システムソフトウェアと
オペレーティング・システム 77－15
マルチメディア通信と分散処理 87－15
（1998. 2. 26）

仮想記憶のバッキングストレージの効率的制御

石井秀浩* 高野陽介 黒岩実 横田実
NEC C&Cメディア研究所

ビデオ、セットトップボックス、PDA等小規模なコンピュータを内蔵した機器の普及とマルチメディアやネットワーク接続といった処理の高度化により、ハードウェア資源の乏しい小規模なコンピュータにおいてもより高度な処理を行うためのオペレーティングシステム技術の重要性が高まりつつある。筆者らは、RT-Machマイクロカーネルをベースとして、このような小規模ながらもある程度高機能なコンピュータのためのOSの研究を進めている。本論文では、Machマイクロカーネルの仮想記憶システムの弱点を明らかにすると共に、この弱点を克服して、さらにアプリケーションがより潤沢に主記憶やそのバッキングストレージを容易に有効活用できる“uncommit”と言われる機能をRT-Machに持たせた事の設計と実装を報告する。

Efficient Control of Virtual Memory's Backing Storage

Hidehiro Ishii Yosuke Takano Minoru Kuroiwa Minoru Yokota
C&C Media Labs, NEC Corporation

Since machines equipped with small scale computers such as VCR, STB, or PDA are getting popular, and they demand richer and richer processing such as multimedia and network connection, importance of operating system technology, which enables applications to do richer processing on such small scale computers of poor hardware resources, is increasing. We are researching such operating system technologies for small scale but richly functional computers based on RT-Mach microkernel. In this paper, we make a weak point of Mach microkernel's virtual memory system clear at first, then report ideas and implementations which overcome the weak point and moreover add “uncommit” function which enables user programs efficiently use main memory and swap area with ease.

1 はじめに

しばらく以前からテレビ、ビデオ、エアコン等のあらゆる機器にコンピュータが入り込んでいて、また近年は、小型ながらもそれらより高度なコンピュータ機器（PDAやセットトップボックス等）も普及してきている。今後は身の回りのあらゆる機器が小型のコンピュータを内蔵していくと考えられ、またそれらの機器は、ネットワークに接続されたりマルチメディア機能を持つなどして、より高度になって行くと思われる。このような状況から、比較的ハードウェア資源の乏しい小型のコンピュータでより高度な処理を行なえるようなオペレーティングシステム技術の重要性が増しつつある。現在の多くの組み込みシステムでは、ハードウェア資源が乏しいために、仮想記憶もないような非常に小さなオペレーティングシステムを利用している。しかし、マルチメディ

*E-mail: ishii@ccm.cl.nec.co.jp

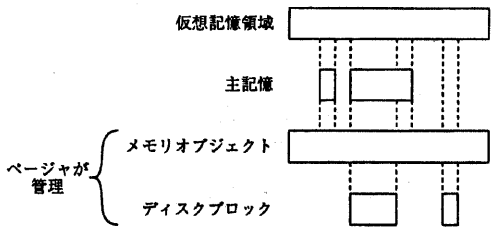


図 1: メモリオブジェクト

アやネットワークの機能が求められるような分野では、必ずしもそのようなオペレーティングシステムで十分とは言えない。そこで筆者らは、マルチメディアやネットワークのようなある程度高度な機能を要する小型のコンピュータ向けのオペレーティングシステム技術の研究を行なっており [黒岩 96]、仮想記憶とリアルタイムをサポートしている RT-Mach マイクロカーネルをそのベースとして採用している。

この一連の研究の中で筆者らは RT-Mach の仮想記憶システムの改良に取り組んでおり [Ishii96, 石井 97a, 石井 97b]、本論文では、主記憶とバッキングストレージの効率的利用のため、仮想領域に対応する主記憶とそのバッキングストレージの解放を任意に行なえるようにした改良について報告する。

2 問題と目的

2.1 Mach の仮想記憶の問題

Mach においては、タスク等の仮想記憶空間内に割り当てられた領域のバッキングストレージを「メモリオブジェクト」と呼ぶ (図 1)。概ねメモリオブジェクトは、連続した仮想領域ごとに一つずつ生成される。メモリオブジェクトは、外部ページャと呼ばれるプログラム[1]が提供し、その実体はスワップ領域等に置かれる。主記憶はメモリオブジェクトのキャッシュとして扱われる。

仮想領域の割り当ては、`vm_allocate()` というカーネルコールで行なう。このカーネルコールは仮想空間内の任意の位置に任意の大きさの領域を割り当てる事ができる。このカーネルコールで割り当てた仮想領域に対応するメモリオブジェクトは、「デフォルト・ページャ」と呼ばれる外部ページャが管理する。

[1]メモリマネージャとも言う。通常はユーザモードのタスクとして実現される。

Mach は、`vm_allocate()` などによって非永続的なメモリオブジェクトが必要になると、デフォルト・ページャにメモリオブジェクトの生成を要求する。メモリオブジェクトの内容を記憶するディスク・ブロックの割り当ては、ページアウト時にオンディマンドで行なわれる。

仮想空間内の任意の領域の解放は、`vm_deallocate()` というカーネルコールで行なう。`vm_deallocate()` は、以下の処理を行なう。

- 仮想空間内の当該領域に対応するページ・テーブル・エントリを無効化する。
- 仮想空間内の当該領域からそれに対応するメモリオブジェクトへの参照を削除する。
- その際、そのメモリオブジェクトへの参照が一切なくなったら、メモリオブジェクトを削除する。

問題は、メモリオブジェクトの内容を記憶しているディスク・ブロックの解放が、メモリオブジェクト自体が削除される時にしか行なわれない事である。しかも Mach における仮想領域の割り当ては、処理の高速化のために、特に指定しない限りは割り当て済みの領域とできるだけ隣接するように行なわれ、また隣接する領域同士は原則として一つにまとめられて一つのメモリオブジェクトに統合される[2]。統合されてしまうと、片方の領域が完全に解放された後も、統合されたすべての領域のすべての部分が解放されるまで、当該メモリオブジェクトが占めるディスク・ブロックは一切解放されない。このように、メモリオブジェクトとそれが占有するディスク領域は大きくなる一方で、その解放は結局タスク自体が終了するまではあまり行なわれないことが多い。

2.2 本研究の目的

本研究では、以上の問題を解決し、またさらにユーザプログラムの便宜を増すため、仮想記憶領域に対応する主記憶とバッキングストレージを任意に解放する機能を Mach に追加する。具体的には、Mach から派生した MKng プロジェクト版 RT-Mach[3] において、`vm_deallocate()` で主記憶とバッキングスト

[2]ただし、領域同士の保護モードが異なるなどの事情がある場合には、まとめられない。

[3]RT-Mach は、Mach にリアルタイム機能を追加したものである。MKng プロジェクト (次世代マイクロカーネル研究開発プロジェクト) は、(社) 情報処理振興事業協会の出資による、慶應義塾大学を中心とした産学共同プロジェクトである。

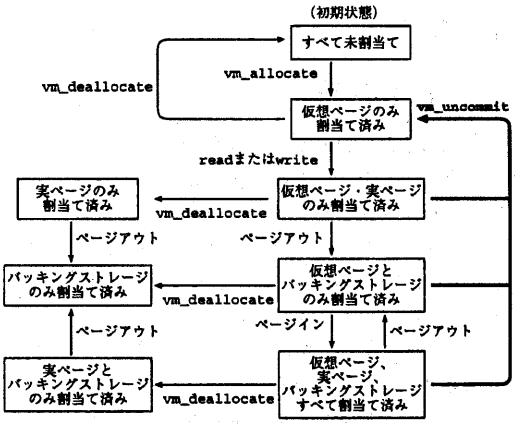


図 2: 仮想ページの状態遷移

レージも解放するようにする改良と、3.2 節で述べる "uncommit" 機能の追加を行なう。

3 仮想記憶の改良の方針

3.1 vm_deallocate() の改良

2.1 節に示したように、従来の`vm_deallocate()` カーネルコールは、主記憶とバッキングストレージを解放しない場合が多い。そこで、`vm_deallocate()` の結果参照されなくなったこれらの領域は即時解放するようにする。

3.2 vm_uncommit の追加

3.2.1 vm_uncommit の機能

仮想ページとそれに対応する実ページとバッキングストレージの状態遷移を図 2 に示す（図中の`vm_deallocate` は、3.1 節による改良を行なわないバージョンの動作である）。`vm_uncommit()` カーネルコールは、指定の仮想領域に対応する主記憶ページとそのバッキングストレージを解放する。呼び出しの形式は、次の通りである。`target_task` は対象とするタスク、`address` は仮想領域の開始アドレス、`size` は仮想領域のバイト数である[4]。

```
kern_return_t
vm_uncommit(
    mach_port_t  target_task,
    vm_address_t address,
    vm_size_t    size);
```

ただし、`vm_deallocate()` とは異なり、ユーザタスクは当該仮想領域を引き続き利用する事ができる。`vm_uncommit()` の後に書き込むと、その時点で実ページが、そしてそのページのページアウト時にディスク・ブロックが、再度割り当てられる。

指定の仮想領域内に copy on write で共有している部分がある場合は、例外的な処理が必要となる。あるタスクの当該領域に対する操作は、他のタスクの当該領域になんら影響を与えてはならないので、`vm_uncommit()` の処理においても、copy-on-write で共有している主記憶やバッキングストレージは解放しない事とする。

3.2.2 vm_uncommit の有用性

ユーザから見れば`vm_uncommit()` は、当該領域を`vm_deallocate()` してすぐに同じ領域を指定して `vm_allocate()` したのと似ている。ただし`vm_deallocate()` と`vm_allocate()` を組み合わせた操作で`vm_uncommit()` の代替としようとすると、次のような問題がある。

1. 前述のように、従来の`vm_deallocate()` の実装では主記憶とバッキングストレージの解放は行なわれないことが多い。

2. 当該領域の`vm_allocate()` が成功する保証はない。`vm_deallocate()` してから`vm_allocate()` するまでに当該仮想領域がたまたま他の目的に割り当てられてしまうと、`vm_allocate()` が失敗してしまう。このような事は、他のスレッドが `vm_allocate()` をアドレス指定無しで呼び出してこの領域が割り当てられてしまったり、IPC のメッセージが当該タスクに届いてマイクロカーネルがメッセージのバッファをこの領域に割り当ててしまったりして起こる危険がある。

3. マルチスレッドのタスクにおいては、`vm_deallocate()` してから`vm_allocate()` するまでの間に他のスレッドが当該領域にアクセスすると、ページ保護違反のエラーになってしまう。

[4]正確には、`address` から`size` バイトの仮想領域がかかるすべての仮想ページが対象となる。つまり、`address` から`size` バイトの仮想領域がページ境界ちょうどになっていなければ、その前後を伸ばしてページ境界に合わせた領域が対象となる。この丸め方は、`vm_deallocate()` の丸め方と同じである

1. は、本研究のvm_deallocate() の改良によって解決できる。2. は、vm_allocate() とvm_deallocate() を組み合わせている限り解決困難である。3. は、ロックなどのメカニズムの導入で解決可能な場合が多いと思われるが、そうすると余計なオーバヘッドを背負い込むことがある。vm_uncommit() は、これらの問題をすべて解決する。

vm_uncommit() は、ユーザタスクがヒープ領域からのメモリ割り当てを管理する場合に有用である。例えば従来の Mach 上のユーザタスクの典型的なメモリ割り当てスキーム（malloc() 関数や free() 関数など）は、次のように動作する。

1. ヒープ領域をまとめてvm_allocate() してプールとし、上位モジュールから割り当ての要求があるとこれを小分けして割り当てる。

2. ヒープ領域が足りなくなると、さらにvm_allocate() を呼んでヒープ領域を拡大する。

3. 上位モジュールから割り当て済み領域の解放要求があると、当該領域をプールに戻す。OS には返さない。

vm_uncommit() を使うと、例えば 3. を次のように変更して不要な主記憶やそのバッキングストレージを活用するようにできる。

3. 上位モジュールから割り当て済み領域の解放要求があると、当該領域をプールに戻し、対応する主記憶とバッキングストレージを解放する。

典型的なメモリ割り当てスキームでは、余剰のメモリを抱え込んでシステムの主記憶とそのバッキングストレージを圧迫するため、システム全体の性能を抑圧したり、場合によっては領域不足のエラーを誘発する事があるが、このようにvm_uncommit() を利用することにより解決できる。

ちなみに、vm_uncommit() で主記憶やバッキングストレージを解放しても、当該仮想領域への再度のアクセスによってそれらが自動的に改めて割り当てられるため、どの領域をvm_uncommit() したかを覚えておくなどの繁雑な処理はユーザタスクには必要ない。

4 実装

4.1 動作の概要

今回はvm_uncommit() の実装を行なった。カーネルやその他の部分は、vm_uncommit() に関して図 3 のようなインタラクションを行なう。

ユーザプログラムがライブラリ（libmach または libmach_rt）の関数vm_uncommit() を呼び出すと、ライブラリはこれをマイクロカーネルに転送する（vm_uncommit() カーネルコール）。マイクロカーネルはこれを受けて、指定領域に対応するページ・テーブル・エントリを無効にし、対応する主記憶ページを解放し、メモリオブジェクト内の対応する部分の解放をページャに要求する（新設の memory_object_uncommit() RPC）。ページャは、メモリオブジェクトの指定部分を記憶しているディスク・ブロックを解放し、処理の完了をカーネルに通知する（新設の memory_object_uncommit_completed() RPC）。

4.2 カーネル — ページャ間プロトコル

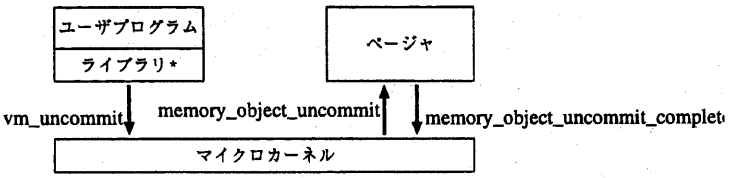
カーネルからページャへのプロトコルの非同期 RPC として、次のものを追加する。

```
kern_return_t
memory_object_uncommit(
    mach_port_t memory_object,
    mach_port_t memory_control,
    vm_offset_t offset,
    vm_size_t   size);
```

この RPC は、メモリオブジェクト内の任意の部分で使用しているディスク・ブロックを解放する要求である。ここで、memory_object は対象とするメモリオブジェクト、memory_control は完了通知の宛先（memory cache control port と言う）、offset は解放すべき部分の開始オフセット、size は解放すべき部分のサイズである。この要求を受けると、ページャは指定部分に対応するディスク・ブロックを解放し、完了したらmemory_control 宛に下記のmemory_object_uncommit_completed() を呼び出す事によって処理の終了を通知する。

また、ページャからカーネルへの非同期 RPC（カーネルコール）として、次のものを追加する。

```
kern_return_t
memory_object_uncommit_completed(
    mach_port_t memory_control);
```



*上記ライブラリは、libmach または libmach_rt

図 3: モジュール間のインタラクション

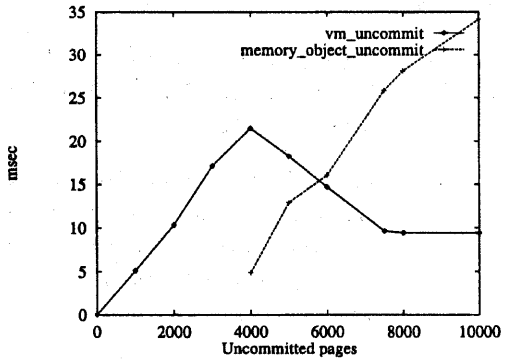


図 4: 処理時間

このRPCは、`memory_object_uncommit()` 要求の処理の完了通知である。ここで `memory_control` は、`memory_object_uncommit()` で完了通知の宛先として指定したポートである。

5 性能評価

ユーザタスクでテスト・プログラムを実行し、`vm_uncommit()` の処理にかかるコストとして下記の処理時間（経過時間）を測定した。

1. `vm_uncommit()` カーネルコールの処理時間
2. `memory_object_uncommit()` RPC の処理時間

測定に使用したテストプログラムは、自らの仮想空間内に仮想領域を確保（`vm_allocate()`）して、その中の初めのページから順に全ページに書き込みを行ない、その後初めの半分のページを解放（`vm_uncommit()`）する。試行のたびに解放するページ数を1ページから10000ページまで変化させ（先に確保・書き込みするページ数はその2倍の2〜20000ページ）、ページ数と処理時間の関係をグラフにした（図4）。ページサイズは4Kバイトで、測定に使用したターゲットマシンはDEC社のHiNote Ultra II (150MHz Pentium、48MB RAM）である。

ちなみに、マイクロカーネルはページャに対して `memory_object_uncommit()` の要求を送るが、これは非同期RPCにしてあるので、ページャの処理を待たない。そしてカーネル内の処理が終了した時点で、ユーザタスクは処理を再開できる。

以下、解放を要求するページ数（グラフの横軸の値）を p とする。`vm_uncommit()` の処理時間を見ると、

- p が1ページから約4000ページまででは、ページアウトが発生せず全ページ（$2p$）が解放時にも主記憶上に存在したため、`vm_uncommit()` の処理時間は p にほぼ比例して増加した。
- p が約4000ページを越えると主記憶が溢れて、確保した仮想領域の後ろのページに書き込むにつれて初めの方のページからページアウトされて行くため、すべての書き込みが終了して解放する時には、解放するよう指定されたページ（初めの半分）のうち主記憶に留まっているものの数は、p の増加につれて減少する。従って `vm_uncommit()` の処理時間も p の増加に従って減少した。
- p が約7000ページになると、解放すべきページが解放時には主記憶に残っていないため、p が増えても `vm_uncommit()` の処理時間は一定となった。

主記憶上に存在するページ辺りの `vm_uncommit()` の処理時間は、図4からおよそ22m秒/4000ページ＝5.5μ秒/ページと言える。

一方、`memory_object_uncommit()` の処理時間を見ると、

- pが約4000ページ未満の場合については、ページアウトが発生しなかったためにメモリオブジェクトが生成されず、よってページャへの要求が発生せず、ページャの処理時間は測定されなかった。
- pが約4000ページを越えると、pの増加に連れてページアウトされるページが増加し、同時に`memory_object_uncommit()`の処理時間も増加した。

メモリオブジェクト内のページの増加に伴う`memory_object_uncommit()`の処理時間の増加率は、概ね$(34-5)$m秒/$(10000-4000)$ページ $= 4.8\mu$秒/ページと言える。

6 筆者らの他の研究との組合せ

第1章で触れた筆者らの研究のうち、タスク間のスワップ領域の融通の研究[Ishii96, 石井97a]では、記憶領域の解放に従来の`vm_deallocate()`カーネルコールしか使えなくてもタスクがスワップ領域の一部を他のタスクに任意に譲れるように、mallocライブラリが記憶領域を確保(`vm_allocate()`)する際に複数の領域が一つのメモリオブジェクトにまとめられてしまわないよう、少々不自然で非効率な実装をせざるを得なかった。しかしこれに本研究の改良版`vm_deallocate()`または`vm_uncommit()`を適用すれば、より自然でより効率良く、スワップ領域のタスク間での融通を実現できる。

また筆者らのオンメモリ圧縮ページャ[石井97b]でも、その発表時のプロトタイプでは、ページャが自身の仮想領域の割り当てを保ちながら主記憶を解放する際に、`vm_deallocate()`と`vm_map()`を組み合わせており、効率と信頼性を少々損ねている。これにも本研究の`vm_uncommit()`を適用すれば、効率と信頼性を得る事が出来る。

7 おわりに

本論文では、Machの仮想記憶システムの弱点を示すと共に、その解消と"uncommit"機能の実装と性能評価を報告した。この改良により、従来より効率良く仮想記憶のバッキングストレージを利用する事ができる。

筆者らの研究の主目的は小型にして高機能なコンピュータに向いたオペレーティングシステムにあるが、本論文で報告した内容はデスクトップ・マシンやサーバ・マシン等の規模のシステムにも適用できる。

謝辞

最後に、様々な情報提供とご指導をくださった慶應義塾大学徳田英幸教授を初めとするMKngプロジェクトのメンバーの皆さんに感謝致します。

参考文献

[Beradat93] P. Bernadat et al. "Real Memory Mach", In *Proceedings of the USENIX Mach Workshop*, 1993

[Ishii96] H. Ishii et al. "Surviving Memory Exhaustion —Memory collection scheme for better efficiency and reliability—", In *Proceedings of RT-Mach Workshop 1996*, Aug. 1996

[Rashid87] R. Rashid et al. "Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures", In *Proceedings of ASPLOS II*, Oct., 1987

[Young87] M. Young et al. "The Duality of Memory and Communication in the Implementation of a Multiprocessor Operating System", In *Proceedings of the 11th Symposium on Operating Systems Principles*, Nov., 1987

[石井97a] 石井他"メモリを高効率で活用するメモリ回収機構" 情報処理学会第54回全国大会講演論文集, 1997年3月

[石井97b] 石井他"MKngプロジェクトにおける組み込みシステム技術:Machマイクロカーネル上のオンメモリ圧縮ページャの設計と実装" 情報処理学会第55回全国大会講演論文集, 1997年9月

[黒岩96] 黒岩他"MKngプロジェクトにおける携帯端末サポート" 情報処理学会第53回全国大会講演論文集, 1996年9月