Vol.2016-OS-136 No.17
2016/3/1

マルチコア環境における優先度逆転を抑制する *AnT* オペレーティングシステムのスケジュール機構

鴨生 悠冬[1]　山内 利宏[1]　谷口 秀夫[1]

概要：優先度に基づくスケジュールは，優先度逆転の現象を抑制することが重要である．特に，マイクロカーネル OS では，AP プロセスだけではなく，OS サーバも優先度逆転の影響を受けてしまうため，その重要性は高い．また，マルチコア環境では，処理の連携と優先性を保証するために，複数のコアを考慮した優先度逆転の抑制が必要である．そこで，本稿では，マイクロカーネル OS である ***AnT*** において，マルチコア環境下での優先度逆転を抑制するスケジュール機構を提案する．

1. はじめに

オペレーティングシステム（以降，OS）は，サービスの要望に則したプロセスのスケジュールが求められる．したがって，プロセスの優先度に基づくスケジュールは，優先度逆転 [1] を抑制する必要がある．特に，マイクロカーネル構造 OS[2][3][4] においては，OS 機能をプロセス（OS サーバ）として実現しており，OS 機能を実現する処理に優先度逆転が生じることの抑制，つまり OS サーバを含めたプロセスの走行環境に応じたスケジュールが求められる．

マルチコア環境において，コア毎に独立したスケジュール機能は，全てのコアを単一のスケジューラでスケジュールする機能と比較して，スケジュールオーバヘッドが小さい．しかし，複数コア上で走行するプロセス間で優先度逆転が生じる．したがって，コア毎に独立したスケジュール機能における優先度逆転の抑制には，スケジュール機能間の連携が必要である．しかし，この実現には，コア間のスケジュール連携のために，コア間通信を必要とする．一方，コア間通信のコストは大きいため，コア間通信回数の削減が重要である．

我々は，シングルコア環境において，OS サーバ間通信での優先度継承による優先度逆転抑制法を提案し，マイクロカーネル構造を有する ***AnT*** オペレーティングシステム（An operating system with adaptability and toughness）（以降，***AnT***）において評価した [5][6]．

本稿では，マルチコア環境において，プロセス間の優先度逆転を抑制し，かつコア間通信回数を削減するスケジュール機構を提案する．

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

2. *AnT* オペレーティングシステム

2.1 基本構造

AnT はマイクロカーネル構造を有する OS である．***AnT*** の基本構造を**図 1** に示し，以下に説明する．

カーネルは，最小限の機能を有する部分である．例えば，スケジュール機能を有する．OS サーバは，OS 機能をプロセス化した部分である．例えば，ファイル管理機能や通信ネットワーク制御機能がある．

マルチコア環境における ***AnT*** は，各コアの独立動作（方針 1）を実現するために，カーネルをコア毎に配置する．各カーネルの軽量化（方針 2）を実現するために，マスタカーネル（以降，m-カーネル）とピコカーネル（以降，p-カーネル）の 2 種類のカーネルを用意する．m-カーネルは，電源投入時に最初に起動するコアを制御し，マイクロカーネルに必要な全機能を有する．一方，p-カーネルは，m-カーネルが制御するコア以外のコアを制御し，スケジュール機能，サーバプログラム間通信機能，およびコア間通信制御機能を有する．処理の高速化（方針 3）を実現するために，

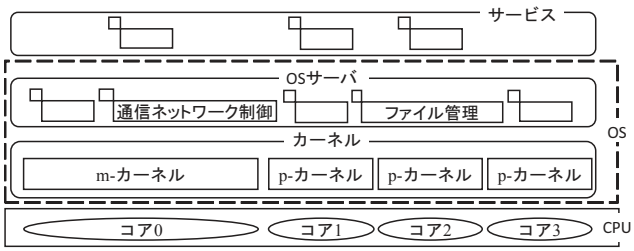

図 1　***AnT*** の基本構造

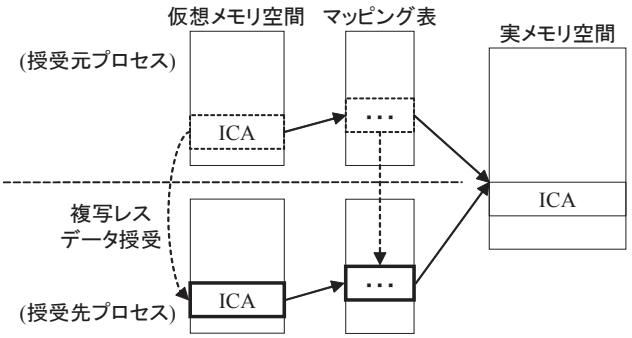

図 2 複写レスデータ授受の様子

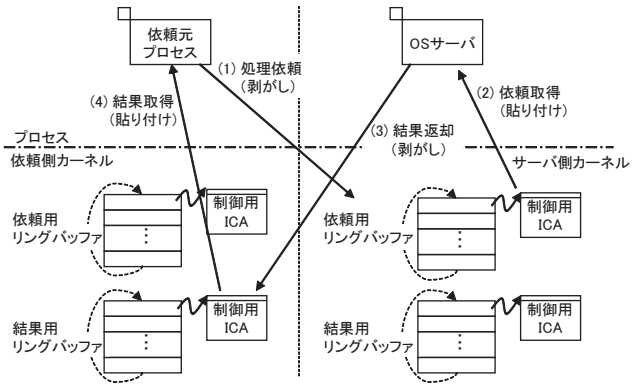

図 3 サーバプログラム間通信の基本機構

高速なサーバプログラム間通信機構を有する．

2.2 データ複写レス授受機能

プロセス間の通信を高速化するため，コア間通信データ域（ICA: Inter-core Communication Area）を利用したデータ複写レス授受機能を有する．ここで，コア間通信データ域の「コア」は，CPU コアではなく OS サーバを意味する．ICA の特徴として以下の 3 つがある．

（1）ページを単位とし，n ページ分の領域の確保と解放
（2）確保した領域 (n ページ) の実メモリ連続の保証
（3）2 仮想空間での領域の貼り替え

ICA は，ページを最小単位として管理される領域であり，ICA へのアクセスは，プロセスごとの仮想空間のページテーブルを通して行われる．ここで，ページテーブルへの書き込みを貼り付けと呼び，ページテーブルからの削除を剥がしと呼ぶ．また，貼り替えとは，剥がしと貼り付けを行うことを意味する．プロセス間の複写レスでのデータ授受の様子を**図 2** に示す．ICA を利用したプロセス間でのデータ授受は，授受するデータを格納した ICA をデータ授受元プロセスの仮想空間から剥がし，データ授受先プロセスの仮想空間へ貼り付けることで行われる．

2.3 サーバプログラム間通信機構

2.3.1 基本機構

サーバプログラム間通信 [7] の基本機構を**図 3** に示す．この機構は，ICA を利用することにより，プロセス間でデータ複写レスでの通信を実現している．具体的には，OS サーバへ渡す引数や通信制御の情報（以降，依頼情報）を制御用の ICA（以降，制御用 ICA）に格納し，扱うデータをデータ用の ICA（以降，データ用 ICA）に格納する．カーネルは，プロセス毎に通信のための依頼用リングバッファと結果用リングバッファを持つ．さらに，同期型と非同期型の通信インタフェースを同様な形式で提供し，両インタフェースを選択して利用できる．基本的な通信の流れを以下に説明する．

（1）依頼元プロセスが処理依頼を行うと，依頼元プロセスの動作するコア上のカーネル（依頼側カーネル）は OS サーバの依頼用リングバッファに依頼情報を格納した制御用 ICA を登録し，依頼元プロセスから制御用 ICA を剥がす．また，OS サーバが WAIT 状態の場合，OS サーバを起床させる．

（2）OS サーバの動作するコア上のカーネル（サーバ側カーネル）は，OS サーバに制御用 ICA を貼り付ける．OS サーバは，依頼用リングバッファから依頼情報を格納した制御用 ICA を取得し，処理を実行する．

（3）OS サーバが結果返却を行うと，サーバ側カーネルは依頼元プロセスの結果用リングバッファに結果情報を格納した制御用 ICA を登録し，自身から結果情報を格納した制御用 ICA を剥がす．また，依頼元プロセスが WAIT 状態の場合，依頼元プロセスを起床させる．

（4）依頼側カーネルは，依頼元プロセスに制御用 ICA を貼り付ける．依頼元プロセスは，結果用リングバッファから結果情報を格納した制御用 ICA を取得し，処理を終了する．

2.3.2 コア間のサーバプログラム間通信

コア間のサーバプログラム間通信は，プロセスを起床させる処理において，コア間通信を伴う．各コアのカーネルは，個別にスケジュールキューを持ち，プロセス実行を制御している．このため，他コアのプロセス起床は，コア間通信を用いて該当コアに依頼する．また，マルチコア環境では，複数のコアから依頼を登録される可能性がある．そこで，依頼用/結果用リングバッファをコア毎に用意する．これにより，排他制御オーバヘッドを削減している．

2.4 コア間通信

コア間通信 [8] は，コア間で共有している領域（以降，共有領域）を利用し通信内容の授受を行う．また，通信内容の登録を通知するために IPI（Inter-Processor Interrupt）を利用する．コア間通信の処理流れを**図 4** に示し，以下に説明する．

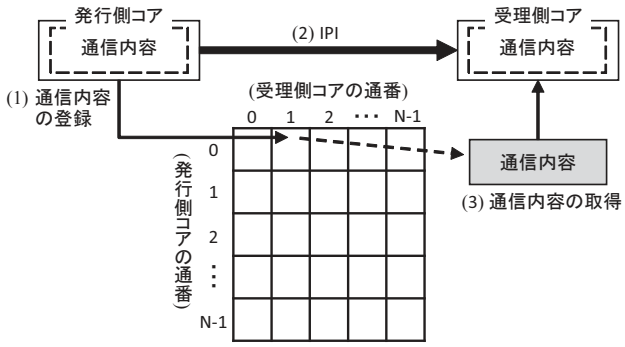

図 4 コア間通信の処理流れ

（1）依頼を発行する側のコアは，通信内容を共有領域に登録する．

（2）発行側コアは，依頼を受理する側のコアに IPI を送信し，受理側コアに通信内容の登録を通知する．

（3）受理側コアは，IPI の受信を契機に，共有領域から通信内容を取得し通信内容の処理を実行する．

共有領域は，N × N（N はコア数）の配列であり，依頼発行側コア（i）と依頼受理側コア（j）のコア間通信は，共有領域の（i, j) 番目のエントリを利用して通信内容を授受する．これにより，複数のコアが 1 つのコアにコア間通信を要求する際の排他処理制御を不要としている．

2.5 スケジュール機能

各コアのスケジュール機能は，プロセスの優先度に基づき当該コア上のプロセスをスケジュールする．また，タイムスライスとプリエンプション機能を持つ．なお，OS サーバの優先度は，AP プロセスの優先度より高く設定する．

AnT は，1 つのコアの処理がネックにならないようにして処理全体を高速化するために各コアの独立動作を設計方針としており，各コアにスケジュール機能を配置している．これは，カーネルで頻繁に動作するスケジュール機能を高速化するためである．各コアのスケジュール機能は独立しており，当該コア上のプロセスのみを制御し，他コア上のプロセス情報を確認するだけで操作しない．例えば，他コア上のプロセスの実行状態を確認する，しかし変更しない．これにより，スケジュール機能間の排他制御が不要になる．一方，各コアのスケジュール機能が全てのプロセス情報を操作する場合，排他制御が必要になる．この排他制御は複雑になると考えられる．また，コア数の増大によって，排他制御オーバヘッドが著しく増大する．

3. シングルコア環境における OS サーバの優先度逆転抑制法

3.1 問題と対処

シングルコア環境の ***AnT*** で生じる OS サーバによる優先度逆転の問題と対処について簡単に説明する [5][6].

（問題 1）OS サーバの依頼キューに先に繋がれた低優先度の AP プロセスの依頼により，後に繋がれた高優先度の AP プロセスの依頼の実行が遅延する．これは，処理依頼や処理結果の取得を FIFO 順に行うことに起因する．

（問題 2）低優先度の AP プロセスの依頼を OS サーバが処理する間，高優先度の AP プロセスの実行開始が遅延する．これは，OS サーバの優先度が AP プロセスの優先度より常に高いことに起因する．

上記の問題に対し，以下の対処がある．

（対処 1）サーバプログラム間通信時に授受する制御用 ICA に依頼元 AP プロセスの優先度を保持させ（以降，ICA 優先度），ICA 優先度が高い順に依頼を取得する．

（対処 2）OS サーバの依頼取得時（契機 1）または OS サーバへの依頼登録時（契機 2）に OS サーバの優先度を ICA 優先度に変更する．

3.2 抑制法

3.1 節で述べた 2 つの対処により，優先度逆転を抑制する制御法を表 1 に示す．可変優先度取得時変更法（VPGET）は，依頼取得を ICA 優先度順に行うとともに，依頼取得時に OS サーバの優先度を ICA 優先度に変更する制御法である．可変優先度登録時変更法（VPSET）は，依頼取得を ICA 優先度順に行うとともに，依頼登録時と依頼取得時に OS サーバの優先度を ICA 優先度に変更する制御法である．ただし，依頼登録時の OS サーバの優先度変更は，ICA 優先度が OS サーバの優先度よりも高い場合に行う．これは，低優先度の AP プロセスの処理依頼によって，OS サーバの優先度が不用意に下がることを防ぐためである．また，依頼取得時の OS サーバの優先度変更は，ICA 優先度が OS サーバの優先度よりも低い場合に行う．これは，OS サーバの優先度を低下させる契機として OS サーバの優先度変更を行うためである．

各制御法について，依頼登録の処理流れを**図 5** に示す．VPGET は，依頼キューの操作を禁止するために割り込み禁止 (1) を行い，依頼登録 (2) 後，依頼先プロセスが WAIT 状態であるか確認 (6) し，WAIT 状態の場合，依頼先プロセスを起床 (7) し，割り込み禁止を解除 (8) し，結果待ち (9) に移行する．VPSET は，処理 (1) と処理 (2) 後，依頼先プロセスが OS サーバであるか確認 (3) し，OS サーバの優先度が ICA 優先度より小さいか確認 (4) する．処理 (3) と処理 (4) を満たす場合，優先度継承 (5) を行う．その後，VPGET と同じく，処理 (6)〜(9) を行う．

優先度継承の条件を満たす場合，VPGET と VPSET の処理の差は，処理 (3)〜(5) である．これらの処理により，VPSET は VPGET より早期に優先度継承を行うことができる．一方，これらの処理は，スケジュールのオーバヘッドである．

表 1 各制御法の概要

制御法	依頼取得順	OS サーバの優先度変更	
		依頼登録時	依頼取得時
可変優先度取得時変更（VPGET）	ICA 優先度順	なし	ICA 優先度に変更
可変優先度登録時変更（VPSET）	ICA 優先度順	ICA 優先度より OS サーバの優先度が低い場合，ICA 優先度に変更	ICA 優先度より OS サーバの優先度が高い場合，ICA 優先度に変更

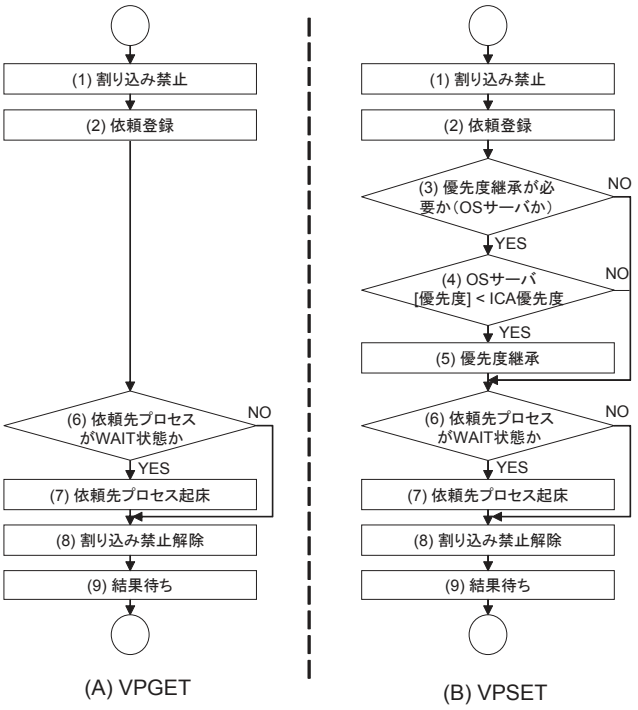

図 5 シングルコア環境における依頼登録の処理流れ

4. マルチコア環境における優先度逆転抑制法

4.1 前提

マルチコア環境の ***AnT*** は，コア毎に独立したスケジュール機能を持つため，複数のコア上で走行するプロセス間で優先度逆転が生じる．このため，スケジュール機能は，1つのコア内に閉じず，複数のコア上で走行するプロセス間の優先度逆転に対処する必要がある．この対処は，スケジュール機能間の連携を必要とする．しかし，各コアのスケジュール機能は独立しているため，他コア上のスケジュール機能により管理されるプロセス情報を操作できない．そこで，各コアのスケジュール機能は，コア間通信によって，他コアのスケジュール機能に操作を依頼し，連携する．しかし，コア間通信のオーバヘッドは大きい．このため，マルチコア環境のスケジュール機能は，コア間通信回数の削減を求められる．

4.2 コア間通信回数の削減

4.2.1 コア間通信の発生条件

マルチコア環境のスケジュール機能において，サーバプログラム間通信時，以下の条件でコア間通信が発生する．

（条件 1）依頼登録時に依頼先プロセスが依頼待ちにより WAIT 状態

（条件 2）結果返却時に依頼元プロセスが結果待ちにより WAIT 状態

（条件 3）依頼先プロセスが OS サーバであり，かつ OS サーバの優先度が ICA 優先度より低

VPGET と VPSET は，サーバプログラム間通信において（条件 1）や（条件 2）を満たす場合，依頼先（元）コアにコア間通信を行い，依頼先（元）プロセスの起床を依頼する．また，VPSET は，（条件 3）を満たす場合，依頼先コアにコア間通信を行い，OS サーバへの優先度継承を依頼する．

4.2.2 依頼先プロセスの起床条件

4.1 節で述べたようにコア間通信のオーバヘッドは大きいため，依頼先プロセスの起床依頼におけるコア間通信の削減が求められる．

依頼先プロセスの起床は，VPGET と VPSET の両方で行われる．ここでは，VPGET を例にプロセス起床依頼におけるコア間通信回数の削減方法を説明する．図 5 より，シングルコア環境の VPGET は，依頼先プロセスの起床条件として，依頼先プロセスの実行状態を利用する．しかし，マルチコア環境の VPGET は，異なるコア上の依頼先プロセスの起床条件として，依頼先プロセスの実行状態を利用できない．これは，依頼元プロセスによる依頼先プロセスの実行状態の確認後，他コアにより依頼先プロセスの実行状態が変更され，依頼元プロセスと依頼先プロセスの両方が WAIT 状態となる可能性があるためである．この問題は，「依頼取得〜WAIT 状態への遷移」と「依頼登録〜OS サーバの状態確認」の処理を同時に行うことができないように排他制御をすれば，解決できる．しかし，この排他制御を用いると，処理の並列性が失われる．さらに，この排他制御は，***AnT*** の設計方針である各コアの独立動作に反する．

そこで，排他制御を抑制した依頼登録の処理流れを**図 6** に示し，以降に説明する．マルチコア環境における依頼登録は，異なるコア上の依頼先プロセスの起床条件として，

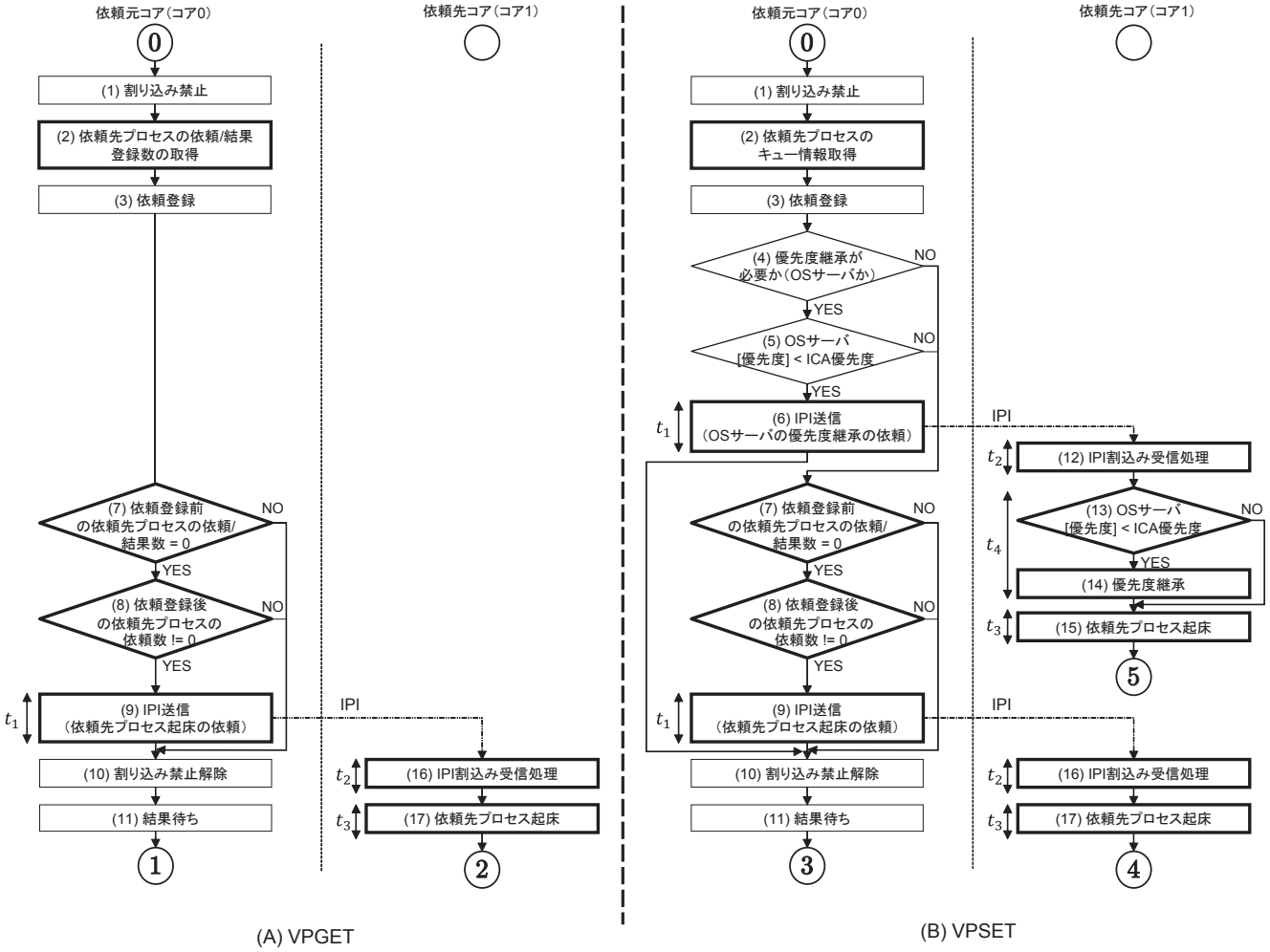

図 6 マルチコア環境における依頼登録の処理流れ

依頼先プロセスの依頼キュー/結果キューに登録されている依頼数/結果数を利用する (7)(8)．ここで，依頼登録前の依頼キューに繋がる依頼数と結果キューに繋がる結果数をそれぞれ $\boldsymbol{Req_{prev}}$，$\boldsymbol{Ret_{prev}}$，依頼登録後の依頼キューに繋がる依頼数を $\boldsymbol{Req_{currnent}}$ とする．以下の全ての条件を満たす場合，依頼先プロセスを起床させる．

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

（条件 1）を満たさない場合，他のプロセスの依頼登録において（条件 1）が満たされるため，他のプロセスが依頼先プロセスを起床している．（条件 2）を満たさない場合，他のプロセスの結果返却において結果返却時の（条件 1）が満たされるため，他のプロセスが依頼先プロセスを起床している．（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，依頼先プロセスは依頼を取得している．すなわち，依頼先プロセスは依頼処理を実行中である．

同様に結果返却時の依頼元プロセスの起床条件を以下に示す．なお，結果返却後の結果キューに繋がる結果数を $\boldsymbol{Ret_{currnent}}$ とする．以下の全ての条件を満たす場合，依頼元プロセスを起床させる．

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

4.2.3 優先度継承

VPSET は，依頼登録時に優先度継承を行い，依頼先プロセスの優先度を変更する．図 6(B) より，マルチコア環境のサーバプログラム間通信において，依頼元コアのスケジュール機能は，コア間通信によって，依頼先コアのスケジュール機能に優先度継承を依頼する (6)．しかし，4.1 節で述べたようにコア間通信のオーバヘッドは大きいため，コア間通信回数の削減が求められる．

ここで，依頼登録時に依頼先プロセスの起床依頼を行うことに着目する．依頼先プロセスの起床依頼は，優先度継承と同じくコア間通信を必要とする (9)．そこで，優先度継承が必要な場合，優先度継承とプロセス起床の 2 つの処理を 1 つのコア間通信によって依頼する (13)(14)(15)．優先

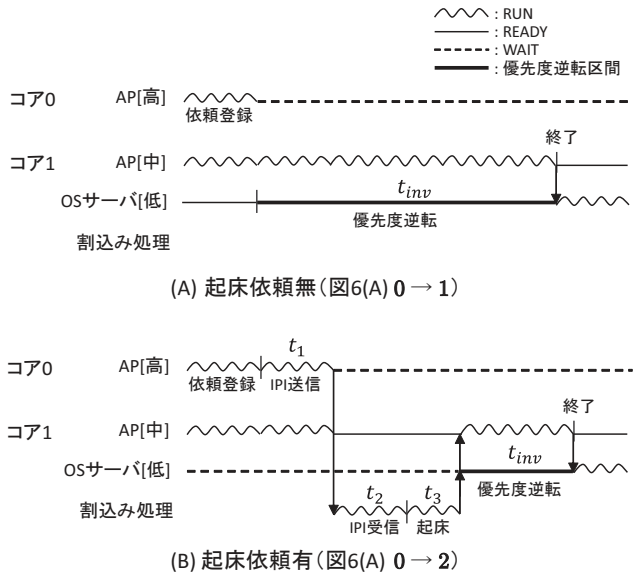

図 7 VPGET における処理の流れと時間

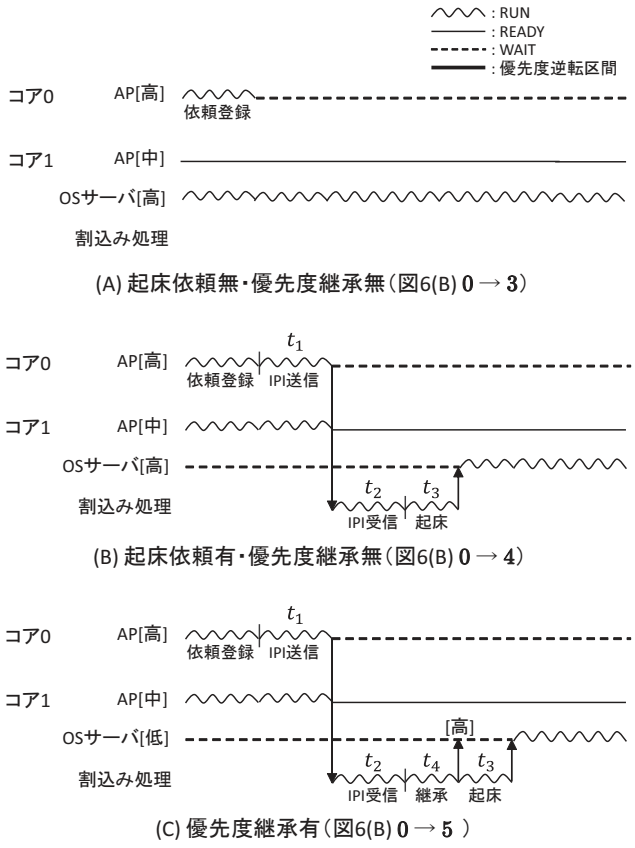

図 8 VPSET における処理の流れと時間

度継承が不要な場合（(4) または (5) が偽），依頼先プロセスが 4.2.2 項で示した起床条件を満たすか確認する (7)．起床条件を満たす場合，処理 (9) を行う．これにより，(6)(9) いずれかの 1 回のコア間通信となり，コア間通信回数を最大 1 回に抑制できる．

4.3 依頼先コアでの優先度継承

シングルコア環境とマルチコア環境の両方で VPSET は，依頼登録時に OS サーバの優先度が ICA 優先度より低いことを確認する（図 5(B)(4)），図 6(B)(5)）．これに加え，マルチコア環境の VPSET は，OS サーバの優先度が ICA 優先度より低い場合，コア間通信によって，依頼先コアのスケジュール機能に優先度継承を依頼する．このとき，依頼先コアでも再度，OS サーバの優先度が ICA 優先度より高いことを確認する（図 6(B)(13)）．これは，複数のコアが同時にコア間通信による優先度継承を行い，優先度逆転が生じる可能性があるためである．

5. 優先度逆転とオーバヘッド

5.1 処理の流れと時間

優先度逆転とオーバヘッドを分析するために，各制御法における処理の流れと時間を示す．VPGET（図 6(A)）における処理の流れと時間を**図 7** に示し，VPSET（図 6(B)）における処理の流れと時間を**図 8** に示す．なお，$\boldsymbol{t_{inv}}$ は優先度逆転時間である．各処理の流れと時間は，以下の場合である．

図 7(A)： 依頼先プロセスの起床依頼を伴わない場合（図 6(A) の **0 → 1**）

図 7(B)： 依頼先プロセスの起床依頼を伴う場合（図 6(A) の **0 → 2**）

図 8(A)： 依頼先プロセスの起床依頼を伴わない，かつ優先度継承を行わない場合（図 6(B) の **0 → 3**）

図 8(B)： 依頼先プロセスの起床依頼を伴う，かつ優先度継承を行わない場合（図 6(B) の **0 → 4**）

図 8(C)： 優先度継承を行う場合（図 6(B) の **0 → 5**）

5.2 比較

図 7 と図 8 を基に，各制御法の優先度逆転とオーバヘッドを表 2 に示す．以降，依頼元プロセスの優先度を $\boldsymbol{AP_r}$，依頼先 OS サーバの優先度を $\boldsymbol{OS}$，依頼先コア上の RUN/READY 状態の AP プロセスの優先度を $\boldsymbol{AP_o}$ とする．なお，図 7 の $\boldsymbol{t_{inv}}$ の値は以下となる．

$$\boldsymbol{t_{inv}} = \begin{cases} \boldsymbol{0} & (\text{if } \boldsymbol{AP_r} \leq \boldsymbol{OS} \text{ or } \boldsymbol{AP_o} < \boldsymbol{OS}) \\ \boldsymbol{> 0} & (\text{if } \boldsymbol{AP_r} > \boldsymbol{OS} \text{ and } \boldsymbol{AP_o} > \boldsymbol{OS}) \end{cases}$$

依頼先プロセスの起床依頼を伴わない，かつ優先度継承を必要としない場合（図 7(A) と図 8(A)）について比較する．優先度継承を必要としない場合，$\boldsymbol{AP_r \leq OS}$ であるため，$\boldsymbol{t_{inv} = 0}$ である．また，依頼登録のオーバヘッドは，図 7(A)，図 8(A) の両方で 0 である．したがって，図 7(A) と図 8(A) は，優先度逆転とオーバヘッドの両方で差がない．

表 2　VPGET と VPSET における優先度逆転とオーバヘッドの関係

起床	継承	AP_r と OS	AP_o と OS	VPGET		VPSET		t_{inv}+オーバヘッド	
				t_{inv}	オーバヘッド	t_{inv}	オーバヘッド	VPGET	VPSET
無	無	$AP_r \leq OS$		0	0	0	0	0	0
	有	$AP_r > OS$	$AP_o < OS$	0			$t_1+t_2+t_3$	0	$t_1+t_2+t_3$
			$AP_o > OS$	>0			$+t_4$	t_{inv}	$+t_4$
有	無	$AP_r \leq OS$		0	$t_1+t_2+t_3$		$t_1+t_2+t_3$	$t_1+t_2+t_3$	$t_1+t_2+t_3$
	有	$AP_r > OS$	$AP_o < OS$	0			$t_1+t_2+t_3$	$t_1+t_2+t_3$	$t_1+t_2+t_3$
			$AP_o > OS$	>0			$+t_4$	$t_1+t_2+t_3+t_{inv}$	$+t_4$

依頼先プロセスの起床依頼を伴わない，かつ優先度継承を必要とする場合（図 7(A) と図 8(C)）について比較する．優先度継承を必要とする場合，$AP_r > OS$ であり，t_{inv} の値は，AP_o と OS の関係によって決まる．$AP_o < OS$ の場合，$t_{inv} = 0$ である．したがって，図 7(A) と図 8(C) の優先度逆転に差はない．また，依頼登録のオーバヘッドは，図 7(A) において 0，図 8(C) において $t_1+t_2+t_3+t_4$ である．したがって，$AP_o < OS$ の場合，図 8(C) における OS サーバの実行開始は，図 7(A) に比べ，$t_1+t_2+t_3+t_4$ だけ遅れる．一方，$AP_o > OS$ の場合，$t_{inv} > 0$ である．したがって，図 7(A) の優先度逆転時間は，図 8(C) に比べ，t_{inv} だけ長い．また，t_{inv} と $t_1+t_2+t_3+t_4$ の大小関係によって，どちらの制御法の OS サーバの実行開始がより遅れるか異なる．$t_{inv} = (t_1+t_2+t_3+t_4)$ である場合，図 7(A) と図 8(C) において，OS サーバの実行開始に差がない．$t_{inv} > (t_1+t_2+t_3+t_4)$ である場合，図 7(A) は，図 8(C) に比べ，$t_{inv} - (t_1+t_2+t_3+t_4)$ だけ OS サーバの実行開始が遅れる．$t_{inv} < (t_1+t_2+t_3+t_4)$ である場合，図 8(C) は，図 7(A) に比べ，$(t_1+t_2+t_3+t_4) - t_{inv}$ だけ OS サーバの実行開始が遅れる．

依頼先プロセスの起床依頼を伴い，かつ優先度継承を必要としない場合（図 7(B) と図 8(B)）について比較する．優先度継承を必要としない場合，$AP_r \leq OS$ であるため，$t_{inv} = 0$ である．また，依頼登録のオーバヘッドは，図 7(B)，図 8(B) のどちらも $t_1+t_2+t_3$ である．したがって，図 7(B) と図 8(B) は，優先度逆転とオーバヘッドの両方で差がない．

依頼先プロセスの起床依頼を伴い，かつ優先度継承を必要とする場合（図 7(B) と図 8(C)）について比較する．優先度継承を必要とする場合，$AP_r > OS$ であり，t_{inv} の値は，AP_o と OS の関係によって決まる．$AP_o < OS$ の場合，$t_{inv} = 0$ である．したがって，図 7(B) と図 8(C) の優先度逆転に差はない．また，依頼登録のオーバヘッドは，図 7(B) において $t_1+t_2+t_3$，図 8(C) において $t_1+t_2+t_3+t_4$ である．したがって，$AP_o < OS$ の場合，図 8(C) における OS サーバの実行開始は，図 7(B) に比べ，t_4 だけ遅れる．一方，$AP_o > OS$ の場合，$t_{inv} > 0$ である．したがって，図 7(B) の優先度逆転時間は，図 8(C) に比べ，t_{inv} だけ長い．また，共通オーバヘッドである $t_1+t_2+t_3$ を除く，t_{inv} と t_4 の大小関係によって，どちらの制御法の OS サーバの実行開始がより遅れるか異なる．$t_{inv} = t_4$ である場合，図 7(B) と図 8(C) において，OS サーバの開始実行に差がない．$t_{inv} > t_4$ である場合，図 7(B) は，図 8(C) に比べ，$t_{inv} - t_4$ だけ OS サーバの実行開始が遅れる．$t_{inv} < t_4$ である場合，図 8(C) は，図 7(B) に比べ，$t_4 - t_{inv}$ だけ OS サーバの実行開始が遅れる．

5.3　まとめ

VPGET と VPSET について，優先度逆転とオーバヘッドの関係をまとめる．

優先度逆転は VPGET にて発生するが，VPSET では発生しない．VPGET における優先度逆転の発生条件は，$AP_r > OS$ and $AP_o > OS$ である．

オーバヘッドは，優先度継承を必要としない場合，両制御法で同じである．一方，優先度継承を必要とする場合，VPSET のオーバヘッドは，VPSET より以下の値だけ大きくなる．

（起床依頼無）$t_1+t_2+t_3+t_4$

（起床依頼有）t_4

したがって，優先度逆転とオーバヘッドによる OS サーバの実行開始は，優先度継承を必要としない場合，同じだけ遅れる．一方，優先度継承を必要とする場合の比較を以下に示す．

（1）（起床無）$AP_o < OS$

VPSET は，VPGET より $t_1+t_2+t_3+t_4$ だけ遅れる

（2）（起床無）$AP_o > OS$ and $t_1+t_2+t_3+t_4 < t_{inv}$

VPGET は，VPSET より $t_{inv} - (t_1+t_2+t_3+t_4)$ だけ遅れる

（3）（起床無）$AP_o > OS$ and $t_{inv} < t_1+t_2+t_3+t_4$

VPSET は，VPGET より $(t_1+t_2+t_3+t_4) - t_{inv}$ だけ遅れる

（4）（起床有）$AP_o < OS$

VPSET は，VPGET より t_4 だけ遅れる

（5）（起床有）$AP_o > OS$ and $t_4 < t_{inv}$

VPGET は，VPSET より $t_{inv} - t_4$ だけ遅れる

（6）（起床有）$AP_o > OS$ and $t_{inv} < t_4$

VPSET は，VPGET より $t_4 - t_{inv}$ だけ遅れる

6. 関連研究

本稿のスケジュール機構は，マルチコア環境において優先度逆転を抑制し，かつコア間通信回数を削減することでオーバヘッドを削減する．

文献 [9] では，Mach[4] の実時間性を向上させるために，実時間処理のためのプロセス間通信ポートを用意している．このポートには，メッセージのキューイング方法や優先度継承するかどうかを設定することができ，優先度逆転抑制法を実現できる．しかし，Mach は，マルチコア環境を想定していない．このため，Mach において提案手法を実現する場合，プロセス間通信をマルチコア環境に対応させ，コア毎に独立したスケジュール機能を配置させ，コア間通信を実現する必要がある．

文献 [10] では，共有資源の操作において発生するデッドロックの対処を参考に，代替資源を用意することで優先度逆転に対処できることを示している．例えば，OS サーバにおける優先度逆転は，優先度の異なる OS サーバを複数用意するか，OS サーバの処理実行中に他の依頼を実行可能にすることで対処できる．***AnT*** では，同一機能の OS サーバを複数用意していない．また，OS サーバの処理実行中に他の依頼を実行可能にすると処理が複雑化する．

文献 [11] では，シングルコア環境の L4[2] において，スレッドのコンテキストを 2 つに分割することでプロセス間通信のオーバヘッドを最小限に抑制し，かつ優先度継承と優先度上限プロトコルを実現している．この手法は，シングルコア環境のプロセス間通信においてコンテキストの利用を工夫しており，マルチコア環境において発生するコア間のプロセス間通信に適用することができない．このため，本稿にて提案するスケジュール機構と組み合わせるのは難しい．

文献 [12] では，分割スケジューリングにおいて優先度継承が無力であることを示し，ロック保持タスクにロック解放待ちタスクと同じコアで動作できる資格を付与する移譲優先度継承を提案している．この手法は，モノリシックカーネルである Linux に実現しており，優先度継承を排他制御によって実現している．ロックに適用している移譲優先度継承を OS サーバに対して実現するべきか否かは，OS サーバの移譲のコストと優先度逆転による影響によって異なる．

7. おわりに

本稿では，コア毎に独立したスケジュール機能を持つ ***AnT*** についてプロセス間の優先度逆転を抑制し，かつコア間通信回数を削減するスケジュール機構を提案した．

コア間通信回数を削減する手法を 2 つ述べた．1 つは，サーバプログラム間通信における依頼先プロセスの起床条件の追加である．これにより，VPGET と VPSET の両方のコア間通信を削減できる．もう 1 つは，サーバプログラム間通信における依頼先プロセスへの優先度継承時に依頼先プロセスを起床させることである．これにより，VPSET のコア間通信回数を最大 1 回に抑制することができる．なお，VPSET におけるマルチコア環境特有の問題として，依頼先コアでも優先度継承が必要か否かを判定することが必要であることを述べた．

また，VPGET と VPSET を比較し，優先度逆転発生時間とこれを抑制するために必要なオーバヘッドの関係を示した．

謝辞 本研究の一部は，科学研究費補助金若手研究（B）（課題番号：25730046）による．

参考文献

[1] Sha, L., Rajkumar, R., Lehoczky, J.P.: Priority inheritance protocols: an approach to real-time synchronization, Computers, IEEE Transactions on, 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] 鴨生 悠冬, 山内 利宏, 谷口 秀夫: ***AnT*** オペレーティングシステムにおける OS サーバ間通信での優先度継承による優先度逆転の抑制法, 情報処理学会研究報告, vol.2015-OS-133, no.15, pp.1-8 (2015.05).
[6] 鴨生 悠冬, 山内 利宏, 谷口 秀夫: ファイル読み込み処理における ***AnT*** オペレーティングシステムの OS サーバの優先度逆転抑制法の評価, 第 14 回情報科学技術フォーラム (FIT2015) 講演論文集, 第 1 分冊, pp.221-222 (2015.09).
[7] 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).
[8] 佐古田 健志, 枡田 圭祐, 井上 喜弘, 谷口 秀夫: マルチコア ***AnT*** における処理分散機能 , 情報処理学会研究報告, vol.2012-OS-122, no.5, pp.1–8 (2012).
[9] 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).
[10] Levine, G.: Priority Inversion with Fungible Resources, ACM SIGAda Ada Letters, Vol.31, No.2, pp.9–14 (2011).
[11] Steinberg, U., Wolter, J., Hartig, H.: Fast component interaction for real-time systems, Real-Time Systems 2005. (ECRTS 2005). Proceedings. 17th Euromicro Conference on, pp.89–97 (2005).
[12] 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).