システムソフトウェアと
オペレーティング・システム 69－8
（１９９５．６．２３）

M^3K プロトタイプシステムの設計

追川 修一　　徳田 英幸

慶應義塾大学環境情報学部
神奈川県藤沢市遠藤 5322

あらまし　現在，アプリケーションの要求は多岐にわたり，一つのポリシー，メカニズムのみが提供される現在のカーネルでは，要求を十分に満たすことはできない．例えばモーバイルシステム上のアプリケーションは，ネットワークとの接続や切断，バッテリ容量の減少といった状況の変化に対応しなければならない．その対応方法はアプリケーションの種類によって異なり，一つのカーネルが全ての対応方法に適した機能を提供することは難しい．従って，アプリケーション自身が，必要とする機能をカーネルに提供することができる必要がある．現在，そのような要求に応えることのできる拡張可能なマイクロカーネル M^3K の研究開発を進めている．本論文では，M^3K のプロトタイプの設計について述べる．

キーワード　マイクロカーネル，オペレーティングシステム，分散システムマルチメディア，リアルタイムシステム．

The Design of M^3K Prototype System

Shuichi Oikawa　　Hideyuki Tokuda

Faculty of Environmental Information
Keio University
Endo 5322, Fujisawa-shi, Kanagawa, 252 Japan

Abstract　Recent applications require a wide variety of functionalities for their efficient execution. Their requirements, however, cannot be satisfied by current kernels which can provide only a single set of policies or mechanisms. For example, applications on mobile systems need to adapt changes in their situations, such as connection/disconnection to a network and the rest of battery life. A way to adapt such changes is different for each application. The single kernel, however, cannot provide all of necessary functionalities for them. Then, applications should provide modules to implement necessary mechanisms. Currently, we are working on a extensible microkernel M^3K for such purpose. This paper describes the design of its prototype.

Keyword　Micro-kernel, Operating System, Distributed System Multimedia, Real-Time Processing.

1 はじめに

これまでのオペレーティングシステムカーネルは，固定されたアブストラクション，そしてそれをサポートする固定されたポリシー，メカニズムのみを提供している．例えばUNIXでは，実行制御のアブストラクションとしてプロセスを提供し，プロセスをサポートするためにスケジューリング，仮想記憶のメカニズムが作られている．マイクロカーネルを基にしたアーキテクチャでは，オペレーティングシステムの持つパーソナリティをカーネルの外に出すために，より一般的なアブストラクションを提供している．そのために，提供されるアブストラクションはより細かく分割されたものになり，それをサポートするために細粒度のインタフェースが提供されている．例えばMach3.0[5]では，UNIXのプロセスは資源管理の単位であるタスクと実行制御の単位であるスレッドに分割されている．しかし，これらはカーネルによって提供される固定されたアブストラクションであることには変わりない．

固定されたアブストラクション，ポリシー，メカニズムが提供されれば，それらはユーザの持つ要求の最大公約数的なものになってしまう．これまでの計算機環境において，テキストデータを処理する上では，一つの決まったポリシーによる資源の公平な共有は十分に機能してきた．

現在，アプリケーションの要求は多岐にわたり，一つのポリシー，メカニズムのみでは，要求を十分に満たすことはできない．例えばモーバイルシステム上のアプリケーションは，ネットワークとの接続や切断，バッテリ容量の減少といった状況の変化に対応しなければならない．その対応方法はアプリケーションの種類によって異なり，カーネルが全ての対応方法に適した機能を提供することは難しい．従って，アプリケーション自身が，必要とする機能をカーネルに提供することができる必要がある．

以上のように，これから重要性がさらに増してくる環境をサポートするには，固定されたアブストラクション，ポリシーのみをサポートしたカーネルでは不十分である．アプリケーション自身がその実行に必要なポリシーやメカニズムをカーネルに提供することができれば，そのアプリケーションに適した実行環境を作ることができる．

そのような要求に応えることのできる拡張可能なマイクロカーネル M^3K を提案し，現在 RT-Mach[15] を基にプロトタイプの開発を進めている．プロトタイプでは，カーネル機能拡張メカニズム，カーネル拡張のためのフレームワークの開発を目的としており，そのためにLKM (Loadable Kernel Module) [14] の導入，カーネルのモジュール化を行なう．本論文では，そのプロトタイプの設計概要について述べる．

2 カーネルの拡張

2.1 静的拡張／動的拡張

カーネルの拡張は，静的な拡張と動的な拡張とに分類することができる．静的な拡張とは，カーネルのリンク時に新しい機能を追加することである．決まったアプリケーションのみが実行される場合には有効であるが，新しいアプリケーションのサポートには，カーネルの再リンク，新しいカーネルを用いてのリブートが必要である．既にUNIX，Machといった既存のカーネルの多くで可能である．

動的な拡張とは，実行中のカーネルに対し新しいモジュールの追加を行なうことである．必要に応じてカーネルのコンフィグレーションを変更することができるため，将来的には使用されるかもしれないが，現時点では使用されない無駄なモジュールをカーネルに入れる必要がなくなり，資源の無駄がなくなる．デバイスドライバなど部分的な拡張は，一部のカーネルで可能になってきている．動的拡張のためのメカニズムについては，次節で詳しく述べる．

ここでは，アプリケーションの要求に応じたカーネルの機能拡張を可能にするため，動的な拡張を対象とする．

2.2 動的機能拡張メカニズム

- LKM (Loadable Kernel Module) [14]
  SunOSにおいて，主にデバイスドライバ，ファイルシステムタイプ，システムコールの追加に使用されている．カーネル拡張のための基本的な機能のみを提供しており，カーネルのリコンフィグレーションと基本的には同等である．追加するモジュールからはカーネル内の全ての資源にアクセスすることができため，モジュールのバグによって，システムのクラッシュが引き起こされることもあり得る．NetBSD[11]，FreeBSD[4]にも実装されている．

- Server Co-Location [3]
  OSF/1 マイクロカーネルにおいて，システム全体のパフォーマンスを向上させ，モノリシックカーネルと同等の性能を得るために開発された．一般にユーザレベルで実行されるサーバをカーネル内で実行させる．マイクロカーネルアーキテクチャでのサーバの機能をカーネル内に取り込むため，カーネルの機能を拡張していると言えるが，マイクロカーネル自体に新たな機能を追加しているわけではない．カーネル内サーバとマイクロカーネルは，サーバがユーザレベルで実行されている場合と同じくRPCにより通信する．この場合のRPCは，全てカーネル内で

実行されるため，それに合わせて最適化されている．

- Apertos [6]
  Apertos オペレーティングシステムでは，オブジェクト指向リフレクティブ計算モデルを用い，デバイスドライバを再構成可能にしている．Apertos の持つリフレクティブアーキテクチャを生かし，デバイスドライバオブジェクト専用の実行環境 DDSL (Device Driver Service Layer) を導入することにより，プログラミングの労力を軽減し，かつデバイスドライバの実行安全性を確保している．

- SPIN [1]
  SPIN オペレーティングシステムでは，カーネル内に Spindle (SPIN Dynamically Loaded Extensions) と呼ばれるモジュールをロードすることにより，システムオーバヘッドを減少させ，アプリケーションの実行効率を向上させている．Spindle はアプリケーションが提供することができ，システムクラッシュを引き起こすようなカーネル内資源へのアクセスや操作を防ぐため，型安全なコンパイラを導入している．

ここでは，カーネル拡張のためのメカニズムの例として，FreeBSD[1]の LKM で用いられている実装方法について詳しく述べる．

LKM のサポートのために，`/dev/lkm` というデバイスが用意されている．LKM の導入，排出は，`modload`, `modunload` コマンドにより行なう．

`modload` コマンドは，`/dev/lkm` デバイスへの `ioctl` を通して，導入するモジュールのためのメモリ領域の確保[2]，使用すべき開始アドレスの取得を行なう．次に，

```
ld  -e _entry -T start_addr -o output_file
    -A /kernel lkm_module
```

により，カーネルとリンクし，モジュール内のシンボルの解決を行なう．リンクされたモジュールはファイル (*output_file*) に書き出さる．そして，再び `ioctl` を通して，カーネル内へモジュールのテキスト領域，データ領域の転送を行なう．転送終了後，モジュールのエントリを `ioctl` を通してカーネル内で呼び出し，モジュールの初期化を行なう．

カーネルは，同時に複数のモジュールを導入することはできないので，`/dev/lkm` が open される時に排他制御をしている．カーネル内には，LKM を管理するための構造体の配列を用意され，その配列のインデックスが導入時にモジュールの ID として返される．モジュールの排出は，その ID を指定することで，モジュールを特定することができる．

問題となるのは，モジュールの初期化を行ない，どのようにカーネルにモジュールを認識させるかということである．LKM では，カーネル内のテーブルに導入した LKM を呼び出すための関数ポインタを埋め込むための関数を，システムコール，VFS，デバイスドライバ，ストリームのために用意している．LKM プログラマは，これらの関数を初期化ルーチン内から呼びだし，適当なテーブルに関数ポインタを埋め込むことになる．従って，UNIX カーネル内でモジュール化の進んだ部分に対し，限られたインタフェースを持つモジュールを組み込むことができるに過ぎない．

動作中のカーネルにモジュールを組み込むためのプリミティブなメカニズムは提供しているという点からは，他のカーネルへの移植は容易であると言える．

3 プロトタイプの設計

3.1 目的

M^3K の達成すべき最終的な目的として，

- 新しい環境，アプリケーションへの適応
- カーネルのモジュール化

といった項目が挙げられる．

これからは，従来のカーネルが想定していなかった新しい計算機環境，アプリケーションに適した実行環境を提供していかなければならない．新しい計算機環境，新しいアプリケーションとして考えられているのは，

- モーバイル環境
- マルチメディアアプリケーション
- 並列アプリケーション

といったものである．これらを，従来の固定された一つのポリシー，メカニズムの基で効率良く実行するのは難しく，効率の良い実行には，アプリケーション自身によるサポートが必要になっている．

様々な要求に応えるための機能をカーネル内に盛り込んでいくと，たとえマイクロカーネルであってもカーネル自身が大きくなり過ぎてしまう．例えば，標準的な最小構成のマシンのデバイスドライバを含む i386 アーキテクチャ用 Mach3.0 MK83 の a.out 形式のファイルの大きさは 400KB 程である．これに，RT-Mach や NORMA などで追加された機能や他の一般的なデバイスのドライバを追加した場合，735KB と倍近くになる[3]．ブート時

[1] FreeBSD と NetBSD の LKM の実装は，ほぼ同一である．

[2] `ld` コマンドを実行してみなければ，確保すべきメモリ領域の大きさがわからないため，適当な開始アドレスを用いて一度プレリンクを行なう必要がある．

[3] `size` コマンドの出力は，以下の通りである．

	text	data	bss	dec
Mach3.0	290784	29164	49988	369936
RT+NORMA	548832	40568	639736	1229136

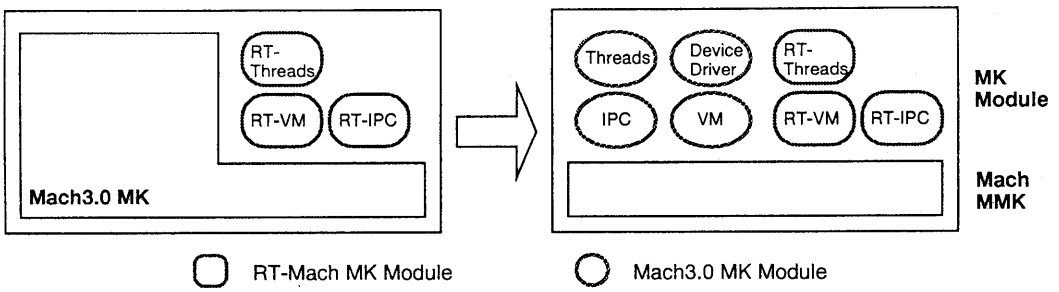

図 1: プロトタイプカーネルのソフトウェアアーキテクチャ

に管理構造体用にとられる領域などもあるため，実際に使用されるメモリ領域は，数倍になると思われる．これに OSF による拡張や Utah 大学の Flex Mach プロジェクト [2] の成果などを取り込んでいけば，さらに大きなものになってしまう．

従って，実際にアプリケーションの必要な機能のみを取り入れたマイクロカーネルを構成可能にし，資源を有効に活用できるようにすべきである．また，今後の機能追加を行ない易くするためにも，ポリシーやメカニズムをモジュール化し，同等の機能を持つモジュールに対するインタフェースを定義する必要がある．

しかし，最初からこれらを達成することのできるマイクロカーネルを構築することは難しい．そこで，必要な機能の一部を持つプロトタイプを構築し，そのためのアーキテクチャ，メカニズムについて検討する．

ここで構築するプロトタイプの目的は，以下の通りである．

- カーネル機能拡張メカニズムの開発
  LKM などの動的カーネル機能拡張メカニズムを RT-Mach に導入する．LKM は UNIX での使用を前提に設計されており，そのまま RT-Mach に適用することも可能ではあるが，マイクロカーネルアーキテクチャとして一般的に有用なものになるとは考え難い．従って，マイクロカーネルアーキテクチャに適した形態の LKM をサポートするアーキテクチャ，メカニズムの開発が必要である．

- カーネル拡張のためのフレームワークの開発
  LKM に見られるように，LKM からはカーネルの全ての資源にアクセスできるようになっていては，機能拡張モジュールの導入は安全ではあり得ない．他のメカニズムのように，限られた形でのみカーネルの機能，資源がアクセスできるべきである．そして，機能追加のための明確なインタフェースが定義されているべきである．

プロトタイプは，RT-Mach[15] を基に開発する．RT-Mach は Mach3.0[5] を基に開発され，リアルタイムスレッド，リアルタイムスケジューリング，リアルタイム IPC，リアルタイム同期の機能が追加されたマイクロカーネルである．既にこれらの豊富なリアルタイム機能が盛り込まれているため，プロトタイプではこれらのモジュール化を行ない，それを通して目的の達成を目指す．

3.2 ソフトウェアアーキテクチャ

Mach3.0，RT-Mach マイクロカーネルの機能は，大きく以下の 4 つに分けることができる．

- IPC
- 仮想記憶
- スレッド／スケジューリング
- デバイスドライバ

Mach3.0 はこれら全ての機能を提供し，完全なマイクロカーネルを構成している．RT-Mach は，Mach3.0 にリアルタイムシステムを構築するための機能を追加したものである．IPC，スレッド／スケジューリング，仮想記憶にリアルタイム機能の追加がされている．

マイクロカーネルのモジュール化の最初のステップは，リアルタイム機能に対して行ない，

- Mach3.0 MK
- RT-Mach 機能拡張モジュール

から，RT-Mach を構成するようにする．RT-Mach 機能拡張モジュールは，必要な機能のみを実際にカーネル内に取り込むことができるように，それぞれ IPC，スレッド／スケジューリング，仮想記憶の機能拡張を含む複数のモジュールに分割する（図 1 左側）．

次に，さらにモジュール化を進め，マイクロカーネルのポリシー，メカニズムを分離することにより，マイクロカーネルは MMK (Meta Micro Kernel) と MKM (Micro

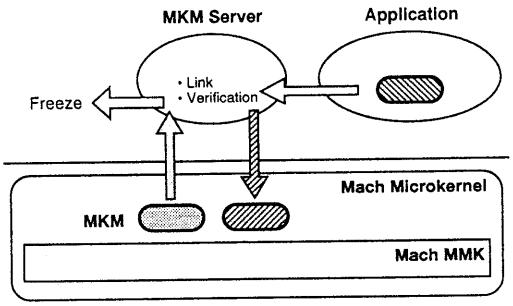

図 2: MKM サーバ

Kernel Modules) から構成されるようにする．Mach3.0 のポリシー，メカニズムは，Mach3.0 MKM により実現され，RT-Mach のポリシー，メカニズムは，RT-Mach MKM により実現される．即ち，

Mach3.0 MK = Mach MMK + Mach3.0 MKM
RT-Mach MK = Mach MMK + RT-Mach MKM

という形態になる（図 1 右側）．

このように，機能を複数のモジュールに分割すると，それぞれのモジュール間に依存関係が生じる．例えば，Mach の IPC は仮想記憶と密接に関係しているため，IPC を使用するためには，仮想記憶モジュールは必須となる．従って，このような依存関係を記述し，そして必要なモジュールを特定するためのメカニズムが必要である．

3.3 MKM の導入／排出

MKM の導入／排出には，それをサポートするサーバを使用する（図 2 参照）．そのためのメカニズムを全てマイクロカーネル内に組み込み，アプリケーションが直接 MKM をマイクロカーネルにロードする方法も考えられるが，拡張性，セキュリティに問題がある．また，カーネル内の機能を増やすことは，マイクロカーネルアーキテクチャや M^3K の目的とも調和しない．

MKM サーバは，アプリケーションまたはカーネルの要求により，MKM をカーネル内に導入する．MKM の持つ依存関係は，カーネルとの協調の基，MKM サーバにより必要な MKM が導入される．また，資源の減少などの理由により一時的に不必要になり排出された MKM の保存も行なう．

MKM をカーネルに導入する場合，MKM サーバは

- MKM の検査
- MKM とカーネルのリンク

を行なう．MKM は，カーネルとのインタフェースは制限されているが，LKM と同様に直接カーネル内の資源にアクセスする．そのため，MKM に含まれる未解決シンボルの解消を行なう必要がある．MKM サーバは，カーネルのシンボルテーブルを利用して，MKM のカーネルとのリンクを行なう．シンボルテーブルは，Mach マイクロカーネルがカーネルデバッガのために持っているものを MKM サーバのアドレススペースにマップすることで，使用できるようになる．

リンク時に，MKM に含まれる未解決シンボルが，カーネルが MKM に提供するインタフェースに限られているかどうかを検査する．それ以外のシンボルを含む場合は，カーネル内の資源に不正にアクセスする可能性があるため，カーネルへの導入を拒否する．正しいインタフェースを使用していても，プログラムのバグなどで，カーネル内のデータを破壊し，クラッシュさせてしまう可能性がある．これは，ソフトウェアフォールトアイソレーション [16] の技術を使用することで，回避することができる．

3.4 MKM の呼出

カーネルから MKM を呼び出せるようにするには，カーネルから MKM の持つインタフェースへの参照を追加する必要がある．LKM では，導入された LKM の初期化時に，LKM 自身によりカーネル内部のテーブルへのインタフェースの追加が行なわれる．最終的には，同様のことが行なわれなければ，カーネルから MKM を参照することはできないが，MKM が直接テーブルを変更するのは，テーブルを破壊してしまう可能性もあるので，避けるべきである．

逆に，MKM からカーネルにインタフェースのリストを渡し，カーネルがそれを登録するようにする．MKM サーバがインタフェースのリストを取り出し，それをカーネルに渡すようにすれば，少なくとも正しいインタフェースのリストが MKM によって提供されているかどうかも，サーバで検査することができる．

4 まとめ

本論文では，拡張可能なマイクロカーネル M^3K のプロトタイプの設計について述べた．M^3K は，アプリケーション自身がその実行に必要なポリシーやメカニズムをカーネルに提供し，そのアプリケーションに適した実行環境の実現を可能にすることを目的としている．また，そのためマイクロカーネル自身のモジュール化を進めることが必要になってくる．

M^3K のプロトタイプは RT-Mach を基に，主にモーバイル環境をターゲットとして研究開発を進めている．カーネル機能拡張メカニズムの開発，カーネル拡張のためのフレームワークの開発を目的とし，LKM (Loadable Kernel

Module) の導入，MKM (Micro Kernel Module) を用いたカーネルのモジュール化を行なう．MKM のカーネルへの導入，排出には，それをサポートする MKM サーバを用いる．

謝辞

この研究は，情報処理振興事業協会 (IPA) が実施している独創的情報技術育成事業「モーバイルコンピューティングのための動的適応可能なソフトウェアアーキテクチャ」プロジェクトのもとに行なわれている．

参考文献

[1] B. Bershad, C. Chambers, S. Eggers, C. Maeda, D. Mcnamee, P. Pardyak, S. Savage, and E Sirer. SPIN - An Extensible Microkernel for Application-specific Operating System Services. Technical Report UW-CSE-94-03-03, Department of Computer Science and Engineering, University of Washington, March 1994.

[2] J. Carter, B. Ford, M. Hibler, R. Kuramkote, J. Law, J. Lepreau, D. Orr, L. Stoller, M. Swanson. FLEX: A Tool for Building Efficient and Flexible Systems. In *Proceedings of the 4th Workshop on Workstation Operating Systems*, October 1993.

[3] M. Condict, D. Mitchell, and F. Reynolds. Optimizing Performance of Mach-based Systems by Server Co-Location: A Detailed Design. OSF Research Institute, Cambridge, MA, August, 1993.

[4] FreeBSD Project. *http://www.freebsd.org/*.

[5] D. Golub, R. Dean, A. Forin, and R. Rashid. Unix as an Application Program. In *Proceedings of the Usenix Summer Conference*, June 1990.

[6] J. Itoh, M. Tokoro. Object-Oriented Device Driver Programming. Japan Society for Software Science and Technology Workshop on Object-Oriented Computing '94, March 1994.

[7] J. Mitchell, J. Gibbons, G. Hamilton, P. Kessler, Y. Khalidi, P. Kougiouris, P. Madany, M. Nelson, M. Powell, and S. Radia. An Overview of the Spring System. In *Proceedings of Compcon Spring 1994*, February 1994.

[8] A. Montz, D. Mosberger, S. O'Malley, L. L. Peterson, T. Proebsting, J. Hartman. Scout: A communications-oriented operating system. Technical Report 94-20, Department of Computer Science, University of Arizona, June 1994.

[9] T. Nakajima, T. Kitayama, H. Arakawa, and H. Tokuda. Integrated Management of Priority Inversion in Real-Time Mach. In *Proceedings of the 14th IEEE Real-Time Systems Symposium*, December 1993.

[10] 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．情報処理，Vol. 35, No. 12, 1994.

[11] NetBSD Project. *http://www.netbsd.org/*.

[12] B. Noble, M. Price, M. Satyanarayanan. A Programming Interface for Application-Aware Adaptation in Mobile Computing. 6th ACM SIGOPS European Workshop, December 1994.

[13] M. Satyanarayanan, B. Noble, P. Kumar, M. Price. Application-Aware Adaptation for Mobile Computing. Technical Report CMU-CS-94-183, Carnegie Mellon University, School of Computer Science, 1994.

[14] Sun Microsystems. *SunOS 4.1.3 Reference Manual*. 1991.

[15] H. Tokuda, T. Nakajima, and P. Rao. Real-Time Mach: Towards a Predictable Real-Time System. In *Proceedings of USENIX Mach Workshop*, October 1990.

[16] R. Wahbe, S. Lucco, T. Anderson and S. Graham. Efficient Software-Based Fault Isolation. In *Proceedings of Fourteenth ACM Symposium on Operating System Principles*, December 1993.

[17] M. Weiser. Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, Vol. 36, No. 7, July 1993.

[18] Y. Yokote, F. Teraoka and M. Tokoro. A Reflective Architecture for an Object-Oriented Distributed Operating System. In *Proceedings of European Conference on Object-Oriented Programming*, March 1989.