システムソフトウェアと
オペレーティング・システム 64−13
マルチメディア通信と分散処理 65−13
（１９９４．５．１９）

連続メディアオブジェクトに対応したメモリ管理機構について

盛合 敏, 木原 誠司, 南部 明

NTT 情報通信網研究所

本論文では，連続メディアオブジェクトを扱うためのメモリ管理機構について述べる．連続メディアオブジェクトを取り扱うためには，大規模なメッセージのリアルタイム通信を効率良くサポートしなければならず，そのためには，メモリ管理機構において，メッセージの移動のためのコストが小さく，意味を失った古いメッセージを次々と捨てる機構が必要である．さらに，QOS を保証するため，予約された資源量が常に利用可能でなければならない．

従来のメモリ管理機構は，これらの要求をすべて満たすには至っていない．そこで，仮想リングバッファ機構とそれを RT-Mach 上でサポートするためのメモリ管理プリミティブを提案する．このプリミティブは高速なネットワーク経由の通信を実現するために有効であることを実験的に示す．

An Extension of Mach Memory Management System for Continuous Media Objects

Satoshi Moriai, Seiji Kihara, Akira Nambu

{moriai,kihara,nambu}@nttbss.ntt.jp
NTT Network Information Systems Laboratories,
Yokosuka-shi, Kanagawa, 238-03, Japan.

In this paper, we describe memory management mechanisms for continuous media objects. An efficient communication facility which crosses protection domains contributes high throughput and low latency of the system. Resource management functions for it enable the system to guarantee QOS of continuous media. It is important to avoid copying messages, to discard out-of-date messages, and to make available the reserved resources. Current memory management systems do not satisfy the conditions mentioned above.

We propose a virtual ring buffer mechanism and some extensions of the RT-Mach virtual memory management system. These mechanisms are suitable for high performance communication on high speed networks.

1 はじめに

分散システム上の連続メディアのアプリケーションは，時間軸上の制約やサービス品質 (QOS: quality of service) の要求を満たしながら，システム上にある様々なエンティティと高速なデータ通信を行う．このためには，遅延時間が小さく，スループットの高いプロセス間通信機構と，QOS を保証するための資源管理システムが必要となる．この分野において多くの研究がなされているが，大半は保護ドメインをまたぐ通信の遅延に主眼がおかれ，スループットや QOS を保証する機構についての研究は多くはない．

著者らは，連続メディアの処理に必須のリアルタイム処理機能を持ったオペレーティング・システムとして Real-Time Mach 3.0 をベースとした連続メディア・ソフトウェア・プラットホームを研究開発中である．本論文では，連続メディアオブジェクトをこのプラットホーム上で扱うための機構のうち，保護ドメインをまたぐ通信機構とそれをサポートする仮想記憶管理機構について述べる．

本論文では，連続メディアオブジェクトを Mach あるいは RT-Mach におけるメモリオブジェクトのように，保護されたメモリ空間の領域と考える．連続メディアオブジェクトの実現モデルには，

- 時刻の経過に従って，メモリの内容が刻々と変化する (Realtime Projection Model) [15].
- 時刻を仮想アドレス上にマッピングし，時刻の経過に従って，アクティブな仮想アドレスが刻々と移動していく [16].

の 2 つが考えられる．前者はビットマップディスプレイにおけるフレームバッファやビデオカメラにおける CCD のアナロジーであり，後者はビデオディスク等のアナロジーである．多様な実世界を表現するためには，いずれのモデルも取り扱える必要があり，メモリ管理機構が重要な役割を担うと考えられる．

連続メディアオブジェクトは，遠隔のシステムとの通信のみならず，ローカルなデバイスとの通信を含め，常に通信の上に成り立つオブジェクトであり，そのメッセージの内容はある特定の時刻において意味を持ち，時機を逸した場合には無意味となってしまうという特徴がある．加えて，動画像のような大規模データを効率良く取り扱わなければならない．すなわち，メモリ管理機構は連続メディアオブジェクトを支えるための大規模なメッセージのリアルタイム通信を効率良くサポートしなければならない．さらに，受信されていても意味を失った時間的に古いメッセージを次々と捨てる機構が必要となる．

2 関連研究

2.1 メモリ管理機構における大規模メッセージ通信のサポート

メモリ管理機構において，大規模メッセージ通信を効率良くサポートする方法については，これまで多くの研究がなされてきた．おおまかには，

- 共有メモリを用いることでメッセージのコピーを減らす
- 仮想記憶のマッピングの書き換えによってメッセージのコピーをさける
- 両者を組み合わせる

の 3 つに分けられる．

共有メモリを用いる方法は，あらかじめ共有メモリ空間を設定しておくことで，効率の良いメッセージの転送が可能である．これには，全てのプロテクション・ドメインから読み書き可能な共有メモリ空間を置く方法 [18] と，特定のプロテクション・ドメインの組みに対して共有メモリ空間を置く方法 [5] がある．いずれの方法を用いても，この共有メモリ空間は他のプロテクション・ドメインからも読み書き可能となるため，セキュリティー上の問題がでてしまう．これを解決するためには，コピーを行うしかない．また，通常の保護された空間でのデータ処理を行うためにも，コピーが行われることになる．

この方法は，効率の良いネットワーク・デバイス・ドライバを作成するために重要なテクニック [19] [7] であるが，アプリケーションまでを共有メモリとするためには，セキュリティー上の危険を犯すか，単一のアプリケーションのみにそのデバイスのアクセスを許すという不自由さに甘んじることになる．

仮想記憶のマッピングの書き換えによる方法として，V [6] や DASH [21] 等でサポートされている page remapping を用いる方法と，Mach [1] [17] でサポートされている copy-on-write を用いる方法がある．page remapping では，メッセージを転送した後，そのメッセージはもはや転送元からは見えなくなる．copy-on-write では，メッセージの転送後にその領域への書き込みが起こるとページの複製が行われるため，通常のコピーと同じセマンティクスを持つことになる．このため，共用メモリでのデータの保護の問題は生じない．しかし，これらを実現するためには，共有メモリと比較して複雑なプログラミングが必要であり，TLB やキャッシュのエントリの無効化の影響 [10] もあって，共有メモリを利用してコピーを行ったほうがコストが小さい場合がある．特に，Mach の copy-on-write はコストが大きい [3].

すなわち，仮想記憶のマッピングの書き換えによる場合は，抽象度の高い汎用的な方法ではなく，メッセージ通信に最適化することが望ましい．Peregrine RPC [11] では，RPC に特化することで遅延時間を小さく抑えることに成功している．また，Fbufs 機構 [8] や temporary mapping [13] によるメッセージ転送機構は，page remapping と共有メモリを組み合わせることで高いメッセージの転送効率と保護された空間を提供している．

2.2 メモリ管理機構における連続メディアオブジェクトのサポート

これまで述べてきた方法は，メモリ管理機構において大規模なメッセージ通信をサポートものであるが，連続メディアオブジェクトを実現するためには，資源の予約管理 (資源のスケジューリング) を行い，QOS を保証する必要がある．

共有メモリを利用して連続メディアのストリームを扱うものとして，Memory-mapped stream 機構 [9] があるが，積極的に QOS を保証するには至っていない．Fbufs 機構 [8] を連続メディアオブジェクトへの適用も考えられるが，複数の連続メディアを同時に扱う場合，QOS 制御が必要である．QOS 制御を行うためには，セッションの概念を持った予約制御 [2] [15] が不可欠である．また，page fault に基づくメモリ管理では，挙動の予測が難しくなるため，連続メディアを扱う上では，page fault が起きないようにしなければならない．さらに，時間切れとなったメッセージは処理がなされていなくても，積極的に廃棄することで常に予約された資源量が確保されていなければならない．

3 連続メディアオブジェクトのための仮想リングバッファ機構

RT-Mach は Mach 3.0 と同様のメモリ管理機構を持ち，copy-on-write といった遅延評価や page fault を契機として働く external pager を基礎としている．これは，寿命の長い大きなオブジェクトには向いているが，連続メディアオブジェクトには適していない．つまり，寿命の短いメッセージのためには高価であり，いつ page fault が起きるか分からないため，QOS の保証が難しくなる．また，copy-on-write は共有したページに書き込みがない限りにおいてコピーを避けるものであり，不要になったページの再利用の機構としては使いにくい．

このため，RT-Mach 上で連続メディアのアプリケーションを作成する場合は，仮想ページを物理ページにくくりつけるか (wire-down)，連続メディア・メモリ・マネージャ [20] を用い，page fault がおきないように時間の進行にともなって次々と仮想ページを物理ページにマッピングしていく，などの手法を用いることになる．しかし，これらの機構は，先に述べたコストの問題，時間切れメッセージの廃棄の問題を解決するものではない．本論文では，仮想リングバッファ機構とそれをサポートする機構を提案する．

仮想リングバッファ機構は，リングバッファ構成により，予約された資源量を確保し，時間切れとなったメッセージを積極的に廃棄する．また，次節以降で述べる RT-Mach に追加したプリミティブを用いて，共有メモリ上での page remapping により効率良くメッセージの転送を行う．これは，ST-II プロトコルサーバ [12] への適用をひとつの目的として開発している．

同一のプロテクション・ドメイン内にある連続メディアオブジェクトのストリームのための仮想リングバッファを実現するには，全ての仮想アドレス空間が共有されていることを利用できる．仮想リングバッファ上の読み書きのサイズが等しいならば，仮想ページプールを用意して，ポインタの配列として実現できる．仮想リングバッファ上の読み書きのサイズが異なるならば，読み書きにおいて scatter / gather 形の処理を行い，仮想メモリ関係のカーネル呼び出しを最小限にとどめることができる．

一方，異なるプロテクション・ドメインにあるストリーム間の仮想リングバッファを実現する場合，仮想リングバッファのための仮想空間をふたつのタスク間で共有してもデータのコピーが避けられないことがある．例えば，プロトコル・サーバではネットワークからの入力をそのデータの内容によって宛先を動的に決めなければならず，共有空間に直接書き込むことはできない．このため，適当な仮想ページにデータを書き込んだ後，その仮想ページを移動することになる．つまり，受け側で **vm_allocate** を実行した後，送り側で **vm_write** と **vm_deallocate** を実行して仮想ページを移動することになる．ところが，**vm_write** 等のシステムコールは，copy-on-write の機構によって実現されているため，高価なものとなっている．また，データを書き込む度に複数回のカーネルとのやりとりが必要となり，コピーの回避による効果が薄れてしまう．また，Mach では共有空間に対する仮想ページの移動がサポートされていないため，仮想リングバッファを複数の異なるプロテクション・ドメイン間の共有空間上に実現することができない．ここでは，効率化のため，以下に述べる新たなプリミティブを利用して実現する．

4 Mach 仮想記憶管理機構の拡張

連続メディアのアプリケーションを作成するためには，仮想ページを物理ページにくくりつける (wire) ことで page fault を避け，また，そのような wired page を常に一定量確保することでネットワークからのメッセージを書き込めるようにしておかなければならない．もし，wired page を他の保護ドメインに移動すると，送り側のドメインでは空きの wired page の総量が減少し，上記の条件が満たされなくなってしまう．これを解決するためには，送り側と受け側で同じ種類のページを交換すれば良い．連続メディアのアプリケーションでは，受け側では送り側と同じ速度でメッセージを処理するため，キュー機構は不要であり，このようなことが可能となる．

RT-Mach 上で仮想リングバッファを実現するために，遅延評価によらない仮想記憶管理のプリミティブを追加する (図 1)．

- **vm_swap** — 仮想ページのマッピング情報を送り側と受け側で交換する．送り側および受け側の仮想ページは，物理ページにくくりつけられていなければならない．
- **vm_steal_in** — 仮想ページを受け側に移動する．このプリミティブは **vm_swap** と似ているが，この呼出しが正常に終了した後は，送り側は割り付けのないページとなる[1]．

これらを実現するにあたり，メモリオブジェクトのセマンティクスとの整合性と，キャッシュや TLB の扱いが問題となる．前者は仮想ページと物理ページがくくりつけられているときのみを扱うことで回避している．後者は性能に対して大きな影響を与える．高い性能を実現するためには，キャッシュや TLB の全てをフラッシュすることなく実際にマッピングが変更された部分のみをフラッシュし，さらに仮想記憶管理の低レベルの部分[2]で実現する必要がある．現在は，仮想記憶管理の高レベルの部分で実現しており，性能的には劣ると考えられる．また，カーネルのバッファ領域はゾーンにより管理されており，その部分とのマッピングの交換はできない．これは，以下で述べる **device_set_incursive_filter** を実現する場合に問題となる．

カーネル空間とユーザ空間の間の共有メモリ (projected buffer) を実現するため，以下のプリミティブ

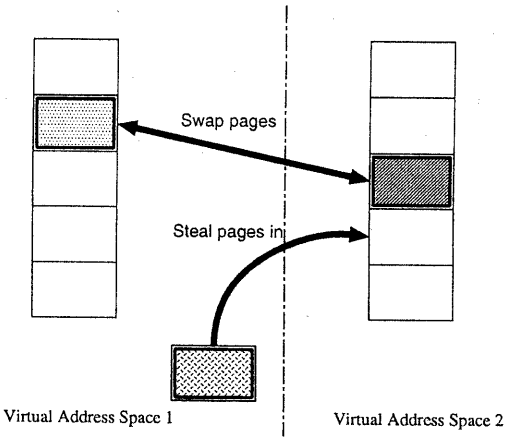

図 1: vm_steal_in と vm_swap

を用意する[3]．

- **vm_allocate_projected_buffer** — カーネル空間に wired page を用意し，それをユーザ空間へ貼り付ける．
- **vm_map_projected_buffer** — 既に存在する projected buffer をユーザ空間へ貼り付ける．
- **vm_deallocate_projected_buffer** — projected buffer を削除する．

共有空間を利用した場合，ひとつの物理ページを指す仮想ページが複数あるため，TLB エントリの無駄が生じ，それらの一貫性を保つためのコストがかかってしまうが，copy-on-write のコストに比べると格段に小さい．projected buffer では共有の関係をカーネルで保持するように実現しているため，送り側または受け側のいずれかが projected buffer 上にあったとしても **vm_swap**, **vm_steal_in** が動作する．

ネットワークからのパケットを仮想リングバッファによって受信するために，以下の 2 つのプリミティブを用意する．

- **device_set_projected_filter** — パケットフィルタの出力を projected buffer で構成されたリングバッファ経由でコピーによって受信する．
- **device_set_incursive_filter** — パケットフィルタの出力をユーザ空間上にあるリングバッファ経由でカーネルから **vm_swap** 相当の機能によってユーザ空間に渡す．

通常の **device_set_filter** は Mach の IPC を利用してパケットフィルタの出力を受信するものである．

以上で述べたインターフェースを利用したプロト

[1] external pager のためのカーネル・インターフェースである **memory_object_data_supply** は **vm_steal_in** と同様に仮想ページの steal-in の機能を持つが，これは external pager の実現にしか用いることができない．

[2] Mach VM における pmap のこと．

[3] 本稿とほぼ同じ機能の Projected buffer を実現するためのカーネル内ルーチンが Mach MK83 に追加されている．

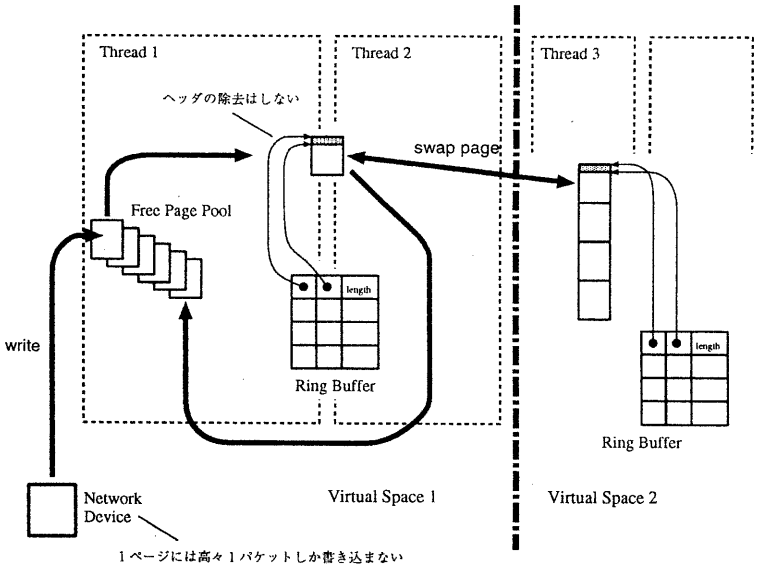

図 2: 仮想リングバッファを利用したプロトコル処理サーバの構成例

コル処理サーバの実現例を図 2 に示す.

5 プロトコル処理サーバによる評価

連続メディア向きのプロトコル処理サーバのプロトタイプを構築し，前節で述べたプリミティブの効果の確認を行った．通信プロトコルとして，Ethernet 上の UDP/IP を用いた．パケットのフラグメンテーションとリアセンブルの処理は省略し，1472 bytes 以下のデータのみを扱う．また，UDP のデータのチェックサムも省略している．パケットの送出側は，10 ms 周期のリアルタイムスレッドによって，指定された個数のパケットを送り出す．受け側は 10 ms 周期で受信スレッドを活性化し，IPC のポートまたはリングバッファに保持されている分のパケットを読み出す．スループットの測定は，データサイズを固定し，10ms 間に送り出すパケットの数を変化させ，パケットの喪失がない最大の数を求めることで行った．テストベッドとしては DECstation 5000/125 を用いた[4].

実験の結果を図 3に示す．仮想リングバッファ機構を用いた方が Mach IPC を用いるよりも高いスループットを得ることができる．仮想リングバッファ機構を用いた場合でもデータサイズが 750bytes で頭打ちとなっているが，これは Ethernet の帯域幅に抑えら

[4]CPU は MIPS R3000A 25MHz．メモリは 64MB.

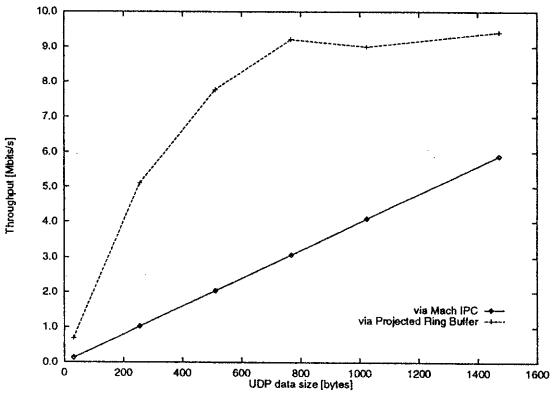

図 3: 仮想リングバッファを利用したプロトコル処理サーバのスループット

れているものと思われる[5]．Mach IPC を用いた場合は，割り込み型でパケットを処理することが可能であり，遅延時間を小さくすることができる．一方，仮想リングバッファ機構を用いた場合は，リアルタイムスレッドでポーリング型で処理する必要があり，オーバーヘッドが少ないためスループットの面では有利であるが，遅延時間はリアルタイムスレッドの周期で抑えられてしまうという欠点がある．

[5]Ethernet を UDP パケットで充たした場合の理論的なスループットは 9.57Mbits/s である．

6 おわりに

本論文では，連続メディアオブジェクトを扱うためのメモリ管理機構について，IPCの観点から考察し，仮想リングバッファ機構とそのためのRT-Machのメモリ管理プリミティブを提案した．本手法は高速なネットワーク経由の通信 [14] [4] を実現するために有効である．ST-IIプロトコルサーバ [12] での利用を含め，詳細な評価を行っていく．また，カーネル内のバッファの管理，デバイスの扱い，QOSの保証など，内部構造に踏み込んだ考察が必要である．

現在，QOSを保証するための資源管理およびアドミッション制御を行う資源管理マネージャの実現を行っており，機会を改めて報告したい．

謝辞

本研究を行うにあたり，貴重なご討論を頂いている慶應義塾大学環境情報学部の斎藤信男教授，徳田英幸助教授，萩野達也助教授をはじめとする開放型基盤ソフトウェア研究開発評価事業「マルチメディア統合環境基盤ソフトウェア」プロジェクトの皆様に感謝いたします．

参考文献

[1] Accetta, M., Baron, R., Bolosky, W. J., Golub, D., Rashid, R., Tevanian, A., and Young, M.: Mach: A New Kernel Foundation for UNIX Development, in *Proceedings of the Summer 1986 USENIX Conference*, 93–112 (1986).

[2] Anderson, D. P.: Meta-Scheduling for Continuous Media, Technical Report UCB/CSD 90/599, UC Berkeley (1990).

[3] Barrera III, J. S.: A Fast Mach Network IPC Implementation, in *Proceedings of the USENIX 1991 Mach Symposium* (1991).

[4] Berenbaum, A., Dixon, J., Iyengar, A., and Keshav, S.: A Flexible ATM-Host Interface for XUNET II, *IEEE Network*, **7**, 4, 18–23 (1993).

[5] Bershad, B., Anderson, T., Lazowska, E. D., and Levy, H.: Lightweight Remote Procedure Call, *ACM Transactions on Computer Systems*, **8**, 1, 37–55 (1990).

[6] Cheriton, D. R.: The V Distributed system, *Communications of the ACM*, **31**, 3, 314–333 (1988).

[7] Dalton, C., Watson, G., Banks, D., Calamvokis, C., Edwards, A., and Lumley, J.: Afterburner, *IEEE Network*, **7**, 4, 36–43 (1993).

[8] Druschel, P. and Peterson, L. L.: Fbufs: A High-Bandwidth Cross-Domain Transfer Facility, in *Proceedings of 14th ACM Symposium on Operating System Principles* (1993).

[9] Govindan, R. and Anderson, D. P.: Scheduling and IPC Mechanisms for Continuous Media, in *Proceedings of 13th ACM Symposium on Operating System Principles* (1991).

[10] Inouye, J., Konuru, R., Walpole, J., and Sears, B.: The Effects of Virtually Addressed Caches on Virtual Memory Design and Performance, *ACM Operating Systems Review*, **26**, 4 (1992).

[11] Johnson, D. B. and Zwaenepoel, W.: The Peregrine High-performance RPC System, *Software – Practice and Experience*, **23**, 2, 201–221 (1993).

[12] Kihara, S., Onoe, Y., Mitsuzawa, A., Moriai, S., Nambu, A., and Tokuda, H.: An Implementation of ST-II Protocol as a User-Level Server on Real-Time Mach, in *Collected Abstracts from the 4th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, 59–62 (1993).

[13] Liedtke, J.: Improving IPC by Kernel Design, in *Proceedings of 14th ACM Symposium on Operating System Principles* (1993).

[14] Maeda, C. and Bershad, B. N.: Protocol Service Decomposition for High-Performance Networking, in *Proceedings of 14th ACM Symposium on Operating System Principles* (1993).

[15] 盛合敏: リアルタイムオブジェクトむきのIPCについて，情処研報，**93**，58 (OS-60, DPS-61)，9–16 (1993).

[16] Nakajima, J., Yazaki, M., and Matsumoto, J.: Multimedia/Realtime Extensions for Mach 3.0, in *Proceedings of 1992 USENIX Workshop on Microkernels and Other Kernel Architectures*, 161–175 (1992).

[17] Rashid, R., Tevanian Jr., A., Young, M., Golub, D., Baron, R., Black, D., Bolosky, W. J., and Chew, J.: Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures, *IEEE Transactions on Computers*, **37**, 8, 896–908 (1988).

[18] Schroeder, M. D. and Burrows, M.: Performance of Firefly RPC, *ACM Transactions on Computer Systems*, **8**, 1, 1–17 (1990).

[19] Smith, J. M. and Traw, C. B. S.: Giving Applications Access to Gb/s Networking, *IEEE Network*, **7**, 4, 44–52 (1993).

[20] 多田, 西尾, 藤井, 堀切, 小野, 斎藤: 実時間カーネルを用いた連続メディアベースの設計，第5回コンピュータシステム・シンポジウム論文集，33–40 (1993).

[21] Tzou, S.-Y. and Anderson, D. P.: A Performance Evaluation of the DASH Message-Passing System, TR 88/452, UCB/CSD (1988).