システムソフトウェアと
オペレーティング・システム 75－13
（１９９７．６．５）

# OSパーソナリティとしてのJavaVMの利用

高野　陽介
日本電気(株)　C&C研究所
〒216 神奈川県川崎市宮前区宮崎4-1-1
E-mail: yt@mmp.cl.nec.co.jp

利用可能な計算機資源の少ない小型情報端末をターゲットとして、RT-Machをベースとしたマイクロカーネル上にJavaアプリケーションとバイナリによるアプリケーションをサポートできる小型のシステムソフトウェアの構築を進めている。このような条件下で、Javaプログラムから OSサーバの利用がより高速に行えること、OSサーバの消費メモリを抑制できることを目的としたマイクロカーネル上のOSサーバ層を実現する一方法を提案する。

# OS Personality Servers on JavaVM

Yosuke TAKANO
C&C Research Laboratories, NEC Corporation
Miyazaki 4-1-1, Miyamae, Kawasaki, Kanagawa 216, JAPAN
E-Mail: yt@mmp.cl.nec.co.jp

We have been developing an operating system, based on RT-Mach microkernel, to serve both for Java applications and for conventional binary applications, running on a small information terminal. This report describes a design of operating system servers on the microkernel, which enables faster server access by Java programs and efficient memory utilization.

## 1 はじめに

アプリケーションの広がりやネットワークへの適応力などを理由に組み込み機器や小型端末でJava言語[1]を利用するシステムの例が増えつつある。Javaプログラムを実行するJavaVM (Virtual Machine)[1]の構築方法は適用するシステムの利用形態に応じて様々である。我々は、小型情報端末をターゲットとして、RT-Mach[2]をベースとしたマイクロカーネル上にJava言語を利用できるシステムの構築を進めている[3][4]。本報告では、マイクロカーネル上でのJavaVMを含むOSサーバの実装の一方法を提案するとともに、その効果を実験システムで評価した結果について述べる。

## 2 システムに対する要請と設計の狙い

RT-Machマイクロカーネル上ではそれに含まれないOS機能は、OSサーバとしてアプリケーションなどど同様のタスクにより実現される。ファイル、ネットワークなどのOS機能がOSサーバに相当する。ターゲットとする小型情報端末に対する要請のもとで、このOSサーバの層をどのように実現すべきかを考察する。OSサーバの実現に際して、我々のターゲットマシン上では以下の2つの要請を満たす必要がある。

1) Javaプログラムおよび通常のバイナリコードによるプログラムの双方にOSサービスを提供できるようにする。

2) OSサーバによるメモリ消費を小さく抑える。

最初の要請を満たすためには、OSサーバ機能をJavaプログラムとバイナリプログラムの双方によってアクセス可能な場所に実装する必要がある。1つの方法は、OSサーバをタスクとして実装し、プログラムを実行するタスクからはタスク間通信によって、OSサーバ機能にアクセスする方法である。しかしながら、Javaプログラムから利用する際にはインタプリタで実行することに起因する実行の遅さに、タスク間通信のためのコンテキストスイッチ、データコピーなどによるオーバヘッドが加わることになり非常に不利である。そこで、本実装では、このオーバヘッドを小さくする手法をJavaアプリケーションの実行環境に適用することで、Java言語による処理スピードの不利面を改善する。

2番目の要請は、消費電力などの問題上、我々のターゲットマシンでは二次記憶をあまり多くは保有できないという状況から発生する。通常のコンピュータシステムではデマンドページングにより物理メモリの消費を小さく抑えることができるが、このターゲットのように二次記憶が得られない環境下ではスワップ領域は確保できないという制約が生じる。

この要請を満たすためには、OSサーバ自体をコンパクトに実現し、またその動的な消費メモリを小さくするような設計にするという方法に加え、デマンドページングの役割と同じくOSサーバ内の利用されていないメモリを見つけ出し、再利用するというメカニズムが求められる。考えうる1つの方法は利用されていないOSサーバのタスクを停止・抹消してしまう方法である。しかしながら、タスクというモジュールの粒度では停止可能な機能を分離しにくい。

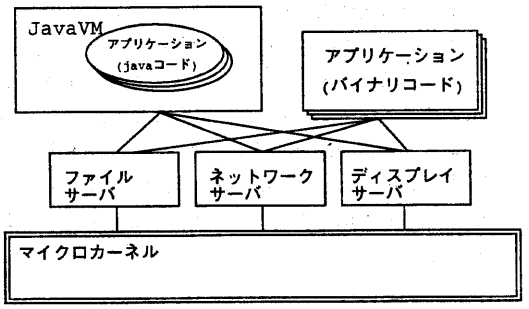
一方、Java言語ではプログラムを比較的小さいコンポーネントの集まりとして記述することができる。JavaVMではそれらのコンポーネントを必要に際してロード・アンロードする機能を提供することが可能である。これを利用しマイクロカーネル上のOSサーバをJavaプログラムとして実装することができれば、利用されていないOSサーバのプログラムに必要に応じて物理メモリを与え、また奪うことができると考えられる。しかし、高い頻度で利用されることが予想されるOSサーバにこの方法を適用することは、上記性能の問題から見ると好ましくない。そこで、利用頻度の低いサーバ機能に限定しJavaVM上でOSサーバの構築を試みる。

以下の節では、この2つの狙いに関してより詳細に説明する。

## 3 JavaVMの実装における性能改善

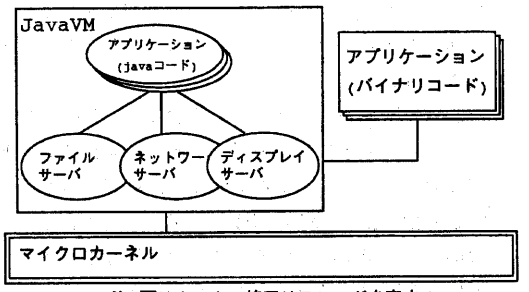
図1は、RT-Mach上に従来の構築方法に基づいて設計した例である。ディスプレイ、ファイル、ネットワークといったサービスを提供するOSサーバが常駐タスクとして実装されるとともに、Java言語で記述されたプログラムをサポートするためにJavaVMがタスクの1つとして実装される。この他に、バイナリコードによるプログラムを実行するア

[1] Javaは Sun Microsystems, Inc. の登録商標です。




(注)□はタスク、楕円はスレッドを表す

図 1: 従来方式による設計




(注)□はタスク、楕円はスレッドを表す

図 2: 新方式による設計

プリケーションタスクが存在する。アプリケーションタスクは直接、Java プログラムは JavaVM を通して、OS サーバにタスク間通信を行うことでサービスを利用する。

これに対し、提案方式による実装を図 2に示す。独立したタスクで実行されていた各 OS サーバは JavaVM のプログラムとリンクし、タスク内部のスレッドによって実行する。これにより、Java プログラムは別のタスクに切り替えることなく、OS サーバのサービスを利用することができる。一方、アプリケーションタスクは従来通り、JavaVM のタスクにタスク間通信を行うことで、JavaVM に内蔵された OS サーバにサービスを利用する。このためには、JavaVM に内蔵された OS サーバは、Java プログラム向けと外部のタスク向けの 2 つの利用口を持つことになる。この際、双方のアプリケーションから見た OS サーバへのインターフェイスは従来方式のものと変更はないようにする。この方法によって、アプリケーション・タスクでのアクセス性能はそのままであるが、Java プログラムからの OS サーバ機能の利用に関して性能の改善が期待できる。

従来からアプリケーションにリンクされる実行時ライブラリに OS 機能を挿入することによる性能改善の試みは存在していた。本報告での試みは、そのバリエーションの 1 つとして、マイクロカーネル上での特定の OS サーバの性能改善を行った例ととらえることもできる。

## 4 OS サーバ実行環境としての JavaVM の活用

デマンドページングを用いない仮定のもとで、サービスの利用頻度の点で OS サーバ機能を分類すると次のようになる。

1. 比較的高い頻度で利用される OS サーバ機能
2. 利用頻度が低く、利用していないときにはその機能を完全に停止することができる OS サーバ機能
3. 利用頻度が低いが、利用していない間でもその機能を (完全には) 停止することができない OS サーバ機能

これらを Java 言語で実行した場合に実行時性能に大きなインパクトを与えるかどうかを検討する。1. は実行時性能にも大きく影響するので、バイナリコードを用い常駐タスクによって実装すべきである。2. と 3. は、利用頻度が低いことから Java プログラムで実装できる候補として考えられる。しかし、2. の場合、バイナリコードを用いた非常駐タスクとしても実装できるので、そちらの方が実行時の性能が良いと考えられる。

3. のケースでは、Java プログラムを用いた OS サーバの実装が有効であると考えられる。例えば、発生頻度の低い外部からのイベントを受け取ってそれに対する処理を行う OS サーバがあったとする。これを、イベントを待機するクラスオブジェクトと処理を行うクラスオブジェクトの 2 つで実装する。前者の待機インスタンスは JavaVM 上に常駐し外

部イベントの監視を行い、後者の処理インスタンスは前者の待機インスタンスによって必要に応じて生成される。

JavaVMには、プログラムをクラスを単位として動的にロード・アンロードする機能を持たせることができる[2]ので、javaで書かれたプログラム部分に関しては、仮想記憶と同様の効果が期待できる。先の例では、後者の処理インスタンスは要求が存在している期間のみメモリ上に作られ、そうでない期間はメモリを占有しない。

3. のケースとして当てはまるOSサーバ機能としては、以下のようなものが考えられる。

- プロセス間通信のための ネームサービス
- PCカードの状態変化に応じて行う制御サービス
- ネットワークの初期化処理
- モバイル環境下での通信制御

また、ネットワーキング機能も、ネットワークに到達できない場所で携帯端末を使うような状況を考えると、取り外し可能な機能ととらえることができるが、ネットワーキング機能の全体をjavaで実現するのは現時点では性能面で十分でないと考えた。

## 5 評価

ここでは、実験システム上で本報告で提案した設計方法の評価を行った結果を述べる。

### 5.1 実験システム

現在、RMK95[5][6](RT-Mach)をベースにそれをさらに小型化したマイクロカーネル上にJDK(Java Developers Kit)1.0.2をベースとしたJavaVMサーバを構築している。RT-Machの利用はそのリアルタイム機能をマルチメディア・アプリケーションなどで利用することがその理由である。JDKで配布されているJavaVMは、本来UNIXやWindows上の一アプリケーションとして起動・動作するものであるが、これを改造してUNIX機能に依存せず、JavaVMサーバに内蔵したファイルサーバを用いてマイクロカーネル上に起動・動作するようにした。先に述べたネットワークサーバ、ディスプレイサーバの実現は現在進行中である。

JavaVMサーバでは、RT-Machの提供するリアルタイムスレッドをマッピングすることによりJavaスレッドを実現した。また、Javaスレッド間の相互排除や同期を行うためのmutexとcondition variableもRT-Machのリアルタイム機能を利用して実装した。

### 5.2 JavaVMへのOSサーバ組み込みの評価

実験システム上でOSサーバをJavaVMタスクに組み込んだことによる効果を調べるため、実験システムのJavaVMサーバに内蔵しているファイルサーバをサンプルとして評価を行った。評価では、図3に示すように、ファイルサーバとそのクライアントの関係に対して、ファイルサーバが、

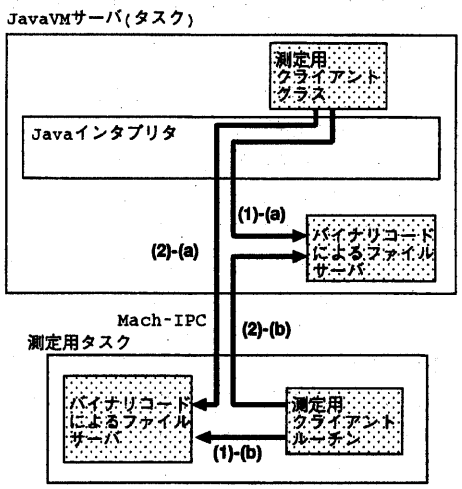



図3: OSサーバ内蔵の評価システム

(1) ファイルサーバがクライアントタスクに内蔵されている

(2) ファイルサーバが別のタスクに存在する

のそれぞれの場合に対して、

(a) クライアントがJavaプログラムである

(b) クライアントはバイナリプログラムである

[2]クラスのアンロード機能はJDK1.1などで実装されている。

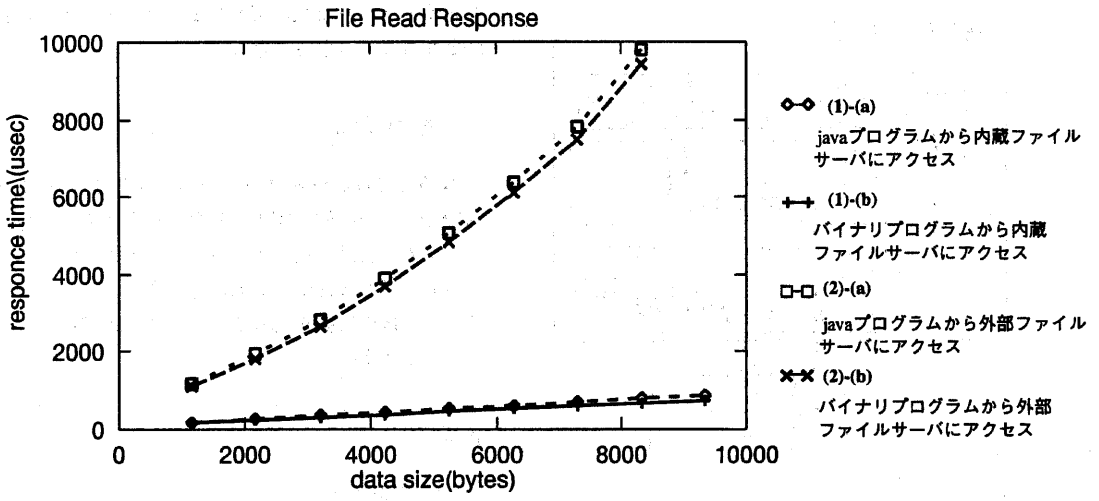



図 4: ファイルサーバのアクセス性能の比較

の場合の計4通りのファイルアクセスの応答時間を測定した。なお、実験システムで本ファイルサーバが扱うファイルシステムの内容は、ディスク装置ではなくプログラムからランダムアクセス可能なROM上に常駐している。また、本ファイルサーバではファイルの先行読み込み、遅延書き込みなどは行っていない。

図4はその測定結果である。4通りのアクセス経路に対して、ファイルの読み込みの応答時間を測定した。評価はPC互換機(Pentium 133MHz)で行った。結果を見ると、別タスクにファイルサーバがある場合に比べて、1/10程度の応答時間でファイルの読み込みが行えることが分かる。本方式によれば、Javaプログラムからの OS サーバの利用に大きな優位性を与えることが可能である。

### 5.3 Java言語による OS サーバの評価

実験システム上でJavaVM上で OS サーバを実装したときのメモリ効率化に関する評価を行った。実験システムではクラスのアンロードの機能はまだ実装していないので正確な評価はできない。そこで、実際に機能的に同等な OS サーバをJavaプログラムとネイティブプログラムの2つで実装し、その大きさを比較することで評価とした。

評価には、プロセス間通信のコネクションを確立するために必要なネームサービスの機能を用いた。このネームサービスは、以下の3つの機能を含む。

- 通信ポートをその名前の対を登録する。
- 名前によって示される通信ポートを取得する。
- 名前によって示される通信ポートを取得する。通信ポートが未登録の場合には登録されるまで実行を中断する。

これらの機能を2つの方法で実装した。この様子を図5に示す。1つは、C言語で記述したバイナリコードでJavaVMタスクに内蔵する形態のものである。もう1つは、Java言語で記述したもので、以下の2つのクラスによって実装した。

- ネームテーブル・コンテナクラス

  名前と通信ポートの対を格納するクラスで、そのインスタンスはJavaVM上に1つだけ存在して常駐する。

- ネームテーブル・ハンドラクラス

  ネームテーブルに対して、前記の3つの機能を適用するクラスで、クライアントからネームテーブルへのアクセスが発生した時に利用される。

以上の機能はアプリケーションタスクから Mach-IPC 経由で到着した RPC 要求を受け付ける JavaVM

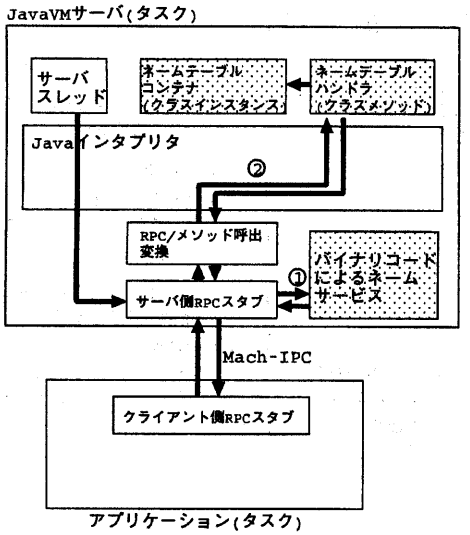



図 5: Java 言語による OS サーバの評価システム

内の受信ルーチンから呼び出される。バイナリコードの場合にはそこから直接手続き呼出しで、java コードの場合には、メソッド呼出しに変換され、JavaVM 内をアップコールすることで起動される。実験結果を表 1に示す。参考までに、Java コードでの通信ポート登録処理に 1290$\mu$s、バイナリコードでの同処理に 330$\mu$s の応答時間を要した。

| 形態 | バイナリコード (text+data+bss) | Java バイトコード | |
|---|---|---|---|
| | | コンテナクラス | ハンドラクラス |
| サイズ | 952 | 350 | 1032 |

表 1: プログラムサイズの比較
(単位はバイト。いずれも Pentium 用のコードでコンパイルでは最適化を行っている)

同じ機能を実装しても Java 言語の方のサイズが大きくなるのは主に Java バイトコードでメソッド間の参照をメソッド名の文字列を用いて行っていることに依る。しかしながら、ネームサービスが利用されていない時のメモリ占有サイズは約 1/3 でありアンロードを行った場合にメモリ節約のメリットが得られることが観察できる。この例で得られる節約は非常に小さいオーダであるが、他の Java 言語化した OS サーバと組み合わせることでより大きな節約となることが期待できる。

## 6 おわりに

本報告では、マイクロカーネル上に JavaVM を含む OS サーバ群を実現する際の一方式を示した。システムの実装が途上であるため、その評価は一部の側面に留まったが、有効性を示すことができた。今後は全体の実装を進めるとともに、細部にわたって評価を行う。

## 謝辞

本研究の機会を与えていただくと共に有益なご指導・ご助言をいただいた C&C 研究所の後藤所長、阪田所長代理、およびターミナルシステム研究部の横田担当部長をはじめとする研究プロジェクトの方々にこの場をかりて深く感謝致します。

## 参考文献

[1] T.Lindholm and Y.Frank: "The Java Virtual Machine Specification", Addison-Wesley(1996).

[2] H.Tokuda,T.Nakajima and P.Rao: "Real-Time Mach: Towards a Predictable Real-Time System", USENIX Mach Symposium, pp.213-221 (1991).

[3] 黒岩, 石井, 高野, 横田: "MKng プロジェクトにおける携帯端末サポート", 第 53 回情処全国大会論文集, 5B-5 (1996).

[4] 石井, 高野, 黒岩, 横田: "メモリを高効率で活用するメモリ回収機構", 第 54 回情処全国大会論文集 (1), pp.223-224(1997).

[5] 徳田, 追川, 西尾, 萩野, 斎藤: "MKng: 次世代マイクロカーネル研究プロジェクト", 第 53 回情処全国大会論文集, 5B-4 (1996).

[6] A.Miyoshi, T.Kitayama, H.Tokuda: "Implementation and Evaluation of Real-Time Java Threads", The 18th IEEE Real-Time Systems Symposium(1997).