B-005

システム設計最適化を目的としたリアルタイム制御用 Linux の性能評価
Performance evaluation of Linux for real-time control for system design optimization

大塚 奎佑† 堀井 圭祐† 山田 竜也† 山本 遼介†
Keisuke Otsuka Keisuke Horii Tatsuya Yamada Ryosuke Yamamoto

1. はじめに

FA 機器や車載機器等の高性能な組込み機器ではリアルタイム処理と情報系処理の両立が求められ，情報系を考慮して OS に Linux が採用される場合がある[1]．Linux は，豊富なソフトウェアの資産を持ち，PREEMPT_RT パッチを適用することでリアルタイム処理も可能である．ただし，リアルタイム処理を行う上では，リアルタイム制御用アプリケーション(リアルタイムタスク)の起床方法やタスク間通信方法などについて注意を払う必要がある．リアルタイム処理を行う際に OS の性能がシステムの要件を満たしていない場合，割込み処理の遅延や処理優先度の逆転など，システムの動作の信頼性が損なわれる．したがって，システム毎のリアルタイム性における要件を満足するための最適な設計が必要である．本研究では，リアルタイム制御を Linux で行う場合に着目し，起床方法の最適化のため，Linux の性能を評価する．

2. 先行研究とその課題

Linux のリアルタイム性能評価に関する先行研究は，いくつかあげられる．BeagleBoard や Raspberry Pi3 上で，PREEMPT_RT パッチを適用した Linux カーネル（バージョン：4.19.10-1-ARCH）のリアルタイム性能を評価した研究では，低コストのシングルボードコンピュータが，リアルタイムタスクに適していることが示された[2]．また，産業用プラットフォーム向けに，Intel Core-i7-6700 上の Linux と Intel Core-i7-6600U 上の PREEMPT_RT パッチが適用された Linux（バージョン：4.14.134）のリアルタイム性能を比較した研究もある．この比較を通じて，リアルタイムタスクが 1ms 以上の周期で動作する場合であれば，低消費電力プロセッサ（Intel Core-i7-6600U）でもリアルタイム制御に適しているといえることが示された[3]．

一方で，これらの先行研究で評価された Linux のバージョンは古く，近年のより新しいカーネルバージョンの Linux と組込み向け x86 系 CPU を組み合わせた場合の性能評価や，タスクの起床方法に着目した性能比較はなされていない．そこで，本研究では x86 系 CPU 上で PREEMPT_RT パッチが適用された近年の Linux カーネルのリアルタイム性能を評価することで，タスク起床方法最適化の指針を得ることを目的とする．

3. 評価方法

3.1 評価項目

本研究では，タスクの起床方法の違いに着目し，3.1.1 から 3.1.3 に示す項目をそれぞれ測定した．なお，各項目における測定回数は 10 万回とし，測定の実行周期は 1ms とした．

† 三菱電機株式会社 情報技術総合研究所
Mitsubishi Electric Corporation
Information Tecnology R & D Center

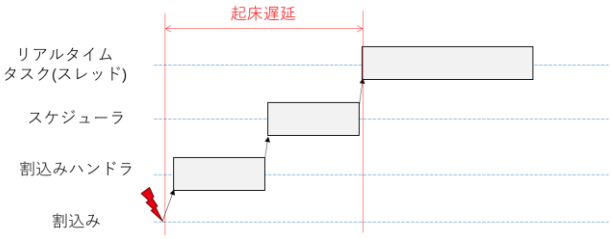

図 1 clock_nanosleep()活用時における
起床遅延の測定箇所

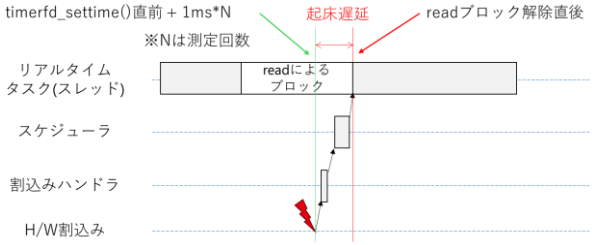

図 2 timerfd 活用時における起床遅延の測定箇所

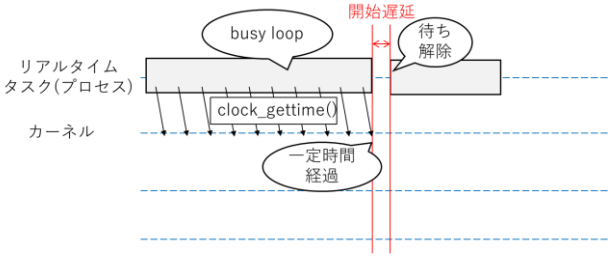

図 3 busy loop 活用時における起床遅延の測定箇所

3.1.1 clock_nanosleep()による定周期タスクの起床遅延

図 1 に clock_nanosleep()活用時における起床遅延の測定箇所を示す．clock_nanosleep()を用いた定周期処理とは，タスクの開始時に次の実行時刻を算出し，その時刻まで clock_nanosleep()によってスリープし，hrtimer の割込みを契機として起床させる方法である．測定には cyclictest (Ver. 2.20)を利用した．

3.1.2 timerfd による定周期タスクの起床遅延

図 2 に timerfd 活用時における起床遅延の測定箇所を示す．timerfd を用いた定周期タスクとは，timerfd_settime()によって設定された時間に達したときにファイルディスクリプタ経由で read()によって停止した処理を hrtimer の割込みを契機として再開させる方法とする．測定には自製のプログラムを活用した．測定プログラムでは，まず，timerfd_settime()の直前に時刻を取得した．次に timerfd を使用して 1ms ごとにスレッドを起床させ，read ブロック解除直後の時刻を取得した．ここで，timerfd_settime()の直前に取得した時刻に 1ms*N（N は測定回数）を加算した時刻

表 1　HW/SW の詳細

項目	内容
CPU	Ryzen Embedded V2546
メモリ	8192MB (DDR4)
コア数	6
OS	Linux 5.15.160
ディストリビューション	Ubuntu 22.04
パッチ	PREEMPT_RT (5.15.160-rt77)

と read ブロック解除直後の時刻との時間差を算出することで，起床遅延を測定した．

3.1.3 busy loop による定周期タスクの開始遅延

図 3 に busy loop により定周期処理を実現する場合における開始遅延の測定箇所を示す．busy loop を用いた定周期処理とは，前回の周期処理が終了してから次の起床時刻に到達するまで busy loop で時刻を確認する方法である．ただし，今回の測定では具体的な周期処理は実行せずに，busy loop だけを行うものとする．

測定には，自製のプログラムを活用した．測定プログラムでは，以下の手順によって開始遅延を測定した．

① 現在時刻を clock_gettime()により取得する
② ①で取得した時刻に "周期(1ms) x 測定回数" を加算する（次回起床時刻の算出）
③ ②の時刻を超過するまで，clock_gettime()を繰り返し実行する
④ ②の時刻を超過した直後の時刻を clock_gettime()により取得する
⑤ ④と②の時刻の差分を起床遅延時間とする
⑥ 繰り返し回数まで②から⑤を実行する

3.2 測定環境

表 1 に本測定における H/W と S/W の詳細を示す．なお，リアルタイム性能評価にあたり，CPU コアは isolcpus オプションを活用しアイソレーションを施したコア（アイソレーションコア）とアイソレーションを施していないコア（非アイソレーションコア）を用意した．非アイソレーションコアは，CPU 負荷を印加する場合に使用する．また，2 種類のコアにおけるリアルタイム性能を比較することで，アイソレーションの効果を確認した．

3.3 CPU 負荷

本研究では，最悪時の性能も把握するため，負荷で処理を阻害された場合についても測定した．なお，負荷を印加するため，iperf3 と stress-ng を活用しており，H/W 割込みやシステムコール，シグナル，メモリ確保，コンテキストスイッチ，ページフォールト，プロセスのスケジュール処理を多発させた．これらの負荷を非アイソレーションコアに対して同時に印加した．

なお，負荷を印加した状態で評価する場合は，アイソレーションコアと非アイソレーションコアのそれぞれにおいて各評価項目を実施した．

4. 測定結果

表 2，表 3 に clock_nanosleep()，timerfd，busy loop の 3 項目における起床遅延の測定結果を示す．表 2 が負荷なしの場合であり，表 3 が負荷ありの場合である．

表 2　タスクの起床遅延測定結果（負荷なし）

	実行コア	最大[μs]	最小[μs]	平均[μs]
clock_nanosleep()	アイソレーションコア	13	2	3
timerfd	アイソレーションコア	18	12	12
busy loop	アイソレーションコア	0.09	0.04	0.06

表 3　タスクの起床遅延測定結果（負荷あり）

	実行コア	最大[μs]	最小[μs]	平均[μs]
clock_nanosleep()	アイソレーションコア	11	3	3
	非アイソレーションコア	21	3	3
timerfd	アイソレーションコア	16	12	12
	非アイソレーションコア	302	10	22
busy loop	アイソレーションコア	0.09	0.04	0.06
	非アイソレーションコア	0.08	0.04	0.06

表 2，表 3 より，アイソレーションコアでの起床遅延は非アイソレーションコアでの起床遅延よりも短くなっていることが分かる．さらに，clock_nanosleep()，timerfd，busy loop のそれぞれの測定結果を比較すると，busy loop による起床遅延が最も短くなることが分かる．

5. 考察

タスクの起床遅延において，clock_nanosleep()による起床遅延の最大値と最小値の差が最も大きくなった．これは，測定プログラムよりも優先度の高いプロセスが動作した影響であり，スリープからの起床はその影響を受けやすくなっていることが考えられる．clock_nanosleep()，timerfd，busy loop の結果の内，busy loop による起床遅延が最も短くなったため，busy loop を活用することでタスクの起床遅延を最も短くできるといえる．一方で，busy loop では，待ち状態のコアを占有するため，リアルタイムタスクの起床遅延時間が 13μs を許容できる場合は，clock_nanosleep()によって起床させるほうが適切である．

また，負荷をかけた場合において，非アイソレーションコアよりもアイソレーションコアのほうが起床遅延は短くなっている．これは，アイソレーションが施されたコアでは，Linux の通常のタスクや割込みが割り当てられなくなるため，負荷の影響を受けにくくなったと考えられる．

6. 結論

本研究では，システム設計の最適化に向けて Linux のリアルタイム性能を評価した．結果として，リアルタイムタスクの起床遅延を抑えるためのシステム設計に対する知見が得られた．今後は，本研究のカーネルバージョンよりも新しい PREEMPT_RT パッチが upstream にマージされたバージョン（6.12）以降における起床遅延を測定する．

参考文献

[1] Giuliano Albanese, Robert Birke, Georgia Giannopoulou, Sandro Schönborn, Thanikesavan Sivanthi, “Evaluation of Networking Options for Containerized Deployment of Real-Time Applications”, 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), (2021).
[2] George K. Adam, “Real-Time Performance and Response Latency Measurements of Linux Kernels on Single-Board Computers”, Computers 2021, Vol. 10, No. 64 (2021).
[3] Jo, Yong Hwan, Choi, Byoung Wook, “Performance Evaluation of Real-time Linux for an Industrial Real-time Platform”, International journal of advanced smart convergence, Vol. 11, Issue 1 (2022).