Vol.2021-OS-151 No.11
2021/3/2

Tender における 2種類の管理単位を持つ資源「実メモリ」の設計と実現

楠 恒輝[1]　山内 利宏[1]　谷口 秀夫[1]

概要：LSI の高集積化により，計算機に搭載される実メモリ量が増加している．また，1 つの計算機上に複数プログラムが動作することによって，必要なメモリ量も増加している．多くのオペレーティングシステムでは，実メモリを 4 KB の単位で管理しているため，使用する仮想メモリ空間の大きさが大きくなると，仮想アドレスと実アドレスの変換表のエントリ数が増加して，TLB のヒット率が低下し，性能が低下する要因になる．そこで，実メモリを管理する単位を 4 KB と 4 MB の 2 種類持つ実メモリ管理を実現することにより，仮想アドレスと実アドレスの変換表のエントリ数の削減を可能にし，TLB のヒット率の向上を目指す．本稿では，***Tender*** において実メモリを 4 KB と 4 MB の 2 種類の単位で管理する資源「実メモリ」の設計と実現方式について述べる．

1. はじめに

LSI の高集積化により，メモリチップが大容量小型化している．これにより，計算機に搭載できる実メモリ量が増加している．また，計算機搭載実メモリ量の増加に伴い，1 つの計算機上で多くのプログラムを実行している．例えば，CentOS 8.3 では，63 個のプロセスが常駐して動作している．

多くのオペレーティングシステム（以降，OS）では，実メモリを 4 KB の単位で管理している．一方，応用プログラム（以降，AP）のプロセスは，多くの機能を提供するために，その大きさが増加している．このため，仮想アドレスと実アドレスの変換表（以降，アドレス変換表）のエントリ数が増加し，TLB のヒット率が低下し，AP プロセスの処理性能が低下する問題がある．

これに対し，実メモリを大きい単位で管理する技術が研究されている [2]，[3]，[4]．実メモリを管理する単位（以降，管理単位）を大きくすることで，仮想アドレスと実アドレスの変換表のエントリ数を削減でき，TLB のヒット率を向上させ，AP プロセスの動作を高速化できる．しかし，メモリの内部断片化が大きくなってしまう問題がある．

Tender オペレーティングシステム [1]（以降，***Tender***）では，実メモリを資源「実メモリ」として独立に管理している．実メモリの管理単位は 4 KB である．

[1] 岡山大学 大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

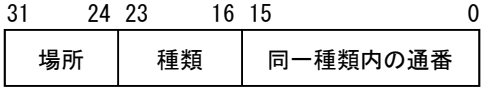

図 1　資源識別子

そこで，本稿では，上記の問題に対処するため，4 KB と 4 MB の 2 種類の管理単位を持つ資源「実メモリ」の設計と実現方式について述べる．また，簡単な評価結果を報告する．

2. *Tender* オペレーティングシステム

2.1 資源の分離と独立化

Tender では，OS が制御し管理する対象を資源と呼び，資源の分離と独立化を行っている．資源は，資源名と資源識別子によって識別される．資源識別子を**図 1** に示す．資源生成時に付与される資源識別子は，資源の場所，種類，および同一種類内の通番を有する．

また，資源は，資源の種類ごとの操作インタフェース（以降，プログラム部品）と個々の資源に関するデータを管理するテーブル（以降，資源管理表）を提供している．プログラム部品は，資源インタフェース制御（以降，RIC）という共通のインタフェースを用いて呼び出される．

2.2 メモリ関連資源

Tender におけるメモリ関連資源の関係について，**図 2** に示し，以下で説明する．

資源「実メモリ」は，固定長で実メモリを管理する資源である．資源「仮想領域」は，メモリイメージを仮想化し

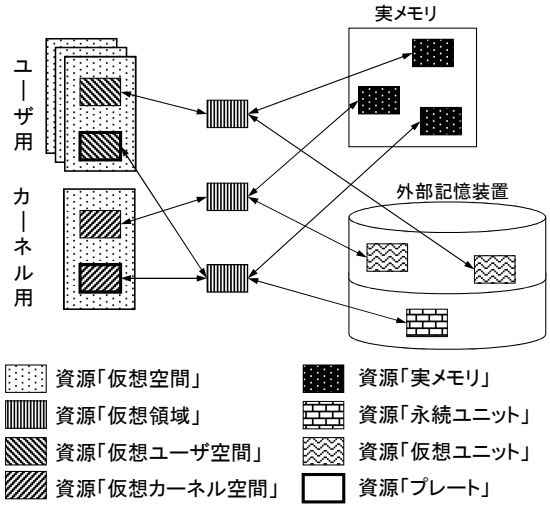

図 2 メモリ関連資源

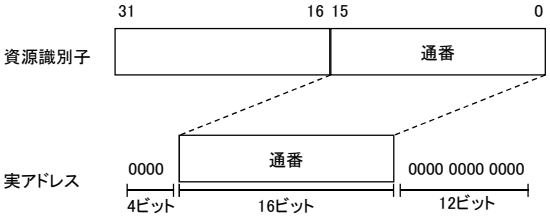

図 3 通番と実アドレスの関係

た資源であり，実体は実メモリもしくは外部記憶装置に存在する．資源「仮想領域」を作成する際に，作成する資源「仮想領域」のサイズに応じて資源「実メモリ」を作成して割り当てる．資源「仮想空間」は，特定のアドレス領域を持つ仮想的な空間であり，仮想アドレスから実アドレスへのアドレス変換表に相当する．資源「仮想ユーザ空間」は，資源「仮想領域」をユーザ用の資源「仮想空間」に貼り付けることで作成される．ここで，「貼り付ける」とは，仮想アドレスを実アドレスに対応付けすることであり，具体的には．当該の仮想アドレスに対応するアドレス変換表のエントリに，実アドレスまたは外部記憶装置のアドレスを設定する．また，仮想アドレスと実アドレスの対応付け解除を「剥がし」と呼ぶ．資源「仮想カーネル空間」は，資源「仮想領域」をカーネル用の資源「仮想空間」に貼り付けることで作成される．

2.3 資源「実メモリ」

資源「実メモリ」は，4 KB 単位で実メモリを管理する．**図 3** に示すように，資源識別子の通番は 16 ビット幅であり，通番と実アドレスを直接関係づけている．通番を 12 ビット左シフトすることで実アドレスが得られる．これにより，資源「実メモリ」を利用する処理を高速化している．しかし，資源「実メモリ」で管理する実メモリの最大サイズは，256 MB（4 KB × 2^{16}）である．

3. 2 種類の管理単位を持つ資源「実メモリ」

3.1 背景

計算機に搭載されている実メモリ量が増加している．管理単位として次の単位がある．

実メモリの管理単位を 4 KB とした場合，走行するプロセスの大きさに合わせてメモリを使用できるため，内部断片化は少ない．しかし，アドレス変換表のエントリ数は多くなるため，TLB ヒット率が低下する問題がある．

一方，管理単位を大きくした（例えば，4 MB）場合，走行するプロセスの大きさ以上のメモリを使用することになるため，内部断片化が発生しメモリ不足が発生する可能性が高くなる問題がある．しかし，アドレス変換表のエントリ数は少なくなるため，TLB ヒット率は向上する．

したがって，各管理単位の利点と欠点を考慮し，2 種類の管理単位を持つ資源「実メモリ」を実現し，両者の利点を生かす方式を実現する．

3.2 要件

2 種類の管理単位を持つ資源「実メモリ」に求められる要件として以下の 2 つがある．

(要件 1)資源「実メモリ」を利用する処理の高速化を保持できること

資源「実メモリ」では，資源識別子の通番と実アドレスを直接関係づけている．これにより，実アドレスを取得する処理を高速化し，資源「実メモリ」を利用する処理を高速化している．このように，2 種類の管理単位を持つ資源「実メモリ」においても，資源「実メモリ」を利用する処理の高速化を保持する．

(要件 2)資源操作のインタフェースを統一すること

各管理単位の得失を考慮し，断片化によるメモリ不足を防ぎつつ，TLB ヒット率を向上させるには，走行するプロセスの使用するメモリの大きさに応じて管理単位を切り替える必要がある．この処理の実現を支援するため，各管理単位のインタフェースを統一する．

3.3 設計

3.3.1 識別子

2 種類の管理単位を 4 KB と 4 MB とし，これらの管理単位を持つ資源「実メモリ」の識別子について述べる．

2 種類の管理単位を持つ資源「実メモリ」の資源識別子を**図 4** に示し，以下に説明する．

(1) 資源識別子の同一種類内の通番の最上位ビットを管理単位ビットとし，管理単位を区別する．管理単位ビットが 0 の場合は管理単位が 4 KB であり，管理単位ビットが 1 の場合は管理単位 4 MB である．

(2) 資源識別子の通番の下位 15 ビットを実アドレスに対

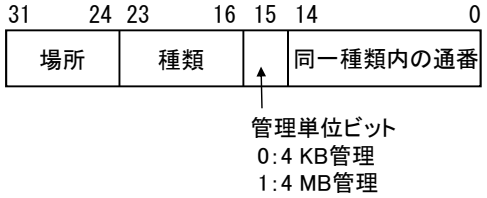

図 4 資源「実メモリ」の資源識別子

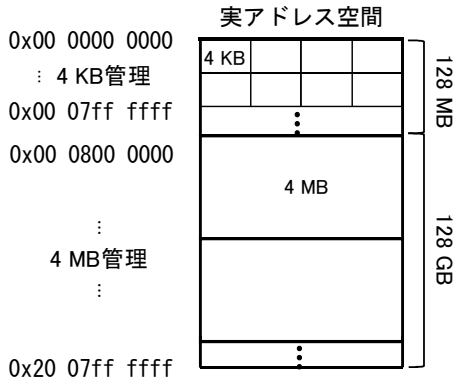

図 5 実アドレス空間

応する情報とする．

また，この資源識別子の構成により管理される実アドレス空間を**図 5** に示す．

(1) 実アドレスが 0 番地から `0x07ffffff`（128 MB）番地までの 128 MB 空間は，管理単位が 4 KB である．

(2) 実アドレスが `0x08000000`（128 MB）番地以降の空間は，管理単位が 4 MB である．この空間の最大サイズは 128 GB である．

上記により，従来と同様に，同一種類内の通番は実アドレスを直接関係づけているため，資源「実メモリ」を利用する処理の高速化を保持でき，（要件 1）を満足できる．具体的には，管理単位ビットが 0 の場合の実アドレスは，通番を 12 ビット左シフトすることで得られる．また，管理単位ビットが 1 の場合の実アドレスは，通番を 22 ビット左シフトしたものに，管理単位ビットが 1 の場合に管理される空間の先頭アドレスを加えた値として得られる．

また，この資源識別子により，2 種類の管理単位を統一して扱うことが可能であり，4 KB の管理単位の操作は従来と同じであり，4 MB の管理単位の操作も同様である．したがって（要件 2）の満足が可能になる．

3.3.2 インタフェース

（要件 2）を満足するために，既存のインタフェースを保存した資源「実メモリ」の OPEN 操作のインタフェースを**表 1** に示し，以下で説明する．資源「実メモリ」の OPEN 操作では，32 ビット整数 `num` は，確保するページ数を示している．そこで，最上位ビットを管理単位のビット（以降，管理単位ビット）とし，0 の場合は管理単位が 4 KB，1 の場合は管理単位が 4 MB であるとする．これにより，4 KB の管理単位の操作は従来と同じである．

表 1 資源「実メモリ」の OPEN 操作のインタフェース

形式	`unsigned int get_pmem(rid, flowid, pmem_common_var, pmem_open_arg)`
引数	`unsigned int rid`： 操作を行う資源の資源識別子 `unsigned int flowid`： 流れ識別子 `void *pmem_common_var`： 実メモリ管理表へのポインタ `struct pmem_open_arg`： 実メモリ管理への引数 `unsigned int num`： 最上位ビット：管理単位 0：管理単位 4 KB 1：管理単位 4 MB 下位 15 ビット：確保するページ数
戻り値	成功：実メモリ資源識別子 失敗：`0xffffffff`
機能	`num` の最上位ビットによって指定された管理単位で，`num` の下位 15 ビットで指定されたページ数の領域の確保を行う．

4. 実現方式

4.1 課題

2 種類の管理単位を持つ資源「実メモリ」の実現にあたり，以下の要求がある．

(要求 1)資源「実メモリ」処理の効率的な処理構造

資源「実メモリ」は，管理単位の種類が 1 つから 2 つに増えても，確保や解放などの処理は共通化して効率的とし，改造量を抑制する．また，さらなる管理単位の種類の実現も容易である処理構造が必要である．

(要求 2)資源「仮想領域」における適切な資源「実メモリ」生成

資源「実メモリ」は，必ず資源「仮想領域」に関連付けられて利用される．したがって，資源「仮想領域」が資源「実メモリ」を生成する際，4 KB と 4 MB をうまく使い分け，両者の得失を生かせる必要がある．

(要求 3)資源「仮想空間」への 4 KB と 4 MB の混在貼り付け処理

OS やプロセスが利用する仮想メモリ空間は，複数の資源「仮想カーネル空間」や資源「仮想ユーザ空間」から構成され，これらは，資源「仮想領域」を資源「仮想空間」に貼り付けることで生成される．一方，資源「仮想領域」には，4 KB 単位の資源「実メモリ」を関連付けたもの，および 4 MB 単位の資源「実メモリ」を関連付けたものがある．ここで，OS やプロセスのメモリ利用を効率化するには，仮想メモリ空間として，4 KB と 4 MB のマッピングが混在できることが望まれる．つまり，資源「仮想空間」が資源「仮想領域」を関連付けて資源「仮想カーネル空間」または資源「仮想ユーザ空間」を作成する際，4 KB 単位や 4 MB 単位の資源「仮想領域」を混在して貼り付けできる必要がある．

これらの要求を満足するには，各要求に対応して以下の各方式を確立する必要がある．

(1) 2 種類の管理単位を扱う資源「実メモリ」の処理方式 (要求 1) を満足するために，資源「実メモリ」の 1 つの

プログラム部品内で 2 種類の管理単位を意識して実メモリを管理する方式である．

(2) 資源「仮想領域」が 2 種類の管理単位を持つ資源「実メモリ」を管理する方式

(要求 2) を満足するために，資源「仮想領域」の情報を用いて，要求された実メモリ量が小さい場合には 4 KB の管理単位で資源「実メモリ」を生成し，要求された実メモリ量が大きい場合には 4 MB の管理単位で資源「実メモリ」を生成することで，管理単位の得失を生かして資源「実メモリ」を管理する方式である．

(3) 資源「仮想空間」が 2 種類の管理単位を持つ資源「実メモリ」を持つ資源「仮想領域」を貼り付ける方式

(要求 3) を満足するために，貼り付け処理に 4 KB 単位と 4 MB 単位でマッピングする機能を実現し，管理単位を意識してマッピングする単位を切り替える方式である．

4.2 対処

4.2.1 想定するアーキテクチャ

想定するアーキテクチャとして，Intel x86 アーキテクチャ（以降，x86 アーキテクチャ）を用いる．x86 アーキテクチャにおいて，4 MB ページを扱うために，Page Size Extention（以降，PSE）を用いる．また，4 GB を超える実メモリを利用するために 36-bit Page Size Extention（以降，PSE-36）を用いる．PSE-36 では，実アドレスは最大 36 ビット幅であるため，使用可能な実メモリは 64 GB に制限される．

4.2.2 2 種類の管理単位を扱う資源「実メモリ」の処理方式

資源「実メモリ」は，ビットマップ方式で実メモリを管理している．ビットマップ方式で 2 種類の管理単位を扱う資源「実メモリ」の処理について**図 6** に示し，以下で説明する．資源「実メモリ」の OPEN 操作では，以下の処理が存在する．

(1) ビットマップ管理表を取得する処理
(2) 管理表内の空き領域を探索する処理
(3) 空き領域を使用中に設定する処理

これらの処理のうち，(2) と (3) については各管理単位で同じ処理を行う．このため，2 種類の管理単位を扱う資源「実メモリ」の実現に必要な処理の変更は (1) のみである．具体的には，各管理単位に応じてビットマップ管理表を用意し，指定された管理単位のビットマップ管理表を取得するのみで良い．これにより，改造量を抑制する．また，さらなる管理単位の種類を実現する際にもビットマップ管理表の種類を増やすのみで良く，実現が容易となる．

4.2.3 資源「仮想領域」が 2 種類の管理単位を持つ資源「実メモリ」を管理する方式

資源「仮想領域」では，**表 2** に示すインタフェースを用いて，仮想領域の大きさを指定して資源「仮想領域」を生成する．そこで，要求された仮想領域の大きさがしきい値 M 以上である場合，管理単位を 4 MB とし，しきい値以下である場合，管理単位を 4 KB とする方式を実現する．例えば，$M = 3.5$ MB とすると，3.5 MB 以上の仮想領域を生成する際に管理単位 4 MB の資源「実メモリ」を生成する．

これにより，使用するメモリ量が小さいプロセスでは管理単位を 4 KB とし，使用するメモリ量が大きいプロセスでは管理単位を 4 MB とするように，管理単位ごとの得失を生かした資源「実メモリ」の生成を行う．

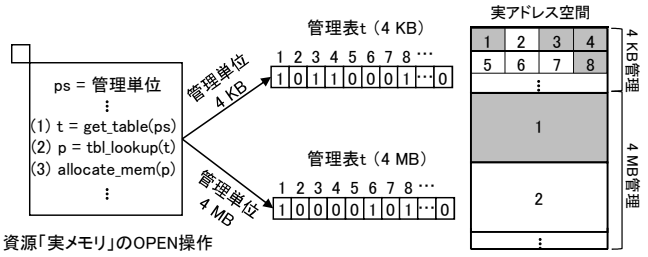

図 6 資源「実メモリ」の OPEN 操作

表 2 資源「仮想領域」の OPEN 操作のインタフェース

形式	`unsigned int create_vr(rid, flowid, vregion_common_var, vregion_open_arg)`
引数	`unsigned int rid`： 操作を行う資源の資源識別子 `unsigned int flowid`： 流れ識別子 `void *vregion_common_var`：仮想領域管理表へのポインタ `struct vregion_open_arg`： 仮想領域管理への引数 　`unsigned int size`： 仮想領域の大きさ [byte] 　`unsigned int mem`： メモリ確保の有無 　`unsigned int dk`： ディスク領域確保の有無 　`unsigned int vr_op`： 予約要否，実メモリ連続指定 　`char *name`： ファイル名（必要な場合のみ）
戻り値	成功：仮想領域資源識別子 失敗：`0xffffffff`
機能	引数 `size` で指定された大きさの仮想領域を作成する．

4.2.4 資源「仮想空間」が 2 種類の管理単位を持つ資源「実メモリ」を持つ資源「仮想領域」を貼り付ける方式

資源「仮想空間」では，ページングを利用して仮想アドレス空間を管理している．**図 7** に，2 種類の管理単位を持つ資源「実メモリ」を用いたページング構造を示し，以下に説明する．

仮想アドレス空間上の任意の 4 MB 境界で管理単位を切り替え可能である．例えば，プロセスが読み込まれる空間には管理単位が 4 KB の実メモリを貼り付け，ワーク領域として使用する空間には管理単位が 4 MB の実メモリを貼り付けることが可能である．この管理単位の切り替えは，ページディレクトリエントリのページサイズビット（PS）の値を変更することで実現する．

管理単位が 4 KB の場合は，仮想アドレスから実アドレスを取得するためにページディレクトリとページテーブルの 2 つのアドレス変換表を必要とする．ページディレクト

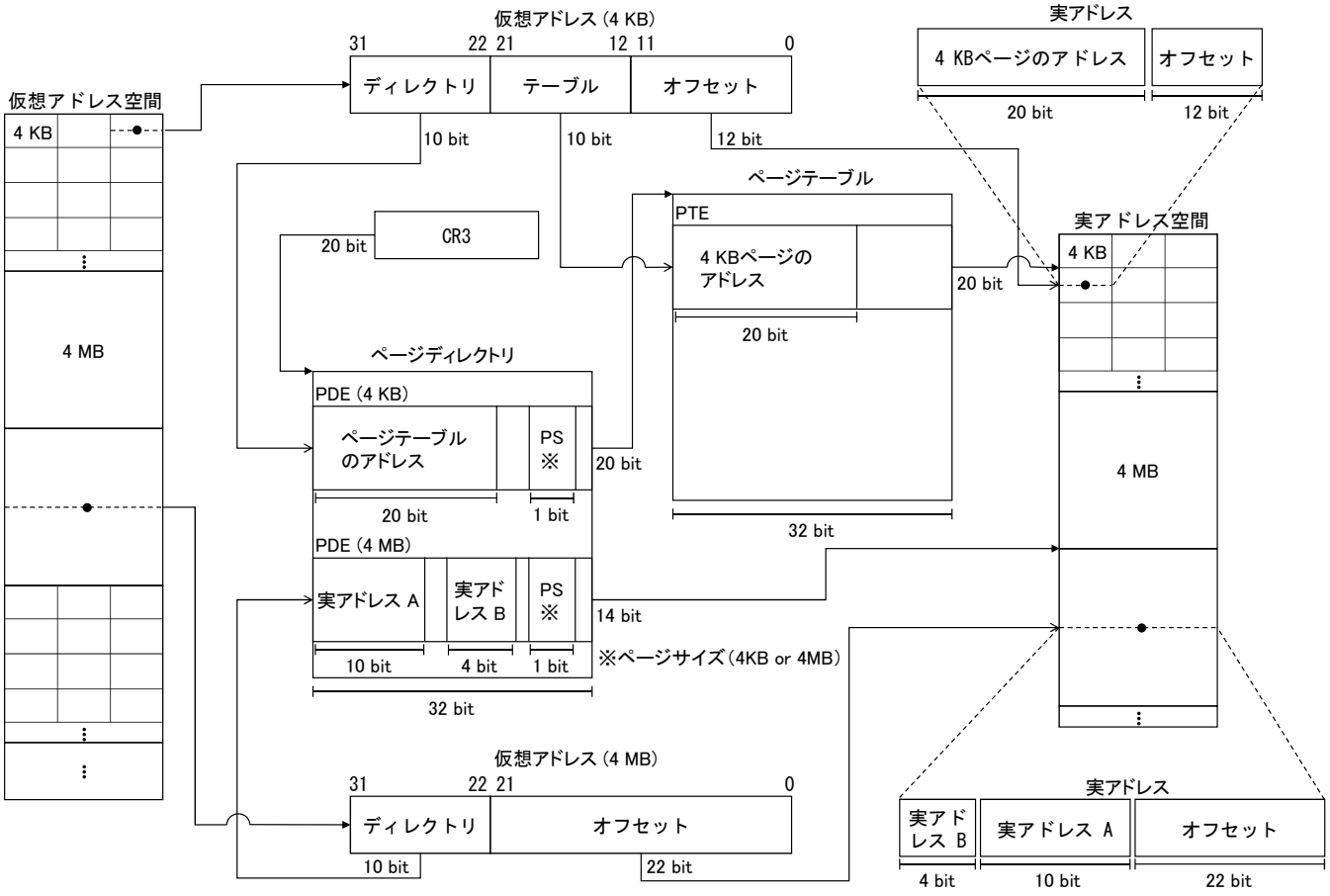

図 7 ページング構造

リは 1,024 個のページテーブルを管理し，ページテーブルは 1,024 個の 4 KB ページを管理する．例えば，4 MB のメモリを使用する際には，少なくとも 1 個のページディレクトリエントリと 1,024 個のページテーブルエントリが必要となる．

一方，管理単位が 4 MB の場合は，仮想アドレスから実アドレスを取得するためにページディレクトリの 1 つのアドレス変換表のみ必要である．ページディレクトリは，1,024 個の 4 MB ページを管理する．例えば，4 MB のメモリを使用する際には，1 個のページディレクトリエントリのみ必要である．これにより，アドレス変換表のエントリ数を管理単位が 4 KB の場合と比較して，削減可能である．

仮想アドレスから実アドレスへの変換の過程を以下に説明する．管理単位が 4 KB の場合，仮想アドレスから実アドレスへの変換は，以下の手順で行われる．

(1) プロセス走行時に設定される CR3 レジスタの値からページディレクトリの先頭アドレスを取得する．
(2) 仮想アドレスの上位 10 ビット（ディレクトリ）を用いて，ページディレクトリエントリ（PDE）のアドレスを取得する．
(3) ページディレクトリエントリからページテーブルの先頭アドレスを取得する．
(4) 仮想アドレスの 21 ビット目から 12 ビット目までの 10 ビット（テーブル）を用いて，ページテーブルエントリ（PTE）のアドレスを取得する．
(5) ページテーブルエントリから 4 KB ページの先頭アドレスを取得する．
(6) 仮想アドレスの下位 12 ビット（オフセット）を用いて，実アドレスを取得する．

一方，管理単位が 4 MB の場合，仮想アドレスから実アドレスへの変換は，以下の手順で行われる．

(1) プロセス走行時に設定される CR3 レジスタの値からページディレクトリの先頭アドレスを取得する．
(2) 仮想アドレスの上位 10 ビット（ディレクトリ）を用いて，ページディレクトリエントリ（PDE）のアドレスを取得する．
(3) ページディレクトリエントリから 14 ビット（実アドレス A と実アドレス B）を用いて，4 MB ページの先頭アドレスを取得する．
(4) 仮想アドレスの下位 22 ビット（オフセット）を用いて，実アドレスを取得する．

このように，管理単位ごとにアドレス変換表の構造を変えることで，2 種類の管理単位を持つ資源「実メモリ」を貼り付ける．

5. 評価

5.1 改造量

2 種類の管理単位を持つ資源「実メモリ」の実現に要した工数を明らかにするために，4.2 節で述べたそれぞれの対処の改造量について評価する．2 種類の管理単位を持つ資源「実メモリ」を実現前後のソースコードについて，ソースコード行数を評価基準として使用し，比較する．

それぞれの対処の改造量を**表 3** に示し，以下に説明する．

(1) 資源「実メモリ」処理

4.2.2 項の対処の実現については，ビットマップ管理表を作成する資源「実メモリ」の初期化処理とビットマップ管理表を扱う資源「実メモリ」の生成処理の改造を行った．新たに追加した処理は，2 つ目のビットマップ管理表を作成する処理と管理単位に応じてビットマップ管理表を切り替える処理の 2 つである．

また，ビットマップ管理表を探索する際に，***Tender*** では前回確保された通番の次の通番から探索を開始することで空き領域の探索を高速化している．この高速化を保持するために各ビットマップ管理表において，前回確保された通番を保存しておくための変数を追加した．

ビットマップ管理表の探索や，空き領域の確保などその他の処理については修正した部分はない．このことから，2 種類の管理単位を扱う資源「実メモリ」の処理方式について実現に要する工数を抑えられたといえる．

(2) 資源「仮想領域」の生成処理

4.2.3 項の対処の実現については，資源「仮想領域」の生成処理の改造を行った．新たに追加した処理は，要求された仮想領域の大きさとしきい値 M を比較し，生成する資源「実メモリ」の管理単位を切り替える処理のみである．その他の処理について修正した部分はない．このことから資源「仮想領域」が 2 種類の管理単位を持つ資源「実メモリ」を管理する方式について実現に要する工数を抑えられたといえる．

(3) 貼り付け処理

4.2.4 項の対処の実現については，資源「仮想空間」の貼り付け処理の改造を行った．新たに追加した処理は，4 MB 単位でマッピングを行う処理と管理単位に応じてマッピングする単位を切り替える処理の 2 つである．また，マッピングを行う際に資源「仮想領域」が持つ資源「実メモリ」の実アドレスを取得する処理がある．この実アドレスを取得する処理について管理単位が 4 MB の場合，通番を 22 ビット左シフトしたものに管理単位が 4 MB で管理される空間の先頭アドレスを加えた値とするように修正した．このように，資源識別子と実アドレスを関係づけることにより，資源「仮想空間」が 2 種類の管理単位を持つ資源「実メモリ」を持つ資源「仮想領域」を貼り付ける方式について実現に要する工数を抑えられたといえる．

表 3 2 種類の管理単位を持つ資源「実メモリ」実現における改造量

処理	改造行数	ファイル数
(1) 資源「実メモリ」処理	55	4
(2) 資源「仮想領域」の生成処理	32	3
(3) 貼り付け処理	53	4

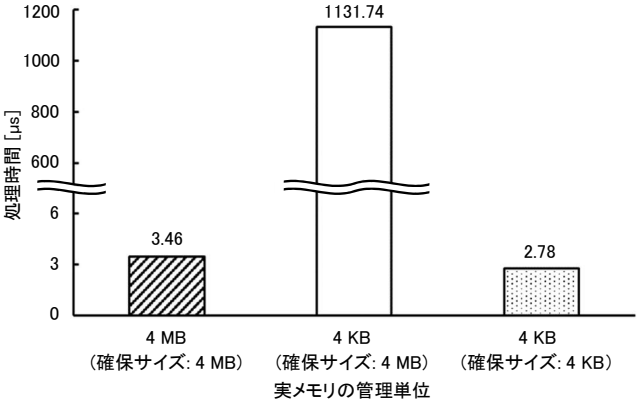

図 8 メモリ確保処理の処理時間

5.2 性能

5.2.1 内容

管理単位が 4 KB の実メモリを扱う場合と，管理単位が 4 MB の実メモリを扱う場合での処理時間の差を示すために，以下の項目について評価する．

(評価 1)メモリ確保処理の処理時間
(評価 2)プロセス生成処理の処理時間
(評価 3)メモリ操作処理の処理時間

(評価 1) では，管理単位の違いによるアドレス変換表の違いがメモリ確保処理の処理時間へ与える影響について示す．(評価 2) では，(評価 1) の効果がプロセス生成処理の処理時間に与える影響について示す．(評価 3) では，管理単位の違いによる TLB ミス回数の違いがメモリ操作処理の処理時間へ与える影響について示す．

本評価については，しきい値 M の値に関係なく管理単位 4 KB の場合，4 KB 単位で実メモリを確保し，管理単位 4 MB の場合，4 MB 単位で実メモリを確保する．

また，評価環境は以下に示す通りである．

- OS： ***Tender*** （シングルコア）
- CPU：Intel Core i7-2600（3.2 GHz）
- メモリ：8 GB

5.2.2 メモリ確保処理

各管理単位で特定サイズ分の資源「実メモリ」を確保し，仮想空間に資源「仮想ユーザ空間」として貼り付ける処理の処理時間を比較する．メモリ確保処理の評価結果を**図 8**

表 4 生成するプロセスのサイズ

通番	プログラム名	テキスト部	データ部	BSS 部
1	prog4	4 KB	4 KB	0 KB
2	prog8	8 KB	8 KB	0 KB
⋮	⋮	⋮	⋮	⋮
n	prog4×n	4×n KB	4×n KB	0 KB
⋮	⋮	⋮	⋮	⋮
50	prog200	200 KB	200 KB	0 KB

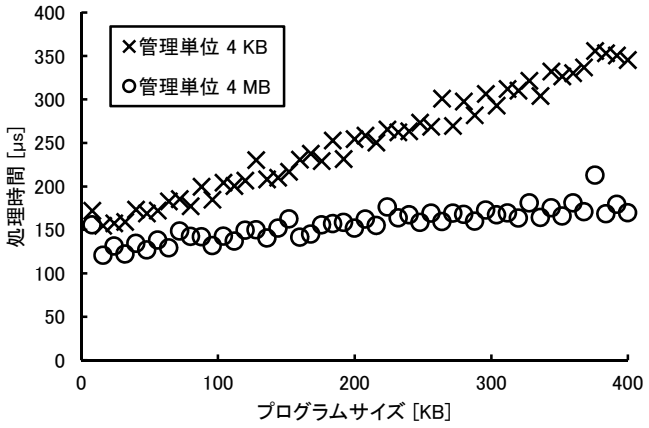

図 9 プロセス生成処理の処理時間

に示し，以下に説明する．

各管理単位で単一の資源「実メモリ」（管理単位 4 KB の場合に 4 KB，管理単位 4 MB の場合に 4 MB）を確保し，貼り付けた際の処理時間は同等（2.8 マイクロ秒≒ 3.5 マイクロ秒）であることがわかる．これは，どちらもアドレス変換表への登録回数が 1 回であることにより，同様な処理となるためである．

また，各管理単位で 4 MB 分の資源「実メモリ」を確保し，貼り付けた際の処理時間は，管理単位が 4 MB の場合に，約 99.7 %程度削減（約 1131.7 マイクロ秒から約 3.5 マイクロ秒）した．

管理単位が 4 KB の場合は，アドレス変換表への登録回数が 1,024 回であるのに対し，管理単位が 4 MB の場合は，アドレス変換表への登録回数が 1 回に減少しメモリ確保処理の処理時間が削減される．また，管理単位が 4 KB の場合は，資源「実メモリ」の生成数が 1,024 個であるのに対し，管理単位が 4 MB の場合は，資源「実メモリ」の生成数が 1 個であるため，資源操作の回数が減少し，メモリ確保処理の処理時間が削減される．

5.2.3 プロセス生成処理

各管理単位で，同一のプロセスを生成する際の処理時間について評価する．プロセスのサイズを**表 4** のように変化させる．また，管理単位が 4 KB の場合は，テキスト部とデータ部にそれぞれ管理単位 4 KB の実メモリを確保し，管理単位が 4 MB の場合は，テキスト部とデータ部にそれぞれ管理単位 4 MB の実メモリを確保する．

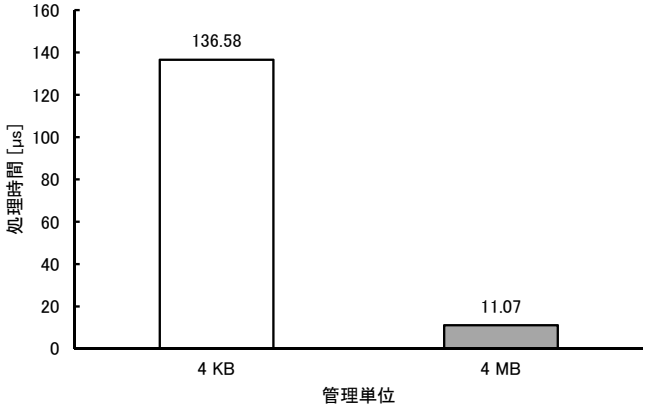

図 10 メモリ操作処理の処理時間

プロセス生成処理の評価結果を**図 9** に示し，以下に説明する．生成するプロセスのサイズが小さい場合，プロセス生成処理の処理時間は管理単位に関わらず同等である．これは，プロセス生成中に確保するメモリのサイズが小さく，メモリ確保処理の処理時間に大きな差がないためである．

生成するプロセスのサイズが大きい場合，プロセス生成処理の処理時間は管理単位が 4 MB の場合に，管理単位が 4 KB の場合と比較して 51 %程度削減（約 350 マイクロ秒から約 170 マイクロ秒）した．

5.2.2 項で示したように，メモリ確保処理の処理時間は管理単位を大きくすることにより，削減可能である．このため，プログラムサイズが大きくなるほどテキスト部とデータ部のメモリ確保処理の処理時間が削減される．

また，プロセス生成処理では，確保したメモリにプログラム内容をコピーする処理を行う．この際，管理単位が 4 MB の場合に管理単位が 4 KB の場合と比較して TLB ヒット率が向上し，処理時間が削減される．

5.2.4 メモリ操作処理

管理単位を大きくしたことによる TLB ミス回数の削減によるメモリ操作処理の処理時間への影響を確かめるために，メモリ操作処理の処理時間について評価する．

メモリ操作処理の評価方法を以下に示す．

(1) 4 MB のメモリ空間へのランダム書き込み（4 B 単位）
(2) ランダム値は，`4096 * ( rand() % 1024 )` のように 4 KB 間隔となるように生成（各管理単位で同じランダム値を使用）
(3) 1,024 回のランダム書き込みの処理時間を測定

メモリ操作処理処理の評価結果を**図 10** に示す．管理単位が 4 KB の場合，4 KB 間隔でメモリ操作されるため TLB ミスは最大 1,024 回発生する．一方，管理単位が 4 MB の場合，TLB ミスは初回メモリ操作時の 1 回のみである．これにより，管理単位が 4 MB の場合に管理単位が 4 KB の場合と比べ TLB ミス回数を削減し，メモリ操作処理の処理時間が 91.8 %程度削減（約 137 マイクロ秒から約 11 マイクロ秒）される．つまり，管理単位が 4 MB の場合に管

理単位が 4 KB の場合と比べて，TLB ミス回数を 1,023 回削減できたとすると，TLB ミス 1 回当たり約 0.12 マイクロ秒処理時間を削減できる．

6. 関連研究

実メモリの管理単位を大きくする機能として，Linux や Windows では，ラージページ（x64 アーキテクチャにおける 2 MB ページ）を確保可能にする機能が存在する [5], [7]．ただし，OS がラージページを意識して管理していないため，ラージページを確保するにはユーザが予めラージページを利用することを意識している必要がある．このため，ラージページ取得時に断片化の影響を受けてラージページが取得できないことがある． ***Tender*** では，このラージページを OS が意識して管理し，割り当てることが可能である．

Linux では，OS が自動的にラージページを割り当てる Transparent Huge Pages[6] という機能も存在する．また，文献 [2], [3] では，この Transparent Huge Pages のラージページ確保と解放のアルゴリズムについて研究されている．また，文献 [4] では，FreeBSD が持つラージページ管理のアルゴリズムについて研究されている．

これらの研究と比較して，2 種類の管理単位を持つ資源「実メモリ」では，管理単位ごとに独立した実メモリ空間を管理している．このため，ラージページの確保に断片化の影響を受けない．これにより，断片化を解消するためのメモリコンパクションを行う必要がない．また，2 種類の管理単位を持つ資源「実メモリ」では，1 GB などの非常に大きな連続領域を必要とするラージページを管理することが容易である．ただし，2 種類の管理単位を持つ資源「実メモリ」では，すべての実メモリ空間でラージページを確保可能にする場合と比較して，多量の実メモリが必要となる問題がある．ただし，今後も計算機が搭載する実メモリ量は増加することが考えられるため，この問題による影響は小さいと考える．

7. おわりに

実メモリの管理単位を 2 種類持つ資源「実メモリ」の設計と実現方式について述べた．管理単位を 2 種類持つ資源「実メモリ」を利用して仮想アドレスと実アドレスの変換表のエントリ数を削減するには，資源「仮想空間」へ貼り付ける処理に変更が必要である．このため，貼り付ける処理において，管理単位が 4 MB の実メモリの場合にはページサイズを 4 MB としてページング構造を設定する．また，貼り付ける処理において，実メモリを貼り付ける際には資源「仮想領域」が用いられるため，資源「仮想領域」において，要求メモリサイズのしきい値によって管理単位を切り替えて資源「実メモリ」を生成するよう変更を加えた．

評価において，管理単位を 4 MB に設定した際に，メモリ確保処理の処理時間を最大で 99.7 %程度削減できることを示し，プロセス生成処理の処理時間を最大で 51 %程度削減できることを示した．また，メモリ操作処理において，TLB ミス 1 回当たり約 0.12 マイクロ秒処理時間を削減できることを示した．

残された課題として，管理単位の違いがプログラム実行の処理時間に与える影響の調査と，管理単位を使い分ける手法の実現がある．

謝辞 本研究の一部は，共同研究（株式会社富士通研究所）による．

参考文献

[1] 谷口 秀夫, 青木 義則, 後藤 真孝, 村上 大介, 田端 利宏, “資源の独立化機構による ***Tender*** オペレーティングシステム ,” 情報処理学会論文誌，Vol.41, No.12, pp.3363–3374 (2000).

[2] Y. Kwon, H. Yu, S. Peter, C.J. Rossbach, and E. Witchel, “Coordinated and Efficient Huge Page Management with Ingens,” in Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI ’16), pp.705–721 (2016).

[3] A. Panwar, S. Bansal, and K. Gopinath, “HawkEye: Efficient Fine-Grained OS Support for Huge Pages,” in Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS ’19), pp.347–360 (2019).

[4] W. Zhu, A.L. Cox, and S. Rixner, “A Comprehensive Analysis of Superpage Management Mechanisms and Policies,” in Proceedings of the 2020 USENIX Annual Technical Conference (USENIX ATC ’20), pp.829–842 (2020).

[5] Persistent Huge Pages in Linux. ⟨`https://www.kernel.org/doc/Documentation/vm/hugetlbpage.txt`⟩ (accessed 2021-02-08).

[6] Transparent Huge Pages in Linux. ⟨`https://www.kernel.org/doc/Documentation/vm/transhuge.txt`⟩ (accessed 2021-02-08).

[7] Windows Large Page Support. ⟨`https://docs.microsoft.com/en-us/windows/win32/memory/large-page-support`⟩ (accessed 2021-02-08).