仮想マシン機構を用いた拡張可能OSにおけるモジュール定義方式

4U−02

高野 了成† 石口 栄一†† 早川 栄一†† 高橋 延匡††
†東京農工大学 工学部 ††拓殖大学 工学部

1 はじめに

ウェアラブルコンピューティングや組込みシステムなど，ターゲットとするハードウェアの性能が幅広く，環境の変化に対する動的な対応が必要なシステムでは，アプリケーションやその用途に合わせてシステムを適合できることが重要である．そこでアプリケーション層において資源管理を行なうマイクロカーネルやOSの各機能をモジュール化し，それらを静的または動的に組み合わせてシステムを構築するコンポーネント指向OSなどが研究されている．

これらのOSは単一のシステム記述言語で実装されることが一般的であるが，ユーザアプリケーション同様に言語の記述性，性能などのトレードオフを判断して，様々な言語によって実装されたモジュールを利用することも有用であると考えられる．そこで我々はOSの各機能を仮想マシン上で実行するモジュールとして記述することによって拡張性とセキュアな実行環境を提供するOSを開発している．

本稿では，資源管理ポリシを仮想マシンとして提供し，仮想マシン上で動作するモジュールの組み合わせにより柔軟なシステム構成を可能とするOSにおけるモジュール定義方式について報告する．本システムでは，各モジュールをインタフェース記述言語によって定義し，実装言語に依存しないモジュールバインディングと名前サービスを提供する．さらに，モジュールバインディング機構のプロトタイプを実装し，動的な拡張と性能のオーバヘッドに対する見積もりを行なった．

2 設計方針

資源管理ポリシを複数のモジュールを組み合わせることで柔軟に構築できるようにするために，次に示す設計方針を立てた．

1. 複数の仮想マシンを用いたOS構成
   アプリケーションに対する実行環境を仮想マシンとして提供し，資源管理ポリシごとに複数の仮想マシンを提供する．
2. 言語中立なインタフェース情報の管理
   アプリケーションの実装言語や実行される仮想マシンに依存せず，モジュール間のバインディングを可能にするためにシステムで一意な名前と型情報管理を提供する．

3 設計とプロトタイプの実装

3.1 システムの概要

本システムはOS層，仮想マシン層，アプリケーション層の三層から構成される．OS層はハードウェアの抽象化を行ない，仮想マシン層はアプリケーションに対して実行環境，資源管理のポリシを提供する [1]．

本システムにおけるモジュールの定義から実行までの流れを図1に示す．モジュールのインタフェースはIDL(Interface Difinition Language)によって記述され，そのインタフェース情報はシステムが管理する名前空間に登録される．また，そのインタフェースを実装したモジュールは各仮想マシン上で実行される．図1で定義しているTimeモジュールは，CとJavaなど複数言語によって実装されたり，同じ言語での実装でも仮想マシンにおけるポリシの違いから異なる仮想時間を提供する実装になることが考えられる．

このように複数の仮想マシン上で動作するモジュールを提供するために，OS層では次の二つの機能を提供する必要がある．

- モジュールの公開インタフェースを管理するシステムで一意の名前空間機構の提供
- モジュールの各実装とその実行環境である仮想マシンの対応付け

3.2 プロトタイプの実装

モジュールバインディングによるシステム構築のトレードオフを見積もるために，本機構のプロトタイプをOS/omicron第4版上にて実現した．実行環境はIntel Celeronプロセッサ(400MHz)，メモリ192MB搭載のPC/AT互換機である．

本プロトタイプではモジュールバインディング方式として，コンパイル時に静的に解決する方式と，V4の仮想マシンインタフェース [2] を利用して実行時に動的に決定する方式を実現した．前者は各インタプリ

A Module Mechanism for an Extensible Operating System using Virtual Machine Concept
TAKANO Ryousei, ISHIGUCHI Eiichi, HAYAKAWA Eiichi, TAKAHASHI Nobumasa
† Faculty of Engineering, Tokyo University of Agriculture and Technology
†† Faculty of Engineering, Takushoku University

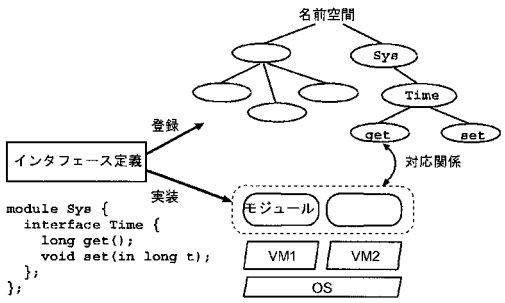

図 1: モジュールと名前空間

タ標準のバインディング機構を呼び出すスタブとして実装され，後者はダイナミックリンクとフォールトを利用した各仮想マシンへの委譲方式を利用して実装されている．

4 評価実験

CからTcl，Wabaで実装されたモジュールへの関数呼出し時間を測定した．その結果を表1に示す．Tclの場合，静的呼出しの41.8μsに対し，動的呼出しには202.4μsと5倍弱の時間がかかった．しかし，動的呼出しにはダイナミックリンクの処理時間が含まれるので，2回目以降の処理時間は135.0μsと3倍強に短縮される．一方，Wabaはオブジェクト指向言語でありメソッド検索のオーバヘッドが大きい．したがって，静的呼出しでも1008.0μsと処理時間が長く，相対的にモジュールバインディング機構のオーバヘッドによる影響が小さかった．

表 1: モジュールバインディング機構の処理時間

	C → Tcl(μs)	C → Waba(μs)
静的	41.8	1008.0
動的 (初回)	202.4	1179.3
動的 (2回目以降)	135.0	1109.9

続いてモジュールバインディングにおける各処理に費した時間の内訳を表2に示す．なお，この結果は主記憶上にモジュールが存在することを想定しており，二次記憶へのアクセス時間を含んでいない．ダイナミックリンクは名前空間からモジュールを検索し，メソッドの未参照シンボルを解決するまでの時間，フォールト処理はモジュールを最初に実行した際に発生するフォールトを取得し，各仮想マシンに処理を遷移するまでの時間，型変換は仮想マシンにてスタブを呼び出し，型変換をしてインタプリト開始の前処理にかかった時間を示している．

表 2: モジュールバインディング処理の内訳

内容	時間 (μs)	割合 (%)
ダイナミックリンク	69.4	34.3
フォールト処理	82.5	40.8
型変換処理	52.5	25.9
合計	202.4	100.0

5 関連研究

コンポーネント指向OSにおいて，各コンポーネントのインタフェースを定義するためにKnit [3]のような独自のコンポーネント定義言語を利用する方式とIDL4 [4]などのIDLを利用する方式が考えられる．Knitはコンポーネントがインポート，エクスポートするインタフェースをUnitと呼ばれる形式で記述し，Cのソースコードへと変換される．また，IDL4はCORBA IDLをベースにしており，アーキテクチャとOSに特化した高速なコンポーネント間通信を行なうスタブを生成することを目的としている．これらの方式はOSを単一のシステム記述言語で実装することを前提としており，本研究の目的のように複数の言語によって資源管理部を構築することはできない．

6 おわりに

本稿では仮想マシン機構を用いた拡張可能OSにおけるモジュール定義方式の設計とそのプロトタイプの実装，評価実験について述べた．

今後は本機構を現在我々が開発しているOS上に実装し，評価を行なう予定である．

参考文献

[1] 石口, 高野, 早川, 高橋: 仮想マシン機構を用いたオペレーティングシステムの実現, 情報処理学会第64回全国大会論文集, 4U-01, (2002)

[2] 高野, 石口, 佐藤, 早川, 高橋: 分散永続オブジェクトを共有するための仮想マシンインタフェースの設計, DICOMO 2001 シンポジウム予稿集, pp.561-566, (2001)

[3] A. Reid, M. Flatt, L. Stoller, J. Lepreau, E. Eide, Knit: Componet Composition for Systems Software, In Proceedings of OSDI 2000, pp 347-360, (2000)

[4] A. Haeberlen, J. Liedtke, Y. Park, L. Reuther, V. Uhlig, Stub-Code Performance is Becoming Important, In Proceedings of First Workshop on Industrial Experiences with Systems Software, (2000)