システムソフトウェアと
オペレーティング・システム 75－1
（1997．6．5）

次世代OSのためのマイクロカーネルトレイアーキテクチャ

盛合敏　　徳田英幸
NTT情報通信研究所　慶應義塾大学

概要: 本報告では，次世代OSのための新しいカーネルアーキテクチャであるマイクロカーネルトレイアーキテクチャを提案する．我々の目標は新しいカーネルの基礎となる再構成および動的拡張が可能なマイクロカーネルとアーキテクチャニュートラルなOS作成キットを構成できるようにするところにある．マイクロカーネルトレイはマイクロカーネル型システムにおけるソフトウェアバックプレーンバスの役割を果たす．OS作成キットを利用することでOS開発者がOSのより重要な部分へ力を注ぐことができるようになる．本報告では研究の最初のステップであるアーキテクチャの設計について述べる．

Micro-Kernel Tray Architecture for the Next Generation Operating Systems

Satoshi Moriai　　Hideyuki Tokuda
NTT Laboratories　　Keio University
moriai@isl.ntt.co.jp　hxt@sfc.keio.ac.jp

Abstract. In this paper, we propose a new kernel architecture, the micro-kernel tray architecture. Our goal is to provide a new kernel foundation that enables an application to extend kernels dynamically and also enables an operating system builder to create a micro-kernel construction kit similar to a compiler construction kit. A micro-kernel tray provides a software backplane bus for micro-kernel based systems. The kernel construction kit would save even more time and would also enable the developer to focus her/his effort on the interesting aspects of operating systems. We report the first step of our research, the design of the micro-kernel tray architecture.

1　はじめに

カーネルモジュールの動的なカーネルへのロードが可能であれば，オペレーティングシステム(OS)を動的に拡張することができる．現代的なOSではカーネルモジュールの動的なロード機能を持っており，この機能はマイクロカーネルベースのシステム[1,5,16]においても有用である．マイクロカーネルシステムでは，カーネル上に様々なOSパーソナリティを構築することが可能であり[7,12,13,8]，さらにライブラリを組み合わせることで，分散共有メモリなどのより高度な機能を提供することも可能である[11]．しかし，常にアプリケーションから要求される機能をユーザレベルで効率良く実現できるとは限らない[14]．マイクロカーネルシステムにおいても，そのモジュラリティをそこなうことなくカーネルそのものの拡張が容易であることが望ましい．

現代的なOSはメカニズムとポリシが分離された固定のアブストラクションの集合によって構成されている．メカニズムは資源を管理したりサービスをアプリケーションに提供するための枠組を提供する．そして，ポリシは資源を管理するアルゴリズムや方策を提供する．このようにポリシとメカニズムを分離することでより柔軟なシステムとなる．つまり，アプリケーションはカーネルに組み込まれているポリシモジュールの一つを選択したり，新たなポリシモジュールをカーネルに組み込むことができる．例えば，新たなファイルシステムモジュールをファイルシステム機構に組み込むことが可能であったり，モバイルアプリケーションは自身の走行環境，つまり，ネットワークの接続状況，バッテリレベル，ハードウェアの構成等に追従して資源の管理ポリシを変えることを可能とすることができる．しかし，アプリケーションは新たなアブストラクションに基づくサービスの提供を受けるために，新たなメカニズムを走行中のカーネルにロードすることは難しい．

この問題を解決するために，新たなパラダイムに基づいたシステムが研究されている．リフレクションアーキテクチャを用いることで，オブジェクトのセマンティクスや動作を定義し，アブストラクションメカニズムを動的に変更することができる[20]．また，より低レベルのアブストラクションを利用することで，アプリケーションンはそれぞれの要求に応じた動作をするようにカーネルを変更することができる[9,2,6]．

ところで，OSの研究者たちは新しいOSの本来興

味がある部分の研究を始める前に，その新しいOSのインフラストラクチャを構築するために多大な労力と時間を費やしてきた．ブートストラップローダ，カーネルスタートアップ，ROMモニタやBIOSとのインターフェース，基本的なメモリ割り当て，コンソール入出力，基本的なロック機構，コンテキスト操作，カーネルデバッガ，デバイスドライバなどを実現するための多くのソフトウェアコードが実際に動作するOSのためには必要である．しかし，これらはほとんどのOSの研究者たちにとっては研究の興味の対象外であったりする．また，様々なOSにとって価値があるような機構が開発されたとしても，開発者が利用していたOSとは異なるOSにプラグインして利用することは困難である．

次世代マイクロカーネル研究プロジェクト (MKngプロジェクト)[18] では，Mach[1] および Real-Time Mach[19] マイクロカーネルを出発点に，新しいカーネルアーキテクチャとしてマイクロカーネルトレイアーキテクチャの設計とそれに基づいたシステムの開発を行なっている．本プロジェクトの目標は，新しいマイクロカーネルの基盤技術，すなわち，再構成および動的拡張が可能なマイクロカーネルを構築するための基盤技術を確立し，コンパイラ作成キットのようなアーキテクチャニュートラルなOS作成キットを提供できるようにすることである．ここで，アーキテクチャニュートラルとはOSパーソナリティやアブストラクションのパラダイムに依存しないことを言う．OS開発者はOS作成キットから取り出した部品と自身で開発した新たな部品を組み合わせることで新しいOSを作成することができる．また，OS利用者は提供されたOSをブート時に再構成したり，モジュールを動的にロードしたりすることが可能となる．このことから，OS作成キットを利用することで，OS開発者はそれぞれがOSの中の興味の対象に力を注ぐことができるようになるわけである．マイクロカーネルトレイは新たなアブストラクションの実現を容易にし，カーネルモジュールの再利用性を高めることに寄与する．

本報告では，マイクロカーネルトレイアーキテクチャの概要と設計方針について述べ，マイクロカーネルトレイ，マイクロカーネル作成キットについて解説する．そして，関連研究との比較を行い，今後の予定について述べる．

2 アーキテクチャの概要と設計方針

2.1 マイクロカーネルトレイアーキテクチャ

マイクロカーネルトレイとは，予め用意されたモジュールや新たに開発したモジュールから，特定の用途に適したOSを組み立てるための枠組みである．マイクロカーネルトレイの目的は，アブストラクションのパラダイムに依存しないアーキテクチャニュートラルな新しいカーネルの枠組を提供することであり，OSのアーキテクチャの枠を超えて，カーネル内のモジュールの再利用性を高めることにある．

図1にマイクロカーネルトレイアーキテクチャの概念図を示す．ユーザはカーネルの機能を拡張するためにマイクロカーネルトレイ上にモジュール (Loadable Kernel Module = LKM) を追加したり，カーネルの機能を縮退するためにマイクロカーネルトレイ上のモジュールを削除したりすることができる．モジュールは静的または動的にマイクロカーネルトレイにリンクされる．

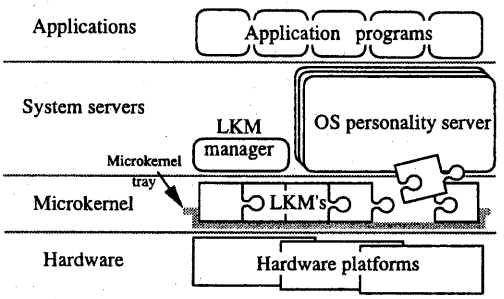

図1. マイクロカーネルアーキテクチャの概念図.

マイクロカーネルトレイはOSを構成するモジュールのソフトウェア的なバックプレーンバスの機能を提供する．言い換えると，モジュールのインターフェースの管理，モジュール間の結合，そして，モジュールの呼出しの機能を提供する．従来のOSで提供されるような何らかのアブストラクションやマイクロカーネルトレイを動作させるためのハードウェアに依存した機能は提供しない．これらはトレイ上に置かれた様々なモジュールによって提供される．

マイクロカーネルトレイを直接利用するのはアプリケーションの作成者であり，アプリケーションユーザがカーネルの機能を変更する場合は，予め登録されたモジュールの追加や削除のみが許されると仮定する．すなわち，任意のモジュールをアプリケーションユーザが追加することはない．そのため，カーネルに動的に追加されたモジュールをカーネルから隔離する機能[2] はマイクロカーネルトレイでは提供しない．もちろん，マイクロカーネルトレイ上のモジュールでそのような機能を提供することは可能である．

マイクロカーネルトレイは「マイクロカーネル＋OSパーソナリティサーバ」というパラダイムを強制するものではなく，カーネルレベルとユーザレベルの境界線の位置を強制するものでもない．例えば，全てのモジュールをカーネルレベルで動作するように作成し，その全てをマイクロカーネルトレイ上に置けば，モノリシックカーネルが実現される．また，基本的なアブストラクションを提供するモジュールをマイクロカーネルトレイ上に置き，それ以外の高度な機能をサーバとして実現すれば，マイクロカーネルベースのシステムとなる．さらに，マイクロカーネルトレイ上にハードウェアを呼び出すモジュールのみを置き，それ以外の全ての機能をライブラリとして実現すれば，ライブラリOS[9] を構築することも可能である．

2.2 OS 作成キット

OS 作成キットとは OS を組み立てるための様々なモジュールやツールの集合である．OS 作成キットの目的は，アプリケーションやアーキテクチャに適したカーネルを OS 作成キットから取り出した様々な部品から組み立てることを可能とすることと，OS 開発者が OS インフラストラクチャの構築に労力を費やすことなく，本質的な部分に力を注ぐことができるようにすることである．

OS 作成キットを構成するアプローチには既存の OS を機能ごとのモジュールに分解していくトップダウン的なアプローチとスクラッチから新たにモジュールを構成していくボトムアップ的なアプローチがある．機能ごとに明確に定義されたインターフェースを持つようにモジュール化がなされている OS を出発点とした場合，トップダウンアプローチが有効である．しかし，多くの OS では様々な機能が相互に依存しているだけではなく，ひとつの機能が様々なモジュールに分散して実現されており，トップダウンアプローチで粒度の小さい独立したモジュールに分解することは容易ではない．また，カーネル生成のための多くのオプションが存在し，それぞれのオプションが直交していないことが問題をさらに難しくしている．一方，ボトムアップ的アプローチはそのような問題点はないが，かなりの労力を費やすであろうし，既存の OS にあるインフラストラクチャを構成している多量なコードが再利用可能であればそれに越したことはない．

本研究プロジェクトでは，マイクロカーネルトレイとそれを直接サポートするモジュールについては，ボトムアップアプローチで構成するが，それ以外の OS インフラストラクチャについては，各種の Mach マイクロカーネルや FreeBSD 等のコードを利用して構成する．また，Real-Time Mach との互換環境を構築するために，Real-Time Mach 内の比較的独立した機能，並列で選択可能な機能，および，過去との互換性のために残してある機能を，粒度の大きなモジュールとして分解する．このようなアプローチで OS 作成キットを構成していく．

3 マイクロカーネルトレイ

マイクロカーネルトレイの基本的な機能は，カーネルモジュールの管理とカーネルモジュールの呼出しである．すなわち，どのモジュールが利用可能か，どのモジュールが利用されていないか，どのモジュールをいつ呼び出すかを管理しなければならない．このためには，ヴァージョン情報を含んだモジュールのインターフェースの情報，モジュールの依存関係および呼出し順序の情報などが必要となる．

マイクロカーネルトレイは C++ や Objective C などのランタイムルーチンに相当する機能を提供する．しかし，これらの一般的なランタイムルーチンと比較すると，マイクロカーネルトレイは

- 静的リンクと動的リンクのサポート，
- OS 実現のための柔軟なモジュール呼出し機構，
- 置き換え可能なメモリ管理機構，

という点が異なる．

3.1 モジュールの管理

C 言語ではモジュールのインターフェースをプログラム内で取り扱う機構がない．また，言語仕様にオブジェクトやモジュールを定義する枠組みを持つ言語でも，ヴァージョン情報や呼出し順序制御の情報をそのままでは定義できないことが多く，モジュールを定義するためには十分ではない．RPC[3,17] や IDL[15] のようなインターフェース定義言語によってモジュールを定義し，それをコンパイルした結果をプログラムに取り込むことで解決する．

マイクロカーネルトレイで扱うモジュールの定義ファイルの形式を図 2 に示す．

```
module-declaration   : module module-name
                           [ : import-list ] {
                           export-list
                       };
module-name          : identifier [ ⟨ version-number ⟩ ]
import-list          : import-name [ , import-list ]
import-name          : module-name [ (identifier) ]
export-list          : export-declaration [ export-list ]
combination-qualifier : : [ combination-type ] identifier
combination-type     : before | after | around | as
                       | precedence(number)
```

図 2. カーネルモジュール定義ファイルの形式．

export-declaration は C 言語の型，変数，および，関数等の宣言である．ただし，**generic** 型の記憶クラス宣言によって，モジュール呼出し機構による間接的な変数や関数のアクセスの指示，言い換えると，呼出しフックの宣言ができる．**generic** 型の関数等の本体の定義では *combination-qualifier* を宣言に後続させなければならない．これはどの呼出しフックからどんな順序で呼び出されるかを指示するものである．

モジュール定義ファイルはモジュール定義コンパイラによっていくつかの種類のインターフェース定義ファイルにコンパイルされ，親言語のコンパイラとリンカによってモジュールのオブジェクトに結合される．インターフェース定義ファイルはインターフェース定義，エクスポートテーブル，フック呼出しマクロ定義，および，ヴァージョン情報からなる．

トレイと複数のモジュールとのリンクは静的または動的に行われる．静的リンクを行う場合，モジュールのインターフェースの整合性のチェックはコンパイラやリンカ等によって行われる．リンク時にモジュール登録テーブルが作成され，このテーブルとインターフェース情報がリンク時にとりこまれる．

モジュールを動的にリンクする場合，モジュールは予めメモリ上にロードされており，ロードされたアドレスやモジュール間の依存関係が与えられているものとする．この作業はトレイ上の別のモジュール（オブ

ジェクトサーバ，名前管理サーバ，オブジェクトローダ）によって行われる．トレイへのモジュールの動的リンクの指示があると，リンクすべきモジュールに含まれるインターフェース情報とトレイが管理している様々なモジュールのインターフェース情報との整合性のチェックが行われ，それが成功するとモジュール本体とインターフェース情報がトレイ上のモジュール登録テーブルに登録される．

マイクロカーネルトレイはモジュール活性化のためのactivatorフックと非活性化のためのdeactivatorフックを提供している．これらのフックは他のモジュールやアプリケーションからの明示的な呼出し指示によって呼び出される．activatorはモジュールに初期コンテキストを設定するために用いられ，deactivatorはモジュールのコンテキストを掃き出させるために用いる．新旧のモジュールを置き換えるためには，deactivatorによって掃き出されたコンテキストを新たにリンクしたモジュールのactivatorによって設定する．

静的リンク用にマイクロカーネルトレイが提供するactivatorフックは複数ある．CPUアーキテクチャやインプリメントに依存するが，物理メモリモード（非ページングモード）や仮想メモリモード（ページングモード）などのためのactivatorフックがあり，マイクロカーネルトレイの起動時に呼び出される．

3.2 モジュールの呼出し制御

モジュール内の **generic** 型関数の呼出し制御は，モジュール定義ファイルで指定された関数結合の指示 *combination-qualifier* に基づいて行われる．**generic** 型関数以外は直接呼出しが行われる．関数結合の種類には，**before, after, around, as** などがある．

関数呼出しディスパッチャは概念的にはbefore, after, primary, aroundの4つのステージからなる．新たにリンクされたモジュール内の関数は，**before** 結合ならbeforeステージの既に登録されている関数の前に，**after** 結合ならafterステージの既に登録されている関数の後に，結合の指定がない場合にはprimaryステージの既に登録されている関数の後に，それぞれ登録される．**precedence** を用いることで登録順を制御することができる．**around** および **as** 結合ならaroundステージにただ一つだけ登録される．原理的にはこれらは再帰的に定義できる．

モジュールを特定しない **generic** 型関数が呼び出された場合，before, primary, afterステージの順に，それぞれの登録順で関数が呼び出される．関数の返り値としては，primaryステージの最後の関数の返り値が用いられる．もし，aroundステージに登録された関数があれば，全ての関数に先立ってそれが呼び出され，それが **around** 結合であれば，そこから順次 before, primary, after のステージに登録された関数を呼び出すことができる．**as** 結合であれば，唯一その関数だけが呼び出される．関数の返り値としては，aroundステージの関数の返り値が用いられる．各関数で途中で強制的に評価を打ち切るように指示することもできる．

4 マイクロカーネル作成キット

現在，OS作成キットとして，マイクロカーネルトレイ基本キット (Tray-kit) と Real-Time Mach システムキット (RMK-kit) の2つのマイクロカーネル作成キットの構築を予定している．

4.1 マイクロカーネルトレイキット

Tray-kit は CPU とメモリからなる，ローカルストレージやネットワークを持たないハードウェア構成のためのものである．Tray-kitの構成を図3に示す．各モジュールの概要は以下の通りである．

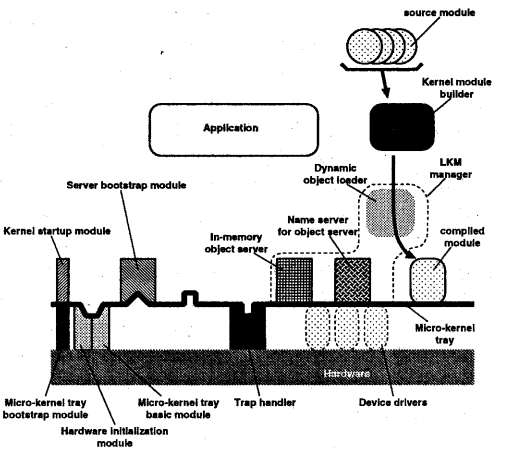

図3. マイクロカーネルトレイキットの構成．

- マイクロカーネルトレイ起動モジュール: ROMモニタから起動され，最小限のハードウェアの初期化を行い，マイクロカーネルトレイを起動する．
- カーネル起動モジュール: ブートコマンドの引数を解析し，ブートパラメタにもとづいてマイクロカーネルトレイを初期化する．
- ハードウェア初期化モジュール: 登録されているデバイスドライバによってハードウェアの検出と初期化を行う．
- マイクロカーネルトレイ基本モジュール: ROMモニタやBIOSとのインターフェス，基本的なメモリ管理 (malloc)，コンソール入出力，スピンロック，コンテキスト操作，トラップベクタ操作，カーネルデバッガなどの機能を提供する．
- サーバ起動モジュール: 登録されたサーバの初期化と起動を行う．
- インメモリオブジェクトサーバ: 様々なオブジェクトのメモリ上の格納庫として働く．各オブジェクトは一意なオブジェクト識別子を持つ．

Tray-kit における LKM マネージャはインメモリオブジェクトサーバ，名前管理サーバ，オブジェクトローダからなる．なお，名前管理サーバとオブジェクトローダについては後述する．

4.2 Real-Time Mach システムキット

RMK-kit は通常のワークステーションやパーソナルコンピュータのハードウェアをサポートする．これは，UNIX エミュレーションや様々なリアルタイムサーバを含む Real-Time Mach システムの全機能を提供する．

RMK-kit は Tray-kit に加え，マルチメディア機能付きの Real-Time Mach カーネル，Lites サーバ (4.4 BSD サーバ)，リアルタイムシェル (RTS)，リアルタイムネットワークプロトコルサーバ (NPS)，4.4BSD コマンドからなる．RMK-kit では，インメモリオブジェクトサーバはキットには含まれず，名前管理サーバやオブジェクトローダは Lites サーバの提供するファイルシステムを利用する．

4.3 名前管理サーバ

オブジェクトサーバのための名前管理サーバはオブジェクトの参照名とオブジェクトに関連した様々な属性情報のデータベースを管理している．名前管理サーバは（インメモリ）オブジェクトサーバ上にある全てのオブジェクトに関する全ての情報を保持しているだけでなく，システムが利用する様々な情報を保持しており，モジュールやアプリケーションがオブジェクトサーバにある最適なオブジェクトを検索することができる．カーネルもしくは特権を持ったアプリケーションのみが名前管理サーバ上に情報を登録することができる．

表 1. 名前管理サーバが取り扱うオブジェクト属性.

属性名	属性値
id	オブジェクト識別子（オブジェクトサーバ用）
path	オブジェクトパス名（ファイルシステム用）
type	オブジェクトのタイプ
name	オブジェクト名
variant	オブジェクトの変種名
version	オブジェクトのヴァージョン
entry_name	エントリシンボル名
routine_id	サービスルーチン番号
trap_id	トラップ番号
device_name	デバイス名
device_id	デバイス番号
capability	ケーパビリティ (Mach IPC port)

オブジェクト参照名の形式は，UFS におけるパス名や URL と同様であり，

path_prefix/system_name/kernel_version/type/name/variant/version

である．*path_prefix* はオブジェクトサーバの指定に用いられ，省略された場合にはローカルなオブジェクトサーバであると解釈される．*system_name* は OS 作成キットやサーバ等の名前であり，マイクロカーネル作成キットの *system_name* は mk である．*variant*, *version* は省略できる．名前管理サーバが扱うオブジェクト属性の例，および，*type*, *name*, *variant*, *version* について，表 1 に示す．

名前管理サーバはシステムコール名，システムコール番号，デバイス名，デバイス番号等も扱う．これにより，OS パーソナリティごとに異なる名前付け規則を統一して扱うことができる．

カーネルおよびアプリケーションはオブジェクト参照名やオブジェクトの属性と属性値の組を指定して検索を行なう．属性値としてワイルドカードを指定した場合には，検索結果としてそれにマッチするオブジェクトのリストが返される．

4.4 オブジェクトローダ

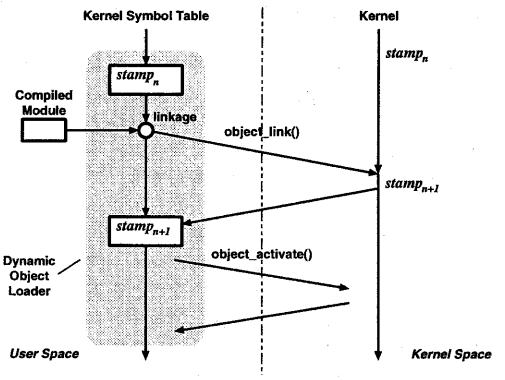

図 4. オブジェクトローダの動作.

図 4 にモジュールのロード時におけるオブジェクトローダとカーネルトレイの動作の例を示す．オブジェクトローダは，モジュール内のデータのリロケーション，シンボル参照の解決，シンボルテーブルの修正，インポートテーブルの生成を行なう．

カーネルトレイはカーネルへのモジュールのロードやアンロードが正常に完了した回数を表すカーネル世代番号を保持している．さらに，ホスト識別子（例えばネットワークインターフェースの MAC アドレス，IP アドレス，ROM が保持している識別番号など），カーネル世代番号，および，現在の時刻からカーネルトレイは（ハッシュ関数を通して）カーネルスタンプを生成し，保持している．カーネルスタンプはネットワークワイドな動作中のカーネルの一意性を保証するものであり，カーネルを変更するためのケーパビリティとして用いられる．

オブジェクトローダは自身が管理するシンボルテーブルから，現行カーネルのカーネルスタンプを取りだし，このカーネルスタンプを利用してカーネルをアクセスする．カーネルトレイは自身が保持しているカーネルスタンプと比較し，両者が一致すればアクセスを許可する．新たなモジュールのロードやアンロードが成功すれば，新しいカーネルスタンプがカーネルトレイから返される．

5 関連研究

カーネギーメロン大学の Mach プロジェクト [1], RT-Mach プロジェクト [19] は，本研究の出発点となったマイクロカーネルを提供している．しかし，マイクロカーネルを動的適応可能にすることは，研究されていなかった．Chorus 社の Chorus カーネル [16] は，単一の OS パーソナリティモジュールのみをサポートしているが，本研究では，複数の OS パーソナリティモジュールを同時にサポートする．

再構成や拡張可能なマイクロカーネルアーキテクチャに関しては，University of Washington の Spin プロジェクト [2]，MIT の Exo-kernel プロジェクト [9]，The Open Group Research Institute[4,8] などで研究開発が進められてきている．従来カーネル内に存在していた機能をカーネル外のユーザレベルで実現する方式，カーネルにモジュールを安全に追加する方式，そして，サーバをカーネル内に移動する方式なども提案されている．しかし，任意のカーネルモジュールを動的に再構成する方式に関しては，アイデア段階に留まっている．本研究では，カーネル内のモジュールをポリシとメカニズムの分離という観点から，従来モノリシックに構成されていたカーネル内モジュールをさらに再構成可能な部品とすることを実現する．このアプローチはレフレクションアーキテクチャによる OS[20] やより低レベルのアブストラクションを用いる OS[9,6] にとっても重要である．

Utah 大学の Flux OS ツールキット [10] は，本研究と同じ動機に基づいているが，Flux OS ツールキットは OS のインフラストラクチャを提供するのみである．本研究のツールキットは，ツールキットから取り出した部品を組み合わせることで，完全な OS を作成できるようにすることを目指している．

6 おわりに

本報告では，次世代OSのための新しいカーネルアーキテクチャであるマイクロカーネルトレイアーキテクチャを提案した．我々の目標は新しいカーネルの基礎となる再構成可能で動的拡張が可能なマイクロカーネルとアーキテクチャニュートラルな OS 作成キットを構成できるようにするところにある．

今後は，いくつかのアーキテクチャで実装を行なった上で，モバイル，マルチメディア，通信プロトコル処理，組み込み型リアルタイムカーネルなど，様々な方面への応用という観点から，評価を行っていく．

謝辞

本研究を進めるにあたり，日頃から熱心に討論して頂いている MKng プロジェクトのメンバの皆様に感謝いたします．

参考文献

[1] Accetta, M., Baron, R., Bolosky, W. J., Golub, D., Rashid, R., Tevanian, A., and Young, M.: Mach: A New Kernel Foundation for UNIX Development, in *Proceedings of the Summer 1986 USENIX Conference*, 93–112 (1986).

[2] Bershad, B. N., Savage, S., Pardyak, P., Sirer, E. G., Fiuczynski, M. E., Becker, D., Chambers, C., and Eggers, S.: Extensibility, Safety, and Performance in the Operating System, in *Proceedings of the 15th ACM Symposium on Operating System Principles*, 267–284, Copper Mountain, CO (1995).

[3] Birrell, A. D. and Nelson, B. J.: Implementing Remote Procedure Calls, *ACM Transactions on Computer Systems*, **2**, 1, 39–59 (1984).

[4] Black, D. and Bernadat, P.: Configurable Kernel Project Overview, in *OSF Research Institute Collected Papers*, volume 4, OSF Research Institute, Cambridge, MA (1994).

[5] Cheriton, D. R.: The V Distributed system, *Communications of the ACM*, **31**, 3, 314–333 (1988).

[6] Cheriton, D. R. and Duda, K. J.: A Caching Model of Operating System Kernel Functionality, in *Proceedings of the First USENIX Symposium on Operating Systems Design and Implementation*, Monterey, CA (1994).

[7] Cheriton, D. R., Whitehead, G. R., and Sznyter, E. W.: Binary Emulation of Unix using the V Kernel, in *Proceedings of the Summer 1990 USENIX Conference*, 73–85 (1990).

[8] Places, des F. B., Stephen, N., and Reynolds, F. D.: Linux on the OSF Mach3 Microkernel, in *FSF Conference on Freely Distributable Software*, 33–46 (1996).

[9] Engler, D. R. and Kaashoek, M. F.: Exterminate All Operating System Abstractions, in *Proceedings of the 5th IEEE Workshop on Hot Topics in Operating Systems* (1995).

[10] Ford, B., Lepreau, J., Clawson, S., Maren, van K., Robinson, B., and Turner, J.: The Flux OS Toolkit: Reusable Components for OS Implementation, in *Proceedings of the 6th IEEE Workshop on Hot Topics in Operating Systems* (1997).

[11] Forin, A., Barrera, J., and Sanzi, R.: The Shared Memory Server, in *Proceedings of the Winter 1989 USENIX Conference*, 229–243 (1989).

[12] Golub, D., Dean, R., Forin, A., and Rashid, R.: Unix as an Application Program, in *Proceedings of the Summer 1990 USENIX Conference*, 87–95 (1990).

[13] Guillemont, M., Lipkis, J., Orr, D., and Rozier, M.: A Second-Generation Micro-Kernel Based UNIX; Lessons in Performance and Compatibility, in *Proceedings of the Winter 1991 USENIX Conference*, 13–22, Dallas, TX (1991).

[14] Lepreau, J., Hilber, M., Ford, B., and Law, J.: In-Kernel Servers on Mach 3.0: Implementation and Performance, in *Proceedings of the 3rd USENIX Workshop on Micro-kernels and Other Kernel Architectures*, Santa Fe, NM (1993).

[15] Nestor, J. R., Newcomer, J. M., Giannini, P., and D, L. S.: *IDL: The Language and Its Implementation*, Prentice Hall (1989).

[16] Rozier, M. and Martins, J. L.: The CHORUS Distributed Operating System: Some Design Issues, in Paker, Y., Banatre, J.-P., and Bozyiğit, M. eds., *Distributed Operating Systems: Theory and Practice*, volume 28 of *NATO ASI Series F*, 261–287, Springer-Verlag (1987).

[17] Sun Microsystems, Inc., : RPC: Remote Procedure Call, Protocol Specification, Version 2, *RFC 1057* (1988).

[18] 徳田英幸, 追川修一, 西尾信彦, 萩野達也, 斎藤信男: MKng: 次世代マイクロカーネル研究プロジェクト, 第 53 回情処全大論文集, 5B-4 (1996).

[19] Tokuda, H., Nakajima, T., and Rao, P.: Real-Time Mach: Towards Predictable Real-Time Systems, in *Proceedings of the USENIX 1990 Mach Workshop* (1990).

[20] Yokote, Y.: The Apertos Reflective Operating System: The Concept and Its Implementation, in *Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications 1992* (1992).