システムソフトウェアとオペレーティング・システム　86－8
(2001. 3. 5)

組込／分散／実時間用 Java における GC 及びスケジューリング機構の設計

関山 守*　宍道 洋**　高橋 栄一*　山口 喜教***　戸田 賢二*

*電子技術総合研究所 情報アーキテクチャ部
〒305-8568 茨城県つくば市梅園 1-1-4

**（株）明電舎 総合研究所
〒141-8565 東京都品川区大崎 2-1-17

***筑波大学 電子・情報工学系
〒305-8577 茨城県つくば市天王台 1-1-1

あらまし　組み込み機器および分散環境上において、実時間処理が可能な Java の処理系を設計する際に検討すべき問題である、実時間 Garbage Collection とスケジューリング機構の設計に関して述べる。

キーワード　実時間 Java、GC、スケジューリング

Design of GC and Scheduling Structure of Real-Time Java for Embedded/Distributed Systems

Mamoru Sekiyama*　Hiroshi SHINJI**　Eiichi TAKAHASHI*
Yoshinori YAMAGUCHI***　and　Kenji TODA*

Electrotechnical Laboratory (1-1-4 Umezono, Tsukuba, Ibaraki 305-8568 Japan)

Central Research Laboratory, MEIDENSHA Corporation
(2-1-17 Ohsaki, Shinagawa-ku, Tokyo 141-8565 Japan)

Tsukuba University (1-1-1 Tennodai, Tsukuba, Ibaraki 305-8577, Japan)

Abstract We discuss designs about structure of Real-Time Garbage-Collection and Task-Scheduling that will be arisen on implementing Real-Time Java on embedded or distributed systems.

key word Real-Time Java, GC, Task-Scheduling

1. はじめに

組込み機器などの実時間性を必要とする局面での Java の適用の要望は近年徐々に増している。それは Java 環境のもつ様々な利点から来るものであり、実時間処理が可能な Java 処理系も現れてきている。しかし、多くの組み込み用 Java 処理系では真に実時間性を必要とする場面においては他の実時間処理系の補助を必要としているというのが現状である。Java の持つ根本的な非実時間性がその要因であり、主には Garbage Collection を用いたメモリアロケーションによる実行時間の非決定性が原因と言える[1]。

また Java がマルチスレッドを目的に設計されているのに対して、実際の実行でのタスク(スレッド)スケジューリングのポリシーは Virtual Machine に依存している。そのため、ユーザー独自のスケジューリングポリシーを実行に反映するためには、各ベンダーから供給される VM をユーザーが「改造」することによってのみ実現される。「きめこまやかな」実時間処理を行いたいユーザーにとっては処理系の変化によって多重の労苦を必要とすることになる。すなわち Java のもつ機種非依存性という利点は VM 依存性によって打ち消されているともいえる。

Java 本来の利点を生かすためにも、実時間処理用 JavaOS は必要であり、組込み機器から分散環境まですべての局面において Java による実時間処理はその Java の利点から有効な手段となりうると 我々は考えている[2]。

本研究の目的は実時間用 Java 処理系の設計と実装である。本稿では実時間 Java の実現のための実時間 GC を実装するための考察およびタスクのスケジューリング機構の実装方式についての提案を行う。

本稿の構成は以下の通りである。第 2 章では我々の実時間 Java の実現方式について概説する。実際の設計手法およびターゲットとしているアーキテクチャ等についても述べる。第 3 章において、実時間 GC 実装について考察する。Kaffe GC を実時間化する為に必要となる変更点およびその設計について説明する。第 4 章ではタスクのスケジューリング機構の実装について述べ、第 5 章でまとめ及び今後の方針について述べる。

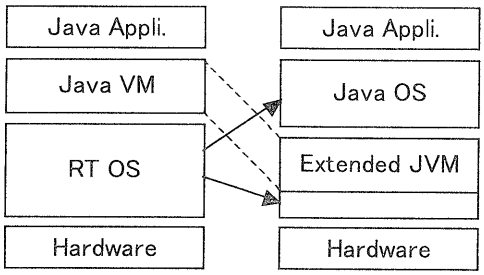

図1:Java実時間環境

2. 実時間 Java の実現方式

2.1. 実時間 Java の設計方針

我々は実時間 Java 実現の方法を次のように計画している。

基本となる実時間 OS およびその上で実行する JavaVM の機能を統合して、さらに分割をおこなう。Java byte code の実行といった Java における必須部分と Hardware に本質的に依存するような部分を一まとめにして、我々は新しい Java 実時間環境として提供する。ユーザー側からは実時間 JavaVM としての環境のみが見えており、ユーザーが独自にその VM 上に Java を用いて実時間 OS を記述することを可能とする。実際のアプリケーションはその実時間 OS 上で実行することが前提となる。(図 1)

このような構想を実現するためには Java 自身の言語の拡張および新しいクラスの設計などが必要となる[3]。さらにもととなる OS および VM のどの機能を環境側にいれるかの振り分けが必要である。

2.2 開発の現状

開発は現在、PC (Intel X86 系) +Linux+Kaffe 上で行っている。これはのちに OS を ART Linux[4]に変更することを念頭においている。また X86 系 CPU 上で開発を行っているのは、将来的に我々の開発した組み込み機器向けプロセッサカードである PPC 上で動作させること

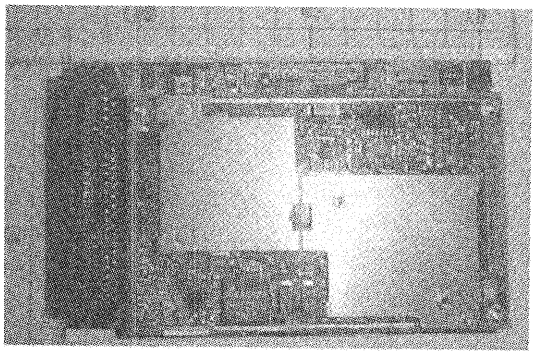

図2：Pentium Processor Card

を目的としている[5]。(図2、スペックは下表)

現在は Kaffe の GC の実時間化および ART Linux のスケジューラと Kaffe の「擦り合わせ」を行っている。

Main CPU	Intel Mobile Module (PentiumII/III/Celron)
ROM	1MB Flush ROM
RAM	128MB DIMM
I/O	PCI,Ether,USB,RS232C,IDE

Pentium Processor Card (PPC) spec

3. Kaffe GC の実時間化

3．1 Garbage Collection の基礎概念

GC の基礎概念とは次のようなものである。

OS は Application の実行中に Application から要求される Object を生成するためのメモリをヒープエリア上の空いている空間に順次確保を行い Object の死亡(使われなくなった時点で死亡) に伴い割り当てたエリアを開放する。Application の実行がすすむに伴ってヒープエリア上の空いている空間は「見かけ上は」少なくなる。これは Application 実行中に死亡した Object に割り当てた空間は再利用されずに放置されるためである。「見かけ上のヒープエリアの空き」が要求されたサイズに足りない場合にその時点までで死亡している部分のエリアを再利用するために生存している Object (メモリ空間上に断片的に存在) をヒープ上に再配置を行い本来使用されていないエリアの回収を終了する。この一連の動作を GC として指す。また Java のように明示的に Object の死亡を宣言しない言語の場合は Object の生死の判断も OS 側が行う必要が出てくる。基本的には Object の生死の判断及びメモリ断片化の解消 (コンパクション) までを含めて GC と呼ぶ。

Object の生死判断に関して種々のアルゴリズムが提案されており、またコンパクションの手法についても様々な方法が提案されている。

上記のようなアルゴリズムに基づいているため、GC を持つシステムは実時間処理には向いていない。まず GC の起こるタイミングは不定期である。次に GC は少なくとも1度は全ヒープエリアの探索を行う必要があるため実行に時間がかかる。また、GC 実行中は Object の生死の変化を避けるために全ての Application の実行を停止することにより「間違い」を少なくすることが多い。

しかし、GC の実時間化の研究は進められてきた。GC のアルゴリズムに中断可能なものを導入することにより実時間処理を行うことを実現したシステムも存在する。中断が可能となることで常に GC を起動することが可能となり、断片化の頻度を抑えることもできると考えられる。

とはいえ現在身の回りに存在する Java VM には、実時間 GC は搭載されていないというのが実情である。我々が今回システムのベースとして用いている Kaffe の GC も実時間化されておらず、実時間化の必要がある。

3．2 Kaffe GC

Kaffe VM での GC は、設計そのものは実時間 GC として多く用いられている「インクリメンタル GC」を目指したものとなっているが、実際には単なる「マークスイープ GC」となっている。マークスイープ GC とは、GC 開始後ヒープ上の全ての Object に対して、ルートから参照によって接続されている全ての Object に印を付ける。(ルートは生存、かつ生存している Object から参照される Object は生存している

と考える)すべてのObjectに対して印を付ける作業が終了したのち印のついていないObjectを死亡Objectとして回収の対象とする、というアルゴリズムである。さらにKaffe VMにて用いられているマークスイープGCでは「3色マーキング」という手法が用いられている。なお「3色マーキング」はインクリメンタルGCでも用いられる手法である。印を付ける際に「白」「灰色」「黒」の三色を用いることからこの名前が付いた。「白」が死亡に、「黒」が生存に、「灰色」が生存かつ探査の起点という意味を与えられている。全てのObjectを「白」くした後、ルートを「灰色」にぬる。以後は「灰色」のObjectから参照されるObject全てを「灰色」に塗り替えた後に基の「灰色」のObjectを「黒」く塗る。この操作を「灰色」のObjectが無くなるまで繰り返し、最後に「白」が死亡と確定できる。

「3色マーキング」はマーキングの一手法であり、利点はどの時点でも中断をすることが出来るという事である。(ただし、アルゴリズム的にという意味であるが。)単純にマークの種類を2種類しか与えない場合中断することは不可能であるが、「3色マーキング」であれば中断後の再開が可能である。だが、Kaffe VMでは実時間処理可能な(あるいは中断可能な)インクリメンタルGCではなく、マークスイープGCを行っている。これはGC中断中に動作する他のスレッドが行う「Objectの参照」の切断・追加・変更の取り扱いが難しい為であろうと考える。

Kaffe VMの現行のGCでは開始からすべての灰色のObjectが存在しなくなり、メモリの開放が終了するまでは中断することはない。このGCのルーチンの中では中断可能な部分が存在すると思われる。また、条件をつけることにより中断可能となる部分も存在する。(ex.「灰色」あるいは「白」のObjectが新しいObjectを生成した場合には問題は存在しない。→後々「黒」くなるであろうから。「黒」のObjectが新しいObjectを生成した場合には取り扱いに注意が必要である。→「黒」Objectから新たに参照を辿ることはありえないため、新しいObjectの色は変更されない。⇒色を「継承」する等の解決策が考えられる。)このようにKaffe VMのGCのメインルーチンを「しらみつぶし」にチェックし、中断できる部分をつくっていくことによって実時間GCを実現することが出来ると考える。

3．3 実時間化の実際

GCとは先にも述べた通り「生死判断」(マーキング)と「コンパクション」(スイープ)から構成されている。ではあるが実際のところ問題とされるのはマーキングの中断性についてである。これはマーキング時のエラーはfatalなものとなる公算が高い(生存させなければならないObjectを回収する等)のに対して、スイープはメモリアクセスが多くルーチンの実行時間は大きいのではあるが、

- ハードウェア構成を工夫する
- Object管理方法を工夫する
- メモリのページングをうまく利用する

といった手法を用いることによってルーチンの負荷低減および実行の中断を可能とすることが出来るためであると考えられる。Kaffe VMでは割り当てるObjectのサイズがメモリページサイズ(Linux OSでは4k byte)を超えるか超えないかによって割り当て手法を変化させている。これにより実際にメモリ上でObjectを移動させること無くコンパクションと同等の効果を得ている。

しかし前述のことからも判る通りGC(特にコンパクション)に実時間性を与えるにおいては、その実装に非常に依存する。ここでの実装とは、ターゲットマシンのHardware構成や、Javaスレッドの実装法、メモリ空間の利用法、割り込みの管理手法といったGC以外の部分も含めての実装という意味である。GCを高速かつ中断可能にして、応答性能を上げるためにはGC以外の部分の設計を非常に注意深く行わなくてはならない。

とはいえ、現時点ではKaffe VMの手法をそのまま継承して実時間性を加えることを考えている。特に欠点らしきものが見当たらないといえる為である。

4． スケジューリング機構

Kaffeでは，デフォルトでは，Javaを起動した際プロセスが1つだけ生成され、そのプロセ

スが、Linux の vtalarm (virtual time)シグナルを利用して、時間で Java のスレッドの切替を行っている。現在、ART Linux では、vtalarm シグナルはサポートされていない。近い将来サポートを行う予定であるため、現行の Kaffe の仕様でも実時間スレッドの実現が可能となる。

しかし、我々の最終的な目標は Java 環境自身がスレッドの管理を行うことである。スレッド管理を Java が自前で行う場合は、そのスレッド間のロックとそれにともなう優先度継承の機構の実現も自前で行うことになる。この自前のスケジューリング機構は、スケジューリングの記述も Java で行うという我々の方針に沿ったものであり、最終的にはこの方向での実現を予定している。

また、ART Linux 上で動作する環境においては、ART Linux のスケジューリング機能を活用したバージョンは実現が容易であり、開発環境としても有用であるので、まず、それを最初に構築する。その後に自前の環境を実現した版に発展させる予定である。ART Linux による Kaffe のスレッドのスケジューリング機構の実現の手法としては以下のようなものを考えている。Kaffe には、スレッドを OS のプロセスとして実現する P スレッドのオプションがあるため、これを指定することにより、Java スレッドを ART Linux の実時間スレッドとして実現する方法により実時間化を進めている。

5. 終わりに

本稿では我々の研究の経過報告として、実時間 Java 処理系開発における現状およびターゲットとなる Kaffe での GC の取り扱い方とその実時間化について説明した。また、ART Linux と Kaffe を統合、分割後の Java 実時間処理環境を提供する際のスケジューリング機構を構成するための手法について論述した。

今後は ART Linux 上で実時間化を進めると共に ART Linux と Kaffe を統合して分割する作業を行う。Java OS 記述のための環境となる部分の定式化を行いつつ Java OS の開発を行う。また、PPC 上に実装を行い組込み環境での開発から PPC でネットワークを構築しての並列・分散環境上での実時間 Java 環境の開発を進める予定である。

参考文献

[1] Nilsen, K., Schmidt, W., "A High-Performance Hardware-Assisted Real-Time Garbage Collection System", Journal of Programming Languages, vol.2, no.1, pp.1-40, January 1994.

[2] 関山, 戸田, 高橋, 山口, 「組込みから並列・分散までの実時間 Java の統一環境の設計と実装」, 電子情報通信学会, RTP99 予稿集 CPSY98-170,pp.33-38,1999 年 3 月

[3] 宍道, 関山, 戸田, 高橋, 山口, 「並列・分散をめざした実時間組込み用 Java の基本設計」, 電子情報通信学会, RTP2000 予稿集, CPSY99-126, pp.23-28,2000 年 3 月

[4] http://esd.etl.go.jp

[5] 戸田, 石綿, 関山, 高橋, 山口, 「市販プロセッサと FPGA から構成される計算ノードをもつ実時間並列システムアーキテクチャ」, 電子情報通信学会, SWoPP99 予稿集 FTS99-35, pp.31-36, 1999 年 8 月