Vol.2017-DPS-169 No.4
2017/1/19

マルチコア環境用スケジューラの優先度逆転とオーバヘッドのトレードオフ評価

鴨生 悠冬[1]　佐藤 将也[1]　山内 利宏[1]　谷口 秀夫[1]

概要：マルチコアプロセッサの普及により，様々なサービスが並列走行し，優先度逆転が生じやすくなっている．特に，マイクロカーネル構造 OS では，OS 機能の大部分をプロセス化しており，この機能の利用時に優先度逆転が生じる．また，コア毎に独立したスケジューラは，マイクロカーネル構造 OS の性能を左右するプロセス切替におけるオーバヘッドが小さい．しかし，このスケジューラにおいて優先度逆転を抑制する場合，コア間通信を必要とするため，コア間通信オーバヘッドが生じる．したがって，優先度逆転時間とコア間通信オーバヘッドは，トレードオフの関係にある．本稿では，この関係を考慮した優先度逆転とコア間通信回数の両方を抑制する優先度継承方式について述べる．また，優先度継承方式を実現し，実測評価した結果について述べる．

1. はじめに

マルチコアプロセッサの普及により，計算機の利用形態が多様化している．これにより，サービスやシステムはますます複雑化し，優先度逆転 [1] が生じやすくなっている．オペレーティングシステム（以降，OS）は，サービスの要望に即したプロセスのスケジュールが求められるため，優先度逆転を抑制する必要がある．ここで，優先度逆転とは，低優先度のプロセスがプロセッサを解放するまで高優先度のプロセスや高優先度のプロセスから処理依頼を受けたプロセスが処理を実行できず，サービスの優先度が逆転する状態である．

計算機の多様な利用を支える高い適応性，および高い堅牢性を有する OS が必要になっている．これを実現する OS のプログラム構造として，マイクロカーネル構造 [2][3][4][5] がある．マイクロカーネル構造は，例外処理や割込み処理といった最小限の OS 処理をカーネルとして実現し，ファイル管理やディスクドライバといった大部分の OS 機能をプロセス（以降，OS サーバ）として実現するプログラム構造である．大部分の OS 機能をプロセスとして実現しているため，利用環境に応じた機能の追加や削除をプロセスの生成と削除として実現できる．また，OS サーバの暴走をプロセスの暴走として対処できるため，システム全体への影響を抑制できる．

[1] 岡山大学 大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

マイクロカーネル構造 OS のプロセスは，OS サーバに処理を依頼することで，OS サーバの実現している OS 機能を利用する．すなわち，OS サーバとプロセス間通信（以降，OS サーバ間通信）を行う．OS サーバは，依頼を受取り，処理を実行する．したがって，依頼元プロセスと OS サーバ間での優先度の継承がない場合，OS 機能の処理は，依頼元プロセスのプロセス優先度に関わらず，OS サーバのプロセス優先度で実行される．このため，依頼元プロセスと OS サーバの優先度が同じでない場合，優先度逆転が生じる可能性がある．

また，マルチコアプロセッサ上で走行するマイクロカーネル構造 OS においては，コア毎に独立したスケジューラを実現することで，スケジュールキューの操作における排他制御を削減することができる．これは，コア毎に独立したスケジューラが他コア上のプロセスのプロセス情報を直接変更せず，コア間通信で変更を依頼するため，スケジュールキューの操作が並列実行されないためである．スケジュールキューの中でもとりわけ READY キューの操作におけるオーバヘッドは，マイクロカーネル構造 OS の性能を左右するコンテキストスイッチの処理時間に影響する．このため，排他制御の削減によるオーバヘッド抑制は非常に重要である．ここで，コア間通信とは，異なるコア上のカーネルに IPI（Inter-processor interrupt）を発行し，他コア上のカーネルに処理を依頼する処理である．

優先度逆転を抑制することはマルチコア環境においても重要である [6][7]．しかし，コア毎に独立したスケジューラにおいて，コアを跨ぐ OS サーバ間通信によって優先度

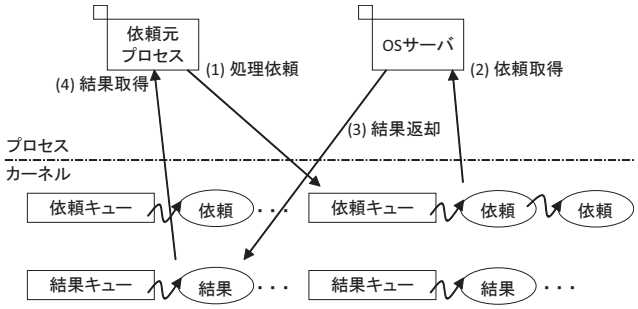

図 1 OS サーバ間通信

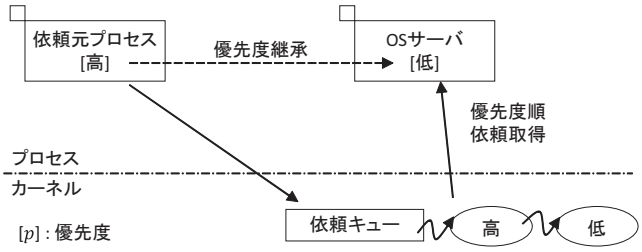

図 2 基本方式

逆転を抑制する場合，他コア上のプロセスのプロセス情報を変更するため，コア間通信が必要になる．コア間通信のオーバヘッドは大きいため，オーバヘッド削減には，コア間通信回数の削減が重要である．したがって，コア毎に独立したスケジューラを有するマイクロカーネル構造 OS には，優先度逆転時間の抑制とコア間通信回数の抑制が要求される．しかし，優先度逆転時間とコア間通信回数は，トレードオフの関係にある．

本稿では，このトレードオフの関係を考慮した優先度逆転時間とコア間通信回数の両方を抑制する優先度継承方式について述べる．提案方式は，既存の優先度逆転抑制法と異なり，コア間通信回数を削減することでオーバヘッドを抑制している．また，提案方式の基本評価として，OS サーバの応答時間の評価を行う．さらに，事例評価として，異種 OS サーバ共存時の応答時間の評価とファイル複写処理共存時のファイル参照性能の評価を行う．上記の評価から，優先度逆転の抑制効果とコア間通信回数の削減効果を明らかにする．

2. 優先度継承方式

2.1 基本方式

OS サーバ間通信の様子を**図 1** に示し，基本的な処理流れを以下に説明する．なお，各プロセスは依頼キューと結果キューを持つ．

(1) 依頼元プロセスは，依頼情報を OS サーバの依頼キューに登録し，処理を依頼する．
(2) OS サーバは，依頼キューより依頼情報を取得し，依頼内容を実行する．
(3) OS サーバは，処理の結果情報を依頼元プロセスの結果キューに登録し，結果を返却する．
(4) 依頼元プロセスは，結果キューより結果情報を取得する．

優先度継承方式の基本方式を**図 2** に示す．基本方式は，以下に説明する優先度順依頼取得と優先度継承により優先度逆転を抑制する．

（優先度順依頼取得）OS サーバは，依頼元プロセスによって登録された依頼を登録順ではなく，依頼元プロセスの優先度順に取得する．この考え方は，EMERALDS[8] で提案されている．

（優先度継承）依頼元プロセスの優先度を OS サーバに継承する． OS サーバは，依頼元プロセスの優先度で依頼された処理（以降，依頼処理）を実行する．

優先度順依頼取得により，依頼を登録順に取得する場合に生じる優先度逆転の問題を回避できる．また，優先度継承により，次の 2 つの場合において生じる優先度逆転を抑制できる．1 つは，プロセス優先度が依頼元プロセス ＜ 他プロセス ＜ OS サーバの場合である．この場合，他プロセスの優先度は依頼元プロセスの優先度に比べ高いにもかかわらず，OS サーバの優先度が他プロセスに比べ高いため，優先度継承を行わないと，OS サーバに依頼した依頼元プロセスの処理が優先され，優先度逆転が生じる．もう 1 つは，プロセス優先度が OS サーバ ＜ 他プロセス ＜ 依頼元プロセス の場合である．この場合，依頼元プロセスの優先度は他プロセスの優先度に比べ高いにもかかわらず，OS サーバの優先度が他プロセスに比べ低いため，優先度継承を行わないと，他プロセスの処理が優先され，優先度逆転が生じる．

優先度継承の契機には，依頼登録時と依頼取得時がある．なお，依頼登録時は，依頼元プロセスの優先度が OS サーバの優先度に比べて高いという条件（以降，継承条件）を満たす場合のみ優先度継承を行う．

2.2 コア間通信回数の削減

コア間通信時間は，処理時間の長い処理である IPI 通信処理を含むため，オーバヘッド削減のためにはコア間通信の利用を抑制する必要がある．コア毎に独立したスケジューラでは，他コア上のプロセスのプロセス情報を変更するためにコア間通信を必要とする．したがって，コアを跨ぐ OS サーバ間通信は，OS サーバの起床や優先度継承において，コア間通信を必要とする．ゆえに，異なるコア上の OS サーバを起床させるコア間通信（以降，起床コア間通信）と優先度継承させるコア間通信（以降，継承コア間通信）の回数を削減する必要がある．

最初に，起床コア間通信の回数の削減について説明する．シングルコア環境では，OS サーバの起床条件に OS サーバの実行状態を利用できる．しかし，マルチコア環境で

は，異なるコア上の OS サーバの起床判定に OS サーバの実行状態を利用すると不具合が発生する可能性がある．これは，依頼元プロセスによる OS サーバの実行状態の確認後，依頼先コアのスケジューラにより OS サーバの実行状態が変更され，OS サーバを起床させるべき際に起床させられない可能性があるためである．このため，コア間通信を行い，依頼先コアのスケジューラへ OS サーバの起床を依頼する必要がある．しかし，処理依頼時に常に起床コア間通信を行うとオーバヘッドが大きい．

そこで，異なるコア上の OS サーバの起床条件として，OS サーバの依頼/結果キューに登録されている依頼数/結果数を利用する．これにより，依頼先コア上の OS サーバの実行状態が変わっても OS サーバの起床が不要な場合を判定できる．ここで，依頼登録前の依頼キューに繋がる依頼数を Req_{prev}，依頼登録前の結果キューに繋がる結果数を Ret_{prev}，依頼登録後の依頼キューに繋がる依頼数を $Req_{current}$ とした時，以下の 3 つの条件（以降，起床条件）を満たす場合に OS サーバを起床させる．

（条件 1）$Req_{prev} = 0$
（条件 2）$Ret_{prev} = 0$
（条件 3）$Req_{current} \neq 0$

（条件 1）または（条件 2）を満たさない場合，OS サーバは他プロセスの依頼処理または結果返却を実行中であり，起床させる必要がない．（条件 3）を満たさない場合，依頼元プロセスの依頼登録から登録後の依頼数の確認までの間に，OS サーバは依頼を取得している．すなわち，OS サーバは依頼処理を実行中である．起床条件は，依頼元プロセスによる依頼登録と起床判定の処理時間が，OS サーバによる依頼取得，依頼処理の実行，および次の依頼取得の処理時間の和に比べ短いことを前提としている．なお，実測により，この前提が保証できることを確認した．

同様に，結果返却時の依頼元プロセスの起床条件について述べる．結果返却後の結果キューに繋がる結果数を $Ret_{current}$ とした時，先に述べた（条件 1）と（条件 2）に以下の（条件 4）を加えた 3 つの条件を満たす場合に依頼元プロセスを起床させる．

（条件 4）$Ret_{current} \neq 0$

次に，継承コア間通信の回数の削減について説明する．基本方式は，依頼登録時に優先度継承を行い，OS サーバの優先度を変更する．しかし，コア毎に独立したスケジューラは他コア上のプロセスのプロセス情報を変更できない．したがって，依頼元コアのスケジューラは，コア間通信によって，依頼先コアのスケジューラへ優先度継承を依頼する．

ここで，依頼登録時に OS サーバの起床依頼を行うことに着目する．OS サーバの起床依頼は，優先度継承と同じくコア間通信を必要とする．そこで，OS サーバが起床条件を満たす場合，または優先度継承が必要な場合，優先度継承と OS サーバ起床の 2 つの処理を 1 つのコア間通信によって依頼する．これにより，コア間通信回数を最大 1 回に抑制できる．

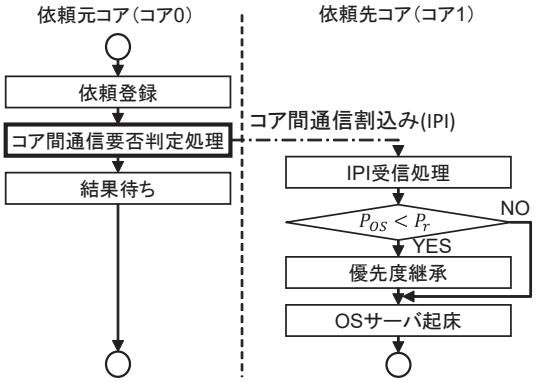

図 3 処理依頼の処理流れ

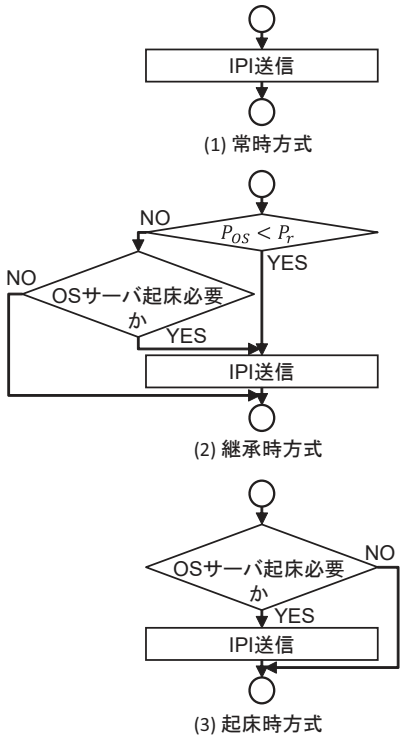

図 4 コア間通信要否判定処理

2.3 トレードオフを考慮した制御方式

コアを跨ぐ OS サーバ間通信の処理依頼の処理流れを**図 3** に示す．なお，依頼元プロセスの優先度を P_r，OS サーバの優先度を P_{OS} とする．依頼元コアは，依頼登録後，コア間通信要否判定処理を行う．コア間通信を行う場合，依頼先コアに IPI を送信し，結果を待つ．コア間通信を行わない場合，結果を待つ．依頼先コアは，IPI を受信後，優先度継承と OS サーバの起床を行う．

コア間通信要否の条件として，起床条件と継承条件がある．したがって，コア間通信要否判定処理として，条件を

表 1 優先度逆転時間とコア間通信回数の関係

通番	起床	継承	逆転条件	常時方式		継承時方式		起床時方式	
			$(P_{OS} \leqq P_o < P_r)$	t_{inv}	コア間通信	t_{inv}	コア間通信	t_{inv}	コア間通信
1	不要	不要	偽	0	1 回	0	0 回	0	0 回
2		必要	偽				1 回		
3			真					> 0	
4	必要	不要/必要	偽/真					0	1 回

用いない場合，継承条件と起床条件を用いる場合，起床条件を用いる場合の 3 つの処理流れが考えられる．これらを**図 4** に示し，以下に説明する．

(1) 常時方式
優先度逆転を最も抑制したい場合，必ずコア間通信を行い，優先度継承を行う．この方式では，最も優先度逆転を抑制できるものの，依頼登録時に必ずコア間通信オーバヘッドが生じる．Intel Xeon E5-2630 v3（8Core，2.4GHz）において，コア間通信オーバヘッドは，約 2 μs であり，処理依頼の処理時間が約 2.5 μs であることから，処理依頼の約 80%を占めることが分かる．

(2) 継承時方式
優先度継承するか否かを判定し，コア間通信回数を削減する．具体的には，継承条件（$P_{OS} < P_r$）または起床条件を満たす場合のみ，コア間通信を行う．この方式は，依頼元コア上で依頼先コア上の OS サーバの優先度を用いて継承条件の判定を行うため，各コア上の独立したスケジュール動作により，判定後に OS サーバの優先度が低下し優先度逆転が生じる可能性，および優先度が上昇し継承が不要になったにもかかわらずコア間通信を行う可能性がある．

(3) 起床時方式
OS サーバの起床条件を満たす場合のみ，コア間通信を行う．この方式は，「OS サーバの起床不要時において，処理依頼による優先度逆転が生じないことがシステム上明らかである場合，依頼登録時に継承する必要はないこと」に着目した方式である．この方式はコア間通信回数を最も削減できる．

2.4 定性的比較

提案方式を優先度逆転時間の抑制効果とコア間通信回数の削減効果の観点から比較する．

各方式における処理流れの差は依頼処理にのみ存在し，起床を不要とする場合において生じる．なお，起床を必要とする場合，差は生じない．それぞれの場合における，コア間通信回数を調べ，コア間通信回数の削減効果について比較する．さらに，優先度逆転が生じる場合と生じない場合についても場合分けすることで，優先度逆転時間の抑制効果を比較する．優先度逆転が生じる条件（逆転条件）は，$P_{OS} \leqq P_o < P_r$ である．なお，P_o は依頼先コア上の RUN/READY 状態のプロセスの優先度である．

表 1 に 3 つの方式の優先度逆転時間 (t_{inv}) とコア間通信回数の関係を示し，以下に説明する．

通番 1 の場合，常時方式のみコア間通信を行い，優先度継承処理を行う．したがって，常時方式のコア間通信回数は他の方式に比べ，1 回だけ大きい．一方，他の方式では優先度継承を行っていないため，依頼登録後から OS サーバの依頼取得までの間に優先度逆転が生じる可能性がある．

通番 2 の場合，常時方式と継承時方式はコア間通信を行い，優先度継承処理を行う．したがって，常時方式と継承時方式のコア間通信回数は，起床時方式に比べ，1 回だけ大きい．一方，起床時方式では優先度継承を行っていないため，依頼登録後から OS サーバの依頼取得までの間に優先度逆転が生じる可能性がある．

通番 3 の場合，常時方式と継承時方式はコア間通信を行い，優先度継承処理を行う．したがって，常時方式と継承時方式のコア間通信回数は，起床時方式に比べ，1 回だけ大きい．一方，起床時方式では優先度逆転が生じる．

通番 4 の場合，全ての方式がコア間通信を行い，優先度継承処理を行う．したがって，各方式の優先度逆転時間とコア間通信回数の差はない．

3. 評価

3.1 実現

コア毎に独立したスケジューラを持つマイクロカーネル構造 OS である ***AnT*** オペレーティングシステム（An operating system with adaptability and toughness）[9] に常時方式，継承時方式，および起床時方式を実現し，Intel Xeon E5-2630 v3（8Core，2.4GHz）の計算機を用いて実測評価した．

3.2 観点と評価内容

まず，基本評価として，依頼を登録順に取得し，優先度継承を行わず，かつ常にコア間通信を行う方式（以降，FPFIFO 方式）と提案方式について処理依頼から結果受取までの時間（以降，応答時間）を比較し，優先度逆転の抑制効果とコア間通信回数の削減効果を明らかにする．

次に，事例評価として，FPFIFO 方式と提案方式について，異種 OS サーバ共存時の処理依頼の応答時間，および

表 2 基本評価の応答時間 (μs)

通番	起床	継承	逆転条件 ($P_{OS} \leqq P_o < P_r$)	FPFIFO 方式	常時方式	継承時方式	起床時方式
1	不要	不要	偽	107.52	56.84	55.93	55.95
2		必要	偽	107.50	56.41	56.39	55.48
3			真	207.93	56.71	56.70	155.70
4	必要	不要	偽	56.42	57.01	56.94	56.93
5		必要	偽	56.44	56.52	56.54	56.52
6			真	156.36	56.52	56.56	56.56

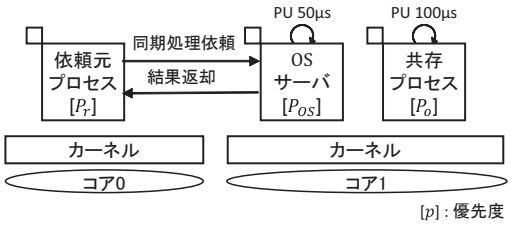

図 5 基本評価モデル

ファイル複写処理共存時のファイル参照性能を比較する．

3.3 基本評価

FPFIFO 方式と提案方式について，応答時間を比較する．基本評価モデルを**図 5** に示す．コア 0 上の依頼元プロセスとコア 1 上の OS サーバで OS サーバ間通信を行い，依頼元プロセスが結果を受取るまでの時間を測定した．共存プロセスとして，コア 1 上に PU 処理を 100 μs 行う他プロセスを配置する．測定において，処理依頼は，OS サーバへ渡す引数および戻り値を無しとし，OS サーバの依頼処理時間を 50 μs とする．起床を必要とする場合と不要とする場合において，OS サーバの依頼数をそれぞれ 0 と 1 として測定した．なお，OS サーバに事前に登録されている依頼の依頼元プロセスの優先度は最低優先度とした．

基本評価における応答時間を**表 2** に示す．表 2 より，以下のことが分かる．

(1) 起床を不要とし，かつ逆転条件を満たさない場合 (通番 1，通番 2)，提案方式の応答時間は，FPFIFO 方式に比べ，約 50 μs 短い．これは，提案方式では依頼元プロセスの優先度順に依頼取得を行うのに対し，FPFIFO 方式では登録順に依頼を取得するためである．したがって，FPFIFO 方式では，OS サーバに事前に登録されていた最低優先度のプロセスによる依頼が実行される．ゆえに，FPFIFO 方式では，依頼元プロセスと OS サーバに事前に依頼を登録していたプロセス間で優先度逆転が生じている．

(2) 起床を不要とし，かつ継承を不要とする場合 (通番 1)，継承時方式と起床時方式の応答時間は，常時方式に比べ，約 0.9 μs 短い．これは，継承時方式と起床時方式では起床条件を用いて起床の判定を行うのに対し，常時方式では常に起床を行うためである．すなわち，上記は，継承時方式と起床時方式のコア間通信回数が常時方式に比べ少ないことに起因する．

(3) 起床を不要とし，継承を必要とし，かつ逆転条件を満たさない場合 (通番 2)，起床時方式の応答時間は，継承時方式に比べ，約 0.9 μs 短い．これは，起床時方式では起床条件のみを用いてコア間通信の要否を判定するのに対し，継承時方式では起床条件と継承条件を用いてコア間通信の要否を判定するためである．したがって，上記は，起床時方式のコア間通信回数が継承時方式に比べ少ないことに起因する．

(4) 起床を不要とし，かつ逆転条件を満たす場合 (通番 3)，常時方式と継承時方式の応答時間は，起床時方式に比べ，約 99 μs 短い．これは，常時方式と継承時方式ではコア間通信による優先度継承を行うのに対し，起床時方式では起床条件を満たさずコア間通信による優先度継承を行わないためである．したがって，上記は，起床時方式において，優先度逆転が生じていることに起因する．

(5) 起床を必要とし，かつ継承を不要とする場合 (通番 4)，提案方式の応答時間は，FPFIFO 方式に比べ，約 0.5 μs 長い．これは，提案方式では，OS サーバの依頼取得時に優先度継承を行い，OS サーバの優先度を低下させ，再スケジューリングが生じるためである．

(6) 起床を必要とし，かつ逆転条件を満たす場合 (通番 6)，提案方式の応答時間は，FPFIFO 方式に比べ，約 100 μs 短い．これは，提案方式ではコア間通信によって OS サーバの起床と優先度継承を行うのに対し，FPFIFO 方式ではコア間通信によって OS サーバの起床のみを行うためである．したがって，上記は，FPFIFO 方式において，優先度逆転が生じていることに起因する．

上記より，提案方式は FPFIFO 方式に比べ優先度逆転を抑制していることが分かる．また，処理負荷が大きく，OS サーバと同一コア上に OS サーバへ依頼するプロセスより優先度の低いプロセスが存在しない場合，起床時方式を用いると良く，存在する場合，継承時方式を用いると良いと推察する．

3.4 事例評価

3.4.1 異種 OS サーバ共存時の応答時間

異種 OS サーバ共存時の応答時間を測定し，FPFIFO 方

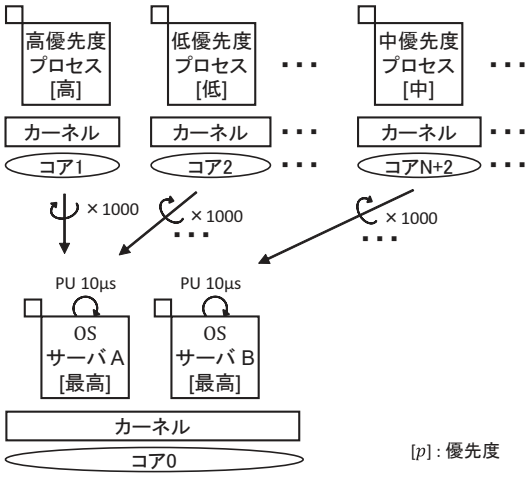

図 6 異種 OS サーバ共存時の評価モデル

式と提案方式を比較する．異種 OS サーバ共存時の評価モデルを**図 6** に示す．コア 0 に OS サーバ A と OS サーバ B，コア 1 に高優先度プロセス，コア 2∼N+1 に低優先度プロセス，コア N+2∼N+M+1 に中優先度プロセスを配置する．N は低優先度プロセス数，M は中優先度プロセス数である．高優先度プロセスと低優先度プロセスは，OS サーバ A に処理依頼を行い，中優先度プロセスは，OS サーバ B に処理依頼を行う．OS サーバの優先度は最高優先度とする．測定において，処理依頼は，OS サーバへ渡す引数および戻り値を無しとし，OS サーバの依頼処理時間を 10 μs とする．コア 1 上の高優先度プロセスとコア 0 上の OS サーバ A で OS サーバ間通信を行い，高優先度プロセスが結果を受取るまでの時間を測定した．なお，低優先度プロセス数（N）を 0∼3，中優先度プロセス数（M）を 0∼2 とする．

1000 回処理時の平均応答時間を**図 7** に示す．図 7(A) より，以下のことが分かる．

(1) FPFIFO 方式の平均応答時間は，低優先度プロセス数に比例して増加する．これは，FPFIFO 方式では，登録順に依頼を取得するためである．したがって，低優先度プロセス数の増加により，OS サーバ A の依頼キューに繋がる低優先度プロセスの依頼数が増加し，この数に比例して，平均応答時間が増加する．ゆえに，FPFIFO 方式では，優先度逆転が生じている．

(2) 提案方式の平均応答時間は，低優先度プロセス数に比例して増加しない．これは，提案方式では，優先度順に依頼を取得するためである．

(3) 低優先度プロセス数 3 の場合，起床時方式の平均応答時間は，常時方式に比べ，約 2.5 μs 短い．これは，起床時方式では起床条件を満たさない場合，コア間通信を行わないのに対し，常時方式では，常にコア間通信を行うためである．

(4) 低優先度プロセス数 3 の場合，継承時方式の平均応答時間は，常時方式に比べ，約 1.0 μs 短い．これは，継承時方式では起床条件と継承条件の両方を満たさない場合，コア間通信を行わないのに対し，常時方式では，常にコア間通信を行うためである．したがって，継承時方式では，低優先度プロセスの処理依頼時にコア間通信を行わないため，OS サーバ A による高優先度プロセスの依頼処理中に IPI 受信割り込み処理が行われない．

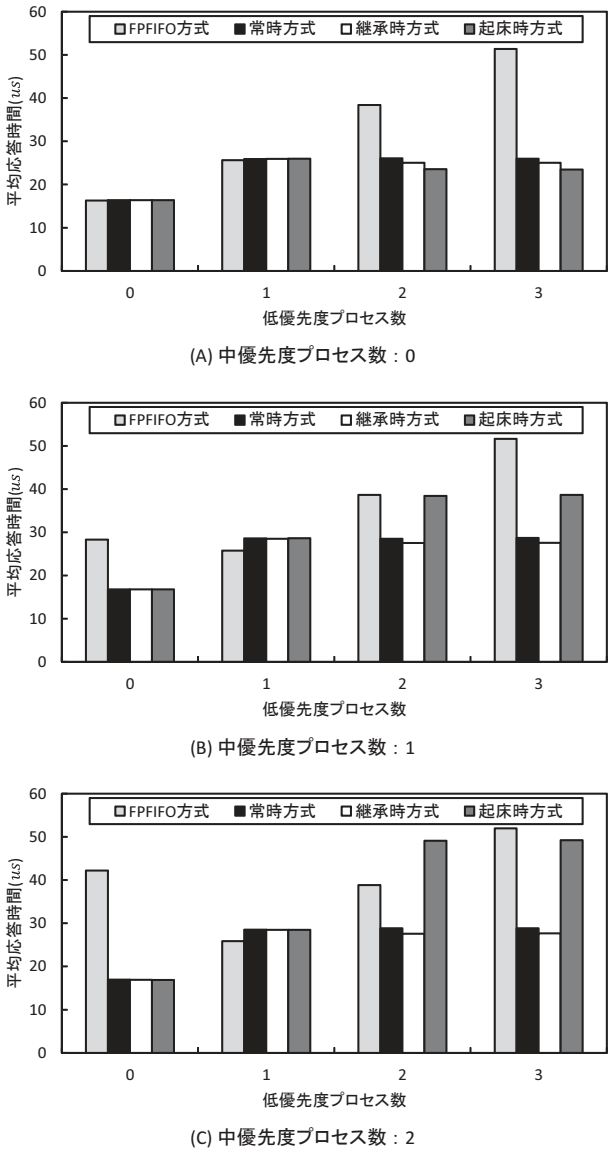

図 7 異種 OS サーバ共存時の平均応答時間

図 7(A)(B)(C) より，以下のことが分かる．

(5) 低優先度プロセス数 0 の場合，FPFIFO 方式の平均応答時間は，中優先度プロセス数に比例して増加する．これは，FPFIFO 方式では，OS サーバに優先度の継承を行わず，OS サーバの優先度が固定であるためである．したがって，中優先度プロセスの依頼先 OS サーバ B の依頼処理によって，高優先度プロセスの依頼先 OS サーバ A の依頼処理が遅延する．ゆえに，この場合，FPFIFO 方式では，優

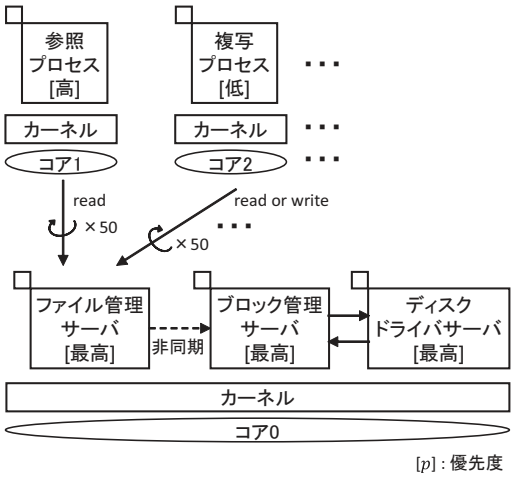

図 8 ファイル複写処理共存時の評価モデル

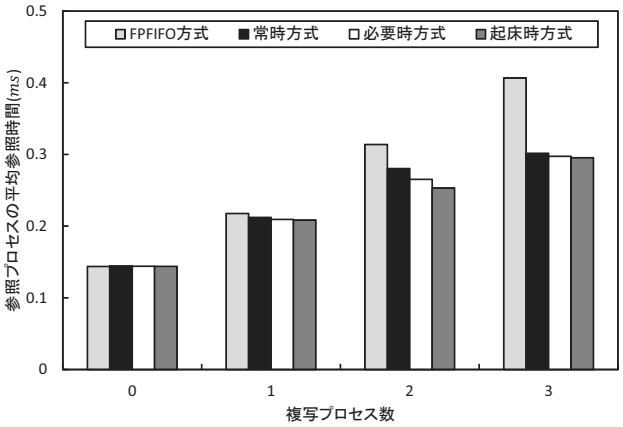

図 9 ファイル複写処理共存時の参照プロセスの平均参照時間

先度逆転が生じている．

(6) 低優先度プロセス数 2 または 3 の場合，起床時方式の平均応答時間は，中優先度プロセス数に比例して増加する．これは，起床時方式では，起床条件を満たさない場合，コア間通信を行わず，優先度継承が行われないためである．したがって，中優先度の OS サーバ B が依頼処理を実行している場合，高優先度プロセスが低優先度かつ依頼数が 1 以上の OS サーバ A に依頼すると，OS サーバ B が依頼処理を完了し依頼待ち状態になるまで，OS サーバ A の依頼処理が遅延する．ゆえに，この場合，起床時方式では，優先度逆転が生じている．

3.4.2 ファイル複写処理共存時のファイル参照性能

ファイル参照を行う高優先度のプロセス（参照プロセス）とファイル複写処理を行う低優先度のプロセス（複写プロセス）を同時走行させ，参照プロセスのファイル参照時間を測定する．ファイル複写処理共存時の評価モデルを**図 8** に示す．参照プロセスは，4KB のファイルを読み込む処理を繰り返し行う．複写プロセスは，4KB のファイルを読み込み，異なるディレクトリ以下に書き出す処理を繰り返し行う．ファイル操作処理に関連するファイル管理サーバ（FS），ブロック管理サーバ（BLK），ディスクドライバサーバ（DK）をコア 0 上に配置し，参照プロセスをコア 1 上に配置し，複写プロセスをコア 2 からコア N+1 に配置する．N は複写プロセス数である．参照プロセスのファイル読み込み処理の時間を測定した．なお，複写プロセス数（N）は，0~3 の場合で測定した．

AnT のファイル操作処理に関連する OS サーバについて，以下に説明する．FS は，i ノードの管理を行う．BLK は，ブロックキャッシュの管理を行う．DK は，外部記憶装置の管理を行う．ファイル参照を行う場合を例にファイル操作の処理流れを説明する．依頼元プロセスは，FS に対してファイル読み込みを同期処理依頼する．FS は，BLK に対して当該ファイルに対応するデータブロックの読み込みを非同期処理依頼する．BLK は，当該データブロックをブロックキャッシュから探索し，AP プロセスに対して返却する．ブロックキャッシュに当該データブロックが存在しない場合，BLK は，DK に対して当該データブロックの読み込みを同期処理依頼する．

50 回処理時の平均参照時間を**図 9** に示す．図 9 より，以下のことが分かる．

(1) FPFIFO 方式の平均参照時間は，複写プロセス数に比例して増加する．これは，FPFIFO 方式が登録順に依頼を取得するためである．したがって，FPFIFO 方式では優先度逆転が生じている．一方，提案方式は，依頼元プロセスの優先度順に依頼を取得するため，複写プロセス数に比例して増加しない．したがって，提案方式は，実 I/O 処理においても，FPFIFO 方式に比べ，優先度逆転を抑制している．

(2) 提案方式の平均参照時間は，複写プロセス数に比例して増加していない．これは，参照プロセスのファイル操作処理依頼時に，DK が複写プロセスの依頼を実行中である確率の増加が参照時間増加の要因であるためである．また，複写プロセス数 2 の場合の平均参照時間に対する複写プロセス数 3 の場合の平均参照時間の増加量がわずかであることから，この確率が 100%に近いと推察できる．すなわち，提案方式の平均参照時間は，複写プロセス数 4 以上の場合も複写プロセス数 3 の場合とほとんど同じであると推察できる．

4. 関連研究

既存のマイクロカーネル構造 OS における優先度逆転の抑制の研究は，シングルコア環境での研究とマルチコア環境での研究に分けられる．

シングルコア環境での研究は，セマフォのような同期機構を対象とした研究（Lynx[10]）と OS サーバ間通信を対象とした研究（QNX[11]，MINIX4RT[12]，RT-Mach[13]，Credo[14]）に分けられる．Lynx は，セマフォにおける優

先度継承を実現することでセマフォにより生じる優先度逆転を抑制している．一方，QNX，MINIX4RT，RT-Mach，および Credo は，OS サーバ間通信における優先度継承を実現することで OS サーバ間通信により生じる優先度逆転を抑制している．しかし，これらの手法は，マルチコア環境を想定していないため，マルチコア環境に適用した場合，優先度継承による排他制御やコア間通信が多発してしまう．

マルチコア環境でのセマフォのような同期機構を対象とした研究として，fiasco[7] がある．fiasco は，他コア上の継承先プロセスを継承元プロセスが走行するコアに移譲させ，優先度継承を行う．継承先プロセスがセマフォによって保護されるクリティカルセクション（以降，CS）を実行している場合，移譲を行えないため，CS の終了の確認または継承先プロセスの実行停止の確認を行う必要がある．fiasco は，この確認にポーリングを用いることで，IPI の送信回数を抑制している．しかし，この手法が有効に働くのは，セマフォによって保護される CS が短時間である場合のみである．多くの場合，OS サーバの依頼処理の実行時間は，セマフォによって保護される CS に比べ長い．したがって，OS サーバ間通信における優先度逆転の抑制に fiasco を適用すると，長時間のポーリング待ちが発生し，システム全体に影響を与えてしまう．一方，提案方式では，依頼登録時と結果返却時に IPI を送信することで優先度継承を行っているため，CS の終了および継承先プロセスの実行停止の確認が不要であり，ポーリング待ちは発生しない．

5. おわりに

本稿では，優先度継承方式である常時方式，継承時方式，起床時方式について述べた．また，コア毎に独立したスケジューラを持つマイクロカーネル構造 OS の ***AnT*** に提案方式を実現し，基本評価と事例評価を行い，優先度逆転の抑制効果とコア間通信回数の削減効果を明らかにした．

基本評価として，処理依頼の応答時間を測定し，FPFIFO 方式と提案方式を比較した．結果より，提案方式は FPFIFO 方式に比べ優先度逆転を抑制していること，常時方式と継承時方式は起床時方式に比べ，優先度逆転を抑制していること，および継承時方式は常時方式よりオーバヘッドが小さく，起床時方式は継承時方式よりオーバヘッドが小さいことが分かった．また，処理負荷が大きく，OS サーバと同一コア上に OS サーバへ依頼するプロセスより優先度の低いプロセスが存在しない場合，起床時方式を用いると良く，存在する場合，継承時方式を用いると良いと推察した．

事例評価として，異種 OS サーバ共存時の応答時間の評価とファイル複写処理共存時のファイル参照性能の評価を行った．異種 OS サーバ共存時の応答時間の評価結果から，起床時方式はコア間通信回数を削減するものの，優先度逆転が生じる場合があることを示した．ファイル複写処理共存時のファイル参照性能の評価結果から，実 I/O 処理時にも提案方式は，FPFIFO 方式に比べて優先度逆転を抑制していることを示した．

参考文献

[1] Sha, L., Rajkumar, R., Lehoczky, J.P.: Priority inheritance protocols: an approach to real-time synchronization, IEEE Transactions, Vol.39, No.9, pp.1175–1185 (1990).
[2] Liedtke, J.: Toward real microkernels，Communications of the ACM, Vol.39, No.9, pp.70–77 (1996).
[3] Tanenbaum, A.S., Herder, J.N., and Bos, H.: Can we make operating systems reliable and secure?, IEEE Computer Magazine, Vol.39, No.5, pp.44–51 (2006).
[4] Black, D.L., Golub, D.B., Julin, D.P., Rashid, R.F., Draves, R.P., Dean, R.W., Forin, A., Barrera, J., Tokuda, H., Malan, G., and Bohman, D.: Microkernel operating system architecture and mach, Journal of Information Processing, Vol.14, No.4, pp.442–453 (1992).
[5] 岡本 幸大，谷口 秀夫: ***AnT*** オペレーティングシステムにおける高速なサーバプログラム間通信機構の実現と評価, 電子情報通信学会論文誌 (D), Vol.J93-D, No.10, pp.1977–1987 (2010).
[6] Björn B. Brandenburg, Andrea Bastoni: The Case for Migratory Priority Inheritance in Linux: Bounded Priority Inversions on Multiprocessors, Proceeding of the 14th Real-Time Linux Workshop (RTLWS2012), pp.67–86 (2012).
[7] Michael, H. and Michael, P.: Helping in a Multiprocessor Environment, In Proceedings of the Second Workshop on Common Microkernel System Platforms, pp.1–5, (2001).
[8] Zuberi, K.M. and Shin, K.G.: EMERALDS: a small-memory real-time microkernel, IEEE Transactions on Software Engineering, Vol.27, No. 10, pp.909–928, (2001).
[9] 井上 喜弘, 佐古田 健志, 谷口 秀夫: マルチコアプロセッサ上での負荷分散を可能にする ***AnT*** オペレーティングシステムの開発 , 情報処理学会研究報告, Vol.2012-DPS-150, No.37, pp.1–8 (2012).
[10] Lynx Software Technology, “LynxOS RTOS,” Lynx Software Technology, http://www.lynx.com/products/real-time-operating-systems/lynxos-rtos/, accessed Oct. 19, 2016.
[11] Dan, H.: An Architectural Overview of QNX, Proceedings of the Workshop on Micro-kernels and Other Kernel Architectures, pp.113–126, (1992).
[12] Pessolani, P.A.,: MINIX4RT: Real-Time Interprocess Communications Facilities, Workshop de Arquitecturas, Redesy Sistemas Operativos, XII Congreso Argentino de Ciencias de la Computación, (2006).
[13] Kitayama, T., Nakajima, T., and Tokuda, H.: RT-IPC:An IPC Extension for Real-Time Mach, USENIX Microkernels and Other Kernel Architectures Symposium, pp.91–104 (1993).
[14] Steinberg, U., Wolter, J., Hartig, H.: Fast component interaction for real-time systems, In Euromicro Conference on Real-Time Systems, pp.89–97 (2005).