M^3K: 拡張可能なマイクロカーネル†

3H－1

追川 修一　西尾 信彦　徳田 英幸

慶應義塾大学環境情報学部

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をユーザに提供してきた．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，実行に必要な資源管理はプロセスを単位として行なっている．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し，様々なパーソナリティを受け入れることができるようにしている．そのためにさらに細かい制御を可能にしているが，固定されたアブストラクションを提供していることには変わりない．例えばMach[3]では，資源管理の単位であるタスク内に，複数の実行制御の単位であるスレッドが存在することを許している．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．カーネルの提供するアブストラクションを制御するポリシーは，ユーザの持つ要求の最大公約数的なものである．これまでのTSS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り資源を公平に共有することが有効であった．

近年のハードウェアの進歩により，汎用のワークステーションにおいても，並列アプリケーションが実行されたり，マルチメディア処理が行なわれるようになってきた．これらのアプリケーションを対象とした時，従来のTSS環境を前提として固定されたポリシーは必ずしも有効でなくなってしまう．並列アプリケーションは，それを実行するスレッドのスケジューリングを，コンパイラの解析結果に従って細かく制御できることが望ましい．しかし，スレッドがカーネルによって提供されるような場合は，それが不可能になってしまう．それを避けるために，アプリケーション内でスレッドを実現するユーザレベルスレッドやファーストクラスユーザレベルスレッド[1]が開発されている．マルチメディア環境において取り扱うデータは，サイズの大きさと時間的な制約を特徴として持つ．これらの特徴は，TSS環境とはかなり異なった資源管理ポリシーを要求する．例えば，一般的に仮想記憶において新たに物理メモリページが必要となった場合，LRUアルゴリズムに従って最近最も使われていないページを再利用しようとする．しかし，マルチメディアデータの場合は，ある時間を過ぎたデータは意味を持たなくなってしまっているため，LRUとは違ったポリシーによって管理される必要が出てくる．

以上のように，これから重要性がさらに増してくるアプリケーション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資源管理に必要なポリシーや機能をカーネルに提供し，カーネルはそれに従って資源管理を行なえば，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，より効率良い実行が可能になる．我々は，アプリケーション毎に適した実行環境を構築することを可能にする拡張可能なマイクロカーネルM^3Kを提案し，必要なメカニズムの構築，実装を行なう．本論文では，機能拡張を可能にするM^3Kのアーキテクチャについて述べる．

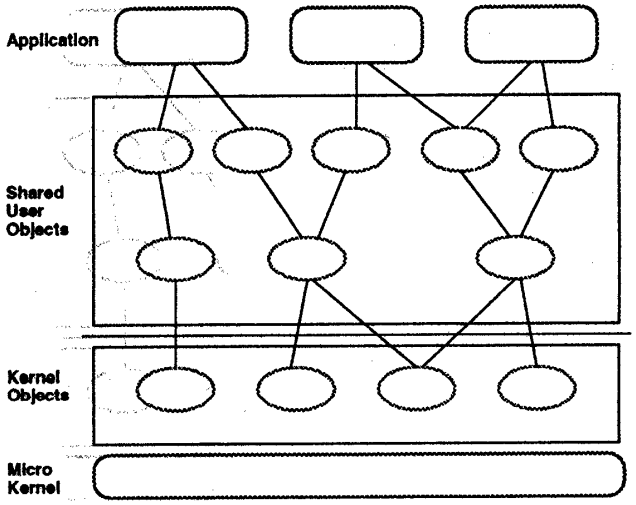

図1: ソフトウェアアーキテクチャ

2 ソフトウェアアーキテクチャ

M^3Kのソフトウェアアーキテクチャは，次の要素からなる（図1参照）．

- アプリケーション
- 共有ユーザオブジェクト
- カーネルオブジェクト
- マイクロカーネル

アプリケーションは，必要な共有ユーザオブジェクト，カーネルオブジェクトの機能を利用して，自身に適した実行環境を作ることができる．共有ユーザオブジェクトは，ユーザ空間に存在し，その機能を必要とするアプリケーションの間で共有される．カーネルオブジェクトは，カーネル空間内に存在し，共有ユーザオブジェクトと同じく，アプリケーション間で共有される．共有ユーザオブジェクトはユーザモードで実行され，カーネルオブジェクトは特権モードで実行される．

共有ユーザオブジェクトは，他のアプリケーションと共有され得るため，ユーザ空間中に連続してそのための空間（共有ユーザオブジェクト空間）を持つ．ある共有

"M^3K: An Extensible Microkernel"†
Shuichi Oikawa, Nobuhiko Nishio, and Hideyuki Tokuda
Faculty of Environmental Information, Keio University, 5322, Endo, Fujisawa-shi, Kanagawa, 252 Japan

† 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「M^3K：超分散超並列環境のためのオペレーティングシステムカーネル」プロジェクトのもとに行なわれた．

ユーザオブジェクトが，複数のアプリケーションと共有されている時，そのオブジェクトはそれぞれのアプリケーションの仮想アドレス空間で，同一の開始アドレスを持つ．即ち，共有ユーザオブジェクト空間は，すべての仮想アドレス空間で共有している．

共有オブジェクトは，アプリケーションの実行環境を提供するためだけでなく，一般にライブラリやマイクロカーネルにおけるサーバとして実現されてきた，他のアプリケーションと共有することができる機能を提供するためにも使用される．共有オブジェクト（の集合）により，マイクロカーネルにおけるサーバの機能を実現することにより，マイクロカーネルでは頻繁に起る異なったアドレス空間に存在するサーバーとの間のIPCを減少させることができる．

共有オブジェクトは，ユーザ空間内，カーネル空間内のどちらかに配置することができる．どちらに配置するかは，共有オブジェクトの実現時に決定される．共有オブジェクトの機能により，どちらに配置するべきかは自ずと決まってくると考えられる．アプリケーションは，独自に共有オブジェクトをシステムに導入することができるが，その場合に配置できるのは，ユーザ空間内のみである．これは，ユーザが定義した共有オブジェクトによるシステムのクラッシュを避けるためである．

マイクロカーネルから，アプリケーションが提供するポリシーや機能を提供する共有オブジェクトを利用できるようにするためには，そのためのインタフェースが必要となる．機能拡張を行なうためには，オブジェクト指向の考え方が有効である．そこで，基礎となるインタフェースを持つクラスを定義する．共有オブジェクトは，そのクラスをスーパークラスとするクラスのインスタンスになるようにすることにより，すべての共有オブジェクトが共通のインタフェースを持つことができる．

3 メカニズム

上記のアーキテクチャを実現するために必要なメカニズムとして，次のものがあげられる．

- 共有オブジェクトのローディング
- 共有オブジェクトのリンケージ
- 共有オブジェクトの保護

アプリケーションが必要とする共有オブジェクトが，共有オブジェクト空間に存在しない場合，新たにロードする必要がある．共通に利用されているファイルシステムが存在する場合，そのファイルシステムから共有オブジェクトをロードすることができる．共有オブジェクト空間内に必要なメモリ領域を割り当て，そこに共有オブジェクトのオブジェクトイメージをロードできる．ファイルシステムをサポートする共有オブジェクトなど，システムのスタートアップに必要な共有オブジェクトは，カーネルのブートイメージの中に組み込むことにより，アドレス空間にロードすることができる．

ロードした共有オブジェクトは，他のオブジェクトの配置にあわせて，未解決シンボルのリンクをする必要がある．例えば，あるクラスのサブクラスのインスタンスである共有オブジェクトは，スーパクラスのコードへの参照を含んでいる．それらは，ロードされるまでは未解決のままである．ロード時に，それらのシンボルのリンクを行なう必要がある．もし，参照先のオブジェクトが存在しない場合は，そのオブジェクトをロードする必要がある．ブートイメージに組み込まれる共有オブジェクトは，組み込み時に未解決シンボルのリンクを行なう．

共有オブジェクトは，アプリケーションや他のオブジェクトから保護されなければならない．言語的には，共有オブジェクトはC++で実装される予定であるが，それぞれのオブジェクトの内部状態を変更するためには，そのオブジェクトで定義されるそのための関数を呼び出すことによって行なうものとする．そのようにすることにより，Sandboxing[6] や ARPC[7] などのような，ソフトウェアによるフォールトアイソレーションのメカニズムを用いることができる．言語的には，内部状態を “private” として実装することにより，そのクラスに実装されるメンバ関数を通してのみ変更できるようにできる．そのメンバ関数で変更できる値かどうかのチェックを行うことができる．

4 今後の予定

今後の予定として，M^3Kにおける共有オブジェクトのクラス階層の詳細な定義を行ない，検討を行なう．並行して，必要なメカニズムの実装に関する検討を行なう．

謝辞

研究を進めるにあたって，有益なコメントを頂きました北陸先端大学院大学中島達夫助教授に深く感謝します．

参考文献

[1] T. E. Anderson, B. N. Bershad, E. D. Lazowska, and H. M. Levy. Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism. In *Proceedings of the 13th Symposium on Operating System Principle*, October 1991.

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] C. Yarvin, R. Bukowski, and T. Anderson. Anonymous RPC: Low Latency Protection in a 64-Bit Address Space. In *Proceedings of 1993 Summer USENIX Conference*, June 1993.

[8] 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.