システムソフトウェアと
オペレーティング・システム 69－7
（１９９５．６．２３）

マイクロカーネルをもとにしたシステム構成に関する考察

岡坂 史紀*　　清水 謙多郎

電気通信大学情報工学科

マイクロカーネルは，本来，オペレーティングシステムの構成をより簡明にする目的で設計された。しかしながら，最近では，さまざまなポリシー，アブストラクションをサポートするために，マイクロカーネル本体を拡張して複雑で扱いにくいものにしてしまう傾向が見られる。本稿では，資源管理のための，利用者レベルサブシステムとカーネルとのインタラクションを増やさずに，マルチメディア応用に適したオペレーティングシステムを構成する枠組みについて検討し，筆者らが開発を進めている Act 8 オペレーティングシステムについて報告する。

A Vision of an Operating System Framework Built on a Micro-Kernel Towards the New Age

Shiki Okasaka　Kentaro Shimizu

The Department of Computer Science
The University of Electro-Communications

1-5-1 Chofugaoka, Chofu-shi, Tokyo, 182, Japan

Micro-kernels were originally designed to introduce modular architecture for the implementation of operating systems. However, recent micro-kernels tend to become complicated and difficult to maintain as they incorporate many policies and many abstractions. In this paper, we discuss a multimedia capable operating system framework which does not require extra resource management interaction between the micro-kernel and application subsystems, and describe our development of an operating system named Act 8.

* mailto: okasak-s@argus.cs.uec.ac.jp

1 はじめに

1.1 問題点

近年のオペレーティングシステムは，研究用，商用を問わず，マイクロカーネルをもとに構成されたものが増えてきた。今後も，マイクロカーネルが主流になるのかどうかはともかく，代表的なマイクロカーネルであるMachに関しては，批判的な議論がある。Presotto[1]らは，適切なアブストラクションを厳選すれば，大きなシステムと同等の機能を備えたより小さなシステムを構成できるとし(Minimalism)，実際にMachマイクロカーネルの半分以下の行数でPlan 9オペレーティングシステムを構築した。Cheriton[2]らは，システムの複雑さをもっともよく表わすのはコードサイズであり，複雑なシステムほど信頼性に問題を起こすとした上で，マイクロカーネルは，依然として複雑過ぎ，エラーを起こしやすいとしている。その理由として以下の点を挙げた:

1) 性能向上を理由に，サーバモジュールがカーネルに戻される。
2) アプリケーション特有の資源管理を行うための拡張がなされる。
3) サーバやネットワークの処理において，複雑な例外処理が要求される。

近年では，マルチメディア処理のため，従来とは異なるポリシー，アブストラクションの必要性が強調される。Cheritonらの主張に従えば，マイクロカーネルは今後さらに複雑化への道をたどることになる。

1.2 目的および方針

マイクロカーネルを十分小さく実現できれば，さまざまな用途をひとつのマイクロカーネルでサポートするのではなく，目的に合わせてマイクロカーネルを置き換えるという方法を採用できる。ただし，用途ごとに新しいインタラクションを導入していたのでは，カーネルを置き換えるたびに，利用者プログラムとして実現したサブシステムまで修正しなければならないという危険が発生する。

筆者らのAct 8オペレーティングシステム[6]では，Minimalismを適用し，マイクロカーネルに実装するモジュールを，デバイスドライバの一部と次の3つのモジュールに限定している:

A －アドレス空間を実現する仮想記憶
C －手続き呼出し型のプロセス間通信
T －カーネルスレッド

さらに，Act 8マイクロカーネルは，マルチメディア環境の実現を考慮し，下記のような実時間サーバ開発の枠組みを提供する:

1) 利用者スレッドにより，サービスが並行に実行される。
2) 利用者スレッドは，サービスを要求したクライアントと同じスケジューリング優先度で実行される。
3) サービス実行における優先度逆転の問題を，基本優先度継承プロトコル[4]により回避する。

従来の利用者スレッドによる並行処理では，スレッドの実行コンテクストを管理するためのカーネルとアプリケーションとの特別なインタラクションが必要であった[3]。Act 8では，利用者スレッドによる並行処理をサーバ開発に限定することにより，アプリケーションとカーネルとの余分なインタラクションを必要としない利用者スレッドを用いたサーバ開発の枠組みを実現している。

本稿では，2章でAct 8マイクロカーネルの3つの基本モジュールについて，3章でサーバ開発の枠組みについて，4章でAct 8の実現と評価について説明する。最後に，関連研究との比較を行い，今後の予定を述べる。

2 Act 8マイクロカーネル

2.1 プロセス間通信

Act 8のプロセス間通信はクライアントサーバ

方式にもとづく手続き呼出し型の通信である。

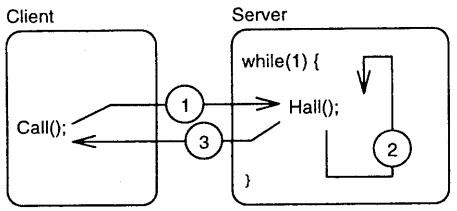

図1 手続き型プロセス間通信

クライアントがサーバを呼び出すには Call システムコールを，サーバがクライアントからの要求を受け取るには Hall システムコールを用いる(図1)。メッセージ通信は，ケイパビリティにもとづく。Act 8 のケイパビリティはサーバプロセスを代表し，カーネルによって管理される。

2.2 スレッド

Act 8 のプロセス間通信は，クライアントとサーバのアドレス空間をスレッドが往来することで実現される。そのため特別な工夫をしなくてもサーバはクライアントの優先度でサービスを実行する(図2)。

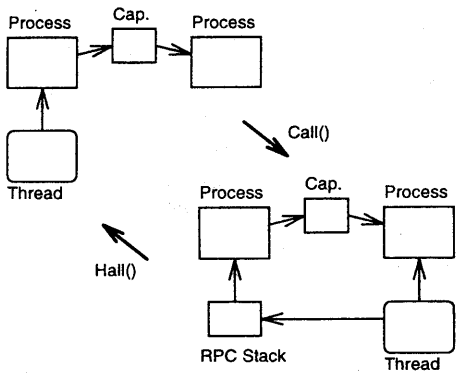

図2 スレッドと手続き型プロセス間通信

カーネルデバイスドライバは，スレッドの再起動をタイマや DMA の割込みなどに連動させることができる。

2.3 仮想記憶サブシステム

今日の計算機には，ワードプロッセッサ，通信ソフトなど普段利用するアプリケーションをすべて収められる程度の主記憶が装備されている。主記憶のサイズよりも大きいプログラムを走らせるという用途や，参照の局所性といった性質は，それほど重要でない。それでも，仮想記憶機構は次のような点で効果的である:

1) ページの共有，コピーオンライトといった技法による処理の効率化。
2) サイズ可変のバッファキャッシュ機構としての利用。

音響，映像といった連続データのファイルを扱う場合には，バッファキャッシュとしての仮想記憶サブシステムもそれに対応する必要がある。特に，ページの空きリストを維持するため，ページ表項目をページ枠を参照していない状態に戻すページ除去方針の選択が重要である。従来の大域的 LRU では，映像を記録したファイルを再生することにより，ほとんどのページ枠が映像ファイルのキャッシュに割り当てられてしまう可能性がある。

最近の UNIX®の madvise システムコールのように，アクセス方式をヒントとしてカーネルに通知する方式では，すべてのプログラムからのヒントをカーネルが解釈し，適切な解を求めなければならず，実現が複雑になる。

Act 8 では，もっとも多くのページ枠をページ表から参照しているメモリセグメント[1]について，参照しているページ枠数がしきい値(多段)以下になるまで，FIFO 順にページ枠を除去する(注: セグメントがプロセスから参照されなくなったとき，すべてのページ枠を空きリストに戻すことはしない)。頻繁に使用されるページは，空きリストに移された後，再びセグメントのページ表項目に戻されるので，周辺装置への負荷が大きく変わることはないと考えられる。

また，Act 8 の外部ページャ[2]インタフェースで

[1] 個々のファイルやプロセスの bss 領域などに対応し，別々のページ表をもつ。

[2] ページの読込み，掃出しなどを行うサーバプロセス。

は，次の2点を可能にしている:

1) ページャが掃出し中のページに参照が発生した場合，掃出し中のページをセグメントのページ表から参照できるようにする。
2) ページャによるページ掃出しが遅いために，空きページ数を維持できない場合，デフォルトのスワップ装置に一時的にページを掃き出す。

この組合わせの実現は，必ずしも単純ではないが，ページ枠およびページ表項目に関する 72 状態からなる状態遷移表をもとに，系統的に実装を進めている。

3 サーバ開発の枠組み

Act 8 のサーバ開発では，ひとつの巨大なモニタに基づくコーディングを用いる。すなわち，サービス実行中の横取りを無視することができる。そのため，UNIX カーネルの記述方法が，サーバ開発においても利用できる。さらに，このシステムは，利用者スレッドによるサービスの並行実行と実時間スケジューリングをカーネルとの特別なインタラクションなしに実現する。

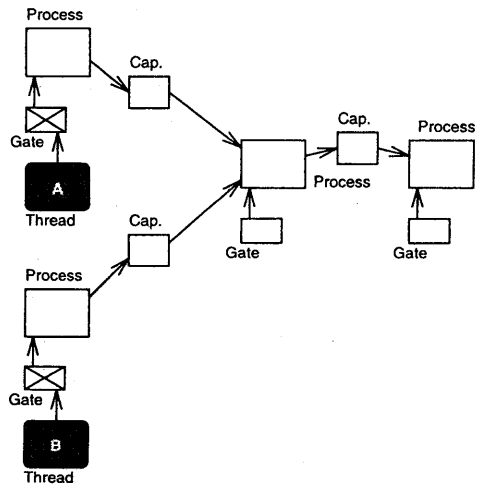

図3 スレッドA, Bのサーバ呼出し

図3は2つのスレッドAとBが同じサーバプロセスを呼び出そうとしている状態を示している。プロセスにはGateがひとつずつ定義されている。Gate は基本優先度継承プロトコルが適用される相互排除ロック(mutex)である。プロセスのアドレス空間内部をスレッドが実行するには，スレッドが Gate をロックしている必要がある。すなわちプロセス内部を同時に実行できるスレッドはひとつに限られる。スレッドはサーバを呼出すとき，まず現プロセスの Gate を解除することによって，現プロセスの並行実行を可能にする。

サーバを呼び出すとき，クライアントプロセスはアドレス空間に配置されているメモリセグメント(複数でも，セグメントの一部分でもよい)をサーバプロセスに共有させることによってデータの受け渡しを行うことができる。サーバがサービスを完了すると，セグメントの共有は解除されるが，このときサーバはセグメントをコピーオンライトにより複製し，サーバアドレス空間に保持することができる。

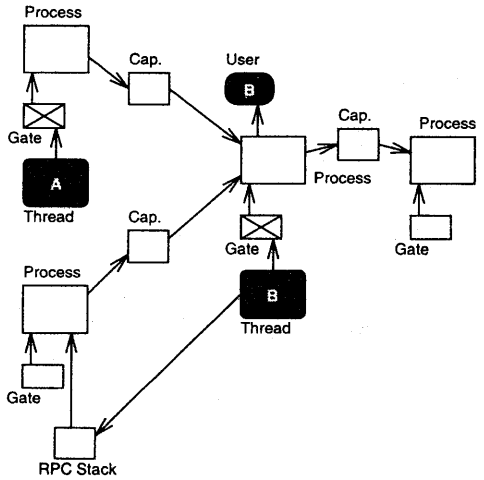

図4 スレッドBによるサーバの実行

図4はスレッドBがサーバを呼び出したあとの状態を示している。サーバは，スレッドBに対応する利用者スレッドBを作成している。スレッドBによるサーバ内部の実行は，この利用者スレッドBによって代表される。

サーバの Gate はスレッドBによってロックされているので，スレッドAが同じサーバを呼び出

すことはできない。しかし Gate では基本優先度継承プロトコルが適用されるため，スレッドAがスレッド B よりもよい優先度をもっていてサーバを呼び出そうとしたときには，スレッドBはスレッド A の優先度を継承してサービスを実行する。そうでなければ，サーバはスレッドBの優先度で実行する。

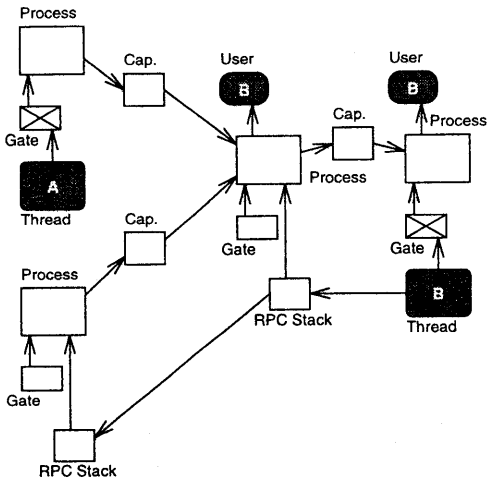

図5 スレッドBによる新しいサーバの実行

図5はスレッドBがさらに別のサーバプロセスを呼び出したあとの状態を示している。このとき元のサーバの Gate のロックが解除される。また，先程と同様に利用者スレッドが新しくサーバ内部に生成されている。元のサーバのゲートが解除されているので，スレッドAがサーバを呼び出すことができる。

図6はスレッドAがサーバプロセスを呼び出したあとの状態を示している。サーバは，スレッドAを代表する利用者スレッドAを作成している。このとき，サーバはスレッドAの優先度でサービスを実行することになる。このようにして，クライアントの優先度にもとづいた，サービスの並行実行が行われる。

図6においては，スレッドAがサービスの実行に失敗しサーバプロセスを終了させるような事態が起こり得る。Act 8 マイクロカーネルは，プロセスの参照数を管理してプロセス構造体を破棄する時期を決定することにより，このような場合にも，スレッドBが最初のクライアントまで問題なく戻れるようにしている。アプリケーションプログラムにとっては，Call システムコールの戻り値から，サーバの異常終了がすぐにわかり便利である。

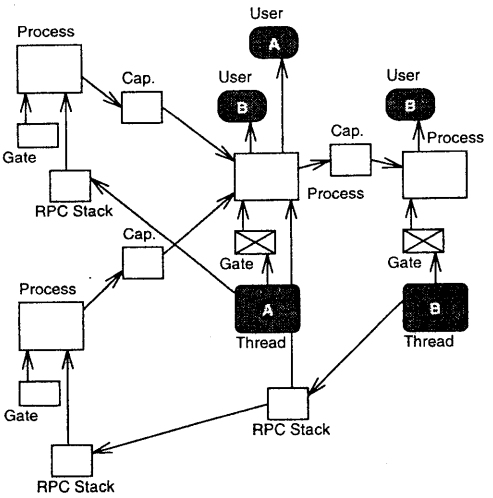

図6 スレッドAによるサーバの並行実行

4 評価

Act 8 マイクロカーネルのプログラムは，デバイスドライバなどハードウェアに依存した部分を除くと，C言語で 6,000 行程度である(カーネルイメージは約 120KB)。

表1. Act 8 の基本性能

	1 回目[μsec]	2 回目[μsec]
Mutex	18.4	10.9
Null-RPC	210	164
RPC-VM	475	384

表 1に，Act 8 の実時間 mutex，実時間プロセス間通信について Intel486™ DX2/66MHz の PC AT™互換機で測定した結果を示す。Mutex, Null-RPC, RPC-VM は，それぞれカーネル内で mutex を獲得しただちに解除する時間，空のサーバ呼出し時間，メモリセグメントの共有による2ページ分(8192 バイト)のデータの受け渡しを含んだサ

ーバ呼出し時間である。

5 おわりに

Cheritonらは，アクティブなオブジェクトのみをカーネル空間にキャッシュすることによって，カーネルを小さく，またより多くの資源を扱えるようにしたCache Kernelを実現した。しかしながら，オブジェクトを利用者空間に退避したり，カーネル空間に回復させたりといった新しいインタラクションを導入したことによって，システム全体として複雑さが増していないのか疑問が残る。

筆者らの006オペレーティングシステムでは，アクティブなスレッドのみをカーネル空間におくことによって，非常に多数のスレッドを生成可能にしたマイクロプロセスを実現した[3]。しかしながら，スレッドの実行コンテクストの退避，回復，優先度の設定を実現するプログラムコードは，カーネル空間の節約という意味では効果的であったものの，システムの簡略化には結びつかなかった。

Act 8 オペレーティングシステムは，アプリケーションとカーネルとのインタラクションをできる限り押さえるという方針のもとで，利用者スレッドによるサービスの並行実行と実時間スケジューリングを可能するサーバ開発の枠組みを実現した。

手続き呼出し型通信において，クライアントとサーバのアドレス空間をスレッドが往来するという方式は，遠隔手続き呼出しの高速化という目的でBershad[5]らにより開発された。Act 8では，これにクライアントの実時間同期制御，利用者スレッドパッケージを統合し提供している。

現在，筆者らは，Act 8上のPlan 9パーソナリティの開発を進めており，シンプルなコマンド環境の中でマルチメディアを利用する可能性について検討している。

参考文献

[1] D. Presotto, R. Pike, K. Thompson, and H. Trickey, "Plan 9, A Distributed System," *Proc. of the Spring 1991 EurOpen Conf.*, May, 1991.

[2] D. Cheriton and K. Duda, "A Caching Model of Operating System Kernel Functionality," *Proc. of the First USENIX Symposium on Operating System Design and Implementation*, November 1994.

[3] 岡坂 史紀, 清水 謙多郎, 芦原 評, 亀田 壽夫, "ユーザプログラムとカーネルの協調に基づくスレッドの設計と実現," 情報処理学会論文誌, Vol. 36, No. 4, April 1995.

[4] L. Sha, R. Rajkumar, and J. Lehoczky, "Priority Inheritance Protocols: An Approach to Real-Time Synchronization," *IEEE Transactions on Computers*, Vol. 39, No. 9, September 1990.

[5] B. Bershad, T. Anderson, E. Lazowska, and H. Levy, "Lightweight Remote Procedure Call," *ACM Transactions on Computer Systems*, Vol. 8, No. 1, February 1990.

[6] 岡坂 史紀, 清水 謙多郎, "対称型マルチプロセッサ用実時間カーネルの設計と実現について," 第 50 回情処全大論文集, 2H-5, March 1995.

ソフトウェア配布について

Act 8 オペレーティングシステム関連のソースコードについては，公開の準備を進めています。詳細については，以下の URL を参照してください。

http://esther.cs.uec.ac.jp/PROJECT/act8.html