システムソフトウェアと
オペレーティング・システム 83－2
（２０００．２．２４）

CEFOSオペレーティングシステムのスレッド管理機構

谷口秀夫† 日下部茂† 棚林拓也‡ 中山大士‡ 雨宮真人†
†九州大学 大学院システム情報科学研究科
‡九州大学 工学部

細粒度マルチスレッドをプロセッサ割り当ての基本単位とし、計算機における内部情報処理と通信の融合を目指した並列分散オペレーティングシステムCEFOS（Communication-Execution Fusion Operating System）について、スレッド管理機構を述べる。具体的には、スレッド管理の基本構造、スレッドスケジュール機構、プリエンプション機構、タイムスライス機構、および同期機構について述べる。

Thread Management Mechanisms of CEFOS

Hideo TANIGUCHI†, Shigeru KUSAKABE†, Takuya TANABAYASHI‡,
Hiroshi NAKAYAMA‡, and Makoto AMAMIYA†
† Graduate School of Information Science and Electrical Engineering, Kyushu University
‡ Faculty of Engineering, Kyushu University

This paper describes thread management mechanisms of an operating system for communication-execution fusion, CEFOS (Communication-Execution Fusion Operating System). CEFOS is based on a fine-grain multi-threading approach. The basic mechanism of thread management, scheduling mechanism, preemption mechanism, time-slice mechanism, and synchronization mechanism are described.

１．はじめに

既存の計算機のオペレーティングシステム（以降、OSと略す）は、計算機内部の情報処理の高速化を目指して構築されているため、通信制御は通信を陽に意識したものになっている。しかし、計算機が単独で動作することは少なくなり、計算機を結合した環境での処理が増加している。一方、計算機ハードウェアの性能向上は目覚しいものの単一の計算機（プロセッサ）での実時間処理には限界があるため、並列分散による実時間処理システムを構築できる必要がある。そこで、我々は、計算機内部の情報処理と通信制御の融合を目指し、並列分散オペレーティングシステムCEFOS（Communication-Execution Fusion OS）を開発している[1]。

CEFOSにおける内部情報処理と通信制御の融合は、処理時間と機能提供インタフェースの両面で実現する。内部情報処理とは、一つの計算機内で行われる処理であり、通信機能を用いず、データの演算を基本とする処理である。通信制御とは、一つ以上の計算機において、プロセッサの割り当て単位であるスレッド間のデータ授受のために行う制御であり、通信パスの概念で制御される。本OS核では、機能提供インタフェースとして、通信制御を内部情報処理へ融合する。具体的には、スレッド間のデータ授受は、通信パスを意識しない形で行えるようにする。さらに、処理時間の上でも内部情報処理と通信

制御を融合する。つまり、一つのスレッド内に閉じた処理、一つの計算機内に閉じた処理、および二つ以上の計算機にまたがった処理について、処理時間の差異をあまり感じさせない環境を実現する。特に、一つの計算機内に閉じた処理と二つ以上の計算機にまたがった処理の処理時間の差異を短くすることを目標とする。

本稿では、上記実現の中核となるスレッド管理機構を述べる。具体的には、スレッド管理の基本構造、スレッドスケジュール機構、プリエンプション機構、タイムスライス機構、および同期機構について述べる。

2．システム規模とスレッド設計方針

システムは、複数の計算機を高速な通信路で結んで構築される。ソフトウェアは、ジョブとプロセスおよびスレッドからなる。これらの様子を図1に示し、以下に説明する。

（1）システム内に数個のジョブが存在する。
（2）ジョブは、1個以上からのプロセス（数十個）からなり、サービスを提供する。
（3）プロセスは、1個以上のスレッド（数百個）からなり、一つの計算機で実行される。つまり、一つのプロセスを構成するスレッド群は一つの計算機で実行される。プロセスには、ユーザモードで走行するユーザプロセスとカーネルモードで走行するカーネルプロセスの両方が存在する。
（4）スレッドは、停止することなく連続して走行し、走行最長時間は1ミリ秒である。

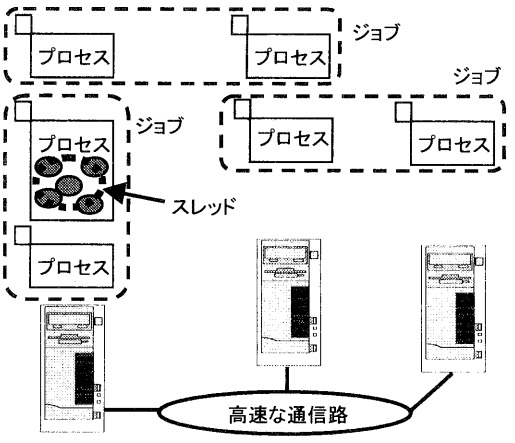

図1 システム概要

スレッドの設計は以下の方針で進めた。

（方針1）軽量スレッドの実現

多数の細粒度なスレッドが同時走行するため、スレッドの軽量化を図る。具体的には、スレッド切り替えの回数を削減するとともにスレッド切り替えの高速化を図る。

（方針2）公平な処理環境の実現

各ジョブが提供するサービスを偏りなく実現するために、プロセス間だけではなくスレッド間を含めた公平な処理環境を実現する。具体的には、タイムスライス機能とプリエンプション機能を実現する。

3．スレッド管理機構

3．1　プロセスとOS核の処理連携

同一プロセス内に複数のスレッドが走行する環境の実現においては、プロセスとOS核の処理の連携を効率的に行うことが非常に重要である。

スレッドを管理する方法は、大きく二つに分類できる[2]。一つはOS核でユーザプロセス内のスレッドを管理する方法（カーネルレベル法と呼ぶ）であり、もう一つはライブラリでユーザプロセス内のスレッドを管理する方法（ユーザレベル法と呼ぶ）である。カーネルレベル法は、多くのOSで実現されており、プロセッサの稼働率の向上や公平な処理環境の実現および複数プロセッサ環境での負荷分散が可能である。ただし、スレッド切り替え時にユーザモードとカーネルモードの間のモード変更が多発する問題がある。ユーザレベル法は、スレッド機能を有しないOS環境で実現されていることが多く、各プロセス内の各スレッド管理機構が同一プロセス内のスレッドを管理している。このため、同一プロセス内のスレッド間では、スレッド切り替え時にモード変更が不要であり、同期処理は保護不要なため高速である。このように、同一プロセス内のスレッド間での処理は高速であるものの、別プロセスのスレッド間での処理には問題がある。OSはプロセスを管理しスレッドを管理しないため、スレッド間での実行の優先度逆転が起こりやすく、また公平な処理環境の実現や複数プロセッサ環境での負荷分散は困難である。このため、文献[2]では、ユーザレベル法を基に上位呼び出し（upcall）機能によりプ

ロセスとOS核の処理連携を強め、ユーザレベル法の問題点に対処している。

従来のOSにおいて、プロセスとOS核の処理連携には、二つの方法が使われている。一つは、プロセスがOS核に処理を依頼するシステムコールである。システムコールは、SVC命令やtrap命令を利用し、プロセスの処理要求をOS核へ通知し、この通知に基づいてOS核は処理を行っている。もう一つは、OS核がプロセスに事象を通知する上位呼び出しである。上位呼び出しは、プロセス内の特定処理を呼び出すもので、マイクロOS核において利用されている。この二つの方法は、いずれも処理オーバヘッドが大きい欠点を持つ。例えば、システムコールは、先に述べたように、モード変更が多発するためカーネルレベル法の欠点になっている。また、上位呼び出しは、マイクロOS核における処理オーバヘッドとして問題になっており、様々な軽減化が研究されている。

そこで、ここでは、要求・情報表示（DRD: Display Request and Data）機能を提案する。DRD機能の特徴を以下に示す。

（特徴1）プロセスとOS核は、共有するメモリ域（CA: Common Area）を保有する。

（特徴2）相互に必要な情報として、処理の要求（request）や処理状態の内容（data）をCAに表示（display）する。

（特徴3）プロセスとOS核は、表示された情報を利用した処理の連携を行う。

さらに、以下のことも可能である。

（特徴4）必要に応じ、プロセスのOS核呼び出しやOS核のプロセス呼び出しを行う。

このDRD機能により、処理オーバヘッドが少ないプロセスとOS核の処理連携が可能になる。具体的には、以下の処理を行う。

（1）プロセスは、OS核への処理依頼内容やOS核が必要とするプロセス内部情報をCAに表示する。

（2）OS核は、プロセスへの事象内容やプロセスが必要とするOS核内部情報をCAに表示する。

（3）プロセスとOS核は、適当な契機で、CAに表示された情報を基に処理を行う。

さらに、必要に応じ、以下の処理を行う。

（4）CAの大きさ不足や緊急を要する場合、相手（OS核またはプロセス）を呼び出す（システムコールまたは上位呼び出し）。

このように、プロセスとOS核の処理連携の手段は、従来の手法の通知ではなく表示を基本とする。つまり、処理は、依頼元主導ではなく受け手主導で行われる。これにより、受け手側は、現在実行中の処理の中断を避けることができる。従来の手法は、依頼元主導であるため、受け手側の処理中断のために行う処理オーバヘッドが大きいのである。また、このDRD機能は、相互の呼び出しを最小限にできる。例えば、プロセスのOS核呼び出しはプロセス内に走行可能スレッドがなくなったときのみにすることが可能であり、OS核のプロセス呼び出しはゼロとすることが可能である。

CEFOSでは、DRD機能を用いて、カーネルレベル法とユーザレベル法の統合した方式を実現し、各方法の問題点を解決する。具体的には、以下の特徴を持つ。

（1）統合システムコール

（2）スケジューラの連携

統合システムコールは、ユーザプロセス内スレッドからのOS核呼び出し回数を削減するものである。スケジューラの連携としては、スレッド実行の優先度逆転期間を短縮するプリエンプション機能のために、プロセス内スレッドスケジューラからOS核内プロセススケジューラへ現最高スレッド優先度を表示し、OS核内プロセススケジューラからプロセス内スレッドスケジューラへ制御移行要求を表示する。これらの詳細については後述する。

3．2　基本構造と機能仕様

CEFOSには、OS核とプログラムを共有するカーネルプロセスおよび共有しないユーザプロセスがある。カーネルプロセスはカーネルモードで走行し、ユーザプロセスはユーザモードで走行する。以降、両者を意味する場合は、単にプロセスと記述する。

プロセスとスレッドの関係について構造面から記述したものを図2に示し、以下にスレッド管理の基本構造と機能仕様を説明する。

（1）プロセスは、一つの仮想記憶空間を持ち、資源操作の単位であり、プロセス優先度を持つ。

（2）プロセスの連続走行時間は、制限され、必要に応じタイムスライスされる。

（3）　プロセスの実行順序は、プロセス優先度に基づき制御され、必要に応じプリエンプションされる。

（4）スレッドは、1プロセス内に一つ以上存在し、プロセッサの割り当て単位であり、スレッド優先度を持つ。なお、同一プロセス内の各スレッドはスタックを共有し、スレッド毎にはスタックを持たない。

（5）スレッドの連続走行時間は、制限されない。つまり、スレッドは無停止で走行する。ただし、スレッドが属するプロセスのタイムスライスにより、スレッドの実行が中断されることはあり得る。

（6）スレッドの実行順序は、同一プロセス優先度を持つプロセス内のスレッドの間において、スレッド優先度に基づき制御される。スレッド優先度における実行の優先度逆転が発生した場合、走行中スレッドの終了を経て実行の優先度逆転は解消される。このように、実行の優先度逆転が生じた直後ではなく、走行中スレッドの終了を経て実行の優先度逆転を解消する。これを、準プリエンプション機能と名付ける。

上記項目（5）の「スレッドは無停止」により、スレッド切り替えの回数を削減できる。また、上記項目（4）の「スレッド毎にスタックなし」、および統合システムコールによるスレッド切り替え時のモード変更回数の削減により、スレッド切り替えの高速化を図る。これらにより、軽量スレッドの実現（方針1）を目指す。上記項目（2）と（3）の「プロセスのタイムスライス機能とプリエンプション機能」、および上記項目（6）の「スレッドの準プリエンプション機能」により、公平な処理環境の実現（方針2）を目指す。なお、準プリエンプション機能は、スレッド切り替えの回数も削減している。

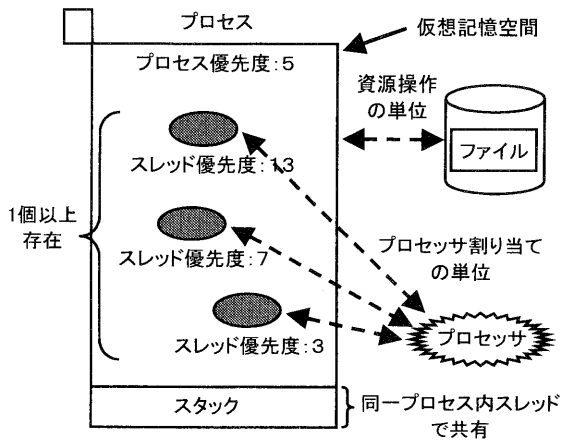

図2　プロセスとスレッドの基本構造

3．3　スレッドスケジュール機構

3．3．1　基本機構

スケジュールの基本機構を図3に示し、以下に説明する。

（1）スレッドの終了は、単なる終了またはOS核の呼び出しである。終了処理は、単なる終了の場合はスレッドスケジューラを呼び出し、OS核の呼び出しの場合はシステムコール一括処理を呼び出す。

（2）スレッドスケジューラは、終了処理またはシステムコール一括処理から呼ばれ、同期処理の情報に基づき、走行可能なスレッドの中からスレッド優先度が最も高いスレッドを選定し走行させる。また、スケジューラの連携として、RUN状態またはREADY状態のスレッドの中でスレッド優先度が最も高いスレッドのスレッド優先度を現最高スレッド優先度として表示する。さらに、OS核から制御移行要求を受けた場合には、システムコール一括処理を呼び出し、OS核への制御移行を促す。

（3）システムコール一括処理は、終了処理からのOS核呼び出し要求を受けた場合、システムコールをまとめる処理を行い、スレッドスケジューラを呼び出す。また、特定数のシステムコール要求がまとまると、統合システムコールの発行をOS核呼び出しに依頼する。さらに、スレッドスケジューラからOS核への制御移行を促された場合、それまでにまとめたシステムコール要求を統合システムコールとしてOS核呼び出しに依頼する。

（4）プロセススケジューラは、各シス

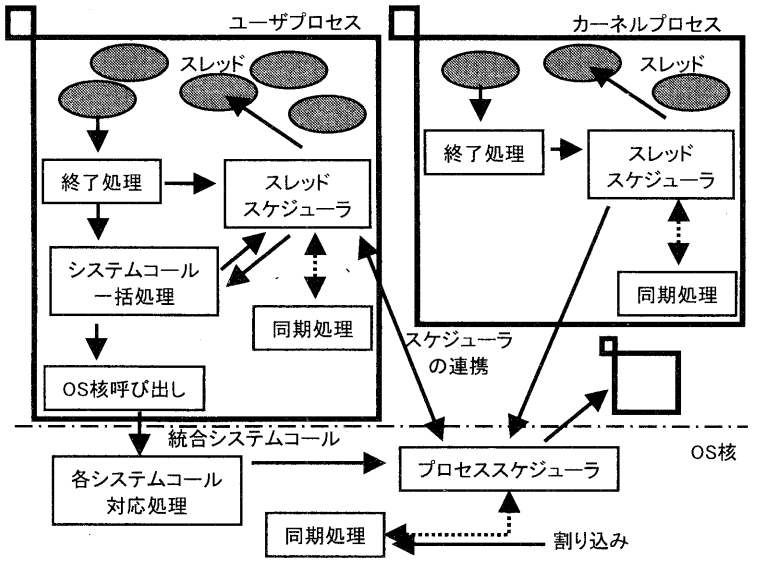

図3 スケジュール基本機構

テムコール対応処理や割り込み処理の終了時点で呼び出され、同期処理の情報に基づき、走行可能なプロセスの中からプロセス優先度が最も高いプロセスを選定し走行させる。また、スケジューラの連携により各スレッドスケジューラから現最高スレッド優先度の情報を取得し、必要に応じ制御移行要求を表示する。

3．3．2　タイムスライス機構

タイムスライス機構を図4に示す。プロセススケジューラは、タイマ割り込みを利用してプロセスの連続走行時間を計測し、連続走行時間が一定時間（タイムスライス間隔）以上になると、強制的にプロセッサを取り上げ、プロセスを再スケジュールする。なお、連続走行時間とは、RUN状態開始から次のWAIT状態開始までの時間からREADY状態の時間を除いた時間であり、RUN状態の途中でプリエンプションによりプロセッサを横取りされてもゼロクリアされない。

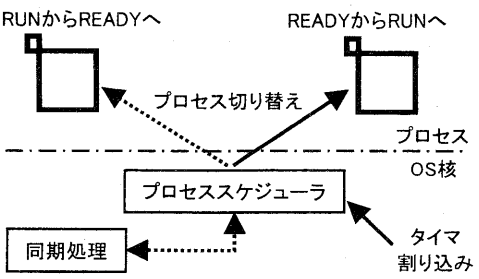

図4 タイムスライス機構

3．3．3　プリエンプション機構

プロセススケジューラは、プロセス優先度に基づきプロセスの実行の優先度逆転を検出し、直ちにプリエンプションの処理を行う。

スレッドの準プリエンプション機能について、プリエンプション機構を図5に示し、以下に説明する。

（1）　各プロセス内のスレッドスケジューラは、RUN状態またはREADY状態のスレッドの中でスレッド優先度が最も高いスレッドのスレッド優先度をOS核へ表示する。

（2）　プロセススケジューラは、走行可能なプロセスの中でプロセス優先度が最も高いプロセスが二つ以上存在する場合、当該プロセス内の現最高スレッド優先度が最も高いスレッドを有するプロセスを選定し、そのプロセスのスレッドスケジューラへ制御移行要求を行う。

（3）　制御移行要求を表示されたスレッドスケジューラは、スレッド切り替えの契機において、OS核へ制御を移行する。

上記のように、準プリエンプション機能では、プロセス内のスレッドスケジューラとOS核内のプロセススケジューラが連携し、スレッド優先度に基づきプロセスのプリエンプションを行う。この場合、実行の優先度逆転が起こり得る最長時間は、タイムスライス間隔または1スレッド走行最長時間である。タイムスライス間隔は1スレッド走行最長時間より長く、1スレッド走行最長時間は1ミリ秒であるため、実行の優先度逆転が起こり得る平均時間は500μ秒である。

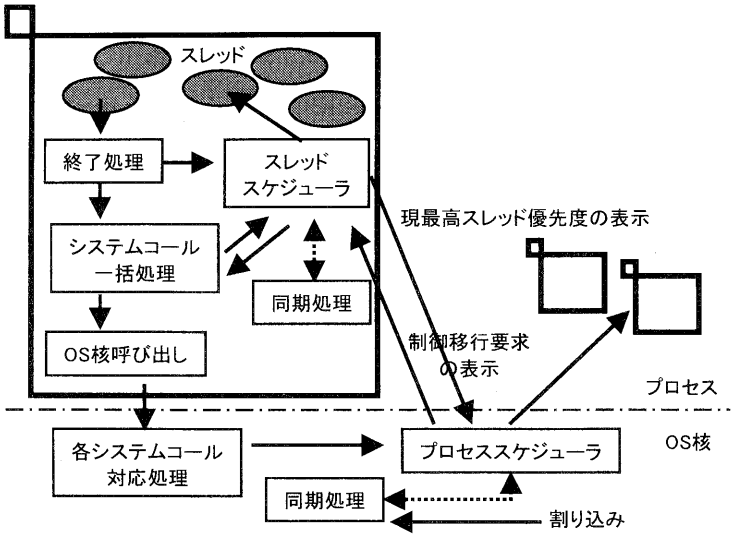

図5 準プリエンプション機構

3．4 同期機構

同期機構の基本方式を以下に説明する。同一プロセス内スレッド間においては、

（1）同期管理表をスレッド間で共有してモード変更を防ぎ、処理の高速化を図る

（2）同期管理表は別プロセスから隔離し、処理の保護を図る

とした。また、別プロセスのスレッド間では、

（3）同期管理表をプロセス間で共有してモード変更を防ぎ、処理の高速化を図る

（4）OS 核が計算機間通信を行い同期を通知し、スレッドは別計算機のスレッドを意識しない

とした。同期機構を図6に示す。同期管理表は、同一プロセス内スレッドからの同期用と別プロセスのスレッドからの同期用からなり、1スレッドのための同期エントリはこの一対からなる。

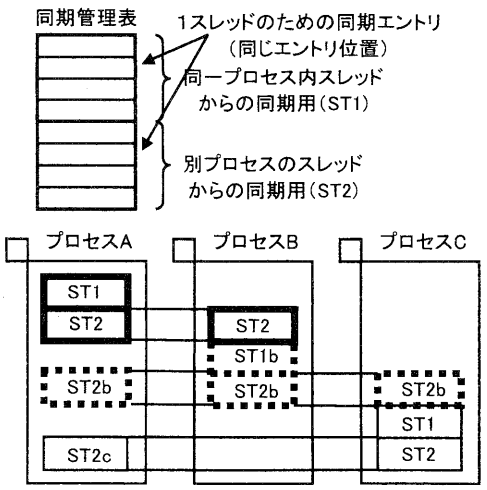

図6 同期機構

4．おわりに

細粒度マルチスレッドをプロセッサ割り当ての基本単位とし、計算機における内部情報処理と通信の融合を目指した並列分散オペレーティングシステム CEFOS について、スレッド管理機構を述べた。

スレッド管理の基本方針は、軽量スレッドの実現と公平な処理環境の実現である。前者のためにスレッド切り替えの回数削減と高速化、後者のためにプロセスのタイムスライス機能とプリエンプション機能の実現、およびスレッドの準プリエンプション機能の実現を図る。さらに、プロセスと OS 核の処理連携において、処理オーバヘッドが小さい要求･情報表示（DRD）機能を提案した。CEFOS では、DRD 機能を用いて、統合システムコールとスケジューラの連携を実現している。

本研究は、通信・放送機構の創造的情報通信技術研究開発推進制度に係る研究開発課題「次世代型インテリジェント・マルチメディア情報通信網の基盤技術に関する研究」による。

参考文献

[1] 日下部茂，富安洋史，村上和彰，谷口秀夫，雨宮真人：”並列分散オペレーティングシステム CEFOS(Communication-Execution Fusion OS)”，信学技報，Vol.99, No.251, pp.25-32 (1999).

[2] Thomas E. A., Brian N.B., Edward D. L., and Henry M. L., “Scheduler Activations: Effective kernel Support for the User-Level Management of Parallelism,” Proc. of the 13th ACM Symp. on OS Principles, pp-95-109 (1991).