3A-5

Android の応答性評価

東山知彦† 増田大樹† 落合真一†
三菱電機株式会社 情報技術総合研究所†

1. はじめに

近年、Android を携帯端末以外の組込み機器に適用する動きが高まっている。組込み機器は機器が求める応答性を確保する必要がある。

一般的に高い応答性を必要とする処理は他の処理に割り込まれないようにLinuxのリアルタイム優先度(固定優先度)で実行する。しかし、Androidアプリは一般ユーザ権限で動作するため、TSSスケジューリングの範囲しか優先度設定できない。そこで我々は特定のアプリの実行優先度をリアルタイム優先度に設定して起動する仕組みを構築した(参考文献[1])。だが、この仕組みを利用した場合でも、アプリがJavaのガーベージコレクション(以降GC)の影響を受けることは避けられない。本稿はGCが動作する際の応答性に与える影響を評価し、影響を小さくするための改善策について述べる。

2. 応答性評価

2.1 評価環境

本稿は表 1、表 2 に示す構成で評価を行った。

表 1 H/W 構成

機能	内容・性能
プロセッサ	TI DM3739 1.0GHz (ARM Cortex-A8)
主記憶	DDR 512MB

表 2: S/W 構成

コンポーネント	バージョン
Android	4.0.3 (Ice Cream Sandwich)
Linux kernel	2.6.37(Android 対応)

2.2 前提条件

Java で GC が動作すると、アプリケーションが動作を停止してしまうため、応答性が必要な処理も GC 完了まで停止することになる。

GC は新たなオブジェクトを生成する等のメモリが必要になった場合に動作する。また、Android 2.3 以降で導入された Concurrent GC はアプリケーションのスレッドと異なるスレッドで GC 処理の一部を実行することで、アプリケーションの停止時間を短くすることができる。

そこで、応答性が必要な処理は新たなオブジェクトを生成せずに処理を行うよう設計することでメモリ確保による GC 発生を抑制し、応答性が必要な処理と GC が必要となる処理とを別スレッドにすることで、GC の影響を最小限に留める対策が考えられる。

2.3 評価方法

応答性評価アプリは周期的に起床する測定スレッドと GC を誘発するための負荷スレッドから構成される。測定スレッドは応答性が必要な処理を想定しているもので、負荷スレッドは GC が必要となるアプリケーションの一般処理を想定したものである。

測定スレッドは Java の TimerTask クラスを使用し、一定間隔で起床して想定した起床時間と実際の起床時間のジッタを測定する。負荷スレッドは無限ループで参照されないオブジェクトを繰り返し作成する。

2.4 評価結果

参考文献[1]の仕組みを利用しリアルタイム優先度で起動した評価アプリでジッタを計測した。GCが発生しない場合の結果を図 1に、GCが発生する場合の結果を図 2に示す。評価スレッドが 12 万回起床する間の応答遅延を測定した結果、GCが発生しない場合の最悪値は 2.7msだったのに対し、GCが発生する場合は 66.0msであった。このことから、優先度を高くしてアプリを起動しても、GCの影響により応答性が大きく劣化してしまう事が分かった。

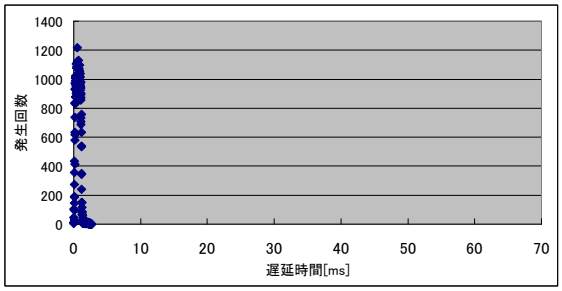

図 1：GC が発生しない場合の応答性評価結果

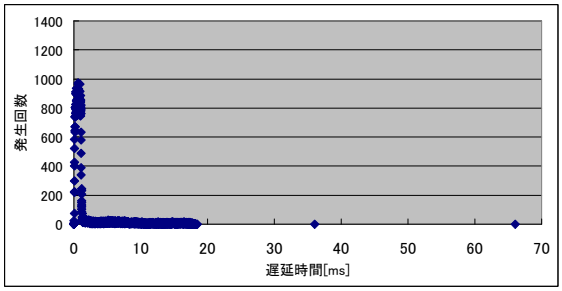

図 2：GC が発生する場合の応答性評価結果

Evaluation of Android realtime performance
Tomohiko Higashiyama, Hiroki Masuda, Shinichi Ochiai
†Information Technology R&D Center, Mitsubishi Electric Corporation

3. 応答性改善方法の検討

3.1 原因の考察

Androidはアプリごとにプロセスが生成され、各プロセスでJava仮想マシン(Dalvik VM)が動作する(図3)。そのため、プロセス起動時に優先度を高くする事で他のアプリがGCで停止しても優先度を高くしたアプリは影響を受けない。

一方 Android アプリ内に目を向けると、開発者が作成した測定スレッドと負荷スレッドの他に、GC スレッドや Binder スレッド(プロセス間通信用スレッド)などのスレッドが動作している。参考文献[1]の仕組みはアプリをリアルタイム優先度で起動するものであるが、アプリ起動時に優先度を高くすると、これらのスレッドの優先度も全て高優先になってしまう。また、設定した優先度はリアルタイム優先度(固定優先度,FIFO スケジュール)のため、各スレッドで一連の処理が終わるまで CPU を開放しない。そのため測定スレッドにスケジュールされず応答遅延が発生していると考えられる。

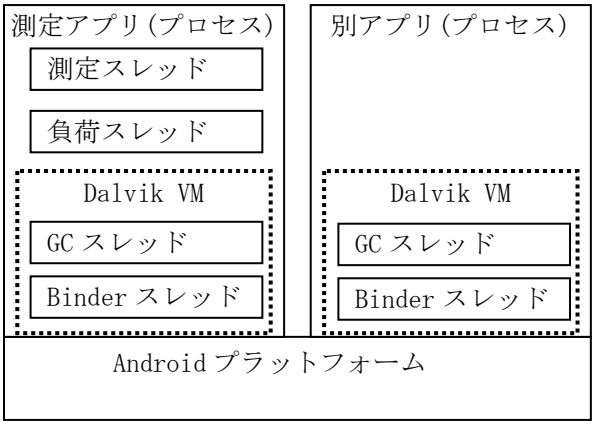

図 3：アプリと各スレッドの関係図

3.2 改善策検討

前節の考察から、GC スレッドや応答性を高くする必要の無い負荷スレッドなどは優先度が低いまま、測定スレッドのみの優先度を上げることで結果が改善すると考えられる。

そこで、スレッド単位にリアルタイム優先度を設定できるようにするために、アプリ起動時には優先度の設定を行わずに、実行時に優先度を変更できるよう Thread クラスの setPriority()メソッドを拡張した。

以下に拡張した setPriority メソッドの使用例を示す。本来の setPriority は引数に 1 から 10 の範囲のみを受けつけ、それ以外は例外を返すものである。拡張した setPriorty()は 100 から 199 の範囲の値が入力された場合に、呼び出し元のスレッドの優先度を Linux の固定優先度「(引数)-100」に設定するようにする。以下の例では、優先度が 99 に設定される。

```
Thread th = Thread.currentThread();
th.setPriority(199);
```

3.3 改善策評価

前節で追加実装した機能を用い、測定スレッドのみをリアルタイム優先度(99)に設定して応答性評価を行った。評価結果を図 4に示す。最大応答遅延は16.8msとなり、改善前の応答遅延を 1/4 程度に抑えることができた。以上から拡張機能を用いて特定のスレッドの優先度をリアルタイム優先度に設定する事でGCの影響を軽減できることがわかった。

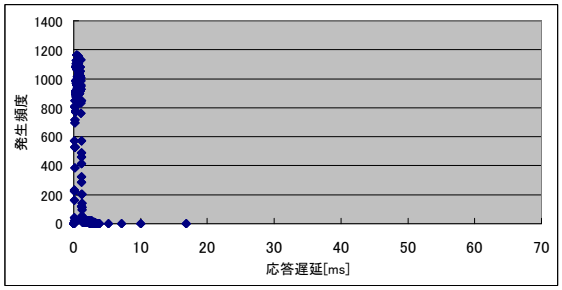

図 4:計測スレッドのみリアルタイム優先度にした場合の応答性評価結果

4. 結論と今後の課題

今回、参考文献[1]の仕組みを利用して、アプリ起動時にリアルタイム優先度に設定した場合に、GCが発生する際のAndroidの応答性評価を行った。その結果、アプリ起動時にリアルタイム優先度に設定したのではGCの影響を受けてアプリの応答性が大きく劣化してしまうことが分かった。

この結果を改善するため、実行時にスレッド単位で優先度を変更する機能を追加実装した。この機能を用いて、特定のスレッドの優先度をリアルタイム優先度に設定して応答性評価を行ったところ、応答性を大きく改善することができた。

今後は更なるGCの影響軽減方法を検討していく予定である。図 1と図 4の結果を比較すると、本稿で実装した機能を適用後もGCの影響により応答性能が劣化していることが分かる。GCの影響を回避するための方法として、1)アプリの中の応答性が重要な部分を、アプリ本体のプロセスと分離して別プロセスとして実行する、2)アプリをネイティブ言語で記述する、などの方法を考えている。

今後、上記の 2 つの実装方式の応答性評価を行い、特定の処理の応答性を確保するための指針を提案していく予定である。

5. 参考文献

[1] 東山知彦、増田大樹、松本利夫：Android のリアルタイム性評価及び改善方法の検討(情報処理学会第 74 回全国大会)