システムソフトウェアと
オペレーティング・システム 87－18
（２００１．６．２９）

組込み用Javaにおけるマルチスレッド機構の設計

駿藤 浩之† 並木 美太郎†

†東京農工大学大学院工学研究科

近年，組込みシステムでJavaを実装する環境を提供するようになってきた．Javaにはプラットフォームに依存しないというプログラムのポータビリティ，オブジェクト指向言語による再利用性，言語仕様による安全性，豊富なAPIが用意されていることによる開発の容易性，という長所があるからである．しかしながら，組込みシステムでJavaを使用するには，リアルタイムJavaスレッドをどのように実現するか，が問題になる．一般にJavaプログラムは，OSのタスクとして動作するJavaVMにより実行され，ガーベッジコレクト，動的クラスロードなどの処理が行われる．このためJavaスレッドの挙動が予測しにくく，Javaスレッドおよびシステム全体のリアルタイム処理のためのデッドラインスケジューリングが困難になる．JavaスレッドもOSの提供する一つのリアルタイムなネイティブスレッドとして実装する方式もあるが，複数のJavaVM間の通信や排他制御処理などでオーバヘッドが増加する問題がある．そこで，本稿では，Javaスレッドの方式を複数考察した．複数のJavaスレッドをOSの一つのタスクであるJavaVMで管理するグリーンスレッド方式，一つのネイティブタスクを一つのJavaスレッドとするネィティブスレッド方式をWabaVM上にそれぞれ実装した．そして，このオーバヘッドを削減するために，カーネルを直接呼び出すダイレクトスレッド方式を提案し，同様にWabaVM上に実装した．その結果，Javaスレッド生成，破棄のオーバーヘッドを約38%削減できた．

Design of MultiThread Mechanism for an Embedded Java

Hiroyuki Sunto† Mitaro Namiki†

†Division for Reserch of Technology,
Tokyo University of Agriculture and Technology

In recent years, an embedded system often offered Java Virtual Machine implementation. Because, the Java Language Specification has the advantage of the portability, safety and object-oriented. Generally Java applications are executed on interpreters called JavaVM running as a task of an OS and processing such as garbage collection,dynamic class loads. Because of these processes, behavior of the Java threads are difficult to predict and schedule deadlines for real-time processing of Java threads and also the whole system become difficult. There is the problem increasing overhead with the communication and exclusive operation between JavaVMs implemented as one real-time naitive OS's task executing on Java thread. Thereupon, we studied some kainds of methods to run Java threads in this paper. It did the green thread method, one native task that manages many Java threads with one JavaVM. Another method is one Java thread equal to one OS's task with one Java VM. Each method is implemented with WabaVM. And, proposing the direct thread method that calls kernel directly, to decreas to this overhead it did implementation on WabaVM similarly. As a result, about 38 % of troubles of Java thread generation, deletion are able to be decreased.

1 はじめに

近年のインタネットの広まりに応じてJavaが考案され，WWWのプログラミング言語として広く利用されている．Javaプログラムは，JavaVM(以下JVM) というインタプリタ上で実行されることから，プログラムがプラットフォーム独立という利点がある．

このプラットフォーム独立という利点を組込みシステムに活かそうという考え方がある．たとえば，情報家電や携帯電話に見られるように，組込みシステム上の応用プログラムをJavaで記述することにより，多様なプラットフォームが存在する組込みシステムのプログラム開発の手間を減らすことができるのではないかと考えられている．また，組込みシステムの記述言語そのものに，Java言語を利用することで，システム開発を容易にできるのではないかと考えられる．

このように，組込みシステム開発にJavaを用いるためには，Javaプログラムを実行するためのJVMが必要不可欠である．組込みシステム向けのJVMの要求を次に示す．

(1) リアルタイム処理に対応できること

組込みシステムでは，機器制御が中心となることから，デバイスからの入力に対してその結果を一定時間以内に処理することが要求される．さらに，動画像や音声処理などのマルチメディアをJavaの応用プログラムで実行するためにもリアルタイム機能は必須となる．そのためには実行時間を見積もれる必要がある．しかし，Javaにはガーベッジコレクトや動的クラスロードがあり，実行時間を見積もることが困難である．

(2) ハードウェア制御ができること

組込みシステムでは外部からのデバイス入力に対して，ハードウェアを制御する機器が多い．Javaないしはネイティブメソッドで，メモリを直接操作したり，デバイスドライバなどを記述できる必要がある．

(3) 小型であること

パーソナルコンピュータ (以下PC) に比べ，製品コストを切り詰めるので，安価なハードウェアが使用される．特にCPU，メモリでの制約が著しい．したがって，サイズが小さいことが要求されることもある．

組込み用JVMへの要求(1)(2)を満たすため，大きく分けて二つのアプローチがある．それはリアルタイム処理用にJavaを拡張するアプローチとハイブリットアプローチが考えられる．

前者は，リアルタイム処理用にJavaおよびJVMを拡張することで，リアルタイム処理とハードウェアを制御する方式である．リアルタイム処理用にJavaを拡張する研究にはRTSJ(The Real-Time Specification for JavaTM)[1]があり，リアルタイムスレッドやスケジューリングクラスなどを仕様として新設している．

後者のハイブリットアプローチはリアルタイム処理，ハードウェア制御をOSのネイティブタスクで実行し，非リアルタイム処理をJavaVMで実行する．ハイブリットアプローチにはJTRON仕様[2]などの研究がある．

一般にリアルタイム処理を行うためには，実行単位となるJavaスレッドおよびOSの提供するネイティブタスクの扱いが重要となる．どのようなアプローチでもJVMを含むシステムは，カーネル (またはOS) 上に構築されので，Javaスレッド，JavaVMをネイティブタスクでどのように実行するかが，リアルタイム機能や実行時性能に大きな

影響を及ぼすと考えられる．JVM をPC に搭載する方法は，グリーンスレッド方式とネイティブスレッド方式の二つに大別できる．時間，リソース要求が厳しくないPC 上ではどちらの方式でも先の (1)(2) で述べた要求をみたせるだろう．しかし，組込みシステムでは，リソース制約を満たした上で，先の要求を実現する必要がある．

そこで，本稿では，リアルタイムの組込みシステムでの JVM を実装方式について考察する．どのような方法が適しているのかを定量的に考察するために，第 2 章でスレッド管理機構の設計方針を示し，第 3 章ではグリーンスレッド方式，ネイティブスレッド方式を比較検討する．第 4 章で実装を示し，第 5 章で実験結果を考察する．

2　スレッド管理機構の設計方針

設計方針では 1 章に示した要求に答えるため，実行単位である Java スレッドをどのように管理するかが問題になる．特に Java スレッドの実現方式が重要である．PC ではグリーンスレッド方式とネイティブスレッド方式が存在する．次に組込み用 JVM を設計する上でマルチスレッド機構の設計方針を示す．

(1) Java スレッドとネイティブタスクの優先度を一元的に管理する
リアルタイム処理の基本的なスケジューリング方式は優先度を参考するものである．Java スレッドとネイティブタスクを 1 元的に管理することができれば，Java スレッドのリアルタイム処理が可能となる．

(2) スレッド操作のオーバーヘッドを削減する
組込みシステムではオーバーヘッドを可能な限り削減する必要がある．また，リアルタイム処理では一定時間以内に処理することが要求される．オーバーヘッドが少ないことが必須条件である．これは本研究の目標であるリアルタイム Java を構築する上では欠かすことはできない．

この他にも JVM の移植性が問題となる．移植性とオーバーヘッド，小型化はトレードオフである．PC では，移植性のほうが重要になる場合もあるが，組込みシステムに適用する場合にはオーバーヘッドの削減と小型化が重視される．

3　スレッド管理機構の実装方式

3.1　グリーンスレッド方式

OS は CPU を仮想化し，複数のタスクをスケジューリングして実行するマルチタスク機能を持っている．この仮想化と同じ機能を JVM が持つことで，JVM は Java スレッドを管理下へ置く．Java スレッドは図 1 のように JVM で処理される．OS

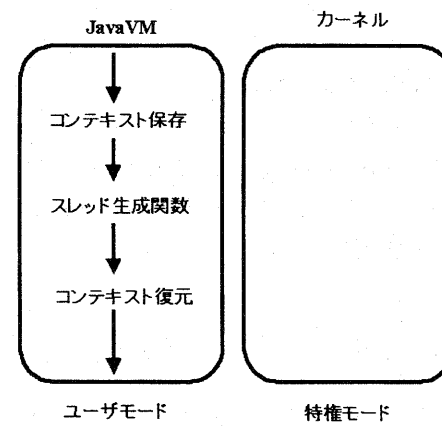

図 1 グレーンスレッドの処理の流れ

の一つのタスクとして実行される JVM に各 Java

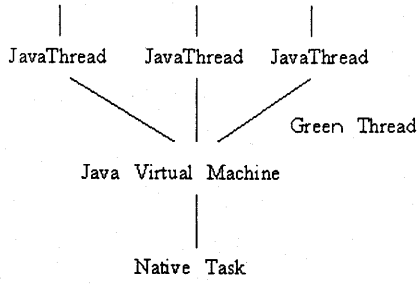

図 2 グリーンスレッド方式の概念図

スレッドを割り当てることにより，Java スレッドをスケジューリングする (図 2)．通常の PC 用 JVM ではユーザスレッドライブラリなどで実現されていることが多い (many to one mapping)．この方式の長所はスケジューリングなどをユーザ領域で実行することで，システムコールなどを使用することなくスレッドを操作できる．これによりシステムコールを呼び出すオーバーヘッドは削減される．短所としては，JVM でスレッド管理をする必要があり，実行サイズが大きい．

また，リアルタイム Java スレッドを実現するには，ガーベッジコレクト，クラスロードの実行時間が予測できない問題がある．さらに，システム全体で考えるとリアルタイムのネイティブタスクとの協調動作や待ちのあるシステムコール処理が困難であり，Java スレッドとネイティブタスクの一元管理が難しい．

3.2 ネイティブスレッド方式

ネイティブスレッド方式とは一つの Java スレッドを OS の一つのネイティブタスクにする方式である．すなわち，1 対 1 で Java スレッドがネイティブタスクに対応する (one to one mapping, 図 3)．OS でネイティブタスクをスケジューリング

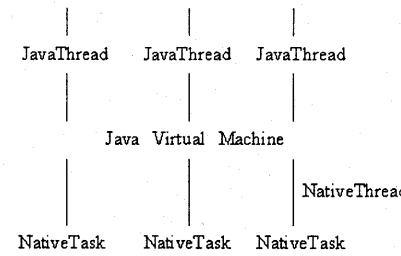

図 3 ネイティブスレッド方式の概念図

することにより，それに対応する Java スレッドもスケジューリングされたことになる．この方式の長所は OS のネイティブタスクと Java スレッドを一元管理できることである．短所は図 4 のように Java スレッドの操作がカーネル内の処理となるので，Java スレッドに対する処理にシステムコールを使用する必要性が生じることである．

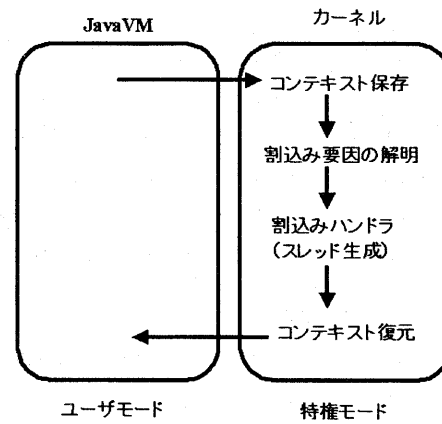

図 4 ネイティブスレッドの処理の流れ

3.3 ダイレクトスレッド方式

前節で述べたネイティブスレッド方式はスレッドを操作するたびにシステムコールを実行するこ

とでオーバーヘッドが大きい．これを解決するために，直接カーネルを呼び出す方式を提案する．これによりシステムコールを呼び出すオーバーヘッドを削減することができる．この方式を本稿ではダイレクトスレッド方式と呼ぶ．この方式の短所はカーネルの関数を呼び出すために，静的にリンクする必要があるので，JVM 自身の移植性が落ちることである．それは図 5 に示すように JVM がカーネル領域にあり，直接カーネルを呼び出すので，サブルーチンコールと同じオーバーヘッドで呼び出すことができる．

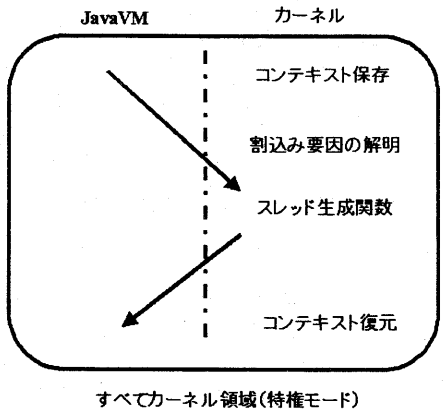

図 5 ダイレクトスレッドの処理の流れ

各方式の比較一覧表を表 1 に示す．

表 1 各方式の比較

各方式	グリーン	ネイティブ	ダイレクト
JVM の移植性	○	○	×
オーバーヘッド	△	×	○
優先度の統一性	×	○	○
全体のサイズ	×	△	○

4 各方式の実装

4.1 全体構成

OS は筆者の所属する研究室で研究開発が進められている組込み用 OS『開聞』[3] を使用した．JavaVM は Java のサブセットである WabaVM[4] に機能を追加して使用する．Waba はマルチスレッド機構をサポートしていないので，マルチスレッド管理機構は新たに実装した．このときにグリーンスレッド，ネイティブスレッド，ダイレクトスレッド方式の三つの方式を実装した．この全体構成をそれぞれ図 6，図 7，図 8 に示す．

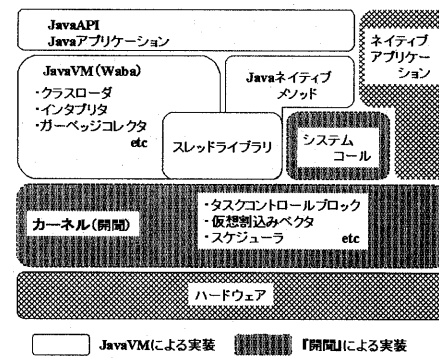

図 6 グリーンスレッド方式の全体構成

4.2 組込み用 OS『開聞』

『開聞』は仮想化のレベルを下げることが可能な設計となっている．グリーンスレッド方式，ダイレクトスレッド方式に完全に対応することができる．すなわち，例外処理，コンテキスト保存などの処理をユーザプログラムで処理が可能である．仮想化のレベルを上げるとタスク管理機能と割込み管理という必要最低限の機能を持つ．

スケジューラは実行可能状態のタスクから最

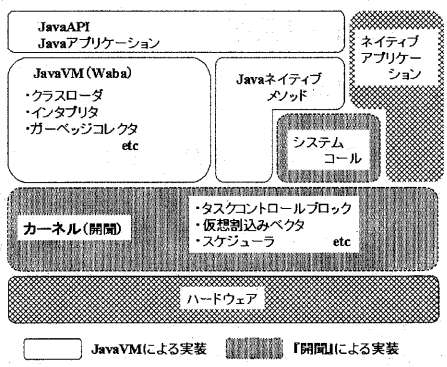

図 7 ネイティブスレッド方式の全体構成

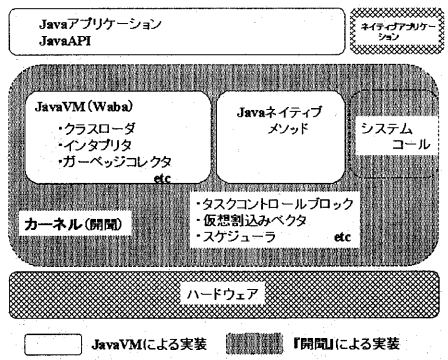

図 8 ダイレクトスレッド方式の全体構成

も優先度の高いタスクを選択して CPU に割り当てる.

4.3 WabaVM

Waba は Java アーキテクチャに制限を加えることにより，PDA 用などで使用されている．小型で，移植性が高い Java 実行環境である．本稿はマルチスレッドの構築を考えているので，マルチスレッドに対応していない Waba にマルチスレッド機能を拡張，三つの方式を実装した．Waba は OS に依存している部分と独立している部分が分かれている．この独立部分は変更を加えずに使用することにした．依存部分に関しては作り直した．依存部分にはクラスローダ，ネイティブスレッド，型などがある．実装する方式は，共通する部分も多く，特にネイティブスレッドを使用する 2 方式は呼び出し以外は，ほぼ同じである．3 方式のプログラムサイズを表 2 に示す．3 方式の処理の流れを図 9 に示す.

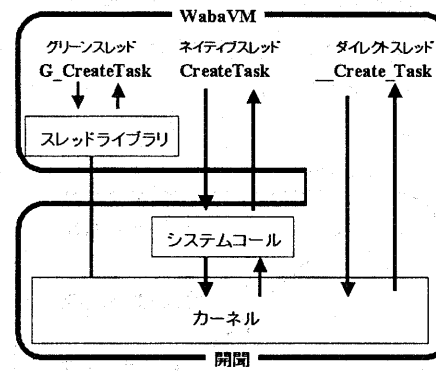

図 9 処理の流れ

表 2 Waba のプログラムサイズ

方式	総行数 (追加行数)	実行サイズ
グリーンスレッド	5728(1368)	80.5
ネイティブスレッド	5538(1178)	75.6
ダイレクトスレッド	5538(1178)	75.6

単位 Kbyte

5 評価

実験に使用したプログラムはスレッドを生成して HelloWorld を表示し，そのスレッドを破棄する．時間計測には純粋にスレッド生成，破棄にかかる時間を計測した．実験に使用した評価用ボードの仕様を表 3 に示す．グリーンスレッド，ネイ

ティブスレッドともメモリ領域は静的に確保した．メモリ確保を静的に実施した目的は，オーバーヘッドを削減するためである．したがって，スレッドの数も静的に決定する必要がある．また，Java 言語仕様でメモリ保護は可能であることから，すべて特権モードで動作させた．スレッド生成，破棄に対する実装の結果を表 4 に示す．

表 3 評価用ボードの仕様

機能	仕様	備考
CPU	SH3(SH7708)	クロック周波数 60MHz
メモリ (ROM)	27256	32K バイト 16 ビット
メモリ (RAM)	HM51w18165	4M バイト 32 ビット
シリアルポート (1/2 ポート)	PD4721	

スレッド生成にかかるグリーンスレッドとネイティブスレッドの差は割込み要因の解明を実行し，割込みハンドラに処理を移す時間であると考えられるが，グリーンスレッドのコンテキスト保存，復元には約 4μsec かかっていることからこの差が縮小した．グリーンスレッド方式ではコンテキストを選択できるため，退避，保存は高速にできることが多いが，本稿ではコンテキスト資源を他の方式と同じにした．未実装ではあるが OS から参照することができれば割込みハンドラを Java 言語で記述できるのではないかと考えたからである．グリーンスレッドとダイレクトスレッドの差はグリーンスレッドのコンテキスト保存，復元が原因であると考えている．ネイティブスレッドとダイレクトスレッドの差は他の部分がすべて同じであるので，コンテキスト退避，保存と割込み要因の解明部分 (システムコール) である．

スレッド破棄にかかるネイティブスレッドと他の方式の差は割込み要因の解明部分である．スレッド破棄はリスト操作だけを実行しているからである．グリーンスレッドとダイレクトスレッドの差はなかった．このときグリーンスレッド，ダイレクトスレッドはコンテキストの退避，復元を実行しない．

表 4 時間計測結果

	スレッド生成		スレッド破棄	
方式	最善値	最悪値	最善値	最悪値
グリーンスレッド	9	10	2	2
ネイティブスレッド	10	14	5	6
ダイレクトスレッド	5	6	2	2

単位 (μsec)

三つの方式を実験して，ダイレクトスレッド方式が一番良い結果となった．その数値的な差を考えると他の方式に大きなメリットがないとダイレクトスレッド方式で設計をすることが組込み用の Java マルチスレッド構築には一番良いと考えられる．ダイレクトスレッド方式はその実装方式のため，JVM 自身の移植性が良くない．現在この問題に対しては『開聞』の移植性を向上することで対処する．

6 おわりに

本稿では組込み用 Java マルチスレッド機構を OS にどのように構築すれば優先度を保持できるか，効率的であるかという点を述べた．

JVM を OS に構築する場合には通常，グリーンスレッド方式，ネイティブスレッド方式がある．本稿では新たにダイレクトスレッド方式を提案した．組込み用 OS『開聞』の上に WabaVM を構築して，各方式を実装し，スレッドの生成，破棄について時間を計測した．その結果，ダイレクトスレッド

方式がグリーンスレッド方式よりも約38%，ネイティブスレッド方式よりも約44%削減できることを示した．PC上では移植性が最も問題とされるが，組込みシステムではオーバーヘッドを削減することが必要である．

参考文献

[1] G.Bollella, B.Brosgol, P.Dibble, S.Furr, J.Gosling, D.Hardin,M.Turnbull, Javaリアルタイム仕様, ピアソン. エデュケーション, 2000.

[2] http://www.itron.gr.jp/SPEC/jtron2-j.html

[3] 萱嶋, 並木, 組込み用OS『開聞』の割込み管理機構, 情報処理学会研究報告, 2000-OS-84, pp.47-54, 2000.

[4] http://www.wabasoft.com/

[5] Y.Gu, B.S.Lee, W.Cai, Evaluation of Java Thread Performance on Two Different Multithreaded Kernels Operating Systems Review, Vol.33, No1, pp.34-46, 1999.

[6] T.Saulpaugh, C.Mirho, インサイドJavaOSオペレーティングシステム, ピアソン. エデュケーション, 2000.

[7] S.Oaks, H.Wong, JAVAスレッドプログラミング, 株式会社オライリージャパン，1997.

[8] T.Lindholm, F.Yellin, The Java仮想マシン仕様, アジソン・ウェスレイ・パブリシャーズ・ジャパン, 1997.

[9] J.Meyer，T.Downing, Javaバーチャルマシン, オライリージャパン，1997.