シングルコア及びマルチコア向けリアルタイム OS の性能とコード共有手法の評価

2019SC025 近藤拓人
指導教員：本田晋也

1　はじめに

RTOS は，リアルタイムシステムを実現するために使用される OS である．組込みシステムはコスト制約が強く可能な限り性能が低い安いマイコンで実現したいという要求があり， RTOS の API(タスクの起動，セマフォの操作関数) の実行時間は短いことが要求される．

一方，組込みシステムにおいても，プロセッサのクロックの上限によりマルチプロセッサシステムが増えている．前述のコスト制約への要求により，製品によっては高機能なものはマルチプロセッサ・低機能なものはシングルプロセッサ構成という場合がある．

RTOS の一種として，TOPPERS プロジェクトから複数種類の RTOS がオープンソースとしてリリースされている．シングルプロセッサ向けには, ASP カーネル (ASP) が，マルチプロセッサ向けには FMP カーネル (FMP) が用意されている．FMP カーネルの設定を変更することで，シングルプロセッサで動作させることは可能であるが，マルチプロセッサ用 RTOS はシングルプロセッサ用 RTOS より実行オーバヘッドが大きいため，OS をそれぞれ用意している．しかしながら，開発効率や保守の観点では，マルチプロセッサ用 RTOS とシングルプロセッサ用 RTOS を分けて開発せず，一つの RTOS で両方をサポートしたいという要求がある．

本研究ではマルチプロセッサ用 RTOS はシングルプロセッサ RTOS と比較してどの程度実行オーバヘッドが大きいか評価し，実行オーバヘッドが増加する点をソースコードを解析して明らかにする．シングルプロセッサ実行時に不要となる，プロセッサ間の排他制御及びプロセッサ ID の取得について省略した場合にどの程度の効果があるか，2 種類のターゲットに対して実施し評価する．

2　背景技術

2.1　RTOS

リアルタイムシステム構築のための OS であり，優先度ベーススケジューリングや優先度継承などのリアルタイムシステム向けの機能や予測可能性，時間制約の管理機能，高速応答性などがある．RTOS を使用するメリットは，ソフトウェアの構造化によって生産性，保守性，信頼性を，論理的，時間的な処理順序の分離によって時間制限付きのソフトウェアの保守性・再利用性を向上させることである．RTOS を使用するデメリットは，RTOS のオーバーヘッドによる実行性能の低下や，RTOS のワークエリア，タスクのスタックエリアによるメモリ消費などがある．

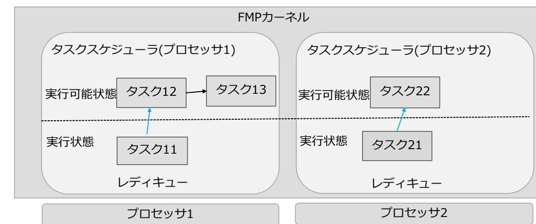

図 1　FMP3 カーネルの概要図

2.2　ASP3 カーネルと FMP3 カーネル

ASP3 カーネル (ASP3) は，TOPPERS 第 3 世代カーネルであり，ITRON 仕様をベースにティックレスと高分解能の時間管理や，外部時刻同期機能，タスク終了要求機能などを持つ．

FMP3 カーネル (FMP3) は ASP3 にマルチプロセッサ対応機能を追加した RTOS である．図 1 に FMP3 の概要図を示す．OS 内部では，プロセッサで実行するタスクはレディキューごとに置かれている． プロセッサ間通信機能や API では，プロセッサ内の通信機能と互換性が必要である．タスクで行うプロセッサ間通信機能は直接操作法で実現する [1].

3　マルチプロセッサ RTOS のシングルプロセッサ向け最適化

マルチプロセッサ RTOS をシングルプロセッサで動作させることを前提とすると，以下の機能を無効とする最適化が可能である．

3.1　ロック

FMP3 では直接操作法を使ってプロセッサ間の排他制御をロックによって行う．ロック単位は 1 つのロックで排他制御を行うリソースの単位である．ロック単位の粒度が小さいときはプロセッサ数が上がりにくいためロックがあまり衝突しない分並列実行しやすい [1]．シングルプロセッサではプロセッサ間の排他制御が必要ないため，FMP3 をシングルプロセッサで実行する場合はロック処理を省略可能である．

3.2　プロセッサ ID 取得

FMP3 ではプロセッサ毎の情報を PCB という構造体に入れている．自分の PCB を取得するために，プロセッサ ID レジスタを読むことでプロセッサ ID を取得してその情報を元にアクセスする PCB を決める．FMP3 をシングルプロセッサで実行する場合は PCB 取得のためにプロセッサ ID レジスタを読む処理，つまりプロセッサ ID 取得を

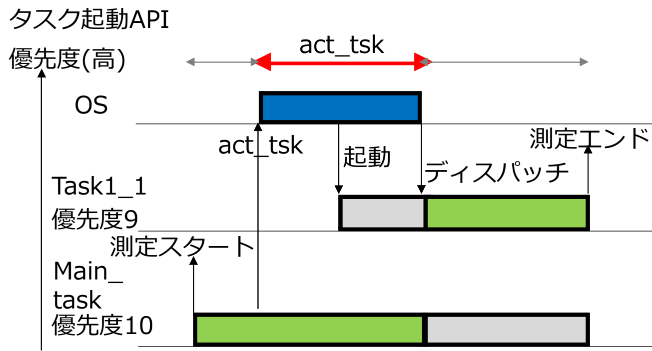

図 2 タスク起動 API のタイムチャート

省略可能である．

4 性能評価

前述の FMP3 に対して最適化手法を実施し，ASP3 と共にその効果を評価する．

4.1 評価項目

各 OS で各種 API の実行時間を評価した．本稿ではタスク起動 API の評価について説明する．

タスク起動 API の評価は，act_tsk で高優先度のタスクを起動してディスパッチするまでの時間を計測する．タイムチャートを図 2 に示す．灰色の矢印部分が測定オーバヘッドで， 赤い矢印部分が測定区間である．

4.2 評価環境

評価環境は Zybo と Pico と呼ばれる性質が異なる 2 種類の組込みシステム向けのプロセッサを用いた．

Zybo：Zybo Z7-10 Xilinx

スマートフォン・TV 等で使用されているものと同等の高速なアプリケーションプロセッサである ARM Cortex-A9 650MHz を 2 個搭載している．ビルドは，Xilinx SDK 2019.1 付属の gcc を用いた．測定はチップ内蔵のタイマーを使用した．

ロックの実現は専用命令を使用し，プロセッサ ID はプロセッサ内レジスタより取得する．

Pico：Raspberry pi pico

家電等に使用されている ARM Cortex-M0+ 133MHz を 2 個搭載している．ビルドは WSL 上で make コマンドを使った．gcc はバージョン 9.2.1 である．時間計測は，10ns 精度のタイマーを用いる．

ロックの実現は専用ハードウェアを使用し，プロセッサ ID はプロセッサ外のレジスタより取得する．

4.3 評価結果

Zybo，Pico を使って ASP3/FMP3 で実施したタスク起動 API の計測結果を求め，図 3，図 4 にそれぞれの最頻値 (ns) を示す．なお，これ以降示す測定結果は，時間計測にかかる時間 (オーバヘッド) を減じた値である．

最頻値の計測結果より，FMP3 の最頻値は ASP3 と比較

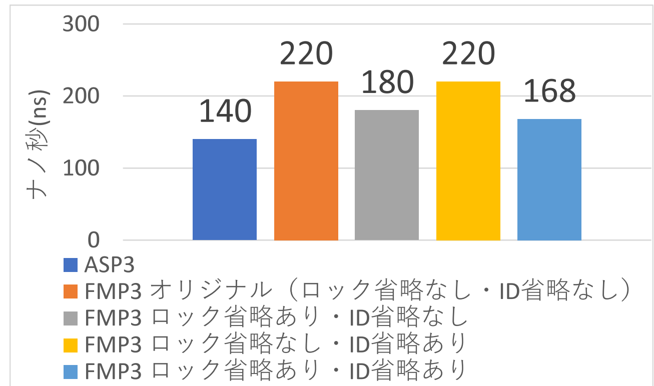

図 3 Zybo：タスク起動 API の最頻値

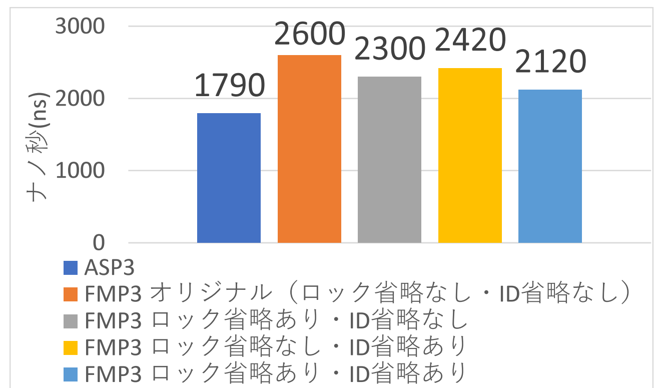

図 4 Pico：タスク起動 API の最頻値

して Zybo では 1.57 倍遅く，Pico では 1.45 倍遅い．ロック省略時の計測結果より，ロック省略なしの場合と比較して Zybo では 18.2%，Pico では 11.5% 高速となった．プロセッサ ID 省略の計測結果より，Zybo ではほぼ効果が無いと言える，Pico では 6.9% 高速となった．これは，Zybo ではプロセッサ ID をプロセッサ内レジスタより取得し，Pico ではプロセッサ ID をプロセッサ外レジスタより取得するためであると考えられる．両者を組み合わせた場合，Zybo では 23.6%，Pico では 18.4% 高速となった．

5 おわりに

本研究ではシングルプロセッサ実行時に不要となる，プロセッサ間の排他制御及びプロセッサ ID の取得について省略した場合にどの程度の効果があるか，2 種類のターゲットに対して実施し評価した．今後の課題は最適化したコードにおける，API の前半部分の実行時間の増加部分を解析することである．

参考文献

[1] 石田利永子，本田晋也，高田広章，福井昭也，小川敏行，田原康宏：TOPPERS/FMP カーネル リアルタイム性と高スループットを実現可能な組込システム向けマルチプロセッサ用 RTOS,” コンピュータソフトウェア, Vol.29,No.4, pp. 219-243 (2012).