

Android のリアルタイム性評価及び改善方法の検討

東山知彦[†] 増田大樹[†] 松本利夫[†]

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 を携帯情報端末以外の組み込み機器に適用する動きが高まっている。一般的に、組み込み機器へ OS の適用を検討する際は、その OS が H/W リソース制約、消費電力、リアルタイム性を満足するか評価する必要がある。その中でもリアルタイム性は特に重要な要素の 1 つである。例えば、制御システムに対し数秒もの応答遅延が発生する可能性のある OS を搭載することはできない。そこで本稿は Android のリアルタイム性を評価し、評価結果を元にリアルタイム性改善方法について検討する。

2. Androidの構成

Android は Linux カーネルをベースとしており、その上にライブラリ群が存在する。本稿では、ライブラリ群と Linux カーネルの領域をまとめてシステム領域と定義する。また、Android は Linux カーネル上で Dalvik という仮想マシンが動作しており、アプリケーションは基本的にこの Dalvik 上で動作する。本稿では Dalvik、アプリケーション、アプリケーションフレームワークをまとめてアプリケーション領域と定義する。

3. リアルタイム性評価方法

3.1 評価環境

今回の評価は、BeagleBoard.org が製造・販売している評価ボード BeagleBoard-xM 上で実施した。Table 1 に BeagleBoard-xM の H/W 構成を示す。

評価対象の Android は TI 社が提供している TI-Android-Gingerbread-2.3-Devkit-1.0^[1] (以下公開版 Android) を使用した。Table 2 にこの Devkit の S/W 構成を示す。

Table 1 BeagleBoard-xM の 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
外部記憶	micro SDHC(8GB)
ネットワーク	100Mbps Ethernet

Table 2 TI-Android-Gingerbread-2.3-Devkit-1.0 の S/W 構成

コンポーネント	バージョン
Android	2.3 (Gingerbread)
Linux kernel	2.6.32(Android 対応)

Evaluation and improvement of Android as a realtime Operating system

Tomohiko Higashiyama, Hiroki Masuda, Toshio Matsumoto

[†]Information Technology R&D Center, Mitsubishi Electric Corporation

3.2 評価方法

評価のために、リアルタイム応答性測定アプリケーション(以下測定アプリ)を作成した。測定アプリはタイマ機能を用いて 30ms の周期起床を行い、想定する起床時間と実際の起床時間のジッタを測定し、ジッタ分布を出力する (Fig. 1)。また、Java のライブラリメソッドである Thread.setPriority() メソッドにより、測定アプリのスレッド優先度を最大にして実行した。

測定中は以下の 2 つを行いシステムに負荷をかけた。

- (1) 測定アプリの起動と同時に別プロセスでサービスを生成し、そのサービスが演算処理を繰り返す。
 - (2) 外部の PC から ICMP フラッティング攻撃を行う。
- (1) は cpu 負荷を (2) は割り込み負荷を意図したものである。

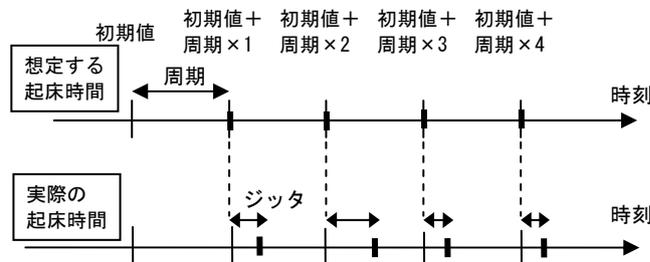


Fig. 1 タイマによるリアルタイム性評価方法

4. 評価結果・考察

4.1 公開版 Android のリアルタイム性能

3.1 節で述べた公開版 Android を対象に、リアルタイム性評価を行った結果を Fig. 2 に示す。

評価の結果、公開版 Android は 30ms の測定周期に対し最大 12.0ms もの遅延が発生することを確認した。

4.2 システム領域のリアルタイム性改善

前節の結果を改善するため、システム領域における改善を試みた。測定アプリはスレッド優先度を最大にしているが大幅な遅延が発生した。Linux カーネルのスケジューリングを確認したところ、カーネルの Preemption Model が [No Forced Preemption] に設定されていることが分かった。そこでカーネルモード実行中でも優先度の高いプロセスが実行可能になった時点で実行できるように、Preemption Model を [Preemptive Kernel] に変更した。

次に、スケジューリング以外の遅延原因を探るため、/proc ディレクトリ以下を確認し、システム全体の状態を分析した。その結果、ICMP 受信時にアライメントエラーが発生していることがわかった。このことから、トラップ処理が頻発し、この処理により遅延が発生している可能性が考えられる。そこで、アライメントエラーが発生しないようにカーネルに処理を施した。

上記 2 点を修正し、リアルタイム性評価を行った。その結果を Fig. 3 に示す。公開版 Android の結果と比べて、ジッタ分布のゆらぎ幅は小さくなった。しかし、最大遅延は 10.2ms であり、未だ改善の必要がある。

4.3 アプリケーション領域のリアルタイム性改善

前節の課題である最大遅延の大きさを改善するため、アプリケーション領域を調査した。測定アプリの /proc/[PID]ディレクトリ以下を確認した結果、プロセス優先度は普通優先度設定になっていた。つまり、Java スレッドの優先度は最大であっても、アプリケーションのプロセス自体の優先度がリアルタイム優先度でなかったために、プリエンプトすることができず、前節の遅延が発生したと考えられる。

普通優先度のプロセスをリアルタイム優先度に設定するには、root 権限が必要である。しかし、Android アプリケーションは一般ユーザー権限で動作するため、自身をリアルタイム優先度に設定することはできない。そこで、シリアルコンソールからのコマンド入力により、外部から優先度を変更し測定を行った。

測定の結果、Fig.4に示すとおり、最大遅延時間を 3.1msに抑えることができた。この結果から、アプリケーションをリアルタイム優先度に設定することで、応答遅延を大きく改善できることが分かった。

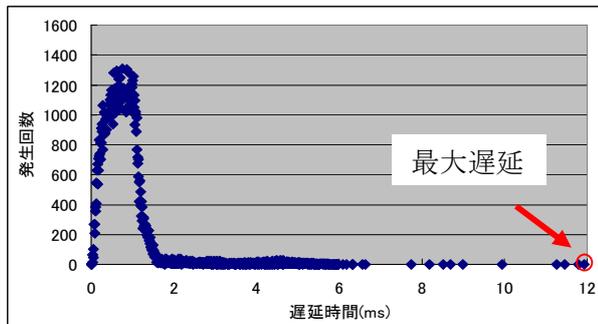


Fig.2 公開版 Android のリアルタイム性評価結果

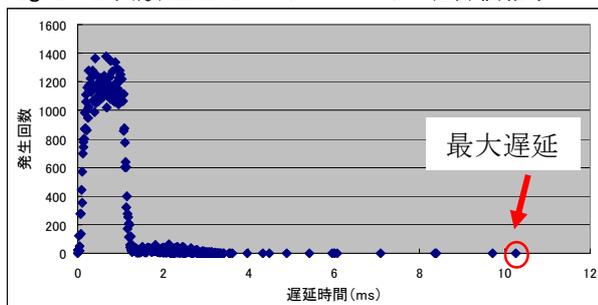


Fig.3 システム領域に修正を施した Android のリアルタイム性評価結果

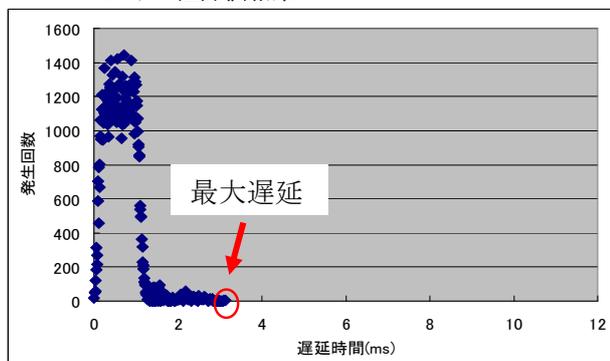


Fig.4 アプリケーション領域に修正を施した Android のリアルタイム性評価結果

4.4 優先度設定自動化の改善試作

前項では、アプリケーション起動後にコマンドライン入力でのリアルタイム優先度に設定したが、製品でコマンドライン入力を行うことは不可能である。そのため、対象のアプリケーション起動時に自動的にリアルタイム優先度設定を行う手法が必要である。考えられる手法として以下のようなものが挙げられる。

- 1) Android では、全てのアプリケーションは Zygote というプロセスから fork されて作られる。その Zygote を root 化し、fork される子プロセスに root 権限を与え、リアルタイム優先度設定を可能にする。
- 2) アプリケーションフレームワークを改造し、特定のアプリケーションをリアルタイム優先度に設定可能にする。
- 3) バックグラウンドで動き続けるプロセスで特定のアプリケーションの起動を監視し続け、起動を検知したらリアルタイム優先度に設定する。

Androidはアプリケーションのプロセス名がクラス名であり、アプリケーションを簡単に特定できるので、今回は3)の手法を試作した。この手法により、ユーザーがコマンドライン入力をせずに、特定のアプリケーションだけをリアルタイム優先度で実行することが可能となった。また、この方法でリアルタイム優先度に設定した測定アプリでリアルタイム性評価を行った結果、Fig.4と同等の結果が得られた。

5. 結論と今後の課題

今回、市販の評価ボード BeagleBoard-xM 上で Android のリアルタイム性の評価を行った。その結果、公開版 Android で 12ms 程度遅延が発生することを確認した。これに対し、1) プリエンプティブモデルを変更する、2) ping 応答時にアライメントエラー発生しないようにする、3) アプリケーションをリアルタイム優先度設定にするという変更を行うことで、最大応答遅延を 3ms 程度に抑えることができた。

今回試作した方法では、事前に決めたアプリケーションはリアルタイム優先度に設定できるが、ユーザーがリアルタイム優先度に設定するアプリケーションを選択することはできない。今後はフレームワークを改造し、ユーザーが任意のアプリケーションをリアルタイム優先度設定できる仕組みを構築する予定である。

また、他の Linux 評価^[2]では 1ms 以下のリアルタイム性も得られており、Android も更なる改善の可能性があり。今回は優先度、異常処理に焦点を絞って改善方法を検討したが、今後は他のリアルタイム性阻害要因の分析を行い、改善方法を検討する予定である。

参考文献

- [1] TI-Android-GingerBread-2.3-DevKit-1.0 ReleaseNotes, http://processors.wiki.ti.com/index.php/TI-Android-GingerBread-2.3-DevKit-1.0_ReleaseNotes
- [2] 徳丸潤一, 摂津敦, 落合真一, 松本利夫: マルチコアシステムにおけるLinuxカーネル2.6のリアルタイム性評価(情報処理学会第73回全国大会)