Unikraftでのプリエンプティブ・スケジューリングの実装とリアルタイム性およびCPUキャッシュへの副作用評価

若山 陽向[1,a)]　朱 義文[1,b)]　松原 豊[1,c)]　高田 広章[1,d)]

概要：組込みシステムではハードウェアコストの削減と異なるOS環境の分離を両立するため仮想化技術が利用されることがあり，省リソース性とリアルタイム性が求められる．既存の選択肢であるLinuxのPREEMPT_RTは多くの汎用機能を備える場合があり，OSのフットプリントが大きい．そこで，必要なOSの機能のみをアプリケーションと一体化し，OSのフットプリントを削減できるUnikraftに着目する．しかし，現行のUnikraftの公式リポジトリでは協調型スケジューリングが採用されており，高優先度処理の実行が遅延する問題がある．そこで，リアルタイム性の改善を図るためプリエンプティブ・スケジューリングを実装し，リアルタイム性を評価した．
その結果，99.99パーセンタイル遅延は最大22 μsとなり，既存のOSと同程度のリアルタイム性を示した．また，プリエンプションに伴うCPUキャッシュミス増加の副作用を明らかにするため，L2キャッシュのMPKIも評価した．その結果，L2キャッシュのMPKIは最大7.6倍に増加したが，ワーキングセットが小さい条件では既存のOSより低いMPKIを示した．以上より，プリエンプション導入後のUnikraftは既存のOSと同程度のリアルタイム性を実現しつつ，特定の条件ではCPUキャッシュ性能の面で優位となる可能性が示された．

1. 背景

1.1 組込みシステムで求められる性質

組込みシステムでは，限られた計算資源の下で効率的に動作することに加え，処理を所定の時間内に完了するリアルタイム性が求められる．物理的な対象の制御では，処理結果の正しさと時間制約の達成がともに重要である [1]．複数の処理がCPUを共有する場合，低優先度処理によるCPUの占有は，高優先度処理の開始を遅らせる可能性がある．したがって，OS自身の資源消費を抑えつつ，高優先度処理の実行開始遅延を低減する必要がある．

1.2 組込みシステムにおける仮想化の利用

組込みシステムでは，ハードウェアコストの削減と処理ごとに異なるOS要求を両立するために仮想化が利用されることがある．例えば，車載システムでは，ECU数や配線の増加に伴うコスト増大が機能集約の動機となる [2]．

また，リアルタイム処理では時間制約が，一般処理では多様な機能や既存ソフトウェアの利用が重視され，OSへの要求が異なる．さらに，安全性要求の異なる機能が混在する車載のMixed-Criticalityシステムでは，重要な機能への障害の波及を防ぐ分離が求められる [2]．具体的には，Mixed-Criticalityシステムでは資源消費による他の実行環境の性能低下を制限する時間的隔離，コードやデータを保護する空間的隔離，障害の波及を防ぐ障害隔離が重要となる [3]．仮想化による複数のOS環境の分離は，これらの要求に対応する構成の一つである [3]．

このような仮想化により各処理のOS環境をVMに分離する構成を行えば，それぞれの要求に適したOSを同一ハードウェア上で利用できる [2]．この構成では，一般処理や管理処理が用いるOS環境を維持しながら，リアルタイム処理には時間制約と必要なOS機能に応じた別のOSを選択できる．本研究は，このようなOS環境の分離を前提として，組込みシステムにおける仮想化環境でのリアルタイム処理を担うゲストOSを評価対象とする．

1.3 汎用OSの問題点

Linuxなどの汎用OSは多様な機能を備える一方，組込み用途では不要な機能も含まれるため，OSのメモリ使用量や実行時のオーバーヘッドが課題となり得る．機能間の依存関係により，不要な機能の除去も容易ではない [4]．さ

[1] 名古屋大学 大学院情報学研究科
Graduate School of Information Science, Nagoya University
[a)] wakayama.hinata.m1@s.mail-u.ac.jp
[b)] shu.yoshifumi@ertl.jp
[c)] yutaka@ertl.jp
[d)] hiro@ertl.jp

ⓒ2026 Information Processing Society of Japan

らに，Linux は，必要な OS 機能に構成を絞った Unikernel と比較してコード規模が大きく，機能間の依存関係も複雑である [4]．そのため，不具合発生時に調査すべきコードや機能間の相互作用が多くなり，原因の特定やデバッグが難しくなる可能性がある．

また，リアルタイム性を高めるために，ロック機構の変更や割込み処理のスレッド化によって実行開始遅延を低減する PREEMPT_RT を適用した Linux がある [5]．しかし，PREEMPT_RT を適用しても不要な機能の除去が容易になるわけではなく，省リソース性の課題は残る．そこで，省リソース性とリアルタイム性の両立に向けて，必要な OS 機能を選択して構成できる OS に着目する．

1.4 Unikernel と Unikraft の特徴と優位性

Unikernel は，アプリケーションと必要な OS 機能を単一のイメージに組み込み，OS のフットプリントを削減する方式である．Unikraft は，メモリ管理やスケジューラなどの OS 機能を独立したライブラリとして提供し，用途に応じて選択・構成できる．静的リンクによって不要なコードを除去できるため，多様な機能を備えた汎用 OS と比較して OS のフットプリントを小さくできる [4]．

また，Unikraft は既存アプリケーションの移植負担を抑えるため，POSIX API および Linux のシステムコール API への対応を進めている [4], [6]．ネイティブにビルドする方式では，musl などの C ライブラリを用いてアプリケーションを再ビルドし，Unikraft とリンクする．この方式では，システムコールを通常の関数呼出しに置き換え，そのオーバーヘッドを低減できる [4]．

これらの特徴から，Unikraft は既存ソフトウェアを再利用しつつ資源消費を抑える実行環境となり得るがリアルタイム性の確保にはスケジューラの検討が必要である．

1.5 Unikraft のスケジューラとプリエンプション導入のデメリット

本研究が対象とする Unikraft の公式リポジトリの標準スケジューラ `ukschedcoop` は，協調型のラウンドロビン方式であり，プリエンプションに対応していない [7]．実行中の低優先度スレッドが自発的に実行権を放棄するか，待機・終了するまで他のスレッドへ切り替わらず，高優先度スレッドの開始が遅れる場合がある．プリエンプティブ・スケジューリングは，低優先度処理を中断して高優先度処理へ実行権を移し，リアルタイム性を改善する．一方，低優先度スレッドが中断中に高優先度スレッドがキャッシュラインを置換すると，再開後のキャッシュミスが増加する可能性がある [8]．したがって，導入効果の評価では，リアルタイム性の改善とキャッシュへの副作用を併せて検討する必要がある．

1.6 研究目的

本研究の目的は，仮想化環境において Unikraft へのプリエンプティブ・スケジューリングの導入によるリアルタイム性の改善とキャッシュ性能への副作用の関係を定量的に明らかにし，他の OS との比較を通じてその特徴を評価することである．具体的には，導入前後の Unikraft と比較対象の OS について，高優先度スレッドの周期的な起床要求に対する遅延と，低優先度スレッドのキャッシュミスを測定する．

2. 関連研究と貢献

2.1 Linux と Unikraft の違い

Linux と Unikraft の違いは，アプリケーションに必要な OS 機能を選択できる粒度にある．Kuenzer らは，Linux では機能間の依存関係が密であり，コンポーネントの除去や置換が難しいことを示している [4]．具体的には Linux ではカーネル全体を必要とする一方，Unikraft は起動処理やメモリ管理など必要な機能だけを組み込み，OS のフットプリントを削減できる．このように，Unikraft は必要な OS 機能を選択することで軽量な構成を実現できる．一方，リアルタイム性を検討するには，スレッドの実行順序や切替契機を決めるスケジューリング方式にも着目する必要がある．

2.2 Unikraft における既存スケジューリング方式

Unikraft におけるプリエンプティブ・スケジューリングの既存実装は非公式には存在するが，公式本流の標準スケジューラは協調型である [7]．主要な実装と本研究の違いを**表 1** に示す．

表 1 Unikraft の標準スケジューラ，公開実装および本研究の比較

実装	主な切替契機	選択方針と適用範囲
標準実装 [7]	自発的な実行権の放棄，待機，終了	協調型ラウンドロビン
Lupu 投稿版（2019）[9], [10]	周期タイマとタイムスライス	優先度別キュー．タイマによる切替えはスライス切れと同優先度の候補の存在が条件
bsdkrun（2026）[11]	周期タイマなどの割込みの復帰経路	既存スケジューラで切替え．カーネルイメージ外のアプリケーションコードが対象
本研究	起床期限に合わせたタイマ割込み	固定優先度．割込み時には同優先度のスレッドへ切り替えない

Lupu は 2019 年に Unikraft のスレッド管理に接続するプリエンプティブ・スケジューラを投稿している [9]．ただし，投稿版のタイマ処理はタイムスライス切れと同優先度の実行可能スレッドの存在を条件とし，高優先度スレッドの起床時に必ず切り替える方式ではない [10]．一方，bsdkrun はアプリケーション実行中に割込み復帰経路から

実行権を放棄させ，既存スケジューラで切り替える [11]．また，LKL 移植は Linux 機能を組み込み，Unikraft とは独立した Little Kernel 由来のスケジューラを用いる [12]．本研究は，Unikraft 自身のスレッド管理を拡張し，起床期限に合わせてタイマを設定し，起床処理後に固定優先度に基づいて切替先を選択する点に特徴がある．

2.3 プリエンプション導入に伴うキャッシュ干渉

プリエンプションの負荷には，割込み処理やコンテキスト保存のコストに加え，キャッシュ干渉が含まれる．Reineke らは，中断した処理のキャッシュラインが置換され，再開後に追加のキャッシュミスが生じる問題を扱っている [8]．したがって，Unikraft でもキャッシュ容量に対する使用領域の余裕を考慮する必要がある．

2.4 本研究の位置づけと貢献

本研究は，2.2 節で述べた機構を実装し，リアルタイム性とキャッシュへの副作用を定量的に比較する研究である．本研究の貢献は以下の 3 点である．

機構の実装 Unikraft への，起床期限に合わせたタイマ設定と割込み時の固定優先度選択によるプリエンプション機構の実装

起床遅延の比較評価 導入前後の Unikraft の比較による起床遅延の改善効果の評価と，既存の OS との比較

キャッシュミスの比較評価 導入前後の Unikraft の比較によるキャッシュミスへの影響の評価と，既存の OS との比較によるワーキングセット条件ごとの特徴の明確化

3. 実装と実験設定

3.1 Unikraft へのプリエンプティブ・スケジューリング実装

Unikraft 0.18.0 を基に，KVM/x86_64 向けの固定優先度スケジューラを実装した．実行待ちキューを 16 段階の優先度別に管理し，待機中のスレッドの最短起床期限にタイマを設定する．期限到来時には起床対象を実行可能にし，現在より高優先度のスレッドがあれば割込み処理内で切り替える．同優先度間では割込みによる切替えを行わない．切替え時は，汎用レジスタや拡張 CPU 状態などを中断スレッドの補助スタックに保存し，再選択時に中断位置へ復帰する．タイマは対応環境で TSC deadline を用い，非対応時は PIT を用いる．プリエンプション禁止に用いる API は協調型の実装から変更せず，割込み要求（IRQ）から強制的にスレッドを切り替える経路を追加した．割込み禁止区間では，入口で元の IRQ 状態を保存して割込みを禁止し，出口で保存した状態を復元する．したがって，低優先度スレッドが自発的に実行権を放棄しなくても，高優先度スレッドを実行できる．

以下，公式リポジトリに公開されている協調型スケジューリング方式の Unikraft を NoRT-Unikraft，本研究で実装したプリエンプティブ・スケジューリング方式の Unikraft を RT-Unikraft と表記する．

3.2 評価対象 OS の選定

本研究では，NoRT-Unikraft，RT-Unikraft，OSv および PREEMPT_RT を，いずれも QEMU/KVM 上のゲスト OS として実行する．NoRT-Unikraft と RT-Unikraft との両者の比較でプリエンプションの影響を調べる．

OSv は，Unikernel の一種でありアプリケーションと OS 機能が単一アドレス空間で動作する VM 向け OS である．Linux 向けの C/POSIX アプリケーションの実行を想定し，プリエンプティブスケジューラの実装も公式 GitHub に明記されている [13]．このため，アプリケーションモデルの近さ，測定プログラムの移植性，切替方式の明確さを重視して比較対象に選定した．また，PREEMPT_RT は割込みのスレッド化などにより高優先度処理の遅延を抑える [5]．PREEMPT_RT をゲスト OS とする KVM 環境の遅延評価や，QEMU/KVM 上での OSv との比較が報告されている [14], [15]．このような VM 上での評価実績を踏まえ，リアルタイム Linux の比較対象として選定した．

以下，特に断りのない限り，PREEMPT_RT は，PREEMPT_RT を適用し，QEMU/KVM 上のゲスト OS として実行する Linux を指す．

3.3 実験環境

測定ホストと仮想化の設定を**表 2** に示す．各ゲストは同一ホスト上の QEMU/KVM で実行し，ハードウェア仮想化を利用する [16]．

表 2 測定ホストと仮想化の設定

項目	設定
CPU	AMD Ryzen 7 8700G（8 コア）
L1 キャッシュ	L1D・L1I とも 32 KiB/コア
L2・L3	L2：1 MiB/コア，L3：共有 16 MiB
ホスト OS	Ubuntu 24.04 系
Linux カーネル	6.8.1-1048-realtime（ホスト・ゲスト）
仮想化	QEMU 8.2.2 / KVM
vCPU 数	各ゲスト 1
QEMU 優先度	SCHED_FIFO 99（本体・vCPU）
ホスト起動設定	`nosmt`，`idle=poll`，`kthread_cpus=0-5`，`nohz_full=6-7`，`irqaffinity=0-5`，`rcu_nocbs=6-7`，`isolcpus=managed_irq,domain,6-7`

測定処理の移動と管理処理との干渉を抑えるため，CPU 配置を固定する．全ゲストで**図 1** に示すように，CPU 0〜

5 には管理処理（housekeeping）を配置し，QEMU 本体を CPU 6，vCPU スレッドを CPU 7 に配置する．さらに，CPU 6〜7 の分離と IRQ の配置を設定し，周波数は 2.0 GHz を上限として制御する．リアルタイム性評価では，ホストの RT スロットリングと NMI watchdog を無効化した．

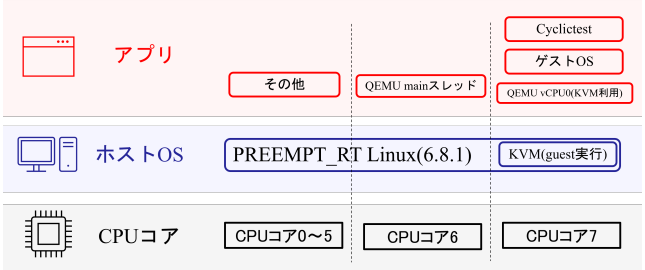

図 1　実行環境と CPU 配置

4. 評価

4.1 リアルタイム性の評価方針

4.1.1 評価目的と測定対象

低優先度スレッドが競合する状況で，プリエンプティブ・スケジューリングが高優先度周期スレッドの起床遅延をどの程度低減するかを評価する．また背景負荷は一定とは限らないため，軽負荷から重負荷まで比較する．

4.1.2 実験条件

高優先度側の実行周期は cyclictest のデフォルト値である 1 ms とする．低優先度側の周期 T と実行時間 E によって，CPU の連続占有時間が定まる．本評価では T を 10 ms に固定し，E を 2，5，8 ms として，CPU 利用率の設定値 E/T が 20，50，80%となる 3 条件を比較する．周期 10 ms は，高優先度側の周期より十分長く，低優先度側の 1 周期中に複数の起床要求が到来するように選んだ．プリエンプションされる状況を作るため，高優先度側の起床は低優先度側の開始から 0.5 ms 後に設定する．このとき，高優先度側の予定起床時刻における低優先度側の残り実行時間は，設定上それぞれ最大 1.5，4.5，7.5 ms となるが，他の処理の影響により実際の遅延はこれらを上回り得る．

OS 別の各スレッドの優先度の設定を**表 3** に示す．Unikraft は内部優先度を 0〜15 の範囲で設定し，値が小さいほうが優先度が高いことに注意する．15 にするとアイドルスレッドと同じとなるため，RT-Unikraft の低優先度側は 10 に設定した．

表 3　OS 別の各スレッド優先度

OS	高優先度側	低優先度側
RT-Unikraft	内部優先度 0	内部優先度 10
OSv, PREEMPT_RT	SCHED_FIFO 99	SCHED_FIFO 1

4.1.3 測定方法と評価指標

高優先度側には cyclictest を用い，予定起床時刻と実行再開時刻の差を測定する [17]．低優先度側では入出力処理や大規模なメモリアクセスによる実行時間の変動を抑えるため，整数演算ループを実行する．各試行で 100 万サンプルを 3 回取得し結果を合算する．また，まれな大きい遅延を捉えるため 99.99 パーセンタイル値を主指標とする．

4.2 キャッシュへの副作用の評価方針

4.2.1 評価目的と測定対象

プリエンプティブ・スケジューリングによる高優先度スレッドのリアルタイム性改善に伴い，低優先度スレッドのキャッシュ性能に生じる副作用を評価する．本実験では高優先度スレッドと低優先度スレッドを同時に動かし，低優先度側の L2 データキャッシュを評価した．

低優先度スレッドに着目した理由は，プリエンプションに伴うキャッシュ性能の劣化を直接評価できるからである．また，L2 データキャッシュを対象としたのは同一 CPU で中断前後に実行されるスレッド間のキャッシュ干渉を CPU に比較的近い階層で観察するためである．LLC を対象とすると，L1 および L2 でのヒット・ミスの影響も考慮する必要があり，干渉の要因の切り分けが複雑になる．L1 は容量が小さく，評価でのノイズの影響が相対的に大きくなることが懸念される．なお，データキャッシュに着目したのは，本評価で用いるスレッドがデータ領域を繰り返し参照する処理であり，プリエンプションに伴うデータの追い出しと再読み込みによる影響を評価するためである．

4.2.2 実験条件

先読みの影響を抑えてキャッシュミスの影響を観測しやすくするため，Luo らの研究に基づきランダムなポインタ追跡を移植した [18]．具体的には，まず 64 B のノードからなる配列を確保し，初期化時に全ノードをランダムな順序で結ぶ単一の巡回路を構成した．次に，リアルタイム性評価で用いた各 OS の評価プログラムにおいて低優先度スレッドの整数演算ループをこの処理に置き換えるとともに，高優先度スレッドにも同じ処理を組み込んだ．そして，両スレッドには独立した配列を割り当てそれぞれの配列を繰り返し追跡させる．これにより，高優先度スレッドの実行が低優先度スレッドの L2 データキャッシュのミスに与える影響を評価する．

また，高優先度側の実行周期はリアルタイム性評価と統一し 1 ms とする．さらに，ワーキングセットが L2 の全ラインを少なくとも 1 回アクセスするように，実行時間を 250 μs に設定する．低優先度側はプリエンプションの発生機会を確保し，中間的な負荷条件で評価するため実行周期 10 ms・実行時間 5 ms とする．ワーキングセットは**表 4** の 3 条件とし，合計サイズが L2 容量 1 MiB 未満，同容量，超過となる場合を比較する．

ⓒ2026 Information Processing Society of Japan

表 4 キャッシュ評価のワーキングセット条件（低優先度側は 256 KiB で固定，L2 容量は 1024 KiB）

条件	高優先度側［KiB］	合計［KiB］
small（容量未満）	128	384
medium（容量相当）	768	1024
large（容量超過）	1280	1536

4.2.3 測定方法と評価指標

L2 データキャッシュの計測には，KVM がゲストに提供する仮想 PMU（vPMU）を用いる．Unikraft と OSv ではゲスト側から vPMU を利用し，PREEMPT_RT ではゲスト内の `perf_event_open()` を介して vPMU を利用する．L2 データミス数には `l2_cache_req_stat.ls_rd_blk_c`，L2 データアクセス数には `l2_cache_req_stat.dc_access_in_l2` のイベントを用いる．

低優先度スレッドを Victim と呼び，その実行区間の計数値を各 OS で次のように取得する．Unikraft では，性能監視カウンタ（PMC）を動作させ続け，Victim のジョブ開始・再開時と中断・終了時にスナップショットを取得する．各実行区間の開始時と終了時の差分を合計し，Victim の計数値とする．OSv では，Victim から他のスレッドへ切り替わる直前にカウンタを停止して計数値を回収し，Victim へ戻る直前にカウンタをリセットして再開する．PREEMPT_RT では，Victim 自身が `pid=0`，`cpu=-1` を指定して `perf_event_open()` を呼び出し，イベントを自身のスレッドに結び付ける．各ジョブの直前にカウンタをリセットして有効化し，直後に無効化して計数値を読み出す．

プリエンプション回数は，各 OS において Victim から他のスレッドへの切替え回数から，周期待ちに伴う自発的な切替えを除いた回数とする．

指標には，Navarro-Torres らのキャッシュ階層評価 [19] に基づき，MPKI を用いる．MPKI（Misses Per Kilo Instructions）は，実行命令 1000 件当たりのキャッシュミス数を表す指標である．実行命令数で正規化することで，命令数当たりのミス発生頻度を比較でき，値が小さいほどその頻度が低く良い結果であることを示す．

また，ワーキングセット条件別に 1 プリエンプション当たりの追加 L2 データキャッシュミス数を評価する．この値は，高優先度側のキャッシュ負荷を無効にした基準条件に対する低優先度側の L2 ミス数の増分を，プリエンプション回数で除した平均値である．MPKI のみでは元から発生するミスとプリエンプションに伴う増分を区別できないため，本指標を併用する．これにより，先行研究で指摘されるキャッシュ干渉による再開後の追加ミス [8] を，ワーキングセット条件間で比較できる．

4.3 リアルタイム性の評価結果

4.1 節の条件で測定した 99.99 パーセンタイル起床遅延を**表 5** に示す．NoRT-Unikraft では，高優先度スレッドに対する設定上の最大妨害時間が 1.5，4.5，7.5 ms と増加するにつれ，遅延は 1521，4531，7545 μs に増大した．一方，RT-Unikraft では 21，21，22 μs と遅延が抑えられた．

表 5 妨害時間別の 99.99 パーセンタイル起床遅延，単位：μs

OS	1.5 ms	4.5 ms	7.5 ms
NoRT-Unikraft	1,521	4,531	7,545
RT-Unikraft	21	21	22
PREEMPT_RT	23	23	23
OSv	20	20	20

低優先度処理の実行時間 8 ms(設定上の最大妨害時間 7.5 ms) における起床遅延の相補累積分布関数（CCDF）を**図 2** に示す．縦軸は横軸の値以上の遅延が生じた割合を表し，両軸は対数目盛である．NoRT-Unikraft では 1 ms を超えても起床遅延は収束しないが，RT-Unikraft，OSv，PREEMPT_RT では，観測された最大遅延はいずれも 100 μs 未満であった．

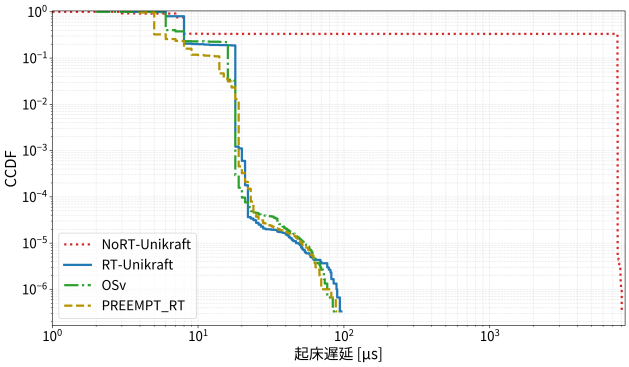

図 2 低優先度処理の実行時間 8 ms（最大妨害時間 7.5 ms）における起床遅延の CCDF

4.4 キャッシュへの副作用の評価結果

4.2 節の周期 1 ms の条件で測定した低優先度スレッドの L2 データキャッシュの MPKI を**図 3** に示す．各構成・各ワーキングセット条件で 100 万回の試行による測定を 3 回実施し，得られた MPKI の平均値を示す．すべての構成で，ワーキングセットが small，medium，large と大きくなるにつれて MPKI が増加した．RT-Unikraft の MPKI は NoRT-Unikraft に対し，small で約 1.02 倍，medium で約 7.26 倍，large で約 7.63 倍となり，large で増加倍率が最大となった．medium と large では，RT-Unikraft・OSv・PREEMPT_RT の MPKI に大きな差は見られなかった．

ⓒ2026 Information Processing Society of Japan

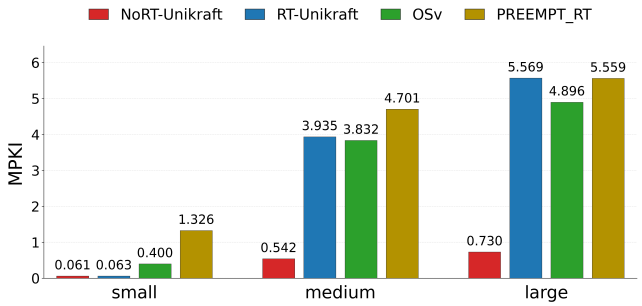

図 3 ワーキングセット条件別の低優先度スレッドの L2 データキャッシュの MPKI

同じワーキングセット条件における，1 プリエンプション当たりの追加 L2 データキャッシュミス数（以下，追加 L2 ミス数と呼ぶ）を**表 6** に示す．NoRT-Unikraft は原理的にプリエンプションが起きず，プリエンプション回数が 0 になるため算出できない．small では，RT-Unikraft は 33.6 回であり，OSv の 157.3 回の約 5 分の 1，PREEMPT_RT の 397.3 回の約 12 分の 1 とキャッシュミスの発生を大幅に抑えられた．一方，medium では RT-Unikraft が最小であったが，OSv との差は約 9.9%，PREEMPT_RT との差は約 12.6%にとどまった．さらに，large では差が減り，最大値と最小値の差は約 1.24%となった．したがって，ワーキングセットが L2 容量未満のときは RT-Unikraft の追加 L2 ミス数が比較的少ないと言える．

表 6 ワーキングセット条件別の 1 プリエンプション当たりの追加 L2 データキャッシュミス数，単位：回／プリエンプション

OS	small	medium	large
NoRT-Unikraft	算出不可	算出不可	算出不可
RT-Unikraft	33.6	1,779.0	2,573.3
OSv	157.3	1,974.9	2,541.7
PREEMPT_RT	397.3	2,034.7	2,567.9

5. 考察

5.1 プリエンプションによる起床遅延改善

起床遅延の改善には，低優先度処理の終了を待たずに高優先度処理へ切り替えるプリエンプションが寄与した可能性がある．4.3 節の表 5 に示したように，低優先度処理の実行時間を 2，5，8 ms に増加させると，NoRT-Unikraft の 99.99 パーセンタイル起床遅延は 1521，4531，7545 μs と次第に増加したが，RT-Unikraft は 21，21，22 μs と遅延が抑えられた．このように，プリエンプションの導入により低優先度処理の実行時間が増加しても RT-Unikraft の遅延はほぼ一定であり，低優先度処理の実行時間の増加に伴う顕著な高優先度処理の遅延増加は見られなかった．

また，PREEMPT_RT の 23 μs や OSv の 20 μs と比較しても，RT-Unikraft の遅延は 22 μs 程度であることから，起床遅延は QEMU/KVM 内の処理が主因である可能性がある．

5.2 プリエンプションによるキャッシュへの副作用

キャッシュへの副作用，つまり MPKI 増加の一因は高優先度処理による低優先度側のキャッシュラインの置換と考えられる．中断中に置換されたデータへのアクセスは，実行再開後に低優先度側の L2 キャッシュミスとなることがある [8]．4.4 節で述べたように，RT-Unikraft の MPKI は NoRT-Unikraft 比で，small，medium，large の順に約 1.02，7.26，7.63 倍であった．本研究の medium・large のようにワーキングセットの合計が L2 容量以上となる条件では，低優先度側のデータが置換されやすくなり，再開後の L2 キャッシュミスが増加したと考えられる．

また，低優先度スレッド側にも時間制約がある場合，再開後の追加キャッシュミスは低優先度処理の実行時間を増加させる可能性がある．この影響を検討するため，キャッシュ関連プリエンプション遅延（Cache-Related Preemption Delay，CRPD）に着目する．Reineke らは，追加ミス数の上界とブロック再読み込み時間を用いて CRPD の上界を評価している [8]．この考え方を参考に，本研究では表 6 の 1 プリエンプション当たりの平均追加 L2 ミス数から，L2 データキャッシュに起因する CRPD を推定する．L2 ヒット時のレイテンシを C_{L2}，L2 でミスし L3 でヒットした場合のレイテンシを C_{L3}（いずれも単位はサイクル），CPU クロック周波数を f_{CPU} とする．本来 L2 でヒットするアクセスが L3 ヒットへ変化すると仮定すると，追加 L2 ミス 1 回当たりの追加遅延 D_{L3} は，

$$D_{\mathrm{L3}} \approx \frac{C_{L3} - C_{L2}}{f_{CPU}} \tag{1}$$

と近似できる．本実験では CPU クロック周波数の上限を 2.0 GHz としており，ここでは 2.0 GHz で動作すると仮定する．式（1）に $f_{CPU} = 2.0 \times 10^9$ Hz と，本実験で使用した AMD の最適化ガイド [20] が示す代表値 $C_{L2} = 14$，$C_{L3} = 50$ を代入すると追加 L2 ミス 1 回当たりの追加遅延は約 18 ns となる．表 6 の平均追加 L2 ミス数に 18 ns を乗じた，低優先度側の 1 プリエンプションでの推定 CRPD を**表 7** に示す．

表 7 低優先度側の 1 プリエンプションでの推定 CRPD，単位：μs

条件	RT-Unikraft	OSv	PREEMPT_RT
small	0.60	2.83	7.15
medium	32.02	35.55	36.62
large	46.32	45.75	46.22

例えば，低優先度スレッドが 3 回プリエンプトされると仮定すると，RT-Unikraft の推定 CRPD の合計は small で約 1.8 μs，medium で約 96 μs，large で約 139 μs となる．高優先度処理の実行時間などを差し引いた期限までの余裕

ⓒ2026 Information Processing Society of Japan

が 100 μs であれば，このモデルでは large の推定追加遅延がその余裕を上回る．一方，余裕が 1 ms あれば，このモデルによる追加遅延は収まる規模である．したがって，副作用を無視できるかは，追加ミス数に加えて実際のプリエンプション回数と期限までの余裕によって決まる．

5.3 Unikraft と他の OS との差異

4.4 節で述べたように，small の追加 L2 ミス数は，RT-Unikraft で 33.6，OSv で 157.3，PREEMPT_RT で 397.3 であった．このようなワーキングセットが L2 容量未満のとき，RT-Unikraft の追加 L2 ミス数が比較的少ない結果となった理由として，システムコールの排除と OS のフットプリントの小ささが考えられる．

Unikernel の最適化がキャッシュ性能の改善につながる事例として，Raza らの Unikernel Linux（UKL）がある [21]．同研究は，システムコールの入口・出口処理を省略するなどの最適化を施した構成において，Linux より L1 データキャッシュへのアクセス数，LLC へのアクセス数および LLC ミス数が減少したことを報告している [21]．

また，各 OS の静的構成を比較するため，ELF のロード可能セグメントのメモリサイズの合計を静的ロード量として測定した．静的ロード量と測定範囲を**表 8** に示す．RT-Unikraft は cyclictest を含めても，OSv の OS 本体・標準ライブラリ等および PREEMPT_RT のゲストカーネル部分より静的ロード量が小さかった．

表 8 各 OS の静的ロード量と測定範囲（○：測定範囲に含む，×：含まない）

対象	OS 本体	標準ライブラリ等	cyclictest	静的ロード量［MiB］
RT-Unikraft	○	○	○	1.409
OSv	○	○	×	7.602
PREEMPT_RT	○	×	×	55.623

本研究の比較対象の OS とは違い，Unikraft は OS 機能をライブラリ単位で選び，フットプリントを削減できるメリットがある [4]．そのため，small では L2 容量に余裕があり，OS 内部処理の参照量が少なければデータを保持しやすいと考えられる．一方，medium 以降は高優先度側のキャッシュ使用量が増えるため，キャッシュ容量に対して OS が占めるサイズよりも高優先度スレッドが占めるサイズの方が支配的になり，OS のフットプリント削減の効果が比較的減少した可能性がある．静的ロード量の差も，Unikraft の小さな OS 構成がキャッシュ干渉の抑制に寄与した可能性と整合する．なお，各 OS ではワークロード領域のページサイズを考慮しておらず，TLB ミスやページウォークの発生頻度に影響する可能性がある．本評価ではこれらの影響を分離していないため，L2 キャッシュミス数の差にはページサイズの違いも寄与している可能性がある．

5.4 リアルタイム性とキャッシュ性能のトレードオフ

4.3 節の表 5 に示したように，低優先度処理の実行時間 8 ms の条件で，Unikraft へのプリエンプション導入により遅延が 7545 μs から 22 μs に低減した．一方，4.4 節の図 3 に示したように，large では，MPKI が 0.7302 から 5.5689 へ増加した．このようなトレードオフは，高優先度処理の割込み実行が実行待ちを抑える一方，低優先度側のキャッシュラインを置換し得るために生じると考えられる [8], [18]．

6. まとめと今後の課題

6.1 まとめ

組込みシステムではリアルタイム性が求められ，またコスト削減と処理ごとの要求の達成に向け，仮想化によって同一ハードウェア上で複数の OS を利用する場合がある．Unikraft は Linux より OS 機能を細かく選択でき，OS のフットプリントを削減できるが，標準スケジューラは協調型でありリアルタイム性が低い．そこで，Unikraft にプリエンプティブ・スケジューリングを実装してリアルタイム性を向上させる．一方，高優先度処理を優先して実行することで，低優先度処理の再開後のキャッシュミスを増加させる可能性があるため両方の評価が必要となる．

本研究では，Unikraft に起床期限と固定優先度に基づくプリエンプティブ・スケジューリングを実装し，リアルタイム性とキャッシュへの副作用を評価した．QEMU/KVM 環境において，導入前後の Unikraft を比較するとともに，OSv および PREEMPT_RT との比較を行った．

リアルタイム性評価では，99.99 パーセンタイル起床遅延は NoRT-Unikraft の 7545 μs から RT-Unikraft の 22 μs に低減した．この値は OSv の 20 μs および PREEMPT_RT の 23 μs と同程度であった．

キャッシュ評価では，L2 データキャッシュの MPKI を測定した．その結果，MPKI の増加倍率が最大となった条件では NoRT-Unikraft の 0.7302 に対して RT-Unikraft は 5.5689 であり，約 7.63 倍に増加した．この増加には，プリエンプションに伴うスレッド間のキャッシュ干渉が影響したと考えられる．また，合計ワーキングセットが L2 容量を下回る条件では，RT-Unikraft の MPKI は 0.0626 であり，OSv の 0.3997 および PREEMPT_RT の 1.3258 を下回った．追加 L2 ミス数で比較しても，RT-Unikraft は 33.6 回であり，OSv の約 5 分の 1 であり PREEMPT_RT の約 12 分の 1 であった．この原因として，Unikraft の軽量な OS 構成に伴うキャッシュ使用量の違いが寄与した可能性がある．

本研究の成果は，仮想化を利用する組込みシステムにおいて，リアルタイム処理を担うゲスト OS の選定に活用できると考えられる．QEMU/KVM 上で PREEMPT_RT や OSv と同程度のリアルタイム性と L2 キャッシュ性能が得

ⓒ2026 Information Processing Society of Japan

られたことは Unikraft を採用候補とする根拠となり，小さいワーキングセットでのキャッシュミスの少なさも補助的な判断材料となる．

6.2 今後の課題

第 1 に，高優先度スレッドと低優先度スレッドの位相関係を変化させ，プリエンプションの発生タイミングが起床遅延に与える影響を評価する．これにより，固定した位相差で得た遅延の改善が，異なる発生タイミングでも維持されるかを確認する．

第 2 に，プリエンプティブ・スケジューリングが TLB に与える影響を評価する．この際，本評価では OS 間でページサイズを考慮していないため，ページサイズも実験条件に入れる必要がある．これを踏まえてプリエンプションに伴う TLB ミスの変化を測定し，キャッシュ以外のメモリ階層への副作用を明らかにする．

参考文献

[1] Chen, K.-H., Günzel, M., Jablkowski, B., Buschhoff, M. and Chen, J.-J.: Unikernel-Based Real-Time Virtualization Under Deferrable Servers: Analysis and Realization, *34th Euromicro Conference on Real-Time Systems (ECRTS 2022)*, Leibniz International Proceedings in Informatics (LIPIcs), Vol. 231, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 6:1–6:22 (online), DOI: 10.4230/LIPIcs.ECRTS.2022.6 (2022).

[2] Bandur, V., Selim, G., Pantelic, V. and Lawford, M.: Making the Case for Centralized Automotive E/E Architectures, *IEEE Transactions on Vehicular Technology*, Vol. 70, No. 2, pp. 1230–1245 (online), DOI: 10.1109/TVT.2021.3054934 (2021).

[3] Cinque, M., Cotroneo, D., De Simone, L. and Rosiello, S.: Virtualizing mixed-criticality systems: A survey on industrial trends and issues, *Future Generation Computer Systems*, Vol. 129, pp. 315–330 (online), DOI: 10.1016/j.future.2021.12.002 (2022).

[4] Kuenzer, S., Bădoiu, V.-A., Lefeuvre, H., Santhanam, S., Jung, A., Gain, G., Soldani, C., Lupu, C., Teodorescu, Ş., Răducanu, C., Banu, C., Mathy, L., Deaconescu, R., Raiciu, C. and Huici, F.: Unikraft: Fast, Specialized Unikernels the Easy Way, *Proceedings of the Sixteenth European Conference on Computer Systems*, ACM, pp. 376–394 (online), DOI: 10.1145/3447786.3456248 (2021).

[5] Siewior, S. A.: Theory of operation, The Linux Kernel documentation (online), available from ⟨https://www.kernel.org/doc/html/latest/core-api/real-time/theory.html⟩ (accessed 2026-10-15).

[6] Unikraft Project: Compatibility, Unikraft documentation (online), available from ⟨https://unikraft.org/docs/concepts/compatibility⟩ (accessed 2026-10-26).

[7] Unikraft Project: Unikraft: Cooperative Round-Robin Scheduler, Unikraft (online), available from ⟨https://github.com/unikraft/unikraft/blob/staging/lib/ukschedcoop/schedcoop.c⟩ (accessed 2026-10-15).

[8] Reineke, J., Altmeyer, S., Grund, D., Hahn, S. and Maiza, C.: Selfish-LRU: Preemption-Aware Caching for Predictability and Performance, *2014 IEEE 20th Real-Time and Embedded Technology and Applications Symposium*, IEEE, pp. 135–144 (online), DOI: 10.1109/RTAS.2014.6925997 (2014).

[9] Lupu, C.: [UNIKRAFT PATCH 00/23] Preemptive Scheduling, Xen Project Minios-devel Mailing List (online), available from ⟨https://lists.xen.org/archives/html/minios-devel/2019-07/msg00065.html⟩ (accessed 2026-10-15).

[10] Lupu, C.: lib/ukschedpreempt: Handle timer interrupts, Xen Project Minios-devel Mailing List (online), available from ⟨https://lists.xen.org/archives/html/minios-devel/2019-07/msg00083.html⟩ (accessed 2026-10-15).

[11] Sandratraina, T.: unikraft: preempt application code, and get MySQL serving SQL, GitHub (online), available from ⟨https://github.com/tsirysndr/bsdkrun/commit/80b48c73d814⟩ (accessed 2026-10-15).

[12] retrage: Linux kernel を Unikernel のライブラリ化する，retrage.github.io（オンライン），入手先 ⟨https://retrage.github.io/2019/05/30/lkl-on-unikraft.html/⟩（参照 2026-10-15）．

[13] OSv Project: Thread Scheduler, GitHub Wiki (online), available from ⟨https://github.com/cloudius-systems/osv/wiki/Thread-Scheduler⟩ (accessed 2026-10-21).

[14] Mathew, D. and Jose, B. A.: Real-Time Performance Analysis and Tuning of Embedded System Virtualization Architecture Based on KVM, *International Journal of Embedded and Real-Time Communication Systems*, Vol. 13, No. 1, pp. 1–20 (online), DOI: 10.4018/IJERTCS.302113 (2022).

[15] Abeni, L.: Virtualized real-time workloads in containers and virtual machines, *Journal of Systems Architecture*, Vol. 154, p. 103238 (online), DOI: 10.1016/j.sysarc.2024.103238 (2024).

[16] QEMU Project: Introduction, QEMU documentation (online), available from ⟨https://www.qemu.org/docs/master/system/introduction.html⟩ (accessed 2026-10-15).

[17] de Oliveira, D. B., Casini, D., de Oliveira, R. S. and Cucinotta, T.: Demystifying the Real-Time Linux Scheduling Latency, *32nd Euromicro Conference on Real-Time Systems (ECRTS 2020)*, Leibniz International Proceedings in Informatics (LIPIcs), Vol. 165, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 9:1–9:23 (online), DOI: 10.4230/LIPIcs.ECRTS.2020.9 (2020).

[18] Luo, Z., Son, S., Bali, D., Amaro, E., Ousterhout, A., Ratnasamy, S. and Shenker, S.: Efficient Microsecond-scale Blind Scheduling with Tiny Quanta, *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, ACM, pp. 305–319 (online), DOI: 10.1145/3620665.3640381 (2024).

[19] Navarro-Torres, A., Alastruey-Benedé, J., Ibáñez-Marín, P. and Viñals-Yúfera, V.: Memory hierarchy characterization of SPEC CPU2006 and SPEC CPU2017 on the Intel Xeon Skylake-SP, *PLOS ONE*, Vol. 14, No. 8, p. e0220135 (online), DOI: 10.1371/journal.pone.0220135 (2019).

[20] Advanced Micro Devices, Inc.: Software Optimization Guide for the AMD Zen4 Microarchitecture, Technical Report 57647, Advanced Micro Devices, Inc. (2023). Revision 1.01.

[21] Raza, A., Unger, T., Boyd, M., Munson, E. B., Sohal, P., Drepper, U., Jones, R., de Oliveira, D. B., Woodman, L., Mancuso, R., Appavoo, J. and Krieger, O.: Unikernel Linux (UKL), *Proceedings of the Eighteenth European Conference on Computer Systems*, ACM, pp. 590–605 (online), DOI: 10.1145/3552326.3587458 (2023).

ⓒ2026 Information Processing Society of Japan