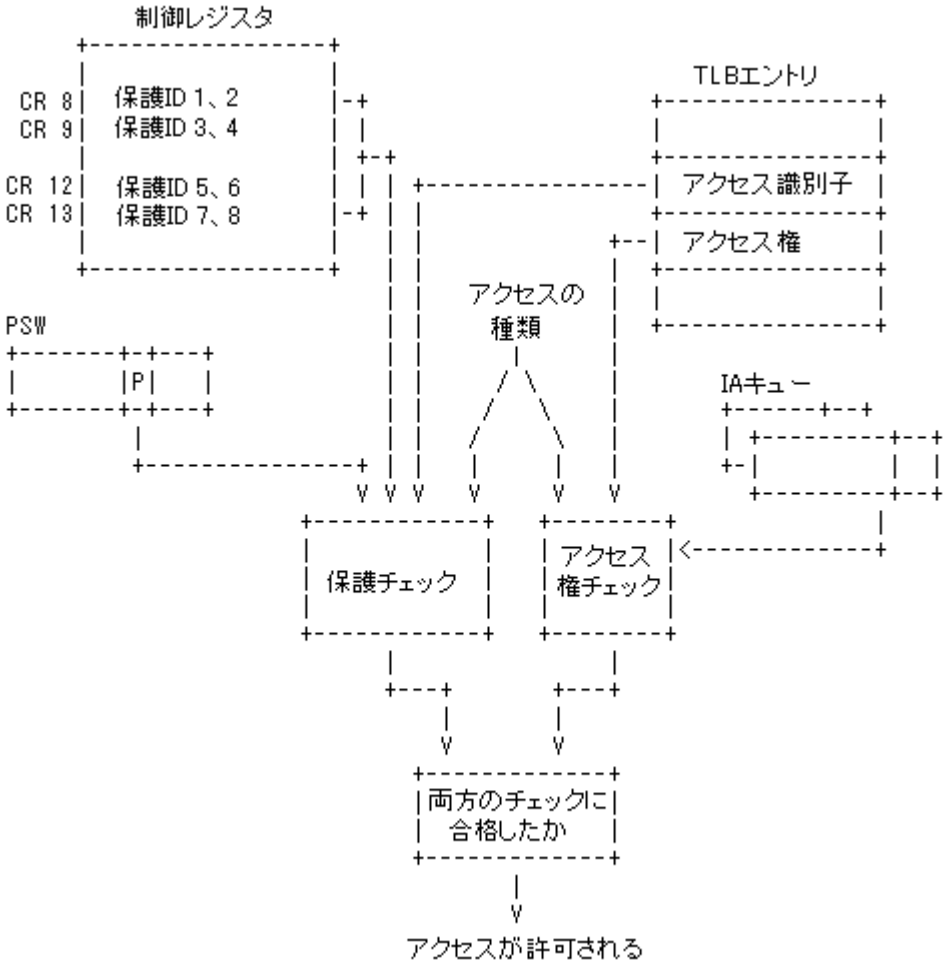
TLB には、アクセス権と保護 ID が含まれます。PA-RISC 2.0 では、最大 8 つの保護 ID を各プロセスに関連付けることができます。これらの ID は、制御レジスタ CR-8、CR-9、CR-12、CR-13 に保持されます（レジスタごとに 2 個）。PA-RISC 1.1 では、各プロセスに関連付けることができる保護 ID は 4 つのみです。

表 4 TLB でのセキュリティ・チェック

セキュリティ・チェック	目的
保護チェック	プロセッサー・ステータス・ワード（PSW）の P ビット（保護 ID 検証対応ビット）をチェック。 ● 設定しないと、あたかも保護チェックに合格してアクセス権の検証チェックに進むかのように、ページに対する保護チェックが省略される。 ● 設定すると、TLB エントリのアクセス識別子が CR-8、CR-9、CR-12、および CR-13 の保護 ID と比較される。
アクセス権チェック	命令の実行に影響を与える許可可能なアクセスの種類と 2 つの特権レベルを含む 7 ビットのフィールドに、アクセス権が保存されている。 ● アクセスの種類は、読み取り、書き込み、実行。 ● 特権レベルは、読み取りアクセス、書き込みアクセス、カーネルおよびユーザー実行についてチェックされる。

次の図は、TLB を介してデータページへのアクセスを制御するチェックポイントを示しています。TLB を介してデータページへのアクセスを制御するために、2 つのチェックが行われます。保護チェックとアクセス権チェックです。両方のチェックに合格すると、TLB により参照されるページへのアクセスが許可されます。

図 13 仮想ページに対するアクセス制御



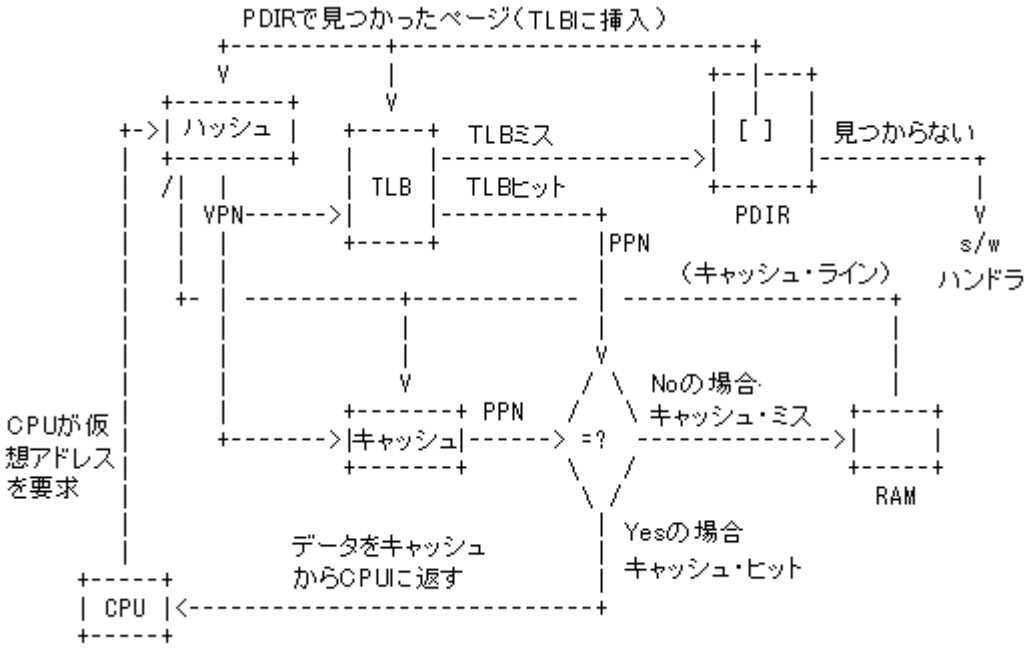
キャッシュのヒットとミス

- キャッシュ・ラインが最初にキャッシュにコピーされた際、その物理ページ番号が、対応するキャッシュ・タグに保存されています。キャッシュ・コントローラは、キャッシュ・タグの PPN と TLB からの PPN を比較します。
  - キャッシュ・タグの PPN と TLB からの PPN が一致する場合、キャッシュ・ヒットが生じます。データがキャッシュ内に存在し、CPU に提供されます。
  - キャッシュ・タグの PPN が TLB からの PPN と一致しない場合、キャッシュ・ミスが生じます。キャッシュ・ミスの場合、仮想ページ上で参照されるバイトがまだキャッシュ内に存在しないため、キャッシュ・ラインがメモリからロードされます（通常の実装では、一度にページ全体がキャッシュにロードされるのではなく、キャッシュ・ラインがロードされます）。データがキャッシュに存在しないため、データがキャッシュからメイン・メモリにロードされる間、CPU は待機しなければなりません。

    2 つの PPN が一致しない場合（TLB ヒットを前提として）、仮想ページ上で参照されるバイトがまだキャッシュ内に存在しないため、キャッシュ・ラインがロードされます。キャッシュ・ミスへの対処にかかる時間は、キャッシュ内に存在するデータがクリーンまたはダーティのどちらなのかによって異なります（キャッシュがダーティな場合、古い内容がメモリに書き出され、新しい内容がメモリから読み込まれます）。キャッシュ・ラインが「クリーン」な（つまり変更されていない）場合、メイン・メモリに書き戻す必要はありません。また、キャッシュがダーティであってメイン・メモリに書き戻す必要がある場合に比べて命令サイクルが少なくなります。

- すべての PA-RISC マシンでキャッシュ・ライトバック（書き戻し）ポリシーが採用されています。これは、キャッシュ・ラインが置き換えられる場合のみメイン・メモリが更新されることを意味します。

図 14 TLB、キャッシュ、PDIR からのページ検索のまとめ



- PA-RISC では、GATEWAY 命令を使用して特権レベルを上げることができます。この命令により、特権レベルを上げるページ間分岐が実行されます。HP-UX でのこの典型例はシステム・コールです。システム・コールにより、特権レベルがユーザーからカーネルに変更されます。

レジスタ

レジスタは、プロセッサーの CPU 内にある高速メモリです。レジスタは、命令制御の流れ、計算、割り込み処理、保護機構、および仮想メモリ管理のためにデータを保持する記憶要素としてソフトウェアで使用されます。

すべての計算はレジスタ間またはレジスタと（命令に埋め込まれた）定数との間で実行されるため、メイン・メモリまたはコードへのアクセスが最小限に抑えられます。このようにレジスタを多用する手法により、PA-RISC システムのパフォーマンスが向上します。このメモリは通常のメイン・メモリよりもはるかに高速ですが、高価でもあります。このため、プロセッサー専用で使用されます。

レジスタは、実行される命令の特権レベルに応じて、特権的または非特権的の 2 種類に分類されます。

表 5 レジスタの種類（PA-RISC 2.0）

レジスタの種類	目的
32 個の汎用レジスタ、サイズはすべて 64 ビット（非特権的）	パラメータ渡しなど、即時の結果または頻繁にアクセスされるデータを保持。次に挙げるのは、PA-RISC または HP-UX で使用されるレジスタ。

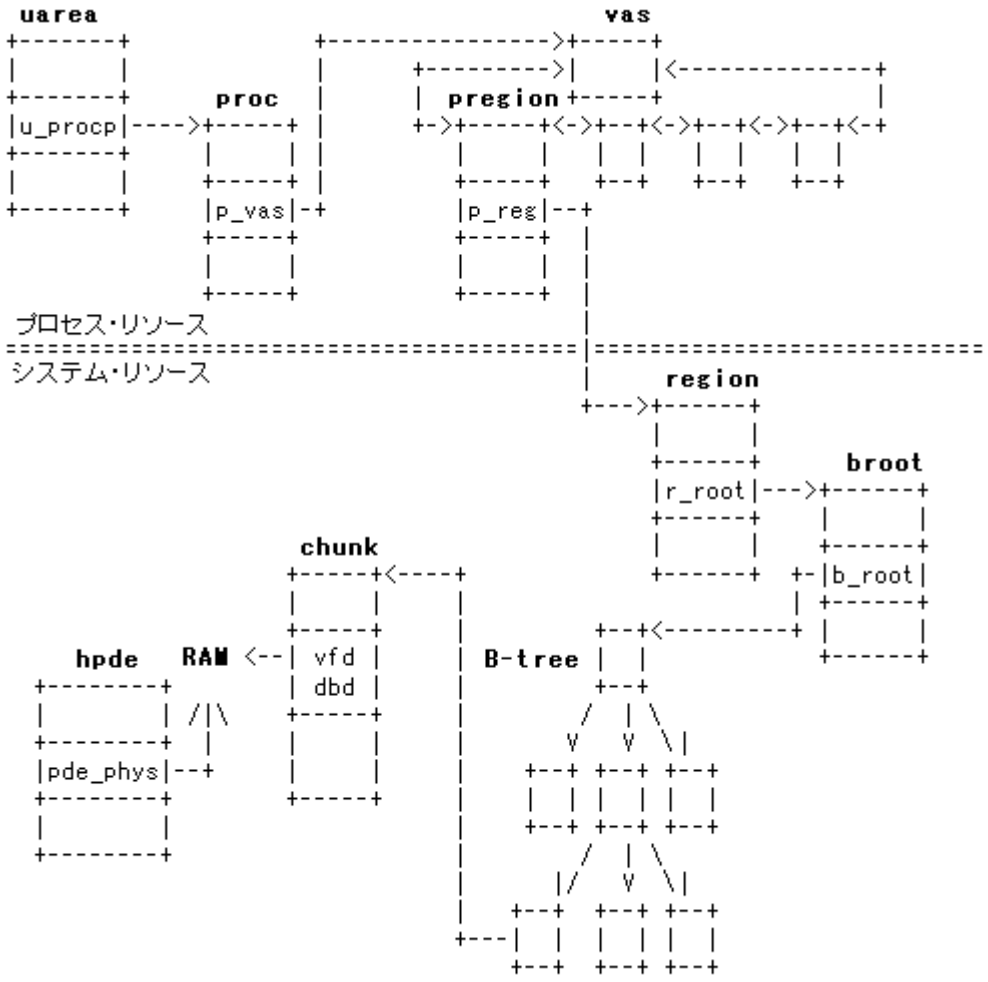
- GR0 - 常にゼロ
- GR1 - ADDIL ターゲット・アドレス
- GR2 - 戻り先の命令の命令オフセットを含むリターン・ポインタ
- GR19 - 引き数 7（arg7）（64 ビット・モード）
- GR20 - 引き数 6（arg6）（64 ビット・モード）
- GR21 - 引き数 5（arg5）（64 ビット・モード）
- GR22 - 引き数 4（arg4）（64 ビット・モード）
- GR23 - 引き数 3（arg3）
- GR24 - 引き数 2（arg2）
- GR25 - 引き数 1（arg1）
- GR26 - 引き数 0（arg0）
- GR27 - グローバル・データ・ポインタ（dp）
- GR28 - 戻り値
- GR29 - 戻り値（double）
- GR30 - スタック・ポインタ（sp）

7 個のシャドウ・レジスタ（特権的）	割り込み時に GR1、8、9、16、17、24、および 25 の内容を保存。これにより、割り込みからの復帰時に復元できる。SHR0～SHR6 の番号が付けられている。
8 個のスペース・レジスタ（SR5～SR7 は特権的）	現在実行中のプロセスのためのスペース ID を保持。 ● SR0 - 外部命令の分岐とリンクに使用される命令アドレス空間リンク・レジスタ。 ● SR1～SR7 - プロセスの仮想アドレス形成に使用。
25 個の制御レジスタ（CR0 および CR8～CR31 の番号付き）、すべて 64 ビット（ほとんどが特権的）	システムのさまざまな状態を反映するために使用され、多くは主に割り込み処理にかかわる。 ● CR0 - フォールト・トレラント・システムにおけるハードウェア障害のソフトウェア回復とデバッギングに使用される回復カウンタ。 ● CR10 - コプロセッサーの存在と可用性を示す、コプロセッサー構成レジスタ（CCR : Coprocessor Configuration Register）と呼ばれる下位 8 ビット。ビット 0 および 1 は浮動小数点コプロセッサーに対応し、ビット 2 は性能モニター・コプロセッサーに対応。 ● CR14 - 割り込みベクター・アドレス（IVA : Interruption Vector Address）。 ● CR16 - 間隔タイマー。2 つの内部レジスタ。一方のレジスタは実装固有の「最大命令速度」の 2 倍と半分の間の速さでカウントされ、他方のレジスタには 32 ビットの比較値が入る。マルチプロセッサー・システムの各プロセッサーには固有の間隔タイマーが備わっているが、同じ周波数で同期化することもクロックを刻む必要もない。 ● CR17 - 割り込み時に命令アドレス空間キューの内容を保存。

	● CR19 - 命令を割り込みハンドラに渡す。 ● CR20、CR21 - 仮想アドレスを命令ハンドラに渡す。 ● CR25 - ハッシュ・ページ・テーブル（htbl）のアドレスを含む。 ● CR26、CR27 - 任意の特権レベルで実行されるコードで読み取り可能だが、特権コードでのみ書き込み可能な一時レジスタ。
32 個の浮動小数点レジスタ、すべて 64 ビットか、64 ビットまたは 32 ビット	計算を保持するデータ・レジスタ。 ● FP-0L - 状態レジスタ。算術モードを制御し、トラップを使用可能にする。また、例外と比較結果を示し、コプロセッサー実装を識別。 ● FP-0R～FP-3 - 実行が完了すると遅延トラップを引き起こす浮動小数点演算に関する情報を含む例外レジスタ。
2 個の命令アドレス・キュー	深さが 2 要素の 2 個のキュー。キューの前部要素（IASQ_Front と IAOQ_Front）が現在の命令の仮想アドレスを形成し、後部要素（IASQ_Back と IAOQ_Back）には次回の命令のアドレスが含まれる。 ● 命令アドレス空間キューに、現在の命令と次回の命令のスペース ID を保持。 ● 命令アドレス・オフセット・キューに、特定のスペースの命令のオフセットを保持。上位 62 ビットには、命令のワード・オフセットが含まれ、下位 2 ビットには命令の特権レベルが維持される。
1 個のプロセッサー・ステータス・ワード（PSW）、64 ビット（特権的）	現在のプロセッサー状態。割り込みが発生すると、PSW（Processor Status Word）が割り込みプロセッサー・ステータス・ワード（IPSW : Interrupt Processor Status Word）に保存され、後で復元される。PSW の下位 5 ビットはシステム・マスクであり、マスク／マスク解除または有効／無効として定義される。PSW により無効にされた割り込みはプロセッサーが無視する。マスクされた割り込みは、マスク解除されるまで保留状態になる。

仮想メモリの構造

図 15 メモリ管理構造



プロセス管理では、pregion までのカーネル構造を使用してプロセスのスレッドが実行されます。uarea、proc 構造、vas、および pregion はプロセスごとのリソースです。つまり、複数のプロセス間で共有されず、これらの構造の重複しないコピーを各プロセスがもちます。

pregion より下のレベルは、システム全体のリソースです。これらの構造は、複数のプロセス間で共有されます（ただし、必ず共有する必要があるわけではありません）。

メモリ管理カーネル構造により pregion が物理メモリにマッピングされ、仮想アドレスを物理メモリに変換するプロセッサー機能がサポートされます。次の表は、メモリ管理にかかわるカーネル構造を示しています。詳細については、後に説明します。

表 6 主要なメモリ管理カーネル構造

カーネル構造	目的

vas	メモリ内のプロセスに関連する構造要素を追跡。プロセスごとに 1 つの vas を維持。
pregion	プロセスごとのリソースで、プロセスに付加された領域を記述。
region	プロセス間で共有できるメモリ常駐システム・リソース。プロセスの B-tree、vnode、pregion を指す。
B-tree	ページ索引とチャンク・アドレスを保存する平衡型ツリー（Balanced tree）。VFD（Virtual Frame Descriptors：仮想フレーム記述子）と DBD（Disk Block Descriptor：ディスク・ブロック記述子）の B-tree のルートに struct broot がある。
hpde	仮想から物理への変換（つまり、VFD から物理メモリへの変換）のための情報。

仮想アドレス空間（vas）

vas はプロセスの仮想アドレス空間を表し、プロセス領域データ構造（pregion）の二重リンク・リストの先頭の役割を果たします。vas データ構造はメモリ常駐です。

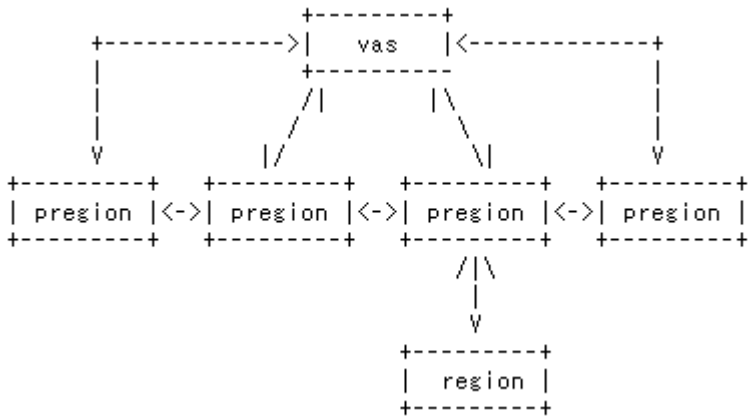
プロセスの作成時、システムにより vas 構造が割り当てられ、そのアドレスが proc 構造のフィールド p_vas に入れられます。

プロセスの仮想アドレス空間は、事実上隣接したページの論理チャンクに分割されます。

pregion の仮想メモリ要素

各 pregion は、プロセスの側からみた仮想アドレス空間の特定部分と、これらのページに到達するための情報を表します。pregion は、メモリまたは二次記憶領域内のページの物理的位置を示す region データ構造を指し示します。また pregion には、プロセスのページがマッピングされる仮想アドレス、ページ使用状況（テキスト、データ、スタックなど）、およびページ保護（読み取り、書き込み、実行など）も含まれます。

図 16 pregion の仮想メモリ要素



プロセスごとの pregion 構造の以下の要素が、仮想メモリ・サブシステムにとって重要です。

表 7 struct pregion の主要な要素

要素	目的
p_type	pregion の種類。
*p_reg	pregion により付加される領域へのポインタ。
p_space, p_vaddr	pregion の仮想アドレスで、仮想スペースと仮想オフセットから構成される。
p_off	region へのオフセットで、ページ内で指定される。
p_count	pregion によりマッピングされるページの数。
p_ageremain、 p_agescan、 p_stealscan、 p_bestnice	メモリ・ページのエージングとスチール（流用）を行うために vhand アルゴリズムで使用。
*p_vas	pregion のリンク先の vas へのポインタ。
p_forw, p_back	アクティブな pregion をウォークするために vhand で使用される二重リンク・リスト。
p_deactsleep	非アクティブ化されたプロセスが休眠状態にあるアドレス。
p_pagein	データをメモリに移すタイミングをスケジュールするために使用される I/O サイズ。
p_strength、 p_nextfault	順次フォールトとランダム・フォールトとの比率の追跡に使用。p_pagein の調整に使用される。

システム・リソースの領域

領域は、ページ・グループを特定のプロセスに関連付けるシステム全体にわたるカーネル・データ構造です。領域には、専用（単一のプロセスで使用）と共有（複数のプロセスで使用可能）の 2 種類があります。必要に応じて、領域データ構造用のスペースが割り当てられます。領域構造はスワップ・デバイスに決して書き込まれませんが、領域構造の B-tree は書き込まれることがあります。

プロセスごとのリソース pregion が領域を指し示します。領域は、メモリ内に存在しないデータ・ブロックが存在する vnode を指し示します。

表 8 領域（struct region）

要素	目的
r_flags	領域フラグ（すぐ後で列挙）。

r_type	• RT_PRIVATE : 複数プロセスで領域を共有できない。PT_DATA と PT_STACKpregion が RT_PRIVATE 領域を指し示す。 • RT_SHARED : 複数プロセスで領域を共有できる。PT_SHMEM とほとんどの PT_TEXT pregion が RT_SHARED 領域を指し示す。
r_pgsz	ページ内の領域のサイズ（現在メモリ内にあるページだけではない）。
r_nvalid	領域内の有効ページの数。これは、B-tree または b_chunk 内の有効な vfd の数と同じ。
r_dnvalid	スワップされる領域内のページ数。システムによりプロセス全体がスワップされる場合、r_nvalid の値がここにコピーされ、後に再びフォールトインする際に、プロセスに必要なページ数の計算に使用される。この情報は、再アクティブ化するプロセスの決定に使用される。
r_swalloc	スワップ・デバイス上のこの領域用に予約され割り当てられるページ総数。vfd/dbd ペア用に割り当てられるスワップ・スペースは計算に入れられない。
r_swapmem、 r_vfd_swapmem	擬似スワップまたは vfd 用に予約されるメモリ。
r_lockmem	vfd/dbd ペア用に割り当てられたロック可能メモリを含め、ロック可能メモリ用の領域に現在割り当てられているページ数。
r_pswapf、 r_pswapb	擬似スワップ・ページ（pswaplist）を使用する領域リストへの順方向ポインタと逆方向ポインタ。
r_refcnt	領域で参照している pregion の数。
r_zomb	変更されたテキストを示すために設定。リモート・システム上で実行中の a.out ファイルが変更されている場合、プロセッサーのキャッシュからページがフラッシュされて、次回のアクセスの試行がフォールトになる。フォールト・ハンドラが r_zomb がゼロでないことを認識し、メッセージ Pid %d killed due to text modification または page I/O error を出力し、プロセスに SIGKILL を送信。
r_off	ページ境界に整列した vnode へのオフセット（ページ単位）。RF_UNALIGNED が設定されていない場合のみ有効。vnode のページ r_off は、領域の B-tree の第 1 チャンクの第 1 エントリから参照。
r_incore	関連するプロセスで SLOAD フラグが設定されている領域を共有する pregion の数。
r_dbd	スワップ・デバイスに書き込まれる B-tree ページ用のディスク・ブロック記述子。これにより、最初のページの位置を指定。ページは、スワップ・スペースの隣接エリアにまとめて保存される。

r_fstore、r_bstore	ブロックの転送元と転送先の vnode へのポインタ。このデータは、領域より上の pregion の種類によって異なる。一般に、r_bstore はページ方式 vnode（システム起動時に初期化されるグローバル swapdev_vp）に設定される。
r_forw、r_back	アクティブな region すべてのリンク・リストへのポインタ。
r_lock	領域構造変更用の読み取りロックまたは読み取り／書き込みロックを取得するために使用される領域ロック構造。
r_mlock	この領域上で mlock 操作をシリアライズ（直列化）するためのロック。
r_poip	実行中のページ I/O の数。
r_root	B-tree のルート。複数のチャンクを参照する場合、r_key が DONTUSE_IDX に設定される。
r_key、r_chunk	vfddbd のチャンクが 1 つのみ必要な（32 ビット・カーネル上で 32 以下のページ、または 64 ビット・カーネル上で 64 以下のページを参照する）場合に、 B-tree 検索（r_root）の代わりに使用。
r_next、r_prev	vnode を共有するすべての領域の循環リンク・リスト。
r_preg_un	領域を指し示す pregion（1 つまたは複数）。
r_excproc	プロセスの r_flags が RF_EXCLUSIVE に設定されている場合の、proc テーブル・エントリへのポインタ。
r_excproc	プロセスの r_flags が RF_EXCLUSIVE に設定されている場合の、proc テーブル・エントリへのポインタ。
r_lchain	メモリ・ロック範囲のリンク・リスト。
r_mlockswap	ロックを扱うために予約されているスワップ。
r_pgszhint	ページ・サイズのヒント情報。
r_hdl	ハードウェア依存の階層構造。

非整列ページのサポート（a.out）

ほとんどの実行可能ファイルのテキストとデータは、4KB のページ境界上で始まります。ファイル内のページがメモリ内のページに直接マッピングするため、HP-UX では、これらのファイルをメモリ・マップ・ファイルとして扱うことができます。

上記のフィールドに加え、struct region には、テキストとデータが（4KB）ページ境界上で整列しない古いバージョンの HP-UX でコンパイルされた実行可能ファイルをサポートするフィールドも含まれます。これらの実行可能ファイルは、r_flags が RF_UNALIGNED に設定されている領域によって参照されます。

表 9 領域によりサポートされる非整列 a.out

要素	目的
r_byte、r_bytelen	a.out ファイルへのオフセットとそのテキストの長さ。
r_hchain	非整列領域のハッシュ・リスト。

領域フラグ

領域の状態を示すさまざまなインジケータが r_flags で指定されます。指定可能なフラグ値の一部を以下に示します。

表 10 領域フラグ

領域フラグ	意味
RF_ALLOC	HP-UX 領域の割り当てと解放は必要に応じて行われるため、常に設定される。空きリストはない。
RF_UNALIGNED	実行可能ファイルのテキストがページ境界上で始まらない場合に設定される。この場合、整列するためにテキストがバッファ・キャッシュによって読み取られ、vfd がバッファ・キャッシュ・ページで指し示される。
RF_WANTLOCK	スレッドがこの領域の vfd のロック（ページ上の I/O のために）を求めたが、すでにロックされていることがわかって休眠状態になった場合に設定される。vfd のロック解除後、このフラグにより、wakeup()が呼び出されて待機中のスレッド（1 つまたは複数）を続行できることが保証される。
RF_HASHED	テキストが非整列であり（RF_UNALIGNED）、ハッシュ・チェーン上にある。領域は r_fstore と r_byte でハッシュされる。各ハッシュ・チェーンの先頭は、texts[]内にある。RF_UNALIGNED フラグを RF_HASHED フラグなしで設定できるが（システムがハッシュされた領域の取得を試みたのに、その領域がロックされていた場合、システムは専用領域を作成）、RF_HASHED フラグは RF_UNALIGNED フラグなしでは設定されない。
RF_EVERSWP、RF_NOWSWP	B-tree がこれまでスワップ・デバイスに書き込まれたことがあるか、現在書き込まれている場合に設定される。これらのフラグは、デバッギングに使用される。
RF_IOMAP	この領域は iomap()システム・コールで作成されたため、exit()の呼び出し時に特別な処理が必要。

RF_LOCAL	リモート・ファイルでローカル・スワップ・スペースが使用されている。
RF_EXCLUSIVE	マッピング・プロセスが領域への排他的アクセスを許可されている。このフラグが設定されている場合、r_excproc が proc テーブル・ポインタに設定される。
RF_STATIC_ PREDICT	コンパイラを最適化するために、テキスト・オブジェクトで静的分岐予測が利用されている。
RF_ALL_MLOCKED	領域全体がメモリにロックされている。
RF_SWAPMEM	領域で擬似スワップが使用されている。つまり、メモリの一部がスワップ用に保持されている。
RF_LOCKED_ LARGE	ラージ・ページを使用して、領域がロックされている。
RF_SUPERPAGE_ TEXT	テキスト領域でラージ・ページが使用されている。
RF_FLIPPER_ DISABLE	カーネルの補助予測が無効になる。パフォーマンス・プロファイリングに使用されるフラグ。
RF_MPROTECTED	領域の一部が、メモリ・マップ・ファイル上で実行されるシステム・コール mprotect の対象になる。

領域のページの検索

領域フィールド r_key、r_chunk、および r_root を使用して、region の個別のページに関する情報を検索します。

各ページは、vfd（メモリ内に存在する場合）または dbd（ディスク上に存在する場合）で表されます。

それぞれのページごとに、vfd と dbd が struct vfddbd にグループ化されます。定義では、vfd の pg_v ビットが設定されている場合、vfd が使用され、pg_v ビットが設定されていない場合、dbd が使用されます。

一般に（個別のページでなく）ページのグループに関する情報が必要なため、ページはチャンクにまとめられます。1 つのチャンクに、仮想フレーム記述子（VFD）とディスク・ブロック記述子（DBD）の 32 または 64 のペアが含まれます。

仮想フレーム記述子（vfd）

仮想フレーム記述子と呼ばれる 1 ワードの構造により、プロセスがメモリ・ページを参照できます。vfd は、プロセスがメモリ内に存在する場合に、pfdat テーブル（ 後述の pfdat_ptr[]）に記述されている物理メモリのページを参照するために使用できます。

図 17 仮想フレーム記述子（vfd）

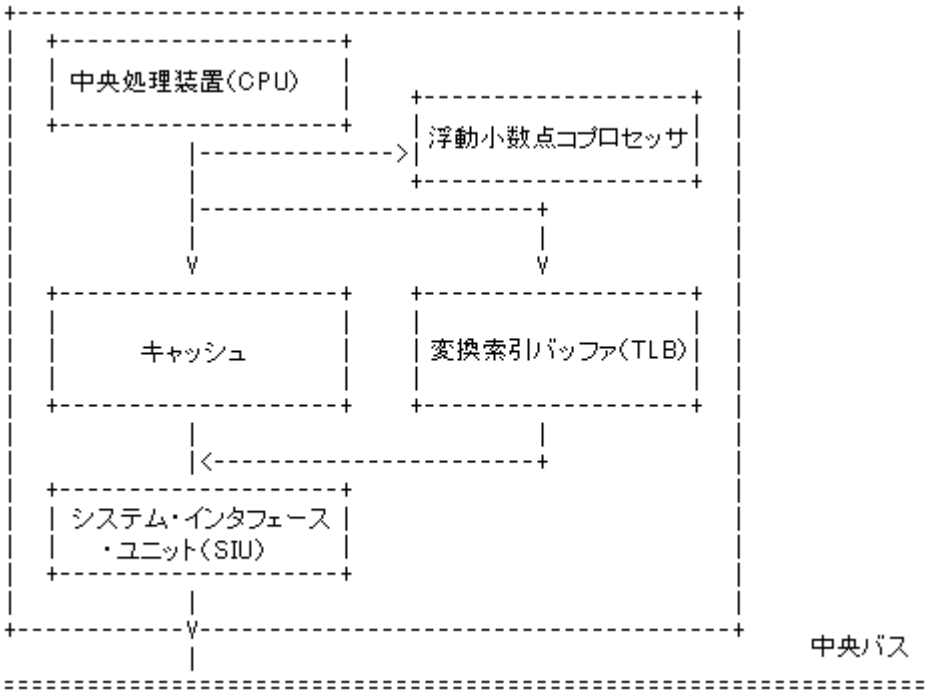


表 11 仮想フレーム記述子（struct vfd）

要素	意味
pg_v	有効なフラグ。設定されている場合、メモリのこのページに有効なデータが含まれ、pg_pfnum が有効になります。設定されていない場合、このページの有効なデータはスワップ・デバイス上に存在します。
pg_cw	「書き込み時コピー（copy-on-write）」フラグ。設定されている場合、ページへの書き込みにより、データ保護フォールトが発生し、その時点でシステムによりページがコピーされます。
pg_lock	ロック・フラグ。設定されている場合、このページで raw 入出力が発生している。データがページとディスクの間で転送されているか、2 つのメモリ間で転送されている。このページに対する次の raw 入出力を開始する前に、カーネルは入出力の完了を待機して休眠状態になる。ディスクに書き込み中のページを読み出すことはできない。
pg_mlock	設定されている場合、ページがメモリ内でロックされており、ページ・アウトできない。
pg_pfnum（pg_pfn のエイリアス）	ページ・フレーム番号。この番号から、このページの正しい pfdat にアクセスできる。

ディスク・ブロック記述子（dbd）

vfd の pg_v ビットが設定されていない場合、vfd が無効であり、データのページがメモリ内でなくディスク上に存在します。この場合、ディスク・ブロック記述子（dbd）により、データへの有効な参照が提供されます。vfd 構造と同じく、dbd の長さも 1 ワードです。

図 18 ディスク・ブロック記述子（dbd）

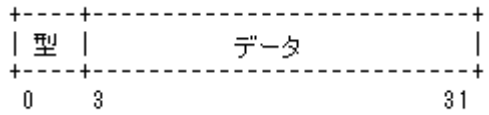


表 12 ディスク・ブロック記述子（struct dbd）

要素	意味
dbd_type	次のデータ型がある。 ● DBD_NONE：このデータのコピーがディスク上に存在しない。 ● DBD_FSTORE、DBD_BSTORE：領域の vnode で指し示される「順方向格納」または「逆方向格納」デバイス上でページを検索できる。（1） ● DBD_DFILL：デマンド・フィル・ページ。スペースは割り当てられない。障害が発生すると、ディスクからのデータで初期化される。 ● DBD_DZERO：デマンド・ゼロ・ページ。要求時に、新しいページが割り当てられ、ゼロで初期化される。 ● DBD_HOLE：スパースな（疎な）メモリ・マップ・ファイルに使用。読み取り時にはページのデータがゼロになる。書き込み時にページが割り当てられ、ゼロで初期化され、データが挿入される。この時点で、dbd_type が DBD_FSTORE に変わる。
dbd_data	vnode のタイプ（jfs、nfs、ufs、スワップ・スペース）に固有のデータ。vnode で指し示されるファイル内のデータを検索するために、ファイル・システム（または、スワップ・スペース管理）のコードで使用。

（1）dbd_type が DBD_FSTORE の場合、データのページは r_fstore で指し示されるデバイス（通常はファイル・システム）のファイルに存在しています。dbd_type が DBD_BSTORE の場合、データのページは r_bstore で指し示されるデバイス（通常はスワップ・デバイス）のファイルに存在しています。

vfds と dbds を 1 つの場所に維持するチャンク

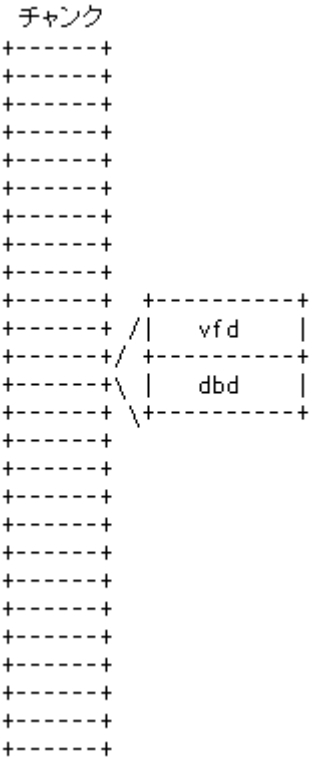
一般に、（個別のページでなく）ページのグループに関する情報が必要なため、ページはチャンクにまとめられます。1 つのチャンクに、仮想フレーム記述子とディスク・ブロック記述子の 32 または 64 のペアが入ります。

- カーネルは、仮想フレーム記述子（vfd）でメモリ内のページを検索する。

- カーネルは、ディスク・ブロック記述子（dbd）でディスク上のページを検索する。
- 定義では、vfd の pg_v ビットが設定されている場合、vfd が使用され、設定されていない場合、dbd が使用される。

vfd と dbd の一対一対応が vfddbd 構造全体で維持されます。この構造には、1 つの vfd（c_vfd）と 1 つの dbd（c_dbd）のみが含まれます。

図 19 vfddbd の配列を含むチャンク



HP-UX 領域では、vfd と dbd のチャンクを使用してページ所有権が追跡されます。

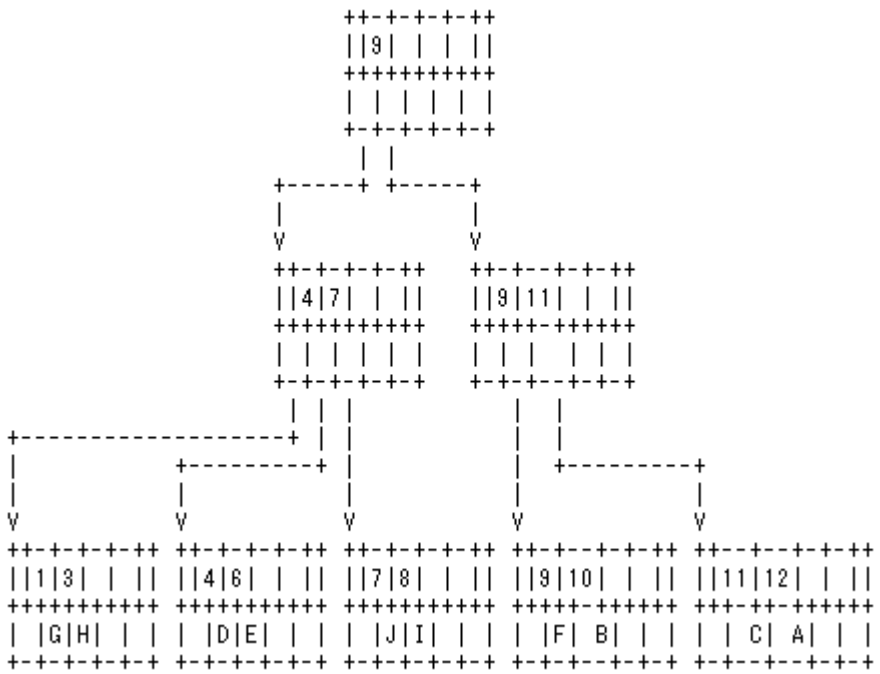
- ページが有効な場合の、仮想ページから物理ページへの割り当て（PDIR に加えてこれが必要です）。ページは変換されないが有効なため、マッピングではなく「割り当て（assignment）」という言葉が使用されます。
- ページのその他の仮想属性（ページがメモリ内でロックされているかどうか、ページが有効かどうかなど）。
- 定義では、vfd の pg_v ビットが設定されている場合、vfd が使用され、設定されていない場合、dbd が使用される。

平衡型ツリー（B-tree）

各領域に、vfddbd の単一配列（chunk）、または B-tree（Balanced Tree）へのポインタのいずれかが含まれます。B-tree と呼ばれる構造により、スパース・データ（疎なデータ）の迅速な検索と効率的な保存が可能になります。bnode のサイズは chunk と同じです。両方とも同じメモリ・ソースから取得できます。領域の B-tree に、ページ索引とチャンク・アドレスのペアが保存されます。HP-UX では、オーダー29 の B-tree が使用されます。

B-tree はキーで検索され、値が生成されます。領域 B-tree では、キーは、領域内のページ番号をチャンク内の vfddbd の数で除算したものです。

図 20 B-tree のサンプル（オーダー= 3、深さ= 3）



B-tree の各ノードには、オーダー+1 のキー（または索引番号）とオーダー+2 の値用の場所が含まれます。1 つのノードがオーダー・キーを含む以上に成長した場合、2 つのノードに分割されます。ペアの半分は元のノードに保持され、別の半分は新しいノードにコピーされます。B-tree ノード・データには、ノードに含まれる有効な要素の数も含まれます。

表 13 B-tree ノードの説明（struct bnode）

要素	意味
b_key[B_SIZE]	bnode の各ページ索引に使用されるキー配列。
b_nelem	bnode 内の有効なキー／値の数。
b_down [B_SIZE+1]	bnode 内の値の配列。この値は、別の bnode へのポインタ（これが内部 bnode の場合）またはチャンクへのポインタ（これがリーフ bnode の場合）のいずれか。

B-tree のルート

struct broot と呼ばれる構造が B-tree の始点を指し示します。

表 14 struct broot

要素	意味
b_root	B-tree の始点へのポインタ。

b_depth	B-tree 内のレベルの数。
b_npages	B-tree の構成に使用されるページ数であり、チャンクと bnode に使用される両方のページがカウントされる。
b_rpages	ルーチン grow_vfdpgs()を使用してカーネルが B-tree 用に予約したスワップ・ページの数。B-tree 構造内の vfd/dbd ペアに割り当てられるスワップ量を示す。
b_list	この領域内の bnode または chunk に使用されるメモリ・ページのリンク・リストへのポインタ。通常、このリストの最初のページには、使用可能な空きスペースがある（b_nfrag がゼロでない場合）。ここから新しい bnode または chunk を割り当て、B-tree に追加できる。
b_nfrag	b_list 内の使用可能な（未割り当て）チャンクの数。チャンクはページの最後から割り当てられるため、ページ内で最後に割り当てられたチャンクの索引でもある（次の使用可能チャンクを取得するために、索引が減分される）。
b_rp	B-tree を使用する領域へのポインタ。
b_protoidx、b_proto1、b_proto2	2 つのプロトタイプ dbd 値と、b_proto1 から b_proto2 への切り替えが行われるページ索引。チャンク・スペースを割り当てる際の時間とメモリ・コストを最小限に抑えるために使用。
b_vproto	「書き込み時コピー」のページ範囲のリスト。これにより、実際の B-tree エントリをすぐに割り当てなくても、ページを「書き込み時コピー」に設定できる。vfd プロトタイプの決定に使用（下記の「vfd プロトタイプ」を参照）。
b_key_cache[]、b_val_cache[]	最後に使用されたキーのキャッシュと、そのキーに関連するチャンクへのポインタ。特定の struct vfddbd の検索時（B-tree の検索前）に最初にチェックされる。

vfd プロトタイプ

struct broot の b_vproto フィールドに、「書き込み時コピー」として扱われるページ範囲のリストが含まれます。これにより、B-tree エントリをすぐに割り当てることなく、ページを「書き込み時コピー」に設定できます。このプロトタイプの型は struct vfdcw です。vfd の作成時、このリストにページがあるかどうかをチェックして、どのプロトタイプを使用するかが決められます。

表 15 struct vfdcw

要素	意味
v_start [MAXVPROTO]	「書き込み時コピー」範囲の最初を示すページ。未使用の場合は、-1 に設定。
v_end[MAXVPROTO]	「書き込み時コピー」範囲の最後。

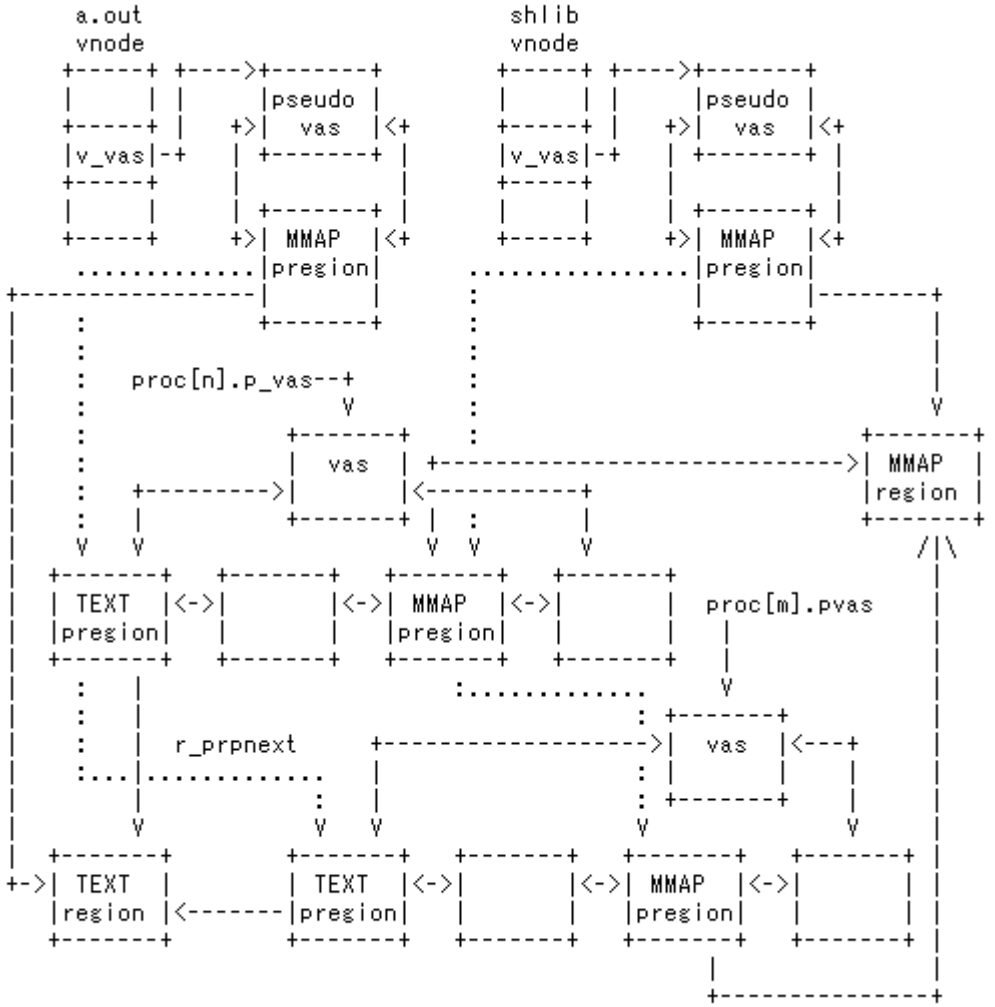
テキストと共有ライブラリ pregion 用の pseudo-vas

ファイルを a.out または共有ライブラリとしてオープンする場合、領域を追跡する最も簡単な方法は、ファイルを実行可能ファイルとして最初にオープンする際に pseudo-vas を作成することです。作成するには、mapvnode()を呼び出し、vas ポインタを vnode の v_vas 要素に保存します。以降、ファイルを実行可能ファイルとしてオープンする場合、仮想アドレス空間が付加される領域の検索に v_vas の非ヌル値が役立ちます。

pseudo-vas のタイプは PT_MMAP であり、関連の pregion の p_flags が PF_PSEUDO に設定されています。この pregion が、この vnode の領域に付加されます。次に、この実行可能ファイルまたは共有ライブラリ（非擬似 pregion）を使用するすべてのプロセスが、タイプが PT_TEXT（a.out）または PT_MMAP（共有ライブラリ）である領域に付加されます。実行可能ファイルとして特定の vnode を使用するプロセスの数は、va_refcnt の pseudo-vas に保持されます。
領域に関連する pregion は、領域要素 r_pregs で始まる二重リンク・リストに連結され、p_prpnext と p_prpprev により pregion 内で定義されます。このリストは p_off（領域への pregion のオフセット）でソートされ、ヌルで終わります。

a.out または共有ライブラリを使用するすべてのプロセスが終了した後でも、領域へのハンドルは残ります。この時点でページを処理できます。

図 21 pseudo-vas 構造のマッピング



ハードウェアに依存しないページ情報テーブル（pfdat）

ページ・フレーム・データ（pfdat）テーブルは、物理メモリの再配置可能ページすべてを表す 2 レベルのテーブルです（カーネルのブート時に恒久的に割り当てられるメモリは表されません）。このテーブルは、ページ・フレーム番号（pfn、PPN と同じ）で索引が付けられた巨大な配列とみなすことができます。

物理メモリ・アドレスが常にページ 0 から始まり、連続して増える場合、単一レベルの配列として実装されます（ハードウェアにこのような連続アドレス範囲が備わっていたため、古い HP-UX リリースでもこの方法で実装されていました）。ただし、最近のシステムの中には、物理アドレスに大きな隙間がある場合があります（たとえば、ページ 0～0x1000 のメモリの次にページ 0x20000～0x21000 のメモリがあるシステムがあります）。すべてのアドレスを表すテーブルは、実際に必要なテーブルよりもはるかに大きくなるからです。

したがって、最初の階層（pfdat_ptr）がサブテーブルへのポインタの配列になります。各ポインタは、ありうる物理アドレス空間の PFN_CONTIGUOUS_PAGES（0x1000）ページ分を表しますが、この範囲に実際の物理メモリがない場合、ポインタはヌルになります（メモリを最大限に節約するために、ブート時に恒久的に割り当てられるメモリは、このテーブルにとって存在しないものとみなされます）。

pfdat 構造の用途は次のとおりです。

- 物理メモリ・アロケータの空きリストを pfdat のリストにリンクする。
- ページング可能な内容（ファイルからのページなど）を含む物理ページのページ・キャッシュを維持する。ページはハッシュ・チェーンでリンクされ、ページ・フォールトを解決する必要がある際に内容を素早く見つけられるようになっている（このようなページは「ハッシュ済み（hashed）」ページと呼ばれることがある）。

表 16 struct pfdat 内の主要エントリ（ページ・フレーム・データ）

要素	意味
pf_hchain	ハッシュ・チェーンのリンク。
pf_devvp（1）	デバイスの vnode。
pf_next、pf_prev	次回と前回の空き pfdat エントリ。
pf_vnext、 pf_vprev	同じ vnode に関連するページのリンク・リストのリンク。
pf_lock	ロック pfdat エントリ（ベータ・セマフォ）。pde（物理から仮想への変換、アクセス権、または保護 ID）の変更中にページをロックするために使用。
pf_pfn	物理ページ・フレーム番号。
pf_use	ページを共有する領域の数。pf_use がゼロに下がると、空きリンク・リストにページを配置できる。

pf_cache_waiting	設定されている場合、ページ上で pf_lock を捕捉するためにスレッドが待機している。
pf_data	pf_devvp 内のこのページを一意に識別するためのディスク・ブロック番号またはその他のデータ。
pf_sizeidx	物理メモリ空きリストにあるラージ・ページの基本ページのサイズを識別。このサイズで、ページを配置する空きリストが決まる。
pf_size	使用中の可変サイズ・ページのサイズ。
pf_flags	ページ・フレーム・データ・フラグ（図 17 を参照）。
pf_hdl	ハードウェアに依存する階層要素（下記の hdlpfdat の説明を参照）

(1) 集合 (pf_devvp, pf_data)がハッシュされます。

ページの状態を示すフラグ

表 17 主要な pf_flag 値

フラグ	意味
P_FREE	ページが未使用（割り当て可能）。
P_BAD	メモリ再配置サブシステムにより、ページが不良だとマークされている。
P_HASH	ページがハッシュ・キュー上にある。
P_SYS	ページがユーザー・プロセスではなくカーネルによって使用されている。このフラグでマークされたページには、動的バッファ・キャッシュ・ページ、B-tree ページ、および動的カーネル・メモリ割り当ての結果が含まれる。
P_DMEM	メモリ診断サブシステムによってメモリがロックされている。dmem ドライバへの ioctl()呼び出しで設定と解除が行われる。
P_LCOW	「書き込み時コピー」によってページが再マッピングされている。
P_UAREA	タイプが PT_UAREA の pregion によってページが使用されている。
P_KERN_DYNAMIC	ページがカーネル動的メモリで使用されている（P_SYS のサブセット）。カーネル動的メモリ空きリストにページが入れられる。

P_KERN_NO_LGPG	ページの再マッピングを求めるユーザーによってページが（カーネル動的メモリとして）割り当てられている（これは、ラージ・ページの一部になることはできない）。P_KERN_DYNAMIC のサブセット。
P_SP_POOL	ページがカーネル動的メモリ・アロケータのスーパーページ・プール空きリストにある。P_KERN_DYNAMIC のサブセット。

ハードウェアに依存する階層ページ・フレーム・データ・エントリ

struct pfdat の pf_hdl フィールドに、各ページに関するハードウェア依存情報が含まれます。これの型は、hdl_pfdat.h で定義されている struct hdlpfdat です。

表 18 struct hdlpfdat

要素	意味
hdlpf_flags	ページの HDL 状態を示すフラグ。フラグの値とその意味は次のとおり。 • HDLPF_TRANS : このページについて、仮想アドレス変換が存在。 • HDLPF_PROTECT : ページがユーザー・アクセスから保護される。このフラグが設定されており、HDLPF_STEAL が設定されていない場合、保存された値（下記）が有効になる。 • HDLPF_STEAL : 保留中の I/O が完了すると、仮想変換を削除する必要がある。 • HDLPF_MOD : hpde の pde_modified フラグの変更に類似。 • HDLPF_REF : hpde の pde_ref フラグの変更に類似。 • HDLPF_READA : 移行における先読みページ。現在の I/O 要求の完了を待機する前に次の I/O 要求を開始する必要があるルーチン hdl_pfault()を示す。
hdlpf_savear	保存されたページ・アクセス権。
hdlpf_saveprot	保存されたページ保護 ID。

仮想メモリから物理メモリへのマッピング

PA-RISC ハードウェアは、TLB を検索することで、仮想アドレスから物理アドレスへの変換を試みます。アドレスを解決できない場合、ページ・フォールト（命令 TLB ミス・フォールトの場合は割り込みタイプ 6、データ TLB ミス・フォールトの場合は割り込みタイプ 15）が生成されます。カーネルは、このフォールトを処理する必要があります。

HTBL

PA-RISC では、ハッシュ・ページ・ディレクトリ・エントリ（hpde）のハッシュ・ページ・テーブル（htbl）を使用して、膨大な仮想アドレス空間内のアドレスが特定されます。ハッシュテーブルのアドレスは、制御レジスタ 25（CR25）に含まれます（reg.h を参照）。

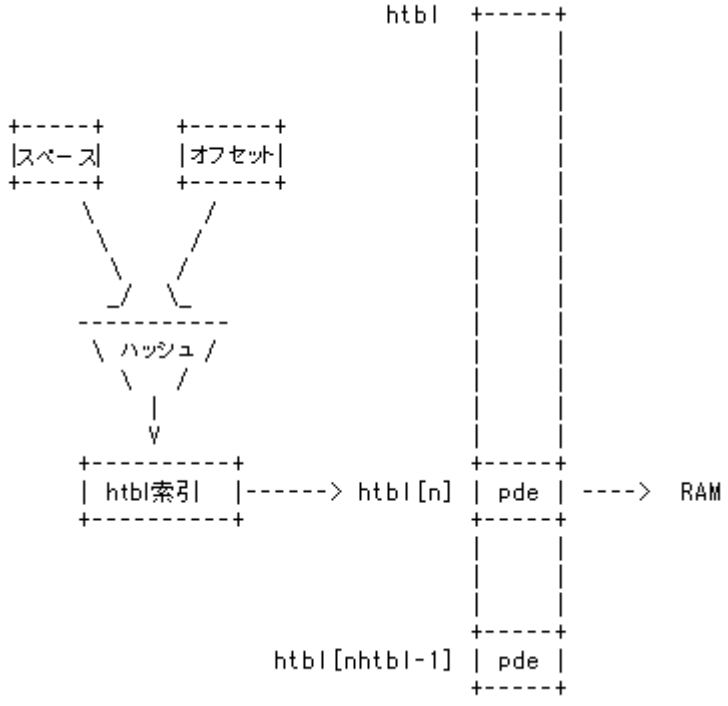
このテーブルの詳細は、「ページ・テーブルまたは PDIR」を、各テーブル・エントリの詳細は、「ハッシュ・ページ・ディレクトリ（hpde および hpde2_0）構造」をそれぞれ参照してください。

注: 歴史的な理由から、このテーブルのエントリが pde、hpde、または pdir と呼ばれることがあります。

htbl 内のアドレスを検索する方法は次のとおりです。

- 仮想スペースとオフセットがハッシュされて、htbl 索引が生成される。
- ハッシュ・アルゴリズムで生成された索引が、htbl への索引として使用される。テーブル内の各エントリは pde と呼ばれ、型は struct hpde になっている。
- 仮想スペースとオフセットが pde の情報と比較されて、エントリが確認される。
- 物理アドレスが pde から検索されて、仮想アドレスから物理アドレスへの変換が完了する。

図 22 htbl エントリからページ・ディレクトリ・エントリへのマッピング



プロセス管理では、pregion までのカーネル構造を使用してプロセスのスレッドが実行されます。uarea、proc 構造、vas、および pregion はプロセスごとのリソースです。つまり、複数のプロセス間で共有されず、これらの構造の重複しないコピーを各プロセスがもちます。

pregion より下のレベルは、システム全体のリソースです。これらの構造は、複数のプロセス間で共有されます（ただし、必ず共有する必要があるわけではありません）。

メモリ管理カーネル構造により pregion が物理メモリにマッピングされ、仮想アドレスを物理メモリに変換するプロセッサー機能がサポートされます。次の表は、メモリ管理にかかわるカーネル構造を示しています。詳細については、後に説明します。

同じ htbl エントリへの複数アドレスのハッシュ

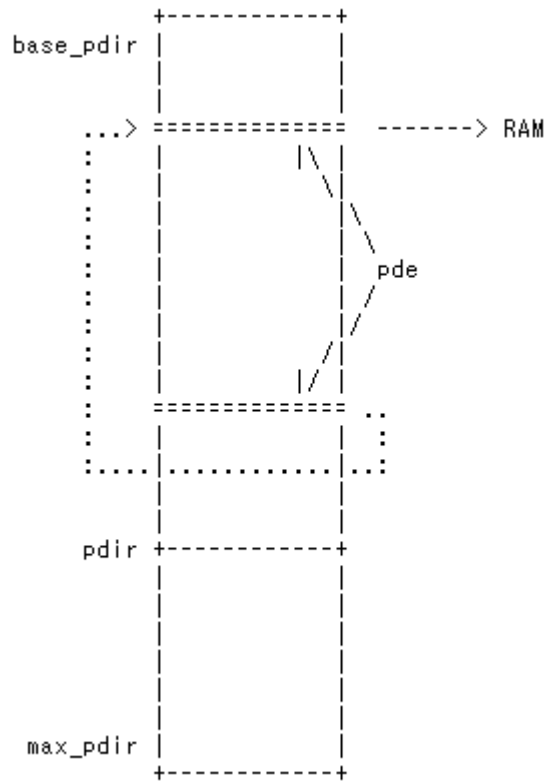
どのハッシュ・アルゴリズムでも、複数のアドレスを同じ htbl 索引にマッピングできます。htbl 内のエントリは、実際には pde のリンク・リストの始点です。各エントリに、別の pde を指し示す pde_next ポインタがありますが、リンク・リストの最後の項目である場合はヌルが含まれます。

実際には、htbl に十分なエントリが含まれ、3 つを超えるリンクを含むまでリンク・リストが成長することはほとんどありません。

各 htbl エントリが、base_pdir から htbl までと pdir（htbl の最後でもあります）から max_pdir まで、pde の 2 つの集合を指し示すことがあります。htbl 全体と周囲の pde をまとめてスパース（疎な）pdir と呼びます。htbl は常に、そのサイズの倍数（つまり、nhtbl * sizeof(struct hpde)の倍数）となるアドレスから始まるように整列されます。

pdir_free_list または pd_fl2_0->head は、現在使用されていないが使用可能なスパース pdir エントリのリンク・リストを指し示します。pdir_free_list_tail または pd_fl2_0->tail は、このリンク・リスト上の最後の pde を指し示します（これらの変数名は、PA-RISC 1.1 の pdir 実装と PA-RISC 2.0 の pdir 実装で少し異なります）。

図 23 複数のアドレスを同じ htbl エントリにハッシュする方法



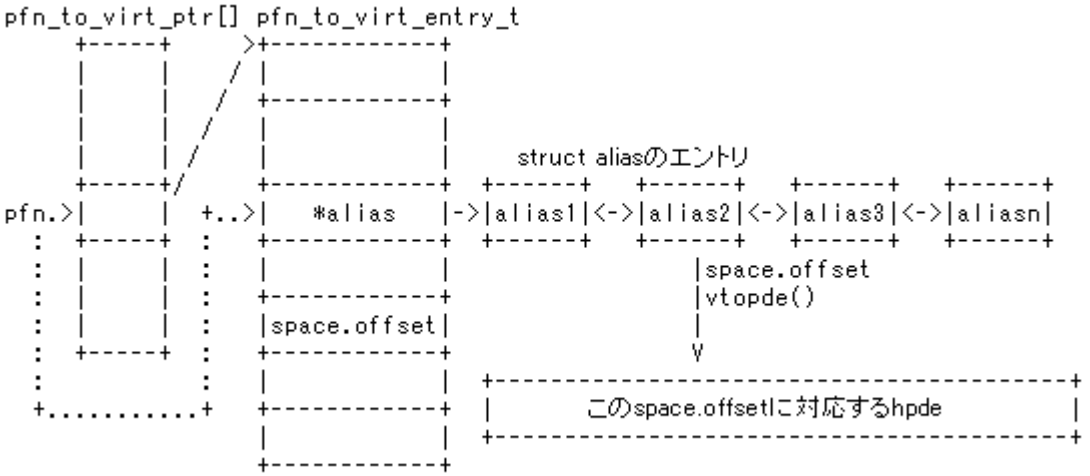
物理アドレスから仮想アドレスへのマッピング

HP-UX では、ハッシュ・ページ・ディレクトリを使用して仮想アドレスを物理アドレスに変換します。

物理アドレスから仮想アドレスへの変換には、pfn_to_virt テーブルが使用されます。これは pfdat テーブルと同様に 2 レベルのテーブルであり、物理メモリの各ページについて 1 つの pfn_to_virt_entry_t エントリを含む巨大な配列とみなすことができます。第 1 レベルのテーブルを pfn_to_virt_ptr[]と呼びます。

各 pfn_to_virt_entry_t には、仮想ページのスペースとオフセット（1 つのページへの単一変換の場合）またはエイリアス構造のリスト（物理ページに複数の仮想アドレス変換がある場合）のいずれかが含まれます。

図 24 物理アドレスから仮想アドレスへの変換



pfn_to_virt_entry_t に、物理アドレスに対応する space.offset（仮想アドレス）、または、それぞれが space.offset ペアを持つエイリアス構造のリストへのポインタが含まれることがあります。

アドレス・エイリアシング

HP-UX は、ほとんどのプラットフォーム上でソフトウェア・アドレス・エイリアシングをサポートしています（ハードウェア・アドレス・エイリアシングは 16MB 境界で実装されますが、ソフトウェア・アドレス・エイリアシングはページ単位で（各ページは 4KB）実装されます）。ソフトウェア・アドレス・エイリアシングは、別のオペレーティング・システムほど頻繁には使用されません。一般に HP-UX では、複数の仮想アドレスに同じオブジェクトをマッピングすることはありません。

テキスト・セグメントを最初に変換する際、テキスト・セグメントにはエイリアスがありません。ただし、同じテキスト・セグメントにプロセスまたはスレッドが属する場合、もう一度変換が必要になることがあります。テキスト・セグメントを共有するプロセスでは、エイリアスを使用しません。「書き込み時コピー」を使用してデータ・ページを共有する専用テキスト・セグメントを持つプロセスでのみ、エイリアスが使用されます。ユーザー・ページのカーネル変換を追加するためにエイリアスが使用されることもあります。

複数の仮想アドレスを同じ物理アドレスに変換する場合、HP-UX ではエイリアス構造を使用して仮想アドレスを追跡します。ページ・フレーム（pfn）のエイリアスが、pfn_to_virt_entry_t から始まるエイリアス・チェーンで維持されます（ラージ・ページでは、ページの基本 pfn に対応する pfn_to_virt_entry_t からエイリアスがリンクされます）。pfn_to_virt_entry_t のスペー

ス・フィールドが無効であり、オフセット・フィールドがゼロでない場合、ゼロでない値がエイリアス構造のリンク・リストの最初を指し示します。各エイリアス構造に、エイリアスのスペースとオフセット、および pde のアクセス権と保護 ID 用の一時保持フィールドが含まれます。エイリアスの基本 pfn の pfdat の pf_lock によって、エイリアス・チェーンの読み取りと変更が防止されます。

特定のエイリアス・スペースおよびオフセットを hpde で検索するために、hpde チェーンとそれに対応する pd_lock に対してスペースとオフセットがハッシュされます。pd_lock が取得されると、vtopde()ルーチンが hpde をウォークして、タグの一致を見つけます。

aa_entfreelist は、空きエイリアス・エントリの二重リンク・リストの先頭です。仮想アドレスから物理アドレスへのこの新しい変換の情報が保存されている aa_entfreelist から、エイリアス構造が取得されます。
グローバル変数 max_aapdir に、システム上のエイリアス hpde の総数が含まれます。エイリアス hpde として使用するためにページが割り当てられると、そのページは返されないため、max_aapdir の値が時間がたつにつれて増加することがありますが、減少することはありません。

使用可能なエイリアス hpde の数は、aa_pdircnt に保存されます。エイリアス hpde が使用または予約されると（後で移動しなければならない場合に備えて、htbl hpde をエイリアス・リンク・リストに入れる場合は、エイリアス hpde が 1 つ予約されます）、aa_pdircnt が減分されます。エイリアス hpde が aa_pdirfreelist に戻されるか予約を解除されると、aa_pdircnt が増分されます。

使用可能なエイリアス構造の数は、aa_entcnt に保持されます。エイリアス構造のグループとして使用するためにページが割り当てられると、そのページは返されません。システム上のエイリアス構造の総数でなく、使用可能な構造の数のみが追跡されます。

ページ可用性の維持

次の 2 つの計算要素によりページ可用性が維持されます。

- あらゆるページング・イベントの原因になるページングしきい値。
- 実際のページングと非アクティブ化を処理する vhand および sched デーモン（システム・プロセス）。

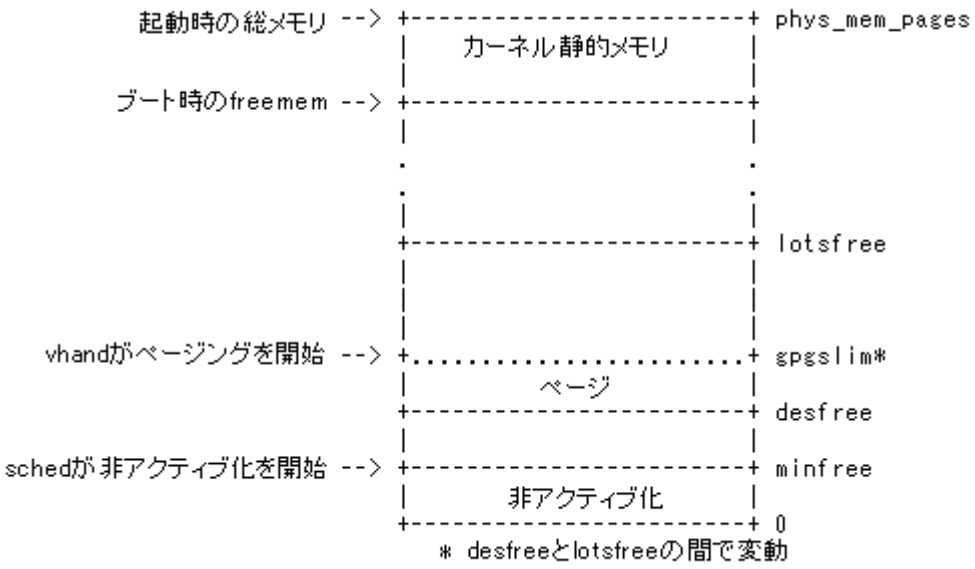
vhand による空きページの監視によって、しきい値を上回る空きページ数が維持され、デマンド・ページング用に十分なメモリが確保されます。vhand は、このページ方式の全体的な状態を管理します。sched は、メモリ内の使用可能ページ数が一定のレベルを下回ると有効になります。後に、vhand と sched の動作について説明します。

注: sched プロセスの通称はスワッパーです。

ページングしきい値

メモリ管理では、さまざまなページング・アクティビティの原因になるページングしきい値が使用されます。次の図は、使用可能メモリの全範囲と、メモリ・レベルが各ページングしきい値を下回った場合に発生するページング・アクティビティを示しています。

図 25 システムの使用可能メモリ



freemem という値は、空きページの総数を表します。

調整可能な 3 つのページングしきい値が setmemthresholds()ルーチンにより初期化されます。

表 19 setmemthresholds()ページングしきい値

ページングしきい値	意味
lotsfree	ページ内で指定される十分な量の空きメモリ。このしきい値を上回ると、ページング・デーモンによるページのスチール（流用）が開始される。
desfree	ページ内で指定される、望ましいメモリ量。このしきい値を下回ると、ページング・デーモンによるページのスチールが開始される。
minfree	ページ内で指定される最低限のメモリ量。空きメモリがこのしきい値を下回ると、sched()が、システムがメモリを求めており、プロセスが実行可能かどうかにかかわらずプロセス全体を非アクティブ化することを認識する。

gpgslim ページングしきい値

gpgslim ページングしきい値は、vhand によりページングが開始されるポイントです。gpgslim は、システムの要求に応じて動的な調整を行います。このしきい値には幅があり、その上限は lotsfree であり、下限は desfree です。lotsfree と desfree は両方とも、システム起動時にシステム・メモリのサイズにもとづいて計算されます。

システムのブート時に、gpgslim は lotsfree と desfree との差の 1/4 のポイントに設定されます（desfree + (lotsfree - desfree) /4）。システムの実行中に、この値は desfree と lotsfree の間で変動します。使用可能メモリの合計と I/O 用に割り当てられたページ数（すぐに解放される）が gpgslim を下回ると、このしきい値を上回る使用可能メモリを確保するために、ほとんど使用されていないページのエージングとスチールが vhand により開始されます。

システムは、gpgslim レベルの空きメモリを維持しようとします。十分なメモリが確保されている場合、より多くのページを解放する必要がないため、gpgslim は低下し始めます。メモリが不足すると（つまり、freemem が頻繁にゼロに達すると）、gpgslim が上昇して、早めにページングが行われ、freemem が頻繁にゼロに達しないようにします。

freemem が minfree まで低下すると、プロセス全体の非アクティブ化が開始されます。

メモリしきい値の調整方法

ページングしきい値は、次のように設定されます。

表 20 ページングしきい値

しきい値	基本値	初期 freemem < 2GB の場合の限度	初期 freemem の 2GB ごとの追加量
lotsfree	freemem の 1/16	32MB	32MB
desfree	freemem の 1/64	4MB	8MB
minfree	desfree の 1/4	1MB	4MB

ページングの起動方法

ルーチン schedpaging()が定期的に実行され、空きメモリと許容メモリの合計（freemem + parolemem）が lotsfree を下回ると vhand が起動されます。schedpaging()が実行される頻度は、vhandrunrate と呼ばれます。この調整パラメータは、デフォルトで 1 秒間に 8 回実行されるように設定されています。

vhand は、reserve_freemem()と allocate_page()によっても起動できます。

reserve_freemem()は、メモリ予約のために呼び出されるルーチンです。十分なメモリを予約できず、freemem + parolemem < gpgslim である場合に、このルーチンは vhand を起動します。

allocate_page()は、実際にメモリを割り当てるために呼び出されるルーチンです。待機できない（割り込みスタック上で実行されていることなどが理由で）コードによって呼び出された場合、このルーチンは要求されたメモリを見つけることができず、

vhand を起動します。また、要求されたメモリを見つけることができなかった場合、呼び出し側が待機できるかどうかにかかわらず unhashdaemon を起動します。

ページアウト・デーモン vhand

vhand の機能は、最近の参照頻度が最も低いページを解放して使用可能メモリを確保することです。また、カーネル・メモリ・アロケータ空きリストのガベージ・コレクションなど、メモリ可用性の維持に関する機能も果たします。

2 針クロック・アルゴリズム

vhand では、2 針クロック・アルゴリズムを利用して、解放するページが決められます。概念的には、このアルゴリズムには、すべてのメモリを通過する 2 つの針（「エージ針（age hand）」と「スチール針（steal hand）」）が存在します。エージ針は、各ページを「最近参照されていない」とマークします。スチール針は遅れて進み、参照されたビットをエージ針がクリアして以降、各ページがアクセスされたかどうか（「最近参照された」とマークされているかどうか）チェックします。アクセスされていないページをスチールする（ページ・アウトし、そのメモリを別のユーザーが使用できるようする）ことができます。

実際の実装では、vhand が、アクティブな pregion リストと呼ばれる pregion の二重リンク・リストに従ってメモリを通過します。vhand は、起動されるたびにすべての pregion を通過するわけではなく、通常は各 pregion 内の一部のページのみ調べます。ファイル・システムのバッファ・キャッシュに使用されるメモリは pregion に関連しないので、vhand のスキャン用のリストにメモリを入れるために、bufcache_preg と呼ばれる特別なダミーpregion が使用されます。

すべてのページを単にスキャンする（たとえば、pfdat を使用して）のではなく、pregion を使用することの利点は、カーネル・メモリとすでに空いているメモリが自動的にスキップされることです。

ただし、取得したすべてのメモリが単一のプロセスに属するという短所があります。つまり、スチール針がそのプロセスの pregion に到達すると、スチールされるすべてのページがその 1 つのプロセスから取得され、そのプロセスのワーキング・セットでページバックが頻繁に発生し、スラッシング状態になります（これが特に悪い状態になるのは、プロセスがたまたま対話的になってユーザー入力を待機する場合です。ユーザーは、プログラムがマウスの動きに応答する前に多数のページインを待つことを望みません）。このような理由から、各 pregion の一部のみが、通過ごとにエージングまたはスチールされ、vhand がアクティブな pregion を複数回通過する必要があるのです。

両針間の適切な距離を維持することが重要です。距離が近すぎると、実際には常に使用されているページがスチールされます。距離が遠すぎると、同じスチール率を維持するために両針がより速く動く必要があります。これは、vhand がより多くの CPU 時間を消費することを意味します。使用可能なページング・バンド幅、スチールする必要があるページ数、すでに解放のスケジュールが決まっているページ数、および vhand の実行頻度にもとづいて、カーネルが両針間の適切な距離を自動的に維持します。

表 21 vhand で使用される pregion 要素

要素	目的
p_agescan	最後のエージ針の位置
p_stealscan	最後のスチール針の位置

p_ageremain	エージングされる残りのページ
p_bestnice	基礎となる領域を共有するすべてのプロセスの最適な nice 値
p_forw、p_back	アクティブな pregion リスト内のリンク

最近参照されていないメモリ・ページを検索し、それらのページを二次記憶領域（スワップ・スペース）に移動するために、2 つの針が物理メモリのアクティブな pregion リンク・リストを巡回します。エージ針が通過した時点からスチール針が通過する時点までに参照されなかったページがメモリから追い出されます。両針は、メモリの必要量に応じて決まる可変速度で回転します。

vhand デーモンは、使用可能な空きメモリ量を判断してページングの開始タイミングを決めます。空きメモリが gpgslim しきい値を下回ると、ページングが行われます。vhand は、供給できるメモリ量を gpgslim まで戻すために十分なページを解放しようとします。このページ・デーモンは、ページをスチールする必要がないほど十分なメモリがある場合に起動された場合でも、ページのエージング（つまり、参照ビットのクリア）を続けます。ただし、このような状況で頻繁に起動されることはありません。

vhand に影響する要因

vhand は、さまざまなワークロード、一時的な状況、およびメモリ構成に反応します。pregion からエージングおよびスチールを行うとき、vhand は次の処理を行います。

- 各 pregion の一定の部分をエージングする。
- pregion のフィールド p_agescan を使用して、最後のエージ針の位置を追跡する。
- pregion のフィールド p_ageremain を使用して、エージングする残りのページを追跡する。
- pregion のフィールド p_stealscan を使用して、最後のスチール針の位置を追跡する。
- 有効なページをもたない vfd/dbd ペアをスワップに追い出す。

エージ針が pregion に達すると、次の region に移動する前にページの一定の部分（デフォルトでは、領域の総ページの 1/16）をエージングします。p_agescan タグにより、エージ針が、前回の通過中に停止した pregion 内の位置に移動できます。p_ageremain が、次の pregion に移動する前に 1/16 の部分を満たすためにエージングする必要があるページ数を示します。

スチール針は、pregion のフィールド p_stealscan を使用して pregion 内のスチール針自身の位置を決め、最後にエージングされて以降参照されていないページの取得を再開します。有効なページが残っていない場合、vhand がメモリを vfd/dbd から追い出します。

エージングとスチールの頻度は、次の要因によって決まります。

- vhand の実行頻度（デフォルトでは、1 秒間に 8 回）
- 使用可能なページング・バンド幅（一定期間内に完了するページ・アウトのグローバルな発生率と比較して決められる）
- システムの空きメモリがゼロになる回数
- ページングしきい値 gpgslim のレベル
- すでに解放のスケジュールが決まっているページ数

特性が優れているスレッドに対しては、vhand に偏りがあります。スレッドが優れているほど、vhand がページをスチールする頻度が高くなります。pregion フィールドの p_bestnice は、pregion を共有するすべてのスレッドの最適な（数値的に最小の）nice 値を反映します。

vhand の起動時に起こること

vhand 変数の詳細は、図 22 を参照してください。

- vhand が SCRITICAL フラグを使用して、システムに欠かせないメモリ・プールへのアクセス権を取得（vhand の SCRITICAL フラグは、最初にプロセス実行が開始される際に設定される）。
- vhand が、エージングとスチールを行うページのページ・カウントを設定。
- vhand が、memzeroperiod に基づいて gpgslim の値を更新。
- vhand が、pageoutcnt を使用して pageoutrate を更新。
- vhand が、エージ針とスチール針の間の望ましい周回差を示す targetlaps を更新。CPU 実行時間が targetcpu の値を下回ると、vhand が targetlaps の値を上げる（最大 15 まで）。CPU 実行時間が targetcpu の値を上回ると、targetlaps が下がる。
- vhand が、1 秒間にエージングするページ数を示す agerate を更新。
- vhandinfoticks がゼロでない場合、診断情報がコンソールに出力される。

注: 次の表の変数を調整することはできません。

表 22 vhand に影響する変数

変数	目的
memzeroperiod	freemem がゼロに到達するまでに許容される最小期間（デフォルト値は 3 秒）。これにより、vhand()の実行中に gpgslim を調整する頻度が決まる。 ● memzeroperiod 以内に freemem が 2 回ゼロに到達しないと、gpgslim が増分される。 ● memzeroperiod 以内に freemem が 2 回ゼロに到達すると、gpgslim が減分される。
pageoutrate	これまでに完了したページアウト数に基づいて計算される現在のページ・アウト発生率。
pageoutcnt	完了したページアウトの最近のカウント。
targetlaps	handlaps に対するスチール針とエージ針の間の理想的な間隔。実行時に調整される。正常な動作中は、ページで使用されているクリア済み参照ビットをリセットするための最大限の時間をプロセスに提供するために、両針をできるだけ離す必要がある。targetlaps はカーネル内で静的変数として定義されるため、シンボル・テーブルには現れない。
targetcpu	vhand がページングに費やす CPU の最大使用率（デフォルトは 10%）。
handlaps	エージ針とスチール針の間の実際の周回差。

agerate	エージングのためにエージ針が 1 秒間に巡回するページ数。システム負荷に応じて頻繁に調整される。agerate はカーネル内で静的変数として定義されるため、シンボル・テーブルには現れない。
stealrate	スチール針が 1 秒間に巡回するページ数。システム負荷に応じて頻繁に調整される。stealrate はカーネル内で静的変数として定義されるため、シンボル・テーブルには現れない。

ページのスチールとエージングを行う vhand

vhand は、その基準を確立してから、pregion のリンク・リストの走査に進みます。時計の針にたとえれば、vhand はその針を動かす準備ができています。

- vhand が、スチールできるページとその数を判断します。
  - スチール針が bufcache_preg を指すと、vhand が stealbuffers()ルーチンでバッファ・キャッシュからバッファをスチールします。グローバル・パラメータ dbc_steal_factor の値で、pregion ページよりもバッファ・キャッシュ・ページの方をどの程度積極的にスチールするかが決められます。dbc_steal_factor の値が 16 の場合、バッファ・キャッシュ・ページは pregion ページと同様に扱われます。デフォルトの値は 48 で、これはバッファ・キャッシュ・ページが pregion ページに比べて 3 倍積極的にスチールされることを意味しています。
  - region に有効なページがない（つまり r_nvalid == 0）pregion をスチール針が指し、region を使用するプロセスがメモリにロードされていない（つまり r_incore == 0）場合、vhand がその B-tree をスワップ・デバイスから追い出します。
  - 上記以外の場合、vhand は p_stealhand と（p_agescan - p_count/16 * handlaps）の間の（割り当てられたスチールまで）未参照ページすべてをスチールします。
  - vhand は、影響を受ける pregion の最後にスチールされたページに続くページ番号に p_stealscan を更新します。
  - 許容されている数のページを vhand がスチールしなかった場合、次の pregion に移動し、システムの要求が満たされるまで上記のプロセスを繰り返します。

- 次に、vhand はエージ針を動かして、選択された番号のページから参照ビットをクリアします。
  - エージ針が bufcache_preg を指すと、vhand は agebuffers()ルーチンでバッファ・キャッシュ内の 16 番目のページをエージングします。
  - vhand は、その領域を使用するすべての pregion の最高の nice 値（数値的に最小の値）を判断します。領域内の各ページについて nice 値が乱数を下回った場合、vhand はそのページのエージングを行いません（高優先順位のプロセス（数値的に小さい nice 値）に属するページほど、エージングの確率が低くなります）。
  - 上記以外の場合、vhand は pde_ref ビットをクリアし TLB を消去することで、p_agehand と（p_agehand + p_ageremain）の間のすべてのページのエージングを行います。
  - 最後に、vhand は、影響を受ける pregion 内でスキャンされた（エージングされた可能性がある）最後のページに続くページ番号に p_agescan を更新します。

注: 次の表の変数を調整することはできません。

sched()ルーチン

sched()ルーチン（通称「スワッパー（swapper）」）は、空きメモリが minfree を下回り、システムがスラッシングしているようにみえる場合にプロセスの非アクティブ化と再アクティブ化を処理します。

注: 非アクティブ化は、スレッドごとに行われます。sched()は、プロセス・レベルに対する非アクティブ化を選択してから各スレッドを非アクティブ化します。

システムが次のような状態であると sched()が判断すると、非アクティブ化が行われます。

- システムのメモリが不足している。つまり、freemem が非アクティブ化しきい値 minfree を下回り、複数のプロセスが実行されている場合。
- スラッシングの兆候がある。つまり、システムのページング発生率が高く、CPU 使用率が低い場合。

システムのメモリ不足またはスラッシングが解消されると、再アクティブ化が行われます。

非アクティブ化または再アクティブ化の対象

非アクティブ化と再アクティブ化の実行は、次の条件によって判断されます。

- プロセスの優先順位。プロセスの優先順位が低いほど（nice 値が高いほど）、非アクティブ化の確率が高くなります。これに対して、優先順位が高いほど、再アクティブ化の確率が高くなります。リアルタイム・プロセスを非アクティブ化することはできません。
- プロセスの状態。長い間休眠状態にあったりメモリ内に存在したプロセスは、非アクティブ化される確率が高くなります。しばらくの間非アクティブ化され、現在実行する準備ができているプロセスは、再アクティブ化される確率が高くなります。
- プロセスの種類。バッチ・プロセス（連続して動作するプロセス）またはシリアライゼーション（直列化）用にマークされているプロセスは、対話型プロセス（一時的に動作するプロセス）よりも非アクティブ化される確率が高くなります。対話型プロセスは、バッチ・プロセスまたはシリアライズされたプロセスよりも再アクティブ化される確率が高くなります。
- 現在の状態の継続時間。

sched()によりプロセスが非アクティブ化されプロセスの実行が防止されるため、新しいページにアクセスする頻度が低くなります。使用可能メモリが minfree を上回ったこととシステムがスラッシングしていないことを検知すると、sched()は非アクティブ化されていたプロセスを再アクティブ化し、引き続きメモリ可用性を監視します。

システムのスラッシングまたはメモリ不足の兆候がある場合、sched()ルーチンがアクティブなプロセス・リストをウォークし、プロセスの種類、状態、メモリ内に存在した時間、および休眠状態にあった時間にもとづいて各プロセスの非アクティブ化優先順位を計算します（バッチ・プロセスと、serialize コマンドによってシリアライゼーションとマークされたプロセスは、対話型プロセスよりも非アクティブ化される確率が高くなります）。次に、非アクティブ化される最善の候補がマークされます。

システムがスラッシングしていない場合またはメモリが不足していない場合、sched ルーチンがアクティブなプロセス・リストをウォークし、プロセスの非アクティブ化期間、サイズ、状態、および種類にもとづいて、非アクティブ化されている各プロセスの再アクティブ化優先順位を計算します。バッチ・プロセスと、serialize コマンドによってマークされているプロセスは、対話型プロセスよりも再アクティブ化される確率が低くなります。最適であると判断されたプロセスが再アクティブ化されます。

プロセス非アクティブ化時の処理

再アクティブ化するプロセスが選ばれると、sched()は次の処理を行います。

- proc struct の SDEACT フラグと thread struct の TSDEACT フラグを設定する。
- 実行キューからプロセスのスレッドを削除する。
- スレッドの uareas をアクティブな pregion リストに追加して、vhand がスレッドをページアウトできるようにする。
- 非アクティブ化対象のプロセスに関連する pregion をすべてスチール針の前に移動して、vhand がすぐにプロセスをスチールできるようにする。
- vhand が、1/16 ではなく pregion 全体からページのスキャンとスチールができるようにする。

最後に vhand が、非アクティブ化されたプロセスのページを二次記憶領域に追い出します。

プロセス再アクティブ化時の処理

プロセスの再アクティブ化が可能になるほど十分なメモリをシステムが解放し、ページング発生率が十分低下するまで、プロセスは非アクティブ化されたままになります。再アクティブ化優先順位が最も高いプロセスが再アクティブ化されます。

再アクティブ化するプロセスが選ばれると、sched()は次の処理を行います。

- アクティブな pregion からプロセスのスレッドの uareas を削除する。
- 非アクティブ化フラグをすべてクリアする。
- uareas にフォールトインする。
- プロセスのスレッドを実行キューに追加する。

自己非アクティブ化

以前の HP-UX の実装では、プロセスがロックを保持したり I/O を実行している場合、またはシグナル受信可能な優先順位にない場合、プロセスをスワップアウトすることはできませんでした。優先順位により非アクティブ化の確率が最も高い場合でも、vhand がそのプロセスを無視していました。

現在では、最適なプロセスをすぐに非アクティブ化できない場合、そのプロセスは自己非アクティブ化とマークされます。つまり、sched()がそのプロセスの proc struct に SDEACTSELF を設定し、そのプロセスの各 thread structs に TSDEACTSELF を設定します。次回、スレッドの 1 つがページをフォールトインしなければならない際に、そのスレッドがプロセスを非アクティブ化します。

スラッシング

スラッシングは、ページング発生率が高く CPU 使用率が低い状態であると定義されます。スラッシングが発生するのは、複数のプロセスが実行されている場合、複数のプロセスが I/O の完了を待機している場合、またはアクティブなプロセスがシリアライゼーションとマークされている場合です。

多量のメモリが要求されているシステム（たとえば、大きなプロセスが多数実行されているシステム）では、ページング・デーモンが非アクティブ化／再アクティブ化およびページのスワップイン／スワップアウトでビジー状態になるため、システムでページングに長時間が費やされ、プロセスの実行に十分な時間が確保されません。

このような事態に陥った場合、何も実行されていないようにみえる程度までシステム・パフォーマンスが急速に低下することがあります。この場合には、生産的な動作よりも多くのオーバーヘッドを実行しているため、システムがスラッシングしているといわれます。

ワーキング・セットが物理メモリより大きい場合、スラッシングが発生します。この問題を解決する方法は次のとおりです。

- 非アクティブ化によって、実行されているプロセスのワーキング・セットを低減する。
- 物理メモリのサイズを増やす。

物理メモリを圧迫する大きなプロセスを追い出すことができず、スラッシングが継続する場合、同時に使用されるページが少なくなるようにアプリケーションを書き直す必要があります（たとえば、アクセスに応じてデータ構造をグループ化するなど）。

シリアライゼーション（直列化）

serialize コマンドによりマークされたプロセスはすべて、シリアル（直列）に実行されます。この機能により、CPU をめぐって競合する大きなプロセスのグループが原因のボトルネック（プロセス処理能力の低下により認識できる）を解消できます。大きなプロセスを一度に 1 つずつ実行すると、システムが CPU とシステム・メモリをより効率的に使用できます。なぜなら、各プロセスのワーキング・セット内で絶えずフォールトが発生することなく、別のプロセスの実行開始時にページをスチールするのみだからです。

システムに十分なメモリがある限り、serialize によりマークされたプロセスの動作はシステム内の別のプロセスと同じです。しかし、メモリが逼迫すると、serialize によりマークされたプロセスは優先順位に応じて一度に 1 つずつ実行されます。各プロセスが一定の間隔で実行された後、シリアライズされた別のプロセスが実行されます。ユーザーはこのシリアライズされたプロセスの実行順序を強制することはできません。

serialize にコマンドを指定して実行することも、PID 値を指定して実行することもできます。また serialize には、指定された PID を通常の時分割スケジューリング・アルゴリズムに返す時分割オプションも備わっています。

シリアライゼーションが不十分でスラッシングを解消できない場合、システムのメイン・メモリを増設する必要があります。

ページャを使用した非アクティブ化

vhand()は I/O 使用状況と CPU 使用率に応じて調整されるため、スワップされるプロセスをページャでフォールトアウトできます。スワッパーは、スワップされるプロセスを非アクティブ化とマークし、そのスレッドを実行キューから除去します。このプロセスは実行できないため、そのページがエージングされると、二度と参照できません。スチール針が 1 周すると、region 内のすべてのページがスチールされます。

メモリが不足気味になると、sched()がルーチン choose_deactivate()を使用して、スワップするプロセスを選択します。このルーチンは、対話型プロセスよりも非対話型プロセスを、実行中のプロセスよりも休眠状態にあるプロセスを、新しいプロセスよりも長く実行されているプロセスをそれぞれ優先的に選択します。

非アクティブ化するプロセスが選択されると、次の処理が行われます。

- プロセスの SDEACT フラグとそのスレッドの TSDEACT フラグが設定される。
- プロセスのスレッドが実行キューから削除される。プロセスが I/O を待機している場合、プロセスの SDEACTSELF フラグとそのスレッドの TSDEACTSELF フラグが設定される。I/O が完了すると、ページング・ルーチンでプロセスが非アクティブ化される。

- プロセス非アクティブ化の時間を記録するために、proc 構造内のプロセスの p_deactime と kthread 構造内のスレッドの kt_deactime が現在の時間に設定される。
- スチール針に対する準備として、プロセスの pregion がアクティブな pregion チェーンに配置される。
- ページ・アウトするために、uarea pregion がアクティブな pregion のリストに追加される。
- グローバル・カウンタ deactive_cnt が増分される。

プロセスのページすべてに対するエージングとスチールが実行されるだけ十分な期間にわたって休止していたプロセスは、事実上すでにスワップアウトされています。グローバルな deactprocs が休止プロセスのリストの先頭を指し示し、そのチェーンが pregion の要素 p_nextdeact に伝わります。

メモリの不足気味の状態が軽減されると、非アクティブ化されたプロセスが再アクティブ化されます。choose_reactivate()ルーチンは、非対話型プロセスよりも対話型プロセスを、休眠状態にあるプロセスよりも実行可能なプロセスを、最近非アクティブ化されたプロセスよりも長期間非アクティブ化されているプロセスをそれぞれ優先的に選択します。

メモリ・リソース・グループ（MRG）

これまで、あたかもすべてのプロセスが同じメモリ・プールを共有するかのように、ページングと非アクティブ化について説明してきました。これは通常の状態であり、以前のバージョンの HP-UX では唯一のオプションでした。

しかし、現在の HP-UX には、メモリ・リソース・グループ（MRG）を使用してプロセス・グループを固有のメモリ・プールに割り当てるオプションがあります。これらのプロセスには実際に、プロセス固有の physmem_pages、freemem、minfree、desfree、lotsfree、gpgslim などが提供されます。

これにより、プロセスのグループが独立してページングできるようになり、プロセス間の干渉が大幅に低減されます。これは、元々個々のサーバ用に記述された複数のアプリケーションが 1 台の大きなサーバ上で実行されるようなサーバ統合で有用です。

メモリ・リソース・グループにより、vhand と sched は、あたかも各 MRG が固有のページャとスワッパーを備えて完全に分離しているかのように動作します（実際の実装はこれより少し複雑です。MRG 間を移動するプロセスとメモリ、ある MRG が別の MRG からメモリを借りることができる機能、任意の単一プロセス（または MRG）に割り当てることができないメモリの使用、およびグローバル・メモリの可用性と個々の MRG メモリの可用性を維持する必要性などを考慮する必要があります）。上記のグローバル変数はまだ存在し、システム全体の状態の要約として機能します。

スワップ・スペース管理

スワップ・スペースとは高速記憶装置（ほとんどの場合ディスク・ドライブ）上の領域であり、仮想メモリ・システムでプロセスの非アクティブ化とページングに使用するために予約されます。システム上に少なくとも 1 つのスワップ・デバイス（一次スワップ）がなければなりません。

システムの起動時に、各スワップ・デバイスの場所（ディスク・ブロック番号）とサイズが 512KB ブロックで表示されます。カーネルを再構成しなくても、システムの実行中に必要に応じて（つまり動的に）スワップを追加できます。

スワッパーはプロセス作成時にスワップ・スペースを予約しますが、ページがディスクに移る必要があるまでディスクからスワップ・スペースを割り当てません。プロセス作成時にスワップが予約されるため、スワッパーがスワップ・スペースを使い果たすことを防止できます。

HP-UX では、物理スワップと擬似スワップの両方を使用して効率的なプログラム実行を実現しています。

擬似スワップ・スペース

スワップ・スペースに使用されるシステム・メモリを擬似スワップ・スペースと呼びます。このスペースにより、物理スワップを割り当てなくてもメモリ内でプロセスを実行できます。擬似スワップを、オペレーティング・システム・パラメータ swapmem_on で制御します。このパラメータのデフォルト値は 1 で、擬似スワップが使用可能なことを示します。

通常は、ページ・アウトしなければならない場合に備えて、システムによるプロセスの実行時にスワップ・スペースがプロセス全体用に予約されます。このモデルに従えば、1GB のプロセスを実行するために、構成済みの 1GB のスワップ・スペースがシステムに存在しなければなりません。これによりシステムがスワップ・スペースを使い果たすことを防止できますが、最低限のスワッピングが生じた場合やスワッピングが生じなかった場合、スワップ用に予約されたディスク・スペースが十分に活用されません。

このようなリソースの浪費を回避するために、HP-UX は擬似スワップとしてシステム・メモリ容量の最大 7/8 までアクセスするように構成されています。これは、システム・メモリが 2 つの機能（プロセス実行スペースとスワップ・スペース）を果たすことを意味しています。擬似スワップ・スペースを利用すれば、2GB のスワップを備える 2GB のメモリ・システムで、最大 3.75GB のプロセスを実行できます。以前と同様に、プロセスがこの拡張しきい値を超えて成長したり、この値を超えるプロセスが作成されると、そのプロセスは失敗します。

スワップに擬似スワップを使用するとページがロックされるため、擬似スワップの量が増えるとロック可能メモリの量が減ります。

アプリケーション全体がメモリに常駐している場合に最良のパフォーマンスが発揮される生産ラインのシステム（コントローラなど）では、擬似スワップ・スペースを使用すればパフォーマンスが向上します。そのために、アプリケーションをメモリ内にロックするか、作成されるプロセスの総数がシステム・メモリの 7/8 を超えないようにします。

作成されるプロセスの数が許容数に近づくと、スラッシングが生じ、システム応答速度が低下することがあります。必要な場合は、/usr/conf/master.d/core-hpux の調整パラメータ swapmem_on をゼロに設定すれば擬似スワップ・スペースを使用不可にできます。

擬似スワップを割り当てられた領域の二重リンク・リストの先頭に、pswaplist というリスト（最後がヌルになっている）があります。

物理スワップ・スペース

物理スワップ・スペースには、デバイス・スワップ・スペースとファイル・システム・スワップ・スペースの 2 種類があります。

デバイス・スワップ・スペース

デバイス・スワップ・スペースは固有の予約済み領域（ディスク全体または LVM ディスクの論理ボリューム）に存在し、一般にファイル・システム・スワップ・スペースよりも高速です。

ファイル・システム・スワップ・スペース

ファイル・システム・スワップ・スペースはマウントされたファイル・システム上に配置され、サイズはシステムのスワッピングのアクティビティによって変わります。ただし、未使用のファイル・システム・ブロックが必ずしも連続していないことがあるため余分な読み取り／書き込み要求が生じることと、コードの追加階層の余分なオーバーヘッドが原因で、スループットはデバイス・スワップよりも遅くなります。

システム・パフォーマンスを最適化するために、ファイル・システム・スワップ・スペースの割り当てと割り当て解除は swchunk サイズのチャンクで行われます。swchunk は構成可能なオペレーティング・システム・パラメータであり、デフォルト値は 2048KB（2MB）です。ファイル・システム・スペースのチャンクがこのページ方式で使用されなくなると、ファイル・システムで使用するために解放されます（ただし、swapon コマンドで事前に割り当てられていない場合）。

ファイル・システム・スワップ・スペースへのスワッピングの場合、スワップ・スペースの各チャンクはファイル・システム・スワップ・ディレクトリ内のファイルであり、その名前はシステム名と swaptab 索引から構成されます（たとえば、becky という名前のシステム上の swaptab[6]は becky.6 となります）。

スワップ・スペースのパラメータ

表 23 構成可能なスワップ・スペース・パラメータ

パラメータ	目的
swchunk	DEV_BSIZE ブロックの数（スワップ・スペース単位）。すべてのシステムでデフォルト値は 2MB。
maxswapchunks	システム上で許容されるスワップ・チャンクの最大数。
swapmem_on	擬似スワップを使用した場合に、物理スワップ・スペースよりも多くのプロセスを作成できるようにするパラメータ。

スワップ・スペースのグローバル変数

スワップ・スペースに関連するカーネル・グローバル変数が多数あります。スワップ・スペース予約にとって最も重要な変数は、swapspc_cnt、swapspc_max、swapmem_cnt、swapmem_max、sys_mem です。

表 24 スワップ・スペースの特性（グローバル変数）

変数	意味
bswlist	空きスワップ・ヘッダ・リストの先頭。

swdevt[]	デバイス・スワップ・テーブル。
fswdevt[]	ファイル・システム・スワップ・テーブル。
swaptab[]	スワップ・チャンクのテーブル。
swapphys_cnt	ディスク上の使用可能な物理スワップ・スペースのページ数。これは、予約されているかどうかにかかわらず、未割り当てページの数になる。下記の swapspc_cnt は、予約されていないページのみの数。
swapphys_buf	使用可能な物理スワップ・スペースのページ数。swapphys_cnt がこの数を下回ると、ページのメモリ上のコピーがディスク上のコピーよりも新しい場合、vhand のエージ針によりスワップ・スペースが解放される（これは、ページをページ・アウトしなければならない場合に、スワップ・スペースを再割り当てする必要があることを意味する）。
swapspc_cnt	使用可能なすべてのデバイスとシステム上で現在使用できるスワップの合計（ページ単位）。スワップが予約または解除されるたびに更新され、デバイスまたはファイル・システムがスワップピングに対して使用可能になるになるたびに更新される。
swapspc_max	システム上の現在使用可能なデバイス・スワップとファイル・システム・スワップの合計（ページ単位）。デバイスとファイル・システムがスワッピングに対して使用可能になるたびに更新される。
swapmem_cnt	現在使用可能な擬似スワップのページ総数。初期値は swapmem_max。
swapmem_max	使用可能な擬似スワップの最大ページ数。初期値は使用可能なシステム・メモリの 7/8。
pswaplist	擬似スワップを使用する領域のリンク・リスト。
maxdev_pri	使用可能なスワップ・デバイスの最高優先順位。
maxfs_pri	使用可能なスワップ・ファイル・システムの最高優先順位。
phys_mem_pages	システム上の物理メモリのページ数。
sys_mem	擬似スワップとして使用できないメモリのページ数。通常の初期値は、使用可能なシステム・メモリ + 25 ページ + sysmem_max ページ。
sysmem_max	使用可能なデバイス・スワップがあるシステムの初期化時に sys_mem（擬似スワップに使用できないページ数）に追加される（ただし、swapmem_max > 0 になる場合のみ）。

maxmem	動的バッファ・キャッシュ用に phys_mem_pages の初期 dbc_min_pct が割り当てられた後、freemem の初期値に設定される。maxmem - swapmem_max は、擬似スワップからスチールされたページをカーネルが返す場合に sys_mem の上限として使用される。
freemem	残りの未予約の空きメモリ・ブロック全体のページ数。
freemem_cnt	メモリを待機するために global_freemem 上で休眠状態にあるスレッドの数（ここでカウントされない別のメモリ待機方法もある）。

スワップ・スペース値

システム・スワップ・スペース値の計算方法は次のとおりです。

- システム上で使用可能な総スワップ
  - デバイス・スワップとファイル・システム・スワップの場合：swapspc_max
  - 擬似スワップの場合：swapspc_max + swapmem_max
- 割り当てられるスワップ
  - デバイス・スワップとファイル・システム・スワップの場合：swapspc_max - [sum（swdevt[n].sw_nfpgs） + sum（fswdevt[n].fsw_nfpgs）]
  - 擬似スワップの場合：swapspc_max - [sum（swdevt[n].sw_nfpgs） + sum（fswdevt[n].fsw_nfpgs）] + （swapmem_max - swapmem_cnt)

HP-UX では、プロセスがスワップ・スペース不足で抹消される原因は、（sbrk()を使用した）データ領域の成長またはスタックの成長のみです。プログラムのテキストでは、スワップは使用されません。

物理スワップ・スペースの予約

スワップの予約は、数値によって行われます。システムの物理スワップ・スペースのページ数には制限があります。HP-UX では、適切なカウンタを減分することでプロセス用のスペースが予約されます。

UNIX システムや UNIX に類似したシステムのほとんどでは、必要に応じてスワップが割り当てられます。しかし、スワップ・スペースが使い果たされているにもかかわらずプロセスのページをスワップ・デバイスに書き込む必要がある場合、プロセスを抹消する以外に方法はありません。HP-UX では、この問題を軽減するために、プロセスが fork または exec される時点でスワップが予約されます。新しいプロセスが fork または exec される際、予約済みの使用可能スワップ・スペースがプロセス全体を処理するには不十分な場合、プロセスが実行されないことがあります。

システム起動時に、swapspc_cnt と swapmem_cnt は使用可能なスワップ・スペースと擬似スワップの合計に初期化されます。

デバイス・スワップまたはファイル・システム・スワップを追加するために swapon()が呼び出されると、新たに使用可能になったスワップ量がページ単位に変換され、2 つのグローバル・スワップ予約カウンタ swapspc_max（使用可能になったスワップ総量）と swapspc_cnt（使用可能なスワップ・スペース）に追加されます。

プロセス用にスワップ・スペースが予約されるたびに（つまり、プロセスの作成または成長時に）、必要なページ数だけ swapspc_cnt が減分されます。カーネルは、必要になるまでディスク・ブロックを実際には割り当てません。

スワップ・スペースが使い果たされている場合（swapspc_cnt == 0）、スワップ予約が要求されると、ファイル・システム・スワップ・スペースの追加チャンクが割り当てられます。これが成功した場合、swapspc_max と swapspc_cnt の両方が更新され、現在と以降の要求を満たすことができます。ファイル・システム・チャンクを割り当てることができなかった場合、擬似スワップが使用可能でなければ要求は失敗します。

（プロセスの終了または縮小が原因で）スワップ・スペースが不要になると、解放されたページ数だけ swapspc_cnt が増分されます。swapspc_cnt は決して swapspc_max を上回らず、常にゼロ以上です。ファイル・システム・スワップのチャンクが不要になると、チャンクがファイル・システムに対して再び解除され、swapspc_max と swapspc_cnt が更新されます。

デバイス・スワップ・スペースまたはファイル・システム・スワップ・スペースが使用できない場合、擬似スワップが最後の手段として使用されます。 swapmem_cnt が減分され、ページがメモリにロックされます。擬似スワップは解放または割り当てされますが、決して予約されません。

スワップ予約スピンロック

rswap_lock スピンロックは、スワップ予約構造 swapspc_cnt、swapspc_max、swapmem_cnt、swapmem_max、sys_mem、および pswaplist を保護します。

擬似スワップ・スペースの予約

調整パラメータ swapmem_on が 1 に設定されている場合、使用可能なシステム・メモリの約 7/8 を擬似スワップ・スペースとして使用できます。擬似スワップは、グローバル擬似スワップ予約カウンタ swapmem_max（使用可能な擬似スワップ）と swapmem_cnt（現在使用可能な擬似スワップ）で追跡されます。物理スワップ・スペースが使い果たされ、追加のファイル・システム・スワップを取得できない場合、swapmem_cnt が減分されてプロセス用の擬似スワップ・スペースが予約されます。

たとえば 256MB システムでは、swapmem_max と swapmem_cnt で約 224MB の擬似スワップ・スペースが追跡され、残りのスペースは、システム使用のためのみに予約されたページ数を示すグローバル sys_mem で追跡されます。

プロセスは、領域ごとのカウンタ r_swapmem を増分することで、プロセスに割り当てられた擬似スワップ・ページの数を追跡します。擬似スワップを使用するすべての領域が、擬似スワップ・リスト pswaplist 上でリンクされます。デバイス・スワップと擬似スワップの両方が使い果たされると（swapspc_cnt==0 かつ swapmem_cnt==0 になると）、プロセスの作成または成長時の試行が失敗します。

割り当てられた擬似スワップ・スペースがプロセスで必要なくなると、解除されたスペース量だけ swapmem_cnt が増分され、r_swapmem が更新されます。

擬似スワップ以外の目的で使用できるメモリが擬似スワップで消費されるため（以降の項を参照）、メモリは節約して使用されます。物理メモリ・スワップ・スペースが最近解放されたかどうか、オペレーティング・システムで定期的にチェックされます。最近解放された場合、システムは擬似スワップ領域の二重リンク・リストをウォークして、擬似スワップの使用から、使用可能な物理スワップのみの使用にプロセスを移行しようとします。swapspc_cnt がゼロに低下するか、別の領域で擬似スワップが利用さ

れるまで、リスト上の各領域について swapspc_cnt が r_swapmem 値だけ減分されます。次に swapmem_cnt が、この移行に成功した擬似スワップ量だけ増分されます。

擬似スワップとカーネル・メモリ

擬似スワップは、システム・メモリの使用をめぐってカーネルと競合します。最初に、使用可能メモリの 1/8（sys_mem ページ数）が擬似スワップに使用できないように設定されますが、これで、カーネル動的メモリとバッファ・キャッシュ・スペースの両方の処理に十分なメモリが確保されるとはとてもいえません。カーネルは、これらの目的のために擬似スワップからメモリをスチールし、ページのスチール時に swapmem_cnt を減分します。swapmem_cnt がゼロに達すると、sys_mem がゼロに達するまで sys_mem からのページの取得が開始されます。

スチールされた擬似スワップが返されると、解除される量が最初に sys_mem に追加されます。sys_mem が最大値（maxmem - swapmem_max）に達すると、返された追加ページだけ swapmem_cnt が増えます。

擬似スワップとロック可能メモリ

擬似スワップはシステム・メモリの使用量と関係があるため、スワップ予約方式にロック可能メモリのポリシーが反映されます。

プロセスがプロセス自体をメモリ内にロックする際、追加メモリが割り当てられるとは限りませんが、ロックされたページを汎用目的で使用できなくなります。そのため、これらのページを考慮して swapmem_cnt が減分されます。プロセスがメモリ内に plock でロックされる場合、swapmem_cnt もプロセス全体のサイズだけ減分されます。

スワップ・スペースの優先順位

スワップに使用可能なすべてのスワップ・デバイスとファイル・システムに 0〜10 の優先順位が付けられます。この優先順位は、デバイスまたはファイル・システムからスワップ・スペースが使用される順序を示します。システム管理者は、swapon(1M)コマンドのパラメータを使用してスワップ・スペースの優先順位を指定できます。

スワッピングは、優先順位が同じデバイスとファイルの両方に対して交互に行われますが、デバイスの方が CPU 時間をより効率的に利用できるため、オペレーティング・システムによりファイル・システムの前にデバイスがスワップされます。

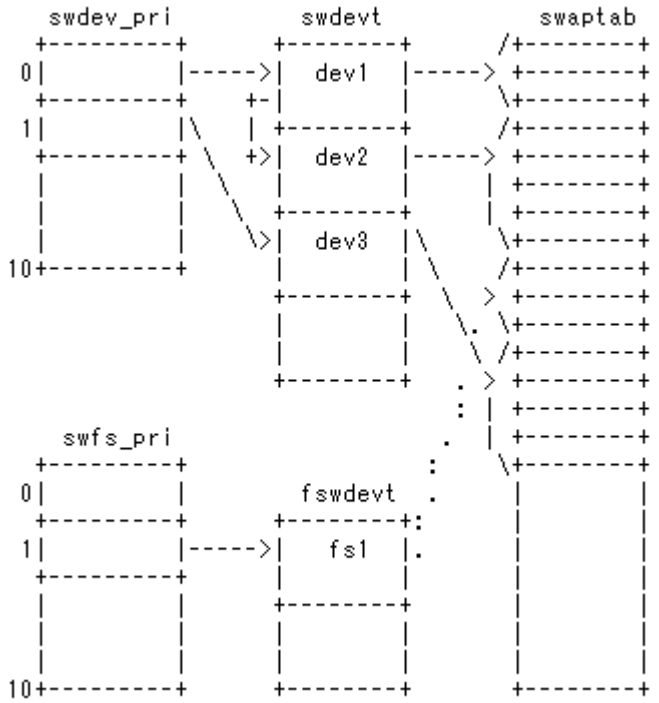
あるデバイスが他のデバイスよりも大幅に遅い場合を除いて、ほとんどのスワップ・デバイスに同じ優先順位を割り当てることをお勧めします。同じ優先順位を割り当てるとディスク・ヘッドの動きが制限されるため、ページング・パフォーマンスが向上します。

3 つのスワップ・スペース割り当て規則

- 優先順位の値が最低のスワップ・デバイスまたはファイル・システムから開始されます。番号が低いほど、優先順位が高くなります。つまり、優先順位 1 のシステムの前に優先順位 0 のシステムからスペースが取得されます。
- 同じ優先順位のデバイスが複数ある場合、スワップ・スペースはデバイスからラウンドロビン方式で割り当てられます。このため、複数のデバイス間でスワップ要求をインターリーブするために、デバイスに同じ優先順位を割り当てる必要があります。同様に、複数のファイル・システムの優先順位が同じ場合も、ファイル・システム間でスワップ要求がインターリーブされます。次の図では、優先順位 0 の 2 つのスワップ・デバイス間で最初にスワップ要求がインターリーブされます。

- デバイスとファイル・システムのスワップ優先順位が同じ場合、ファイル・スワップ・スペースの前にデバイスからすべてのスワップ・スペースが割り当てられます。このため、優先順位 1 のファイル・システムからスワップが割り当てられる前に、優先順位 1 のデバイスが割り当てられます。

図 26 スワップ場所の選択



スワップ・スペース構造

HP-UX では、次のデータ構造を使用してスワップ・スペースが割り当てられます。

- 同じ優先順位のスワップ・デバイスのリンクに使用されるデバイス・スワップ優先順位配列（swdev_pri[]）。swdev_pri[n]のエントリは、優先順位 n のスワップ・デバイス・リストの先頭になる。swdev_pri[]構造の最初のフィールドは、このリストの先頭になる。swdevt[]構造の sw_next フィールドにより、各デバイスが適切な優先順位リストにリンクされる。
- ファイル・システム・スワップ優先順位配列（swfs_pri[]）。この配列は swdev_pri[]と同じ役割を果たすが、ファイル・システム・スワップ優先順位用。
- 基本スワップ・デバイス情報の設定に使用されるデバイス・スワップ・テーブル（struct swdevt）。
- 補足ファイル・システム・スワップ用のファイル・システム・スワップ・テーブル（struct fswdevt）。
- 使用可能チャンクのスワップ・テーブル（struct swaptab）。スワップ・スペースの使用可能ページを追跡。
- スワップ・ページのマッピング（struct swapmap）。この構造のエントリと swaptab が組み合わされてスワップ・ディスク・ブロック記述子になる。

次の表は、struct swdevt の要素の詳細です。

表 25 デバイス・スワップ・テーブル swdevt[]（struct swdevt）

要素	意味
sw_dev	メジャー番号（上位 8 ビット）とマイナー番号（下位 24 ビット）で定義される実際のスワップ・デバイス。
sw_flags	複数のフラグ。SW_ENABLE フラグは、このデバイス上でスワップが使用可能になっていることを示す。
sw_start	ディスク上のスワップ領域へのオフセット（KB 単位）。
sw_nblksavail	スワップ領域のサイズ（KB 単位）。
sw_nblksenabled	スワップに使用可能なブロック数。swchunk の倍数でなければならない（デフォルト値は 2MB）。
sw_nfpgs	デバイス上の空きスワップ・ページ数。ページが使用または解放されるたびに更新される。
sw_priority	スワップ・デバイスの優先順位（0～10）。
sw_head, sw_tail	このスワップ・デバイスに関連付けられる最初と最後の swaptab[]エントリの索引。
sw_next	この優先順位の次のデバイス・スワップ・エントリ（swdevt）へのポインタ。特定の優先順位のデバイスすべてをラウンドロビン方式で使用するために、swdev_pri 内のポインタの更新に使用される循環リストとして実装される。

次の表は、struct fswdevt の要素の詳細です。

表 26 ファイル・システム・スワップ・テーブル fswdevt[]（struct fswdevt）

要素	意味
fsw_next	この優先順位の次のファイル・システム・スワップへのポインタ。循環リストとして実装される。
fsw_flags	複数のフラグ。FSW_ENABLE フラグは、このファイル・システム上でスワップが使用可能になっていることを示す。
fsw_nfpgs	このファイル・システム・スワップの空きスワップ・ページ数。ページが使用または解放されるたびに更新される。
fsw_allocated	このファイル・システム上でスワップ用に割り当てられた swchunks の数。
fsw_min	ファイル・システム・スワップが使用可能になる際に事前に割り当てられる最小 swchunks。

fsw_limit	ファイル・システム上で許容される最大 swchunks。ゼロに設定されている場合、無制限。
fsw_reserve	このファイル・SYSTEM 上でスワップ以外の用途で予約される最小ブロック（サイズ fsw_bsize の）。
fsw_priority	ファイル・システムの優先順位（0～10）。
fsw_vnode	スワップ・ファイルが作成されるファイル・システム・スワップ・ディレクトリ（/paging）の vnode。
fsw_bsize	このファイル・システム上で使用されるブロック・サイズ。予約されるスペース fsw_reserve の量の決定に使用される。
fsw_head、fsw_tail	このファイル・システム・スワップに関連付けられる最初と最後のエントリの swaptab[]への索引。
fsw_mntpoint	ファイル・システム・マウント・ポイント。fsw_vnode を文字で表したもの。ユーティリティ（swapinfo(1M)など）とエラー・メッセージに使用。

swaptab 構造と swapmap 構造

2 つの構造がスワップ・スペースを追跡します。swaptab[]配列は、スワップ・スペースのチャンクを追跡します。swapmap エントリに、スワップ情報がページごとのレベルで保持されます。デフォルトでは、swaptab が 2MB のスペース・チャンクを、swapmap がこの 2MB チャンク内の各ページをそれぞれ追跡します。

swaptab[]配列の各エントリには、一意の swapmap へのポインタ（st_swpmp）があります。swapmap エントリには、swaptab 索引への逆方向ポインタがあります。swaptab エントリ（デフォルトでは 2MB または 512 ページ）で表される各ページについて 1 つのエントリが swapmap にあります。 つまり、swapmap のサイズは swchunk と一致します。

空きスワップ・ページのリンク・リストは swaptab エントリの st_free から始まり、空き swapmap エントリの sm_next それぞれが使用されます。スワップのページが必要になると、カーネルがこれらの構造をウォークします（vm_swalloc.c の get_swap()ルーチンを使用して）。このルーチンは、実際にはチャンクを配置する他のルーチンなどを呼び出します。

- まず、最低優先順位から、swdev_pri[].curr（swdevt エントリを指し示す）を調べる。
- sw_nfpgs がゼロである（空きページがない）場合、ポインタ sw_next に従ってこの優先順位の次の swdevt エントリを取得する。
- これらに空きページがない場合、swfs_pri[].curr（この優先順位のファイル・システム・スワップ）に移動して、fsw_nfpgs をチェックして空きページがあるかどうかを調べる。
- これにも失敗した場合、次の優先順位に移って再試行する。
- 空きページがある swdevt または fswdevt が見つかると、sw_head または fsw_head から各 swaptab エントリの st_next をたどりながら、ゼロ以外の st_nfpgs を持つ swaptab エントリが見つかるまで、そのデバイスの swaptab リストをウォークする。
- st_free が、この swaptab 内の最初の空き swapmap エントリ（および最初の空きページ）を指し示す。

- get_swchunk()ルーチンにより、swaptab 索引の dbd_data の 14 ビットと swapmap 索引の 14 ビットでディスク・ブロック記述子（dbd）が作成される。この領域内の r_bstore がディスク・デバイス swapdev_vp に設定され、dbd が DBD_BSTORE とマークされる。

  スワップからのフォールトインが発生すると、ファイル・システムから同じプロセスのフォールトインが続いて発生する。r_bstore と dbd_data が一緒にハッシュされ、ソフト・フォールトが発生していないかどうかチェックされ、次に devswap_pagein()が呼び出される。devswap_pagein()ルーチンは、14 ビット swaptab 索引と 14 ビット swapmap 索引として dbd_data を使用して、ディスク上のページの位置を判定する。

これで、スワップからのページの取得に必要な情報がすべて格納されます。

図 27 swaptab 構造と swapmap 構造

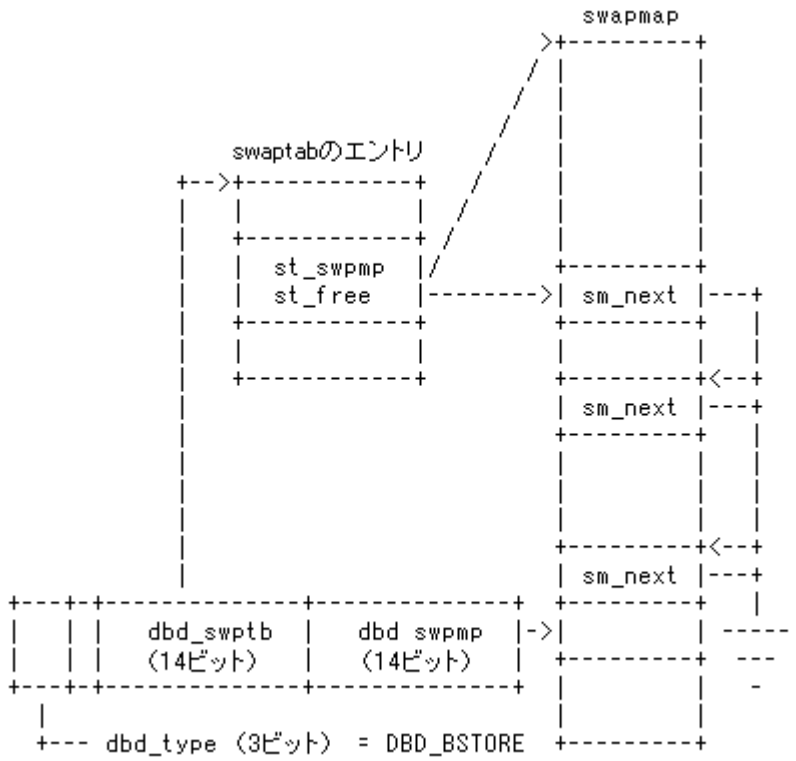


表 27 スワップ・テーブル・エントリ（struct swaptab)

要素	意味
st_free	チャンク内の最初の空きページへの索引。各エントリがスワップの 4KB ページにマッピング。
st_next	同じデバイス・スワップまたはファイル・スワップの次の swaptab への索引。リストの最後では、st_next は-1。
st_flags	ST_INDEL：チャンクが削除中であることを示すファイル・システム・スワップ・フラグ。そのチャンクからのページは割り当てられない。swapdel()ルーチンでのみ設定される。 ST_FREE：チャンクのページが使用中でないため、チャンクを削除できることを示すファイル・システ

	ム・スワップ・フラグ。リモート・スワップの場合、すぐにチャンクを削除してはならない。このフラグの設定時は、st_free_time を現在時間プラス 30 分（1800 秒）に設定する必要がある。30 分が経過すると、チャンクを解放できる。この期間中にチャンクが必要になった場合は、フラグをクリアできる。 ST_INUSE : swaptab エントリが変更中。
st_dev、st_fsp	swdevt[]エントリへのポインタ、または swaptab エントリを参照する fswdevt[]へのポインタ。
st_vnode	デバイスまたはスワップ・ファイルの vnode。
st_nfpgs	この（swchunk）swaptab エントリの空きページ数。
st_swpmp	スワップ・ページのこの swchunk を定義する swapmap[]配列へのポインタ。
st_free_time	リモート fs チャンクを解放できるタイミングを示す（ST_FREE フラグの説明を参照）。

表 28 スワップ・マップ・エントリ（struct swapmap）

要素	意味
sm_ucnt	ページを使用するスレッドの数。ゼロまで減分されると、スワップ・ページが解放され、空きページ・リンク・リストを更新できる。
sm_next	swapmap[]内の次の空きページの索引。sm_ucnt がゼロの場合のみ有効。これは、この swapmap エントリが、swaptab の st_free で始まるリンク・リストに含まれることを意味する。

デマンド・ページングの概要

以前に説明したように、プロセスを実行するには、すべての（データやテキストなどの）region をセットアップする必要があります。ただし、プロセスで必要になるまでページはメモリにロードされません。実際のページがアクセスされるときのみ、変換が確立されます。

コンパイルされたプログラムのヘッダには、データ領域とコード領域のサイズに関する情報が含まれます。fork と exec によって、コンパイルされたコードからプロセスが作成される際、カーネルがプロセスのデータ構造をセットアップし、プロセスの命令がユーザー・モードから実行開始されます。現在メイン・メモリにないアドレスにプロセスがアクセスを試みると、ページ・フォールトが発生します（たとえば、メモリ内にないページから実行を試みることがあります）。カーネルが実行をユーザー・モードからカーネル・モードに切り替え、必要な仮想アドレスを含む pregion を検索してページ・フォールトの解決を試みます。次にカーネルが pregion のオフセットと region を使用して、ページでの読み取りに必要な情報を検索します。

変換がすでに存在せず、ページが必要な場合、pdapage()ルーチンが変換（スペース ID、ページ内のオフセット、ページに割り当てられた保護 ID とアクセス・パーミッション、およびページの論理フレーム番号）を追加します。このルーチンは次に、必要に応じてページに取り込まれ、変換のセットアップ、テーブル内でのハッシュ、および残りのすべての処理を行います。

またカーネルは、要求されたページをロードする空き物理ページをメイン・メモリ内で検索します。使用可能な空きページがない場合、要求されたページ用のスペースを確保するために、選択された使用済みページがページ・アウトされます。次にカーネルは、ディスク上のファイル・スペースから要求されたページを取得（ページイン）します。また、プロセスで必要になることがある追加（隣接）ページも頻繁にページインされます。

次に、カーネルはページのパーミッションと保護をセットアップし、ユーザー・モードに戻ります。プロセスが命令を再実行し、この時点でページが検索され、実行が継続されます。

デマンド・ページングには、物理メモリより大きなプロセスが許容されるという柔軟性があります。この方式の短所は、プロセッサーに求められるページングが複雑だということです。ページ・フォールトを処理するために命令が再開可能でなければなりません。

デフォルトでは、すべての HP-UX プロセスは必要に応じてロードされます。デマンド・ページングのプロセスでは、実行前にプログラムがあらかじめロードされません。プロセスのコードとデータはディスク上に保存され、ページが増分されるときに必要に応じて物理メモリにロードされます（ほとんどアクセスされないルーチンとコードがプログラムに含まれることが多いです。たとえば、プログラムの大きな割合を占めるエラー処理ルーチンはアクセスされないことがあります）。

書き込み時コピー

システムでより効率的にプロセスを操作できるように、現在の HP-UX には EXEC_MAGIC プロセスの「書き込み時コピー（copy-on-write）」が実装されています。以前のシステムでは、プロセスが fork されるたびにプロセスのデータ・セグメント全体がコピーされ、fork 回数が増えるにつれてデータ・セグメントとコード・セグメントのサイズが増加していました。物理ページの変換が 1 つのみ維持されます。1 つの親プロセスで 1 つの物理ページを指し示し、読み取ることができますが、コピーできるのはページへの書き込み時のみです。子プロセスにはページ変換がなく、読み取りアクセスまたは書き込みアクセスのためにページをコピーする必要があります。

「書き込み時コピー」とは、親の region 内のページが、必要になるまで子の region にコピーされないことを意味します。親と子の両方が、同じページの共有を心配することなくページを読み取ることができます。ただし、親または子のいずれかがページへの書き込みを行うと、新しいコピーが書き込まれ、他方のプロセスには元のページが保持されます。

EXEC_MAGIC の実装の詳細は、ホワイト・ペーパー 『HP-UX Process Management』 (HTML) (PDF) を参照してください。

スワップ・スペース管理

プロセスが fork される際、その親プロセスの複製コピーにより、子プロセスの基礎が形成されます。

領域の種類によって異なる複製方法

カーネルルーチン procdup()を使用して、システムが親プロセスの pregion リストをウォークし、子プロセス用に各 pregion を複製します。この方法は、region の種類によって異なります。

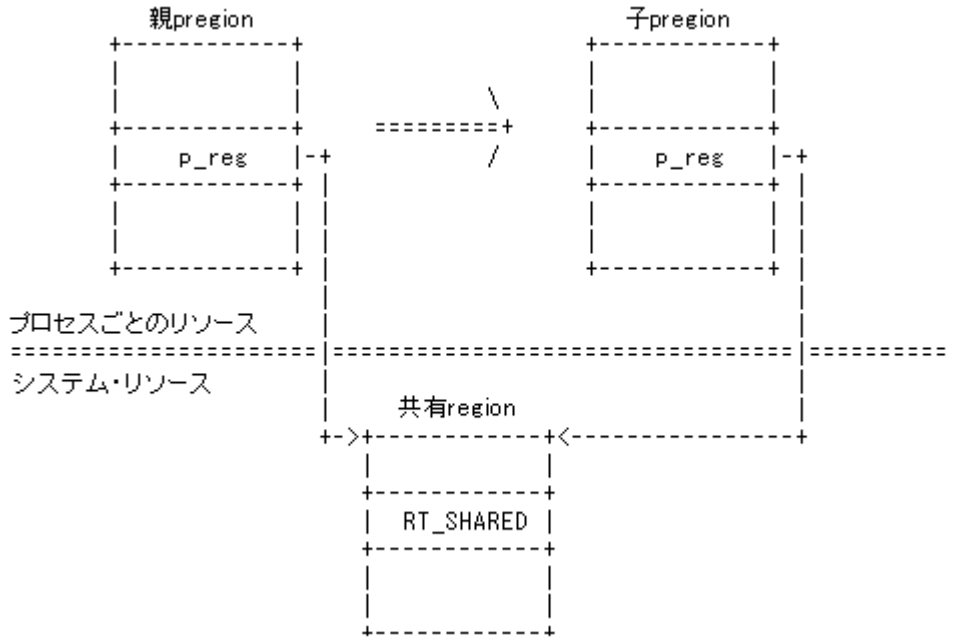
- region の種類が RT_SHARED の場合、新しい pregion が作成され、親の region に付加される。
- region の種類が RT_PRIVATE の場合、まず region が複製され、次に、新しい pregion が作成されて新しい region に付加される。

共有 region の pregion の複製

種類が RT_SHARED の region は、親と子で共有されるため、pregion と region に対する変更はわずかです。新しい pregion のみ作成し、共有 region に付加する必要があります。

- 新しい pregion が割り当てられ、親 pregion から子 pregion にフィールドがコピーされる。
- vhand で使用される pregion 要素（p_agescan、p_ageremain、および p_stealscan）がゼロに初期化され、子 pregion が stealhand の直前のアクティブな pregion チェーンに追加される。これで、子 pregion がスチールされることを防止できる。
- region にアクセスするメモリ内の pregion の数と、region にアクセスするメモリ内またはページングされた pregion の数を反映するように、region 要素 r_incore と r_refcnt が増分される。

図 28 共有 region の pregion の複製



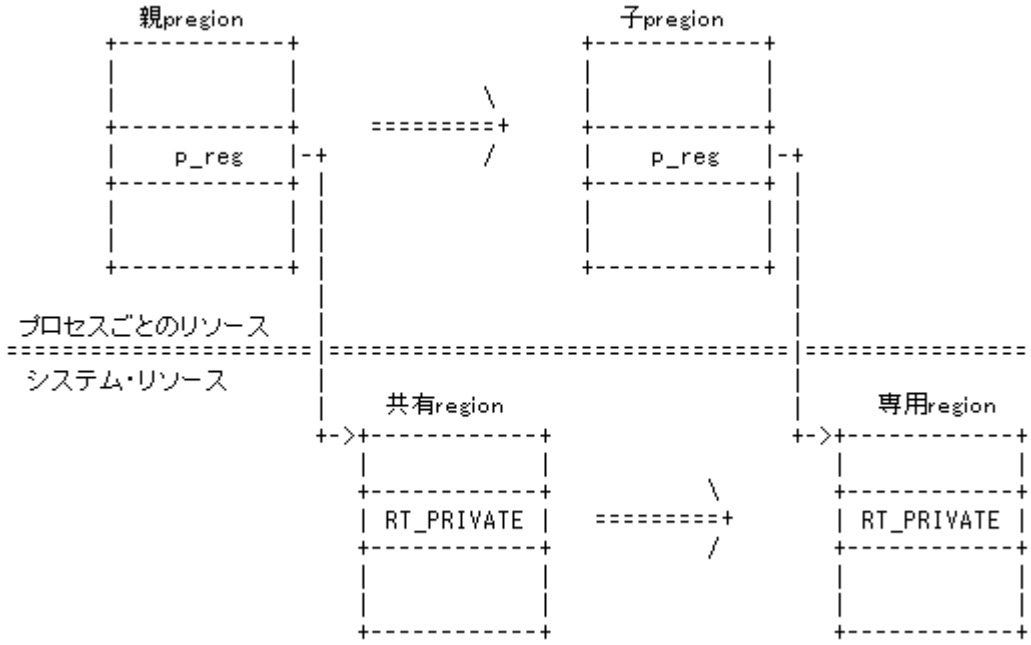
専用 region の pregion の複製

RT_PRIVATE region のコピー手順は、RT_SHARED region よりも複雑です。

- 新しい region が割り当てられる。
- 子 region のポインタが設定される。

  - 順方向格納ポインタ r_fstore が親と同じ値を指し示すように設定され、vnode の参照カウント（v_count）が増分される。
  - 逆方向格納ポインタ r_bstore がカーネル・グローバル swapdev_vp に設定され、その v_count も増分される。
- 子 region が、アクティブな region のリンク・リストの最後に付加される。
- スワップが予約される。使用可能なスワップ・スペースが不十分な場合、fork()が失敗し、エラーENOMEM が返される。
- region の B-tree 構造が初期化され、完全に満たされた B-tree 用の十分なスワップ・スペースが予約される。
- 親の vfd プロトタイプ値と dbd プロトタイプ値が子の B-tree ルートにコピーされる。
- region のすべてのページが「書き込み時コピー」されるように、親 region と子 region の両方の vfd プロトタイプが設定される。
- B-tree に追加された新しい vfddbd ペア用に vfd で「書き込み時コピー」フラグ（pg_cw）を設定しなければならないことを示す B-tree 要素 b_vproto が設定される。
- vfddbd のチャンクが子の B-tree 用に作成され（親の B-tree 内の vfddbd の各チャンクと等しい）、プロトタイプ値で満たされる。pg_cw ビットは、子 B-tree のチャンク内のすべてのデフォルト vfd について「書き込み時コピー」にすでに設定されている。

図 29 RT_PRIVATE region の複製



vfd が有効な場合の「書き込み時コピー」の設定

子 region 内の vfddbd のチャンクを使用する前に、すべてのエントリの有効性をチェックする必要があります。

- vfd が有効でない（その pg_v が設定されていない）場合、親の vfd の pg_cw を設定し、子にコピーする必要がある。pg_lock が親内で設定されている場合、ロックが継承されないため、子内の設定を解除する必要がある。

vfd が有効であれば、低レベル構造がさらに変更されます。

- 有効ページ数を反映するように、子 region 内の r_nvalid 要素が増分される。

- vfd には、pfdat[]配列への索引である pfn（ページ・フレーム番号）が含まれます。pfdat エントリ pf_use カウント（このページを使用する region の数）を増分する必要があります。
- 親 vfd の「書き込み時コピー」ビットが設定されていない場合、ページへの変換が「書き込み時コピー」として動作するように pde を設定する必要がある。

ページとスワップ・イメージの調和

ページがスワップ・デバイスに書き込まれ、その後に変更されている場合は、スワップ・デバイス・データがメモリ内のデータと相違します。 dbd のタイプを DBD_NONE に設定して、ディスク・ページとメモリ内のページの結合を解除する必要があります。次回、スワップ・デバイスにページを次回書き込む際に、ページが新しい位置に割り当てられます。

これで、親の B-tree の観点からすべてが「書き込み時コピー」にセットアップされます。

子 region の「書き込み時コピー」ステータスの設定

- 子の r_swalloc が、予約された region と B-tree のページ数に設定される。
- r_prev と r_next が、子 region を親 region にリンクするように設定される。
- カーネルが、親 pregion から新しいスペースをコピーするのではなく、pregion 用の新しいスペースを選択する。これにより、2 つの範囲の仮想アドレス（スペースは異なり、オフセットは同じ）から 1 つの範囲の物理アドレスへの変換が確立される。
  - 親プロセスがその仮想アドレスにアクセスすると、そのアドレスが TLB から消去されているため TLB ミス・フォールトが発生する。
  - 子プロセスがその仮想アドレスのいずれかにアクセスすると、そのアドレスが TLB に以前存在しなかったため TLB ミス・フォールトが発生する。

プロセスのアドレス空間の複製

- procdup()が、fork タイプ、親プロセス（pp）、子プロセス（cp）、親スレッド（pt）、および子スレッド（ct）に基づいてプロセスの複製コピーを作成する。

  procdup()が、子の uarea 用のメモリを割り当てる（実際には、procdup()は、uarea を作成する createU()を呼び出すルーチン）。

- procdup()が、実行されているプロセスの種類（fork または vfork）にもとづいて親の仮想アドレス空間を複製する dupvas()を呼び出す。

- プロセスが fork で作成された場合、dupvas()が親プロセスの仮想アドレス空間を複製する。プロセスが vfork された場合、親の仮想アドレスが使用される。

  dupvas()が、（複製する必要があるデータ・オブジェクトがどれであっても）各専用データ・オブジェクトを検索し（テキスト、メモリ・マッピング、データ・オブジェクト、グラフィックスで求められる特別な留意事項がある）、特殊オブジェクトの複製が終了すると、専用 region または共有 region のどちらを処理するかに応じて private_copy()または shared_copy()を呼び出す。

  - 共有 region の場合、共有されていることを示すために shared_copy で region 上の参照カウントが増分される。
  - 専用 region の場合、private_copy が region をロックし、dupreg()の呼び出しにより region を複製できるようにする。
- dupreg()が子用の新しい region を割り当て、親の vfd と region 構造全体を複製する。次に、do_dupc()を呼び出して region の下のエントリを複製する。
- do_dupc()が親／子関係をセットアップし、その関係を複製することで、子を「書き込み時コピー」にセットアップする。また、do_dupc()は、親の region が有効であることを確認し、子を「書き込み時コピー」に設定する。さらに、変換を rx（読み取り実行）のみに設定し、region 内の vfddbd の組み合わせすべての情報を複製する。
- 次に、do_dupc が hdl_cw()を呼び出して、子のアクセス権を更新し、子を「書き込み時コピー」にする。

これが完了すると、子プロセスが親プロセスの複製バージョンとして存在します。子プロセスが子のアドレス空間に付加されて、親に依存しなくなります。

子プロセスの uarea の複製

プロセスの各スレッドに、固有の uarea があります。プロセスが fork()する場合、新しいプロセスにはスレッドが 1 つのみ存在し、そのスレッドに uarea が必要です。procdup()が createU()を呼び出して、この uarea を作成します（uarea pregion は dupvas()でコピーされないため、親のスレッド（および、関連する uarea）の数に関係なく、子の uarea は 1 つのみになります）。

createU()ルーチンが、子プロセスの uarea とアドレス空間を生成します。fork されたプロセス用に最後に uarea がセットアップされ、pregion の複製コードの中間で子プロセスが再開されることが防止されます。このプロセスが vfork されると、exec()中に uarea が作成されます。それまで、子プロセスは親プロセスのスレッドの uarea を使用します。

- ユーザー・プロセスが FORK_PROCESS で作成される際、子の uarea に変更される親の uarea の作業コピー用に一時的スペースが割り当てられる。この一時的スペースは、uarea が新しい region にコピーされた後、解放される。データがコピーされた直後、fork()が親の uarea の u_pcb 内の savestate を更新する。（vfork()では、exec()中に uarea が作成されて savestate がすぐに変更されるため、この更新は行われない）。
- 新しい uarea 用に region が割り当てられ、そのデータ構造が初期化される。また、その r_bstore 値がスワップ・デバイスに設定され、新しい region がアクティブな region のリストに追加される。uarea にはすでにデータが入っているため、uarea には r_fstore の値が設定されない。
- uarea の pregion 用にスペースが割り当てられ、初期化される。各 uarea には、一意のスペース ID がある。新しい pregion は、PF_NOPAGE フラグでマークされる。uarea pregion はアクティブな pregion のリストに追加されないため、vhand の影響は受けない。プロセス全体がスワップアウトされる場合のみ、uarea のページがスワップ・デバイスに書き込まれる。
- 作成された pregion は、vas に接続されている pregion のリンク・リストに付加される。そのポインタが r_pregs に保存され、p_prpnext がヌルに設定され、r_incore と r_refcnt が 1 に設定される。
- uarea と B-tree ページ用にスワップ・スペースが予約され、デフォルトの dbd が DBD_DFILL に設定されると、uarea ページ（UPAGES）が割り当てられる。各ページには、1 ページの物理メモリが必要（すぐに使用可能なメモリがない場合は休眠状態となる）。pfn が vfd に保存され、pg_v が有効として設定される。また、r_nvalid が増分され、pde が物理から仮想への変換用に作成される。pfdat エントリの P_UAREA フラグと HDLPF_TRANS フラグが設定され、dbd が DBD_NONE に設定される。
- 子のスレッド構造内のポインタ kt_upreg が、このスレッドの uarea pregion を指し示すように設定される。

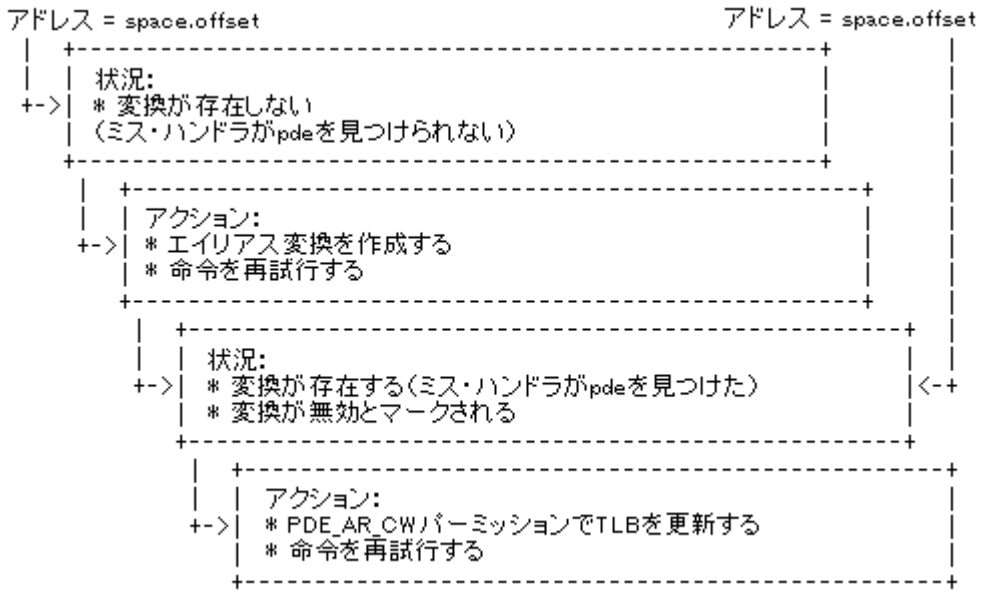
これで子の実行が成功すると考えられます。setjmp()呼び出しでコピーされ pcb_sswap で指し示された uarea に、現在の状態が保存されます。このため、子が resume()ルーチンを最初に呼び出すと、このルーチンが pcb_sswap がゼロ以外であることを認識し、longjmp()を行ってここに戻ります。次に、子が戻り値 FORKRTN_CHILD で procdup()から返されます。

親のオープン・ファイル・テーブルが子にコピーされ、コピー済みの uarea が実際の pregion にコピーされます。このコピーによって TLB ミス・フォールトが発生し、その結果 pregion の pde が TLB に書き込まれます。そのため、uarea の仮想アドレスが、セットアップされたばかりの物理ページに関連付けられます。戻り値 FORKRTN_PARENT で procdup()から返されることで、このプロセスが完了します。

親の「書き込み時コピー」ページからの読み取り

親が読み取りのためにその RT_PRIVATE ページの 1 つにアクセスすると、プロセッサーが、カーネルにより割り込みとして処理される TLB ミス・フォールトを生成します。TLB ミス・フォールト・ハンドラが hpde を検索し、その情報（新しいアクセス権を含む）をプロセッサーの TLB に挿入します。PDE_AR_CW によってユーザー・モードの読み取りアクセスが許されるため、割り込みから戻る際、プロセッサーが読み取りを再試行し、成功します。

図 30「書き込み時コピー」ページに対する初回の読み取り



子の「書き込み時コピー」ページからの読み取り

子が読み取りのためにそのページの 1 つにアクセスするとき、まだ何もセットアップされていないため、TLB ミス・ハンドラは仮想アドレスの hpde を検索しません。この仮想アドレスは、pregion でセットアップされています。「アクセス時コピー（copy-on-access）」（現在のデフォルト）を行わない場合、ページが必要になると、エイリアス指定された変換を行う必要があります。

- 最初に save_state が作成される。
- vas ポインタが取得され、このアドレスのページを含む pregion を見つけるためにスキップ・リストが検索される。
- このページが複数の仮想アドレスに変換される場合、適切なエイリアスが得られる。
- 子領域が読み取りのためのページへのアクセスに失敗し、TLB ミスが発生するが、ミス・ハンドラが変換を見つけて TLB にロードする。

- ルーチンが割り込みから戻り、ページの読み取りに成功する。

ページのフォールトイン

領域の初期化時、ディスク・ブロック記述子（dbd）の dbd_data フィールドが必ず DBD_DINVAL (0x1fffffff)に設定されます。プロトタイプ dbd_type 値は、次のように設定されます。

- テキストと初期化されたデータの場合、DBD_FSTORE。
- スタックと初期化されていないデータの場合、DBD_DZERO。

ページが最初に読み取られる際、物理ページ（および、スパース PDIR 内のその変換）がまだ存在しないため、TLB ミス・フォールトが発生します。このページを取り込み、フォールトが発生した命令を再開するのがフォールト・ハンドラです。フォールト・ハンドラは、ページが有効かどうかを判断するために、フォールトが発生しているプロセスのどの pregion にフォールトが発生しているアドレスが含まれるかを判別します。最終的にフォールト処理コードによって、主要な仮想フォールト処理ルーチン virtual_fault()が呼び出されます。このルーチンに渡される引き数は、フォールトを引き起こした仮想アドレス、pregion、および読み取りまたは書き込みアクセスを示すフラグです。

カーネルがページの vfd と dbd を見つけるために B-tree を検索します。vfd フラグに有効ビットが設定されている場合、別のプロセスがすでにアドレスをメモリに読み込んでいます。領域内で r_zomb フラグが設定されている場合、カーネルが Pid %d killed due to text modification またはページ I/O エラーメッセージを出力し、SIGKILL を返します。SIGKILL は、ハンドラによりプロセスに送られます。

スタックまたは未初期化データのページのフォールトイン

dbd_type 値が DBD_DZERO に設定されている場合（スタックと初期化されていないデータの場合と同様に）、プロセスが「書き込み時コピー」ビットをゼロに設定します。次にカーネルが、ページがシステム・プロセスまたは高優先順位のスレッドのどちらに関係があるかチェックします。どちらにも関係がなく、メモリが不足気味の場合、プロセスに関連付けられた優先順位まで空きメモリが下がるまでプロセスは休眠状態になります（最悪の場合、メモリが desfree を上回るまで、スレッドが待機することがあります）。

プロセスが再開すると、精度を保つために vfd ポインタと dbd ポインタが調べられます。空き pfdat エントリが物理メモリ・アロケータから取得され、その pfn (pf_pfn)が vfd に配置されます。また、vfd の有効ビットが設定され、領域の r_nvalid カウンタ（有効なページ数）が増分されます。次に、ページ数がゼロにされ、仮想から物理への変換がスパース PDIR に追加されます。最後に、カーネルが dbd_type を DBD_NONE に、dbd_data を 0xfffff0c に変更します。

テキストまたは初期化済みデータのページのフォールトイン

プロセスの DBD_FSTORE ページ上に仮想フォールトがある場合、カーネルは vnode への r_fstore ポインタを使用して、呼び出すファイル・システム特有の pagein()ルーチン（たとえば、ufs_pagein()、nfs_pagein()、cdfs_pagein()、vx_pagein()）を決めます。pagein()ルーチンは、メモリ・ページの空きリストからの正しいページの回復、またはディスクからの正しいページでの読み取りに使用されます。

pagein()ルーチンは、フォールトが発生しているページに関する情報を vm_pagein_init()ルーチンから取得します。vm_pagein_init()ルーチンは vfd/dbd の取得、領域索引のセットアップ、および有効なページがまだ存在しないことの確認を行います。

1 つのページを予約する必要があります。ゼロが満たされたページ（スパース・ファイル）またはページ・キャッシュのいずれかでページ・フォールトをローカルに満たすことができるかどうかを判定するために、vm_no_io_required()が呼び出されます。

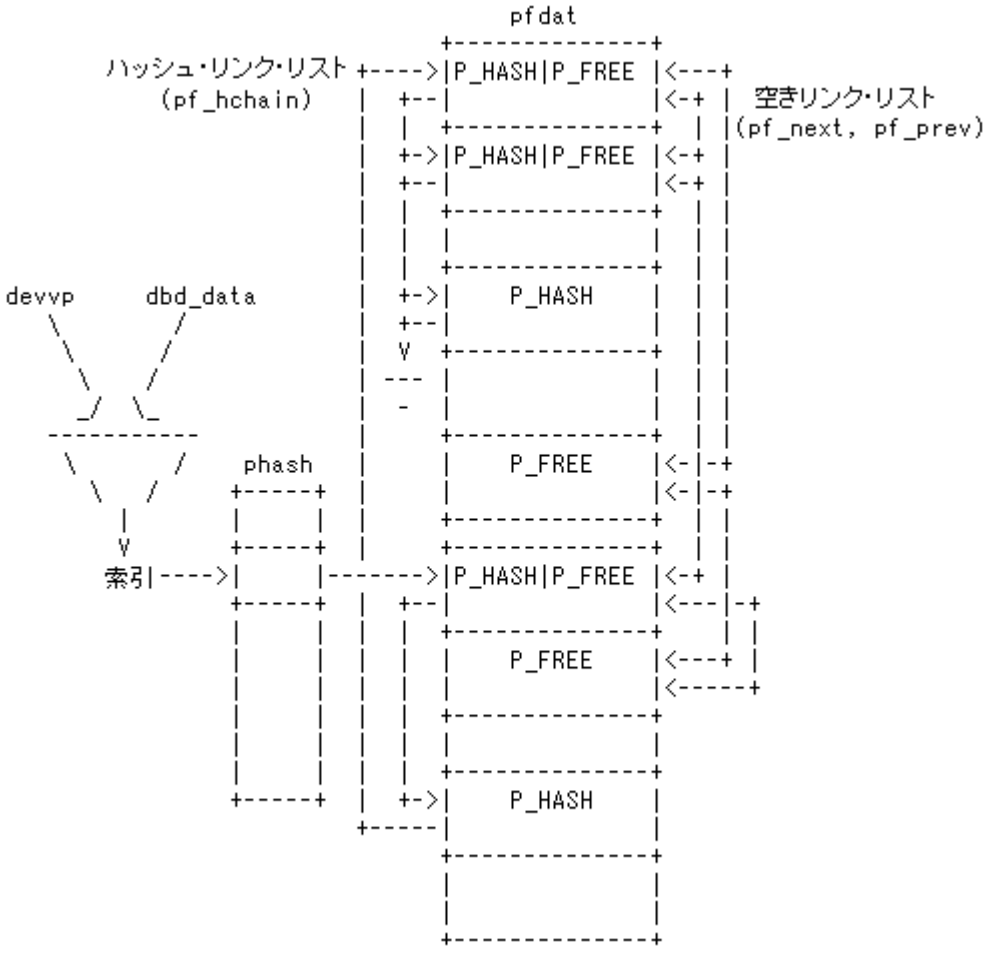
vm_no_io_required()は、lgpg_cache_lookup()を呼び出して、ページ・キャッシュ内にフォールトが発生したページがあるかどうかチェックします。

lgpg_cache_lookup()は、pageincache()を使用して基本ページを検索し、次に lgpg_lookup()を使用して基本ページが適切な大きさのページの一部であるかどうか判断します。

pageincache()は、vnode ポインタとデータをハッシュして、phash[]内の pfdat ポインタを選択します。また、pfdat エントリの pf_hchain チェーンをウォークして、一致する vnode ポインタ（pf_devvp）とデータ値（pf_data）を検索します。一致が見つかると、空きリストから削除されます。

ページ・キャッシュ内でページが見つかると、その領域の有効なページ・カウント（r_nvalid）が増分され、vfd が pfn（pf_pfn）で更新されます。また、そのページの仮想から物理への変換がスパース PDIR に追加されます（変換が削除されている場合）。

図 31 DBD_FSTORE ページのフォールトインに対するページ・キャッシュのチェック



ディスクからのテキストまたは初期化済みデータのページの取り出し

必要なページがページ・キャッシュ内で見つからない場合、pagein()ルーチンが dbd を参照して、フェッチ対象のページを確認します（この情報は vm_no_io_required()によって dbd に保存されています）。一般に pagein()ルーチンは、フォールトが発生している単一のページよりも多くのページの読み取りを試みます。つまり、4KB よりも大きなページの使用（利用可能なメモリやファイル属性などの状況により、そうすることが可能な場合）と、順次アクセスをしているファイルからの余分なページの単純な先読みの両方を行います（そのファイル上で次にページ・フォールトが発生したときに、余分なページを使用できるようにします）。

メモリのページ（1 つまたは複数）が物理メモリ・アロケータから割り当てられ、仮想から物理への変換がスパース PDIR に追加されます。また、ディスクからページへの I/O がスケジュールされ、プロセスが非先読み I/O の完了を待機して休眠状態になります（プロセスは先読み I/O の完了を待機しません）。vfd が有効とマークされます。dbd の dbd_type は DBD_FSTORE に設定され、dbd_data はディスク上のブロック・アドレスに設定されたままになります。

ページ・データにゼロが満たされのか、それとも空きリストまたはディスクのいずれから取り出されたのかにかかわらず、ページ・ディレクトリ・エントリ（pde）が触れられます。命令が再試行されて、TLB ミス・フォールトが発生します。ミス・ハンドラが変更された pde データを TLB に書き込みます。命令がまた再試行されて成功します。

仮想メモリと exec()

システムで exec()が実行されるとき、仮想メモリシステムでは、古い pregion または region がクリーンアップされ、新しい pregion または region がセットアップされます。

vfork()からのクリーンアップ

vfork()からのクリーンアップは簡単です。

- 子プロセスが実行されるが、そのリソースは親プロセスから借りられる。
- カーネルが新しい vas を取得して、子プロセス（p_vas）に付加する。
- fork タイプが FORK_PROCESS の場合と同様に、親プロセスの uarea とスタックがコピーされ、子プロセスの uarea 用に pregion と region が作成される。また、親プロセスのカーネル・スタックを使用していたスレッドが新しい子プロセスのカーネル・スタックを使用するようになる。
- 子プロセスが親プロセスのリソースを返す。
- 次に、カーネルがテキストやデータなどを追加する。

古い pregion の処理：dispreg()

exec()が FORK_PROCESS タイプの fork の後で呼び出された場合、最初に複数の領域を処理する必要があります。通常は、まだ必要な PT_UAREA pregion を除くすべての pregion が処理されます。ファイルがファイル自身に対して exec()を呼び出している場合、わずかのプロセスが保存され、PT_TEXT 領域と PT_NULLDREF 領域も維持されます。

- deactivate_preg()を使用してアクティブな pregion リストから削除することで、pregion が非アクティブ化される。非アクティブ化されている pregion を agehand が指し示し、stealhand がアクティブな pregion リスト内の次の領域を指し示している場合、agehand の順序が stealhand を超えないように、agehand が 1 つの pregion だけ後方に移される。非アクティブ化されている pregion を agehand または stealhand が指し示している場合、両方の針が 1 つの pregion だけ前方に移される。
- hdl_detach()が呼び出されて、プロセスのアドレス空間から領域を切り離す処理のハードウェアに依存する部分が処理される。特に、これがアドレス空間への最後の参照である場合、そのリソースを解放する必要がある。
    - hdl_detach()が、領域に対して保留中の I/O（つまり、r_poip = 0）の完了を待機するために wait_for_io()を呼び出す。これにより、現在別の目的に割り当てられているページを変更する I/O 要求は返されない。
    - hdl_detach()が、すべての仮想アドレス変換を削除するために、領域の B-tree の各チャンクに対して do_deltransc()を呼び出す。有効な vfd それぞれについて、do_deltransc()が hdl_deletetrans()を呼び出す。hdl_deletetrans()は pddpage()を呼び出して、以下の処理を行う。
        - キャッシュをフラッシュする。
        - hpde を無効にする（スペースを-1 に設定し、アドレス、pde_phys（pfn）、pde_ref、および pde_os を 0 に設定する）。
        - hpde が htbl エントリでない場合、hpde がハッシュ・リストから空きリストに移される。 hpde が HTBL hpde であって未使用の場合、hpde を変換先のリンク・リストで満たし、コピーされた hpde を解放する試みがなされる。
        - pfn_to_virt テーブルから変換を削除する。

    - これが共有アドレスであった場合、アドレス空間アロケータに返される（つまり、別の region 用に仮想アドレスを再使用することが可能になる）。
- pregion ポインタが r_pregs リストから削除され、pregion で使用されていたメモリが解放される（つまり、カーネル・メモリ・アロケータに返される）。
- 領域の r_incore 要素と r_refcnt 要素が減分される。r_refcnt がゼロの場合、領域も解放される。

region を解放する必要がある場合に呼び出されるルーチン freereg()が、以下の処理を行います。

- 以下の処理を行うために、pgfree()を呼び出す。
    - 領域に対して保留中の I/O（つまり、r_poip = 0）の完了を待機するために wait_for_io()を（再び）呼び出す。これにより、現在別の目的に割り当てられているページを変更する I/O 要求は返されない。
    - region の B-tree に（再び）移り、region の有効なページすべてを解放する（freepfd()）ために、B-tree の各チャンクに対して do_freepagesc()を呼び出す。
        - pfdat の pf_use フィールドが減分される。
        - 物理ページがエイリアス指定されていない場合、その pf_use が 0 になる。つまり、別の用途のために物理ページを解放できる。その P_FREE フラグが設定され、物理ページが物理メモリ・アロケータに返される。カーネルのグローバル・パラメータ freemem が増分される。別のプロセスがメモリを待機している場合、最初のプロセスがページを獲得できるように、それらのプロセスがすべて起動される（この競争に負けたプロセスは再び休眠状態になる）。
    - これが共有アドレスであった場合、アドレス空間アロケータに返される（つまり、別の region 用に仮想アドレスを再使用することが可能になる）。
- r_bstore が swapdev_vp の場合、B-tree 構造用に予約されたスワップ・ページ（ページ数は r_root->b_rpages）と同様に、予約済みのスワップ・ページ（r_swalloc）が解除される。
- 領域要素 r_root と r_chunk がカーネル・メモリ・アロケータに返される。
- activeregions が減分される。region が、アクティブな region リストとその vnode に関連付けられた region のリストから削除され、region struct 自体がカーネル・メモリ・アロケータに返される。

新しいプロセスの生成

メモリ構造を作成中のプロセスが実行可能ファイルとしてファイルを最初に使用するプロセスである場合、その実行可能ファイルの vnode の v_vas はヌルであり、pseudo-vas、pseudo-pregion、および region の作成が必要になります。そうでない場合は、pseudo-vas の参照カウントが更新されます。

- PT_TEXT pregion が付加される領域は、実行可能ファイルの種類によって異なる。
  - 実行可能ファイルが EXEC_MAGIC でない場合、PT_TEXT pregion は pseudo-vas region に付加される。
  - 実行可能ファイルが EXEC_MAGIC である場合、VA_WRTEXT がプロセス vas で設定され、pseudo-vas の region が RT_PRIVATE region として複製される（RT_PRIVATE region について説明したすべての手順が実行される）。また、必要になる前にスワップが予約されないように RF_SWLAZYWRT が新しい region で設定され、PT_TEXT pregion がその region に付加される。
  - 両方のケースで、新しいスペースが pregion の仮想アドレスに付加される。
  - PT_NULLDREF pregion は、PT_TEXT と同じスペースを使用するグローバル region（globalnullrp）に付加される。
  - 実行可能ファイルのデータ部分の最初を指し示す r_off を使用する RT_PRIVATE 領域として、pseudo-vas の領域が複製される。PT_DATA pregion がこの領域に付加される。これが EXEC_MAGIC 実行可能ファイルである場合は PT_TEXT pregion のスペースが使用され、そうでない場合は新しいスペースが割り当てられる。
- DBD_DZERO dbd を使用する PT_DATA pregion が bss（初期化されていないデータ領域）のサイズだけ増分される。これにより、初期化されていないデータ領域の最後に b_protoidx が設定され、b_proto2 が DBD_ZERO に設定される。より多くのスワップが予約される。
- ユーザー・スタック用に 3 つのページの専用領域（SSIZE +1）が作成される。dbd プロトタイプ値が DBD_DZERO に設定され、PT_STACK pregion が USRSTACK で付加される。PT_UAREA pregion のスペースが使用される。
- 共有ライブラリがプロセスにリンクされる際、2 つの PT_MMAP pregion が作成される。それは、KERNELSPACE のスペースを持つ第 3 クワドラントにマッピングされたテキストを含む RT_SHARED pregion と、PT_DATA pregion のスペースを持つ関連データ（ライブラリ・グローバル変数など）を含む RT_PRIVATE pregion。
- VA_WRTEXT が設定されている場合、データ pregion には、それを超えるとテキストが終了する（第 1 クワドラントと第 2 クワドラント内の）最初の使用可能アドレスが割り当てられる。そうでない場合は、第 2 クワドラント内の最初の使用可能アドレスが割り当てられる。

仮想メモリと exit()

仮想メモリの観点からすると、exit()は exec()の最初の部分に似ています。プロセスに割り当てられたすべての仮想メモリ・リソースが破棄されますが、新しい仮想メモリ・リソースは割り当てられません。

したがって、vfork の子が exec()を実行する前に exit()で終了した場合、親プロセスに返すリソースを除き、仮想メモリからクリーンアップすべきリソースはありません。vfork 以外の子が終了した場合、dispreg()の呼び出しによって仮想メモリ・リソースが破棄されます。