システムソフトウェアと
オペレーティング・システム 73－17
（１９９６．８．２７）

カーネル機能の動的構築が可能なOSの設計

柏木一彦 最所圭三 福田晃

奈良先端科学技術大学院大学
〒630-01 奈良県生駒市高山町8916番地の5
{kazuhi-k,sai,fukuda}@is.aist-nara.ac.jp

オペレーティングシステムのカーネルを設計する場合，高速化・資源の有効利用と拡張性・柔軟性・移植性とのトレードオフを考慮しなければならない．前者に重点をおいた場合は単層カーネル構成が，後者に重点を置いた場合はマイクロカーネル構成が選択されるであろう．

そこで，本稿では拡張性や柔軟性等を維持しつつ，高速化を実現するカーネルの構成を提案する．今回は本方式の有用性を確かめるため，既存のオペレーティングシステム上にプロトタイプを実装し，システムコールの処理速度を測定して本方式を評価した．その結果，全体的にシステムコールの処理速度が上昇し，本方式の有用性が確認できた．

Design of Operating System with Dynamically Constructing Kernel Functions

Kazuhiko Kashiwagi, Keizo Saisho, Akira Fukuda
Graduate School of Information Science,
Nara Institute of Science and Technology
8916-5,Takayama-cho, Ikoma, Nara, 630-01, Japan
{kazuhi-k,sai,fukuda}@is.aist-nara.ac.jp

There are two approaches to design and implement an operating system kernel: a monolithic kernel and a microkernel. The monolithic kernel has advantages of high performance and high utilizability of computer resources. The microkernel has ones of flexibility, expandability, and portability. Design choice of which kernel is employed is decided by the tradeoff between these advantages.

In this paper, we propose a kernel architecture which enables to construct its functions at run time in order to take in the both advantages. The availability of the kernel is confirmed by implementing a prototype on the existing operating system and measuring the performance of some system calls.

1 はじめに

オペレーティングシステム (以下 OS と表す) のカーネルを構築する場合，いくつか構成法が存在する．従来はカーネルを 1 つのモジュールとする単層カーネル構成であった．この構成は，実行速度や計算機資源の利用効率という点では優れている．ただし，拡張性・移植性・柔軟性が低い．このため，拡張性・移植性・柔軟性等を考慮して，カーネルをいくつかのモジュールに分割し運用するマイクロカーネル構成が研究されるようになった．この構成では，複数のモジュールで協調作業を行なうために，メッセージを用いる．メッセージ伝達が頻発する場合，メッセージ伝達のオーバヘッドが全体の処理に占める割合が多くなり，実行効率が落ちる．

そこで，本稿では，上記 2 種類の長所に着目し，拡張性・移植性・柔軟性を考慮しつつ実行速度を改善できるように，運用時にカーネル機能の動的構築を可能とする OS の構成法を提案する．

本稿においては，既存の教育用 OS をベースにして，目的の OS カーネルのプロトタイプの開発および性能評価を行なった．システムサーバをカーネル内に追加したカーネルで，一部のシステムコールの処理速度の比較を行なった結果，通信のオーバヘッドが劇的に改善され，特に処理が小さいシステムコールの性能向上が目立った．

以下，第 2章では従来のカーネル構成の特徴を述べ，第 3章で本稿で提案しているカーネル機能を動的に構築する OS について説明する．第 4章で目的の OS カーネルを実現するためのプロトタイプの設計方針を示し，第 5章で今回行なったプロトタイプの実装と性能評価について述べる．第 6章で今後の課題等を述べる．

2 従来の OS

OS のカーネルを構築する場合，いくつか構成法が存在する．従来はカーネルを 1 つのモジュールとする単層カーネル構成であった．

この構成の特徴は以下である．

- 実行速度や計算機資源の利用効率という点で優れている．
- 機能の変更や拡張が容易ではない．
- ハードウェア依存の部分が散在し移植性が低い．
- 分散環境や並列環境に容易に適応できない．

このため，拡張性・移植性・柔軟性等を考慮して，カーネルをいくつかのモジュールに分割し運用するマイクロカーネル構成が研究されるようになった [1, 2, 3, 4]．OS は，必要最小限の機能で構成されるマイクロカーネルと，その他の機能をサポートするシステムサーバとに分割されている．

この構成の特徴は以下である．

- 機能の変更や拡張をシステムサーバ単位で行なえるので，拡張性が高い．
- システムサーバ単位で分散環境や並列環境に対応できるので，柔軟性が高い．
- ハードウェアに依存するコードがマイクロカーネル内にのみ存在する場合が多いので，移植性が高い．
- モジュール間でメッセージ伝達が頻発するとそのオーバヘッドで実行効率が落ちる．
- 各モジュール内に同様の冗長コードが存在し，計算機の利用効率が悪い．

3 機能の動的構築が可能な OS

3.1 概要

2章で述べたような，従来の単層カーネル構成，マイクロカーネル構成の OS の長所と短所を考慮して，移植性・拡張性・柔軟性のある OS カーネルの構築は非常に有用である [5, 6, 7, 8]．

そこで本研究ではカーネル機能を独立したモジュールとして実現し，必要に応じてそれらの

機能をカーネル実行中に追加・削除・置換することが可能なOSカーネルを提案する[9, 10].

以下，本研究におけるカーネル機能の追加・削除・置換の概念を示す.

- 追加

  カーネル機能の追加は以下の2通りを考える.

  (1) ユーザ空間からカーネル空間への移動

  図1に示すように，プロセスとして運用されていたカーネル機能をカーネル空間へ移動させ運用する.

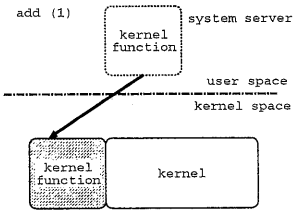

図 1: 追加 (1) の概念

  (2) カーネル機能自体の追加

  図2に示すように，プロセスとして運用されていない場合，カーネル機能自体をファイル上からカーネルのメモリ空間内へ追加する.

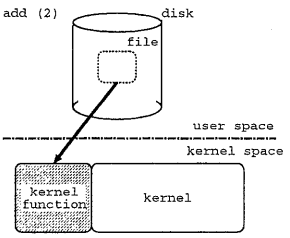

図 2: 追加 (2) の概念

- 削除

  カーネル機能の削除は以下の2通りを考える.

  (1) カーネル空間からユーザ空間への移動

  図3に示すように，カーネル空間内で運用されていたカーネル機能をユーザ空間へ移動させ，システムサーバの様にプロセスとして運用する.

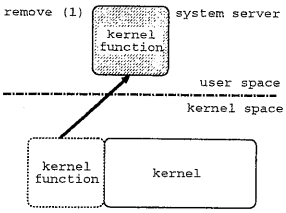

図 3: 削除 (1) の概念

  (2) カーネル機能自体の停止

  図4に示すように，カーネル機能自体を停止し，それまでにそのカーネル機能が保有していた計算機資源も同時に開放する.

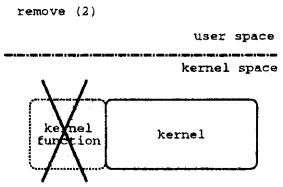

図 4: 削除 (2) の概念

- 置換

  カーネル機能の置換は以下の2通りを考える.

  (1) 旧カーネル機能の移動

  図5に示すように，旧カーネル機能をユーザ空間へ，新カーネル機能をカーネル空間へ移動させる.

  (2) 旧カーネル機能の停止

  図6に示すように，旧カーネル機能を停止させそのまま新カーネル機能へ置き換える.

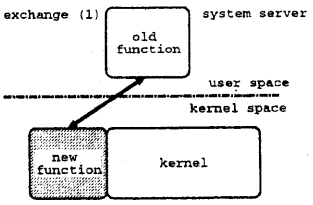

図 5: 置換 (1) の概念

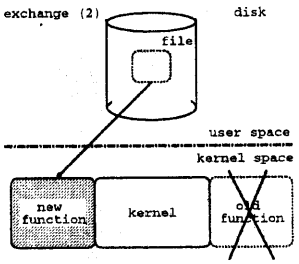

図 6: 置換 (2) の概念

4 プロトタイプの設計

4.1 設計方針

カーネル機能の動的構築を行なうために必要な事項を以下に示す.

- 追加・置換機能をカーネルに認識させる方法

  カーネル機能を追加したり新たな機能に置換する場合, 他のカーネル機能に認識させなければならない. 手法はいくつかあるが, 本研究では, 各カーネル機能の相互に対して, 各々のメモリ空間を認識させる方式を採用した. 具体的には, カーネル機能がカーネル内に存在する場合には, 各カーネル機能に対して, 相互に該当する機能の直接呼び出しを可能とすることである. ただし, カーネル機能をプロセスとして運用する場合は, プロセス間通信を行なえるようにしなければならない.

  この方式では, 主記憶のみ利用するカーネルや, 単一仮想アドレス空間をサポートしているカーネルでは, 有効な実現方法である. 今回は, プロトタイプの実装に用いた既存の OS の特徴から, 本方式が有用であると判断した.

- データの管理

  追加・削除・置換を行なう場合, 以下のようなデータの管理が必要となる.

  - 各カーネル機能が提供する機能に関する情報
  - 置換する場合に相互に授受が行なわれるカーネル機能内部のデータ

  これらの情報を管理するため, 後述するカーネル機能マネージャを, カーネル内にカーネル機能毎に実装することにした.

- データや変数等の保護

  データや変数等の保護を実現する方法はいくつか存在するが, 本研究ではオブジェクト指向の導入を検討した.

  オブジェクト指向であれば, 言語レベルで情報の隠蔽や保護といったことが可能である. また, 各カーネル機能をオブジェクトとしてとらえ定義することで, モジュラリティが高まる. なおかつ, オブジェクト指向を導入すれば, インタフェースの統一も可能となるので, 有効である.

5 プロトタイプの実装

提案した OS のカーネルのプロトタイプを実装するにあたり, 既存の OS のカーネルをテストベッドとして利用することにした.

具体的には, 教育用 OS の Minix[11] を用いた.

5.1 実装

以上述べてきたことを踏まえて, 実際に Minix のソースコードを変更し, カーネル機能を追加・削除できるような構成のカーネルのプロトタイプを実装した.

5.1.1 システムコールの実装

カーネル機能を動的に追加・削除するために以下のような処理内容のシステムコールを作成した．ただし，今回は置換機構の実装が不十分であるので，そのためのシステムコールは実装していない．

- プロセスが自身をカーネル内に追加する．
- あるプロセスをカーネル内に追加する．
- ファイルとして存在する所定のプログラムをカーネル機能としてカーネル内に追加する．
- あるカーネル機能をカーネルから分離してプロセスにする．
- あるカーネル機能をカーネルから消去する．

5.1.2 カーネル機能マネージャの実装

カーネル機能の追加・削除を実現する上で，カーネル内にカーネル機能を管理する機構が必要となる．具体的には以下の役割を担う機構である．

- 管理すべきデータの保持
- カーネルや他のカーネル機能とのインタフェース
- サービス要求のインタフェース
- サービス要求の正当性の検証

そこで，上記の役割を担うカーネル機能マネージャを，カーネル内に実装した．マネージャはカーネル機能毎に存在する．カーネル機能運用の概念を図7に示す．

5.2 システムサーバ

Minixにおけるシステムサーバであるメモリマネージャ(以下MMと表す)およびファイルシステム(以下FSと表す)をカーネル機能としてとらえ，カーネルに追加・削除することにした．

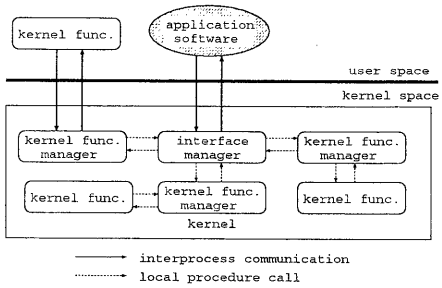

図7: カーネル機能運用の概念

MMおよびFSでは，システムコール処理依頼がくると，自分で処理できない機能があれば，当該処理をプロセス間通信を介してカーネルに委託する．よって，MMおよびFSをカーネルに追加した場合，カーネルと通信を行なう部分を，直接カーネル内の該当する関数や手続きを実行するように変更しなければならない．しかも，カーネルから分離し通常のシステムサーバとしての運用に戻した場合でも，再びカーネルと通信をするように戻せなければならない．

そこで，今回設計した追加・削除の機構を用いて，MMおよびFSを運用時に追加や削除させ，本システムが問題無く動作するかどうか実験した．結果は問題無く動作することを確認できた．

5.3 性能評価

今回設計・実装したカーネル構成が有用であるかどうかを確認するために，カーネル機能を追加した後のシステムコールの処理速度を測定した．実際には，エディタとシェルを例に取り，それらのなかで利用頻度の高いシステムコールについて，通常のMinixの構成とで比較して見た．結果は，全体的に処理速度は向上し，なかには約2倍向上したシステムコールも見られた．以下に，評価に用いたテストプログラムの実行時間の測定結果を示す．各データの単位は秒である．表1はsbrkを30000回実行するプログラムsbrkで，表2はforkした後，子プロセスで/bin/lsをexecleするループを30000回実行

するプログラム mfork である.

表 1: sbrk 実行時間 (30000 回実行)

Minix	実行時間
オリジナル	74.2 秒
新 OS	34.6 秒

表 2: mfork 実行時間 (30000 回実行)

Minix	実行時間
オリジナル	52.8 秒
新 OS	41.8 秒

結果から推測できることは，プロセス間通信がボトルネックになっているシステムコールについては，劇的な効果が見られるが，カーネルやカーネル機能内でのシステムコールの処理自体がボトルネックとなっているシステムコールについては，それ程効果が見られなかった.

6 おわりに

本論文では，カーネル機能の動的構築が可能な OS の構成を提案・設計し，プロトタイプを既存の OS 上に実装した．そして，システムコールの処理時間を測定することで，有用性を確認した.

今後は，以下のような様々な問題を解決していく予定である.

(1) 置換機構の実現

旧カーネル機能から新カーネル機能へのデータの受渡し等を実現する.

(2) 保護機能の強化

カーネル機能の適性検査，全く同じカーネル機能の重複登録の処理，別のカーネル機能としての追加といったなりすましの処理等を実現する.

(3) 無駄なコードの削減

各カーネル機能内に存在する冗長なコードを削除する.

参考文献

[1] M. Rozier et al.: Overview of the CHORUS Distributed Operating System, Proc. of the USENIX Workshop on Micro-Kernels and Other Kernel Architectures,pp.39-69 (1992).

[2] R.F. Rashid et al.: Mach: A System Software Kernel, Comput. Syst. Eng. (UK), vol.1, no.2-4, pp.163-169 (1990).

[3] D. Hildebrand: An Architectural Overview of QNX, Proc. of the USENIX Workshop on Micro-Kernels and Other Kernel Architectures, pp.113-126 (1992).

[4] J. Mitchell et al.: An Overview of the Spring System, Proc. of Compcon Spring 1994, pp.122-131 (1994).

[5] Brian N.Bershad et al.: SPIN - An Extensible Microkernel for Application-specific Operating System Services, Proc. of SIGOPS European Workshop, pp.74-77 (1994).

[6] 保木本 晃弘: オブジェクト指向に基づくモーバイルコンピューティングに適したカスタマイズ可能な OS 構築のための枠組, Workshop of Object Oriented Computing'94 (1994).

[7] Y. Yokote: The Apertos Reflective Operating System: the Concept and Its Implementation, SIGPLAN Not. (USA), vol.27, no.10, pp.414-34 (1992).

[8] J. Lepreau et al.: In-Kernel Servers on Mach 3.0: Implementation and Performance, Proc. of the Third Usenix Mach Symposium (1993).

[9] 柏木 一彦, 最所 圭三, 福田 晃：OS 機能のクラス化・モジュール化-機能の動的構築が可能な OS の構成を目指して-, 情報処理学会 OS 研究会, pp.70-76(1995).

[10] 柏木 一彦, 最所 圭三, 福田 晃：機能の動的構築が可能な OS におけるカーネル機能の考察, 情報処理学会コンピュータシステムシンポジウム, pp.9-15(1995).

[11] Andrew S.Tanenbaum: Operating System - Design and Implementation, Prentice Hall (1987)