システムソフトウェアと
オペレーティング・システム 70－6
（１９９５．８．２２）

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–

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

従来の単層カーネル構成やマイクロカーネル構成のOSでは，ユーザやアプリケーションの多様な要求に応じるため，OSに様々な機能が実装された．その結果，OS自体が複雑化することとなり，同時に様々な欠点が生じるようになった．

そこで，従来のOSの欠点を克服するため，我々は機能を動的に構築するOSの構成を提案する．

提案するOSを研究する上でカーネル機能のモジュール化は非常に重要である．そこで，本稿ではその一環として，オブジェクト指向の概念を導入し，カーネル機能をオブジェクトとして定義し，さらに実装したカーネルの評価を行なった．

Building Class Hierarchy and Implementing Modules of OS Functions
– Torward at Operating System with Dynamical 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

For satisfying various requests of users and application softwares, Many functions were implemented on an operating system. As a result, operating system itself has become complex and some week points have been appeared on it.

We propose the constructing way of the operating system which can be constructed its kernel functions dynamically in order to get over the week points of usual operating systems.

It is very important to divide kernel structure into some modules in order to accomplish proposing OS. We introduce a concept of object-oriented design and evaluate implemented kernel that we build objects as a part of its functions.

1 はじめに

従来のOSのカーネルは単層構成になっている．このため，ユーザの多様なニーズに応じて，必要とされる多くの機能を取り込んだ結果，OS自身が肥大化し，様々な問題が生じてきた．

そこで，単層カーネル構成を見直し，新たなカーネルの構成法としてマイクロカーネル構成の研究が行われるようになった．マイクロカーネル構成のOSは，カーネルでは必要最小限の機能のみを提供し，他の多様な機能はシステムサーバとして運用するという構成になっている．しかし，マイクロカーネル構成のOSにもサービス処理にかかるコストの増大などの欠点が存在する．

本稿では，機能の動的構築が可能なOSの構成法を検討し，より効率的な計算機資源の運用およびユーザやアプリケーションソフトへのより柔軟な対応を提案している．

以下第2節では従来のOSの構成を，第3節ではカーネル機能を動的に構築するOSについて，第4節ではカーネル機能のモジュール化について，第5節では今回行なったカーネル機能のクラス化とモジュール化について，第6節では結果をそれぞれ述べる．

2 従来のOS

UNIXに代表される従来のOSの構成である単層カーネル構成は，近年ユーザの多様なニーズに応えるために，必要とされる多くの機能を取り込むこととなり，以下のような問題が指摘されている．

- OSが肥大し，OS自体が消費する計算機資源が多い．
- 実際の運用面において，機能の追加・変更が容易ではなく，それにかかるコストが大きい．
- ソースの解析が容易でなくなったため，OSを研究する上でのテストベッドとはしがたくなった．

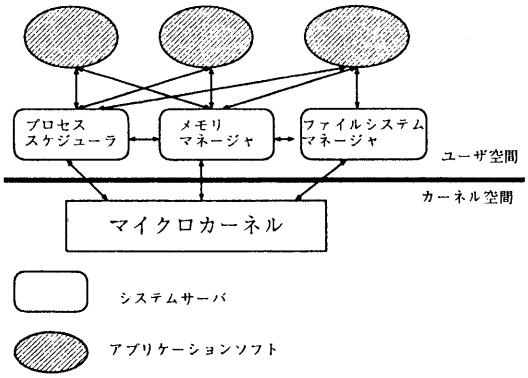

図1: マイクロカーネルの概念図

そこで，上記の欠点を克服するために，構成の見通しをよくし，修正・変更を容易にするマイクロカーネル構成のOSの研究が行われるようになった[1, 2, 3, 4, 5, 6].

Mach[6]に代表されるマイクロカーネル構成のOSでは，カーネルは必要最小限の機能のみを保持し，その他の様々な機能をカーネル外部でシステムサーバやモジュールとしてユーザ空間で運用している．そして，全体で1つのOS環境としてユーザに見せ，OSとしてのサービスを提供することで，ユーザの要求に応えようとするものである．これはまた，カーネル機能を分割しモジュール化することで，構成の見通しの良さを狙ったものである[1, 7, 8]．マイクロカーネル構成のOSの概念図を図1に示す．

マイクロカーネル構成のOSは，単層カーネル構成のOSでは実現が不可能であったアプリケーションに依存した規模のシステムの構築や，アプリケーションに適したリソース管理ポリシの選択やモジュールの最適化を行なうための基盤を提供する[9].

しかし，現在のマイクロカーネルの仕様では，以下のような欠点が存在する．

- カーネル機能変更時に伴うOSの再構築・再起動の必要性

  システムサーバとして実行しているカーネル機能の追加・変更・削除等を行なう場

合，一度カーネルを再構築し，OSを再起動し直す必要がある．

- OS機能のサービスの停止

  マイクロカーネル構成のOSの場合，システムサーバが何らかの原因で異常終了や停止をしてしまうと，その時点でシステムサーバが行なっていたサービスが停止し，最悪の場合カーネル自体の停止も引き起こしてしまう．

  単層カーネル構成のOSの場合，事態はさらに深刻で，あるカーネル機能の停止はOS自体の停止を意味している．

- 新機能の試験運用および性能比較

  従来のカーネル構成では，新たなカーネル機能を設計・実装した場合，その機能をテストするには，カーネルの再構築・再起動が必要であり，新旧のカーネル機能との機能比較を行なうのが非常に不便である．

- 不必要なサービスの実行

  カーネル内やシステムサーバ内でユーザが必要としないサービスを含んでいても，該当するサービスを停止または削除する等で冗長な部分をなくすことができない．

- システムサーバ実行状態の不適切さ

  ユーザが真に必要としかつ高速に処理して欲しい機能が，ユーザ空間でシステムサーバとして運用されている場合があり，ユーザの要求に適切に対応できるとは限らない．

- サービス処理にかかるコストの増大

  システムサーバとしてユーザ空間で運用している機能モジュールが実際に処理を行なおうとする場合，カーネルとの通信のオーバヘッドが大きく，また，アドレス空間の切替えにかかるコストも大きいので，処理のコストが大きくなる．

3 機能の動的構築が可能なOS

前述の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の欠点を回避・克服するために，OSおよびカーネルのあり方を再考し，より効率的なOSカーネルの構成を研究することが必要である．

そこで，カーネルが提供する機能を動的に構築することにより前述の欠点を改善することを検討した．カーネルとしての必要な機能を可能な限りシステムサーバとして運用し，必要に応じてそれらのシステムサーバをカーネル実行中に追加・置換・削除できるようにする．

具体的には，以下のような方法を提案する．

(1) カーネル機能の追加・置換・削除

カーネル機能をカーネル実行中に動的に追加・置換・削除し，カーネルを再構築・再起動する必要をなくす．

(2) 代替サービスへの運用変更

あるシステムサーバが異常終了や停止をすることがあっても，その代わりとなるシステムサーバへの動的置換を行ない，カーネル機能の提供を続行する．これにより，OSのフォールトトレランシを上げる．

(3) 新機能の試験運用および機能評価

カーネルを動的に構築することで，新たに設計・実装したシステムサーバと古いサーバとの同時実行が可能となり，比較・検討が容易になる．

ここでいうカーネル機能の追加・置換・削除とは，従来のマイクロカーネル構成のOS等のようにユーザ空間で1つのユーザプロセスとして機能していたシステムサーバを，以下のような方法で操作するという意味である．

これは，アドレス空間を越えた通信のオーバヘッドやアドレス空間切替えにかかるコストの削減を狙ったものである．

カーネル機能の追加・置換・削除の概念図を図2に示す．

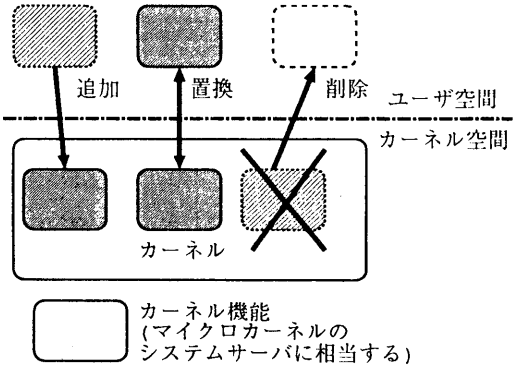

図 2: カーネル機能の追加・置換・削除

- 追加

  システムサーバを，ユーザ空間で運用していればカーネル空間へ移動させる．まだ運用していなければカーネル空間で起動し，カーネル空間で運用する．

- 置換

  一方がユーザ空間で運用され他方がカーネル空間で運用されている同等の機能を持つシステムサーバ同士を置換する場合，後者をユーザ空間へ移動させ，前者をカーネル空間へ移動させることで置換を行なう．

- 削除

  カーネル機能として不適切と考えられるモジュールを削除する場合は以下の 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

  (2) モジュールの停止

カーネルが提供する機能を動的に構築できれば，以下のような利点が生じる．

(1) 計算機資源管理の効率化

不必要となったハードウェアを制御していたモジュールや保持している機能自体が不必要となったモジュールを削除することで，計算機資源を効率良く管理・利用することが可能となる．

(2) ユーザ (アプリケーションソフト) の要求への柔軟な対応

アプリケーションやユーザが必要に応じてカーネルに機能変更の要求を出すと，カーネルはその要求に基づいてシステムサーバの追加・置換・削除を行ない，動的に環境を変更するので，アプリケーションやユーザの要求を柔軟に満たすことが可能となる．

(3) フォールトトレランシの向上

カーネルが動的にシステムサーバを置換するので，カーネルにシステムサーバの異常検知ルーチンを実装し，異常を検出したものを直ちに代わりとなるシステムサーバに置換させることで，フォールトトレランシを上げることが可能となる．

上記の方法が実現できれば，信頼性の高い，アプリケーションやユーザの側から見てさらに効率の良い OS となると考えている．

4 カーネル機能のモジュール化について

機能の動的構築が可能な OS を研究する上でカーネル機能のモジュール化は非常に重要な課題である．

そこで，オブジェクト指向の概念を採用し，カーネル機能をクラス定義し，その上でモジュール化することを検討した．

オブジェクト指向を導入する理由として，オブジェクト指向の概念である，

- 全ての資源をオブジェクトとして統一的に扱う
- クラス階層を構築して，資源を共有する

や，オブジェクト指向の利点である，

- モジュール化・カプセル化がしやすい

- 抽象化しやすく，分かりやすい

が，カーネル機能のモジュール化に非常に有効であると考えているからである．

カーネル機能のモジュール化を研究するために，以下のような事項を検討しなければならない．

(1) カーネル機能の限定

カーネルの機能をプロセス間通信のみに限定し，その他OSとして必要な機能を全てユーザプロセスとして実現できるかを研究する．

QNX[3] ではカーネルの機能としてプロセススケジューリングおよびプロセス間通信を提供し，その他の機能は全てユーザプロセスとして定義しているが，プロセススケジューリングもユーザプロセス空間で実行させることは可能かを研究する．

そのときに問題となるのが，従来のOSではプロセススケジューリングの際に行なわれるプロセスのコンテキストスイッチをカーネル内で行なっているということである．カーネルからプロセスのコンテキストスイッチをどのようにして分離するかを検討しなければならない．

(2) プロセスの定義の考察

プロセスをオブジェクトとして定義することを研究する．Chorus[2] では，アクターというオブジェクトがUNIXにおけるプロセスとして定義されているが，アクターと同等のプロセスを想定し，カーネルの動的構築に適した定義を研究する．

(3) プロセス間通信の考察

プロセス間通信の実装方法をオブジェクト指向のメッセージ通信で実現できないか研究する．オブジェクト指向のメッセージ通信が応用可能であれば，比較的容易にプロセス間通信の機能が提供できるのではないかと考えている．

(4) OS機能のクラス化

必要な機能全てをクラス (オブジェクト) として定義できないかを考察する．そうすることにより，カーネルおよびOSの機能のモジュール化がさらに容易になる．

(5) クラスで定義したOS機能のモジュール化

(4) においてクラス (オブジェクト) で定義したOSの機能を実際にOSの機能としてモジュール化し，システムサーバとしてカーネルから分離させることが必要となる．ここで考慮すべき点は，OS機能はクラス階層を構築して生成するので，各システムサーバ間で継承が存在する場合，メタクラスとサブクラス間で階層の一貫性を保持するように考慮しなければならない．不整合が発生した場合，各機能が正常に動作しないので，モジュール間で継承の不整合が発生することは避けなければならない．

(6) モジュールのサービス毎の定義

MinixやUNIXのようなOSの構成では，ユーザやアプリケーションソフトにOS機能を提供するのはシステムコールによるインタフェースによって行なわれるので，ユーザやアプリケーションソフトの実行効率に着目した場合，OS機能のみに重点を置いてモジュール化するよりも，提供するサービス (システムコール) にも重点を置いてモジュール化した方が効率が良いと考えている．

(7) モジュールの分割の大きさ

(6) で，システムコールなどのユーザに提供するサービス単位でモジュール化を検討するとしたが，モジュールの粒度を検討する必要がある．なぜなら，モジュール間の通信のオーバヘッドやアドレス空間の切替えにかかるコストは無視できないからである．よって，それらを可能な限り削減できる粒度を目指す．

(8) モジュール追加の方法

モジュールの追加はカーネル空間からユーザ空間へ移動させることで達成する．これは通信のオーバヘッドの削減と，アドレス空間切替えにかかるコストを削減するためである．

(a) カーネル機能として適切なモジュールの選択

どのモジュールがカーネル機能として追加するのにふさわしいか，または，どのモジュールをカーネル機能としてカーネル空間に移動させるべきかを決定するのは非常に重要な事項である．

SPIN[4] では，カーネル機能として追加するモジュールである Extender の適正は，開発言語である Modula-3 により型検査を行ない，決定している．

(b) モジュールの追加・置換・削除アルゴリズム

追加・置換・削除を行なうためのアルゴリズムを決定することも非常に重要である．

モジュールを追加・置換・削除するタイミングは非常に重要である．特に削除の場合は，削除対象となるモジュールが他からのサービス実行要求を受けている場合があるので，そういった場合の対処法も考えなければならない．

5 カーネル機能のクラス化・モジュール化

カーネル機能の動的構築が可能な OS の構成を研究する上での足掛かりとしてカーネル機能をクラス定義し，オブジェクトモジュールとして運用することを行なった．

具体的には，研究を進めていく上で OS のテストベッドとして Minix[10] を使用し，オブジェクト指向の概念を導入するためにオブジェクト指向言語の C++を用いて Minix を記述しなおした．

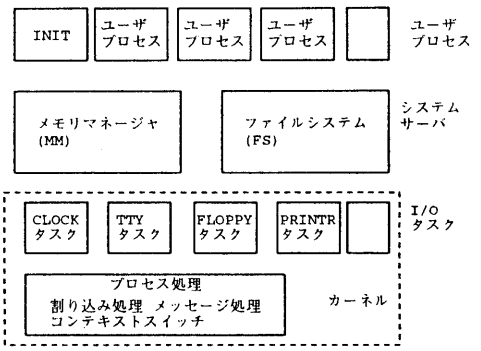

図 3: Minix の概念図

Minix の概念図を図 3に示す．

Minix は，カーネルのかなりの部分がすでにマイクロカーネル化されているので，カーネル機能のモジュール化の研究という側面から見て非常に適していると考えている．

また，オブジェクト指向言語 C++を用いているのは，オブジェクト指向におけるクラス階層の構築が，OS の機能の動的変更を研究する上で有効ではないかと考えているからである．カーネル機能をクラスで定義し実際の機能モジュールをオブジェクトとして生成し，オブジェクトモジュールとして運用することによりモジュール化を押し進めようというものである．

今回は Minix のシステムサーバであるメモリマネージャ(MM) およびファイルサーバ (FS) 内の各サービス機能のクラス階層を構築して定義し，各機能をオブジェクトモジュールとして運用することを行なった．

現状では各オブジェクトモジュールは MM および FS にリンクされ，MM や FS と同じアドレス空間で運用している．

MM および FS で定義したオブジェクトモジュールを以下に示す．

- MM で定義したオブジェクトモジュール

  Alloc, Break, Exec, Fork_exit, Getset, Putc, Signal, Trace, Utility

- FS で定義したオブジェクトモジュール

  Cache, Device, Filedes, Inode, Link, Misc, Mount, Open, Path, Pipe, Protect, Putc, Read, Stadir, Super, Time, Utility, Write

評価に使用した環境は，以下の通りである．

- ハードウェア

  SuperSPARC20

- OS

  SunOS-4.1.3_U1

- コンパイラ

  gcc-2.7.0 (オプティマイズなどのオプションは無し)

- Minix

  SunOS-Minix-1.5

評価として，以下のような実験を行なった．

通常のC言語で記述されたMinixと今回C++言語で書き換えたMinixとを比較検討するために，fork した後親プロセスと子プロセスで for ループを 1000 万回実行した時の親プロセス子プロセス両方の実行時間を 20 回実行し平均を出し，同じ操作を 3 回繰り返した後 3 回の平均を算出した．

fork システムコールは各種レジスタ，プロセステーブル，メモリの各処理を行なうためMinixが提供しているシステムコールの中で最もオーバヘッドがかかるものであり比較テストに適していると考えた．また，SPARC チップ自体の処理速度が非常に高速であるため，さらに比較テストを容易に行なえるように for ループを 1000 万回実行した．

実行時間の単位はSPARCにおけるハードウェアクロック割り込みの 1 tick=10(msec) としている．

ただし，今回使用した Minix は SunOS 上で 1 つのプロセスとして動作する SunOS-Minix を用いているので，クロック割り込みはハードウェアクロック割り込みではなく，SunOS からのシグナル割り込みであるが，ほぼSPARC のハードウェアクロック割り込みと同等の数値と判断して差し支えないと考えている．

測定結果を表 1 に示す．

表 1: 平均実行時間

平均実行時間	通常	変更後
親プロセス	20.42	20.25
子プロセス	20.27	20.37

この評価実験では，親プロセスの実行に関して，通常の C 言語で記述されたコードよりも，今回行なった MM および FS を C++言語を用いてクラス定義しオブジェクトモジュールを生成する構成の方が予想に反して良い結果が得られた．

この結果については，現在調査検討中である．今後，fork 以外のシステムコールや for ループ以外の関数等を実行し，今回の結果の分析を行なう予定である．

6 おわりに

本稿では，カーネル機能の動的構築が可能な OS の構成を提案し，その一環として OS にオブジェクト指向の概念を導入し，各カーネル機能をクラス階層を構築して定義し，オブジェクトモジュールとして運用することを行なった．

今後は，まず現状で MM や FS と同じアドレス空間で運用している各オブジェクトモジュールを MM や FS からは切り離し独立した 1 つのプロセスとして運用することを行ない，最終的には各オブジェクトモジュールをカーネル機能として動的に追加・置換・削除可能にしたいと考えている．

参考文献

[1] D.L.Black et al. : Microkernel Operating System Architecture and Mach, Proc. of the USENIX Workshop on Micro-kernels

and Other Kernel architectures, pp.11-30 (1992).

[2] 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).

[3] 鈴木治 : 次世代 OS アーキテクチャ, 工学社 (1994).

[4] Brian N.Bershad et al.: SPIN - An Extensible Microkernel for Application-specific Operating System Services , Proc. of SIGOPS European Workshop, pp.74-7 (1994).

[5] J. Mitchell et al.: An Overview of the Spring System, Proc. of Compcon Spring 1994, pp.122-31 (1994).

[6] 乾 和志, 菅原 圭資:分散OSMach がわかる本, 日刊工業新聞社 (1992).

[7] R.Zajcew et al. : An OSF/1 UNIX for massively Parallel Multiprocessors, Proc. 1993 Winter USENIX, pp.449-468 (1993).

[8] D.P.Julin : Generalized Emulation Services for Mach 3.0 - Overview, Experiences and Current Status, Proc. of the USENIX Mach Symp., pp.13-26 (1991).

[9] 中島達夫, 保木本晃弘: 移動計算機環境におけるアプリケーションとオペレーティング・システム支援, Jouhoushori, Information Processing Society Japan, (1994).

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

[11] 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).

[12] Y. Yokote:The Apertos reflective operating system: the concept and its implementation,SIGPLAN Not. (USA), vol.27, no.10, pp.414-34 (1992).