Vol. 36 No. 8　　情　報　処　理　　Aug. 1995

解　説

日本におけるオペレーティングシステム研究の動向

1.5 並列および分散応用を対象とした分散型OS ReSC†

新城　靖††　清木　康†††

1. はじめに

共有メモリ型マルチプロセッサや単一プロセッサが高速ネットワークで結合されたハードウェア環境が広く普及してきた．そのような環境では，従来の逐次応用プログラムに加えて，並列応用プログラムや分散応用プログラムが利用されるようになってきている．ReSCは，そのようなハードウェア環境を，逐次から並列/分散までさまざまな応用プログラムで共有するための分散型オペレーティングシステムである[4),6)~10)]．

ReSCシステムの設計目標は，各応用プログラムの要求に応じてシステムの機能を利用可能にすることである．この目標を達成するために，ReSCでは，2種類の調和を実現している．この論文では，まず調和の考え方を，調和が実現されていない状況（競合している状況）と対比することで説明する．そして，ReSCシステムにおける具体的な調和の実現方法について説明する．

2. 競合と調和

オペレーティングシステムの主要な仕事の1つに，資源割当ての最適化がある．ここで資源割当てとは，高速処理，スループットの改善，利用者間の公平などを実現することを目的として，CPU時間やメモリなどの計算機資源をいくつかの主体に配分することである．例として，CPUのスケジューリング，ディスク・キャッシュの置換え，仮想記憶のページ置換えがあげられる．

近年，数多くの並列応用プログラム，および，分散応用プログラムが開発されてきた．並列/分散応用プログラムの特徴は，内部に複数の処理単位を含んでいる点，および，自ら資源割当ての最適化を行う点にある．

単一プログラミング・システムとして開発されてきた並列/分散応用プログラムを，多重プログラミング環境，すなわち，オペレーティングシステム上で動作させる場合，資源割当てに関してさまざまな問題が生じる．たとえば，システムのCPUスケジューラは，複数の利用者間の公平性を実現するために，各プロセスに対して均等にCPU資源を割り当てようとする．これに対して，個々の並列応用プログラムは，自分自身の高速処理を目的として，内部の処理単位の中で特定のものにより多くのCPU資源を割り当てようとする．このように，オペレーティングシステムの資源割当てを行う部分と並列/分散応用プログラム内の資源割当てを行う部分の方針に食違いが生じる．この状況を，**「システムと並列/分散応用プログラムが競合している」**と呼ぶことにする．

上記の競合は，別の局面では，逐次応用プログラムと並列/分散応用プログラムの間の競合となって現れる．たとえば，逐次応用プログラムは，内部に資源割当てを行う部分を含んでいない．そのため，（分散型）オペレーティングシステムの資源割当て機能を利用して，負荷が小さい計算機上で実行されることを望む．一方，並列/分散応用プログラムは，内部にCPUやディスクの資源割当てを行う部分を持っている．そのため，自らプロセスやデータの位置を制御することを望む．このようにシステムのある機能が，逐次プログラムにとっては必要であるが，並列/分散応用プログラムにとっては必要ではないことがある．この状況を，**「逐次応用プログラムと並列/分散応用プログラムが競合している」**と呼ぶことにする．

† The ReSC Distributed Operating System for Parallel and Distributed Applications by Yasushi SHINJO (Department of Information Engineering, University of the Ryukyus) and Yasushi KIYOKI (Institute of Information Sciences and Electronics, University of Tsukuba).

†† 琉球大学情報工学科
††† 筑波大学電子・情報工学系

ReSCシステムの研究目的は，上の2つの競合を解消し，次の2種類の調和（harmonization）を実現することに集約されている[9]．

（1） **システムと並列/分散応用プログラムの調和：** オペレーティングシステムは，並列/分散応用プログラムと競合することなく，それらに対して，高速処理や高信頼性を実現するためにスケジューリングやマッピングを制御する機能を提供する．

（2） **逐次応用プログラムと並列/分散応用プログラムの調和：** 1つの並列/分散ハードウェア環境の中で，並列/分散応用プログラムと逐次応用プログラムを同時に利用することを可能にする．

図-1 に，ReSCシステムの構成と調和の実現が示されている．システムは，分散カーネル，外部サーバ，並列シェル，そして，軽量プロセス・ライブラリ（並列応用プログラムにリンクされる）から構成されている．このように，システムの機能が階段状に提供されている点に特徴がある．従来のシステムでは，主に逐次応用プログラムを対象として設計されていたため，左側の高いレベルの視点（分散透明性）しか提供されなかった[1),11)]．このため，システムと並列/分散応用プログラムの機能に重複があった．その重複している部分で，競合が生じる．ReSCでは，右側の低いレベルの視点も提供されている．これは，並列/分散応用プログラムは，システムに干渉されることなく資源割当ての最適化を行うことが可能であることを示している．同時に，逐次プログラムにとっては，外部サーバと並列シェルを利用することにより，従来の分散型オペレーティングシステムと同じ視点が実現されていることを示している．

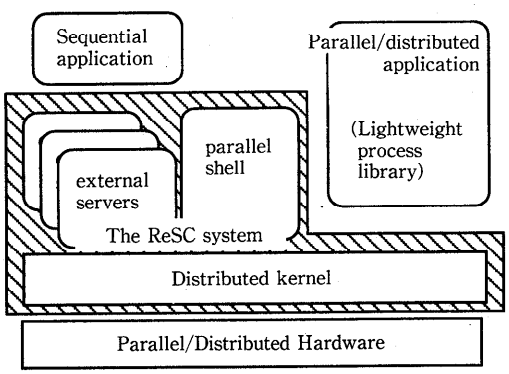


図-1 ReSCシステムにおける調和の実現

ReSCのカーネルには，プログラムの保護，プロセス間通信，および，複数の応用プログラム間の公平性を実現するための資源割当てモジュールが含まれている．各応用プログラムは，カーネルより割り当てられた資源を自由に利用することができる．

3. 軽量プロセス

多重プログラミングのオペレーティングシステムにおいて，プロセスという言葉は，資源割当てと保護の単位を指すものとして用いられる．軽量プロセス（lightweight process），あるいは，スレッド（thread）は，資源割当てと保護の単位としてのプロセスとは異なる，1つの応用プログラム内部の並列処理の単位としてのプロセスである．軽量プロセスは，単一プログラミングの並列/分散応用プログラムにおけるプロセスと対応する．

ReSCでは，マイクロプロセスと仮想プロセッサという概念を用いて軽量プロセスを実現している[10)]．**マイクロプロセス（microprocess）**は，並列処理の単位としてのプロセス，すなわち，軽量プロセスである．各マイクロプロセスは，重量プロセスの中にあり，独立のプログラム・カウンタとスタックを備えている．マイクロプロセスの生成・消滅，マイクロプロセス間の同期・通信など，すべての操作は，利用者レベルにおいて行われる．オペレーティングシステムのカーネルは，いっさい介在しない．したがって，それらの操作が高速に実行される．このことは，軽量プロセスをサブルーチンと同等に扱い，保護の機能を設けていないことによる．

仮想プロセッサ（virtual processor）は，カーネルが1つの利用者プロセスに対して複数の実プロセッサを割り当てるためのエントリである．各仮想プロセッサは，1つの利用者プロセスと対応しており，その利用者プロセスに割り当てられた資源，利用者識別子，アクセス権，プロセスの優先順位などを共有している．並列応用プログラムは，複数の仮想プロセッサを生成する．カーネルは，それらの仮想プロセッサに対して複数の実プロセッサを割り当てる．それらの仮想プロセッ

サは，複数のマイクロプロセスを並列に実行する．

マイクロプロセスは，層構造を持つライブラリにより実現されている[10]．これにより，さまざまなハードウェア・アーキテクチャの違い，ソフトウェア環境の違いを吸収することが容易になっている．さらに層構造のライブラリにより，システムと並列応用プログラムの調和が実現されている．すなわち，並列応用プログラムは，それぞれの要求に従ってライブラリを部分的に利用することができる．さらに，応用固有のスケジューリングを実現するために，ライブラリの一部を利用してスケジューラを構築したり，ライブラリ内のスケジューラを自分自身で定義した同じ外部仕様を持つものと入れ換えることが可能になっている．

共有メモリ型マルチプロセッサにおいて，仮想プロセッサを提供するカーネルが実現されている[6]．そのカーネルの構成法の特徴は，カーネル自身を1つの並列プログラムとしてとらえ，カーネルをカーネル内の軽量プロセスの集合として構築している点にある．

4. マッピング・コントローラと並列シェル

ネットワーク上に分散した複数のサイト（ネットワーク上のプロセッサ）を用いて並列処理や分散処理を行う応用プログラムにおいて，プロセスやデータをサイトへ割り当てることは，非常に重要である．サイトへ割り当てることをマッピング，割当ての方針を決定するモジュールをマッピング・コントローラ（mapping controller）と呼ぶ[9]．マッピング・コントローラは，システムの資源の利用状況を観察し，プロセスやデータのサイトへの割当てを最適化する．たとえば，CPU処理が中心のプログラムのマッピング・コントローラは，CPUの能力が高く負荷が軽いサイトを探し，プロセスを生成する．

ReSCでは，各並列/分散応用プログラムが固有のマッピング・コントローラを構築している[9]．ReSCシステムは，それらのマッピング・コントローラに対して資源割当ての最適化に必要な情報を積極的に提供し，逆に干渉することはしていない．この方針の利点として，各応用プログラムがその応用の性質を利用するようなマッピング・コントローラを容易に構築することができることがあげられる．さらに，マッピングの制御に，システム定義の汎用の記述ではなく，応用固有の記述を用いることも可能である．これらの利点は，マッピングに関して，システムと並列/分散プログラム間の調和が実現されることを意味している．

マッピング・コントローラを持っていない応用プログラムに対しては，並列シェル（parallel shell）が提供されている．並列シェルは，協調して動作する複数の逐次プログラム（コマンド）のためのコマンド・インタプリタである．並列シェルは，ReSCシステムの構成要素であると同時に，カーネル上で動作する1つの並列応用プログラムでもある．並列シェルは，内部にマッピング・コントローラを持たない（逐次）応用プログラムに代わり，マッピング・コントローラとしての役割を果たしている．このことは，逐次プログラムと並列/分散プログラムの間の調和が実現されることを意味する．並列シェルのプロトタイプは，イーサネットにより結合されたワークステーション上に実現されている[9]．

5. オブジェクトの堆積

ReSCカーネルが提供している基本的な機能は，オーバヘッドが小さく，効率を重視する並列/分散応用プログラムに適している．しかしながら，逐次応用プログラムの高度な要求をすべて満足させることはできない．たとえば，ファイルのキャッシングやファイルの複製（replication），オブジェクトの移動（migration）といった機能を提供していない．ReSCでは，このような高度な機能は，カーネル外のサーバにおいてオブジェクトの堆積というモデルを利用することで提供される．オブジェクトの堆積により，逐次応用プログラムと並列/分散応用プログラムの調和が実現される．同時にオブジェクトの堆積は，逐次応用プログラムを対象とした分散型オペレーティングシステムを構成するための方法と見ることもできる．

オブジェクトの堆積（object-stacking）は，object-basedシステムにおいて，複数のオブジェクトが提供する機能を統合するためのモデルである[8]．オブジェクトの堆積では，個々のサーバは，単純な機能を持つオブジェクトを提供する．

それらのオブジェクトは，オブジェクト識別子と遠隔手続き呼出しにより結合され，オブジェクトの層（堆積）を形成し，それらの機能を合せ持つようなオブジェクトとなる．

オブジェクトの堆積の基本的な考え方は，**一様なインタフェース**を持つオブジェクトを**積み重ねる**ことである．インタフェースとは，オブジェクトが受け付けることができる手続きの集合である．積み重ねるとは，上位層のオブジェクトが下位層のオブジェクトの識別子を保持し，かつ上位層のオブジェクトが自分自身の機能を実現するために下位層のオブジェクトを呼び出すことである．たとえばフィルタ機能を持つファイル・オブジェクトを積み重ねることを考える．フィルタの例としては，文字列の置換え，選択，ソートがあげられる．UNIXのパイプと同様に，積み重ねられたオブジェクトの機能は，統合され利用される．

フィルタ機能を持つファイル・オブジェクトの実現においては，ストリームを入出力とする外部のフィルタ型プログラムが利用されている[4]．これにより，サーバ自身を書き換えることなく，新たなフィルタ機能を追加することが可能となっている．この実現においては，ファイル単位のキャッシングが利用されている．キャッシングの利用により，行単位やファイル単位のフィルタの実現が容易になり，同時に，性能も改善されている．

6. おわりに

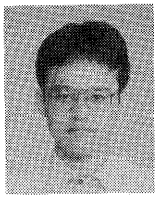
この論文では，ReSCシステムの設計方針，軽量プロセス機能，マッピング・コントローラに対する態度，オブジェクトの堆積によるファイル・システムについて説明した．応用プログラムの要求に従ってシステムの見え方を変えるという考え方は，最近の並列計算機用のオペレーティングシステムにも見られる[2),3)]．1994年9月に開催されたACM SIGOPSのヨーロッパ・ワークショップでは，“Matching Operating Systems to Application Needs”というテーマが掲げられた．今後とも調和という概念は，透明性に代わりオペレーティングシステム設計の目標となるであろう．

ReSCシステムでは，2つの調和を実現するために，システムの機能を選択的に利用可能な形（ライブラリ，シェル，外部サーバ）で提供するという方法が用いられていた．応用プログラム間の相互作用は，カーネルを通じた間接的なものとなっていた．今後は，システムや応用プログラムがより密に相互作用を行うことでそれらの間の調和を実現する方法の研究が期待される．

参考文献

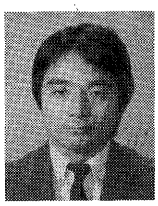
1) Goscinski, A.: Distributed Operating Systems — The Logical Design, Addison — Wesley (1991).
2) 福田：並列オペレーティング・システム，情報処理，Vol. 34, No. 9, pp. 1139-1149 (1993).
3) 萩野，徳田，斉藤，松岡，米澤，砂原，多田，柴山：超並列計算機のためのオペレーティング・システムの構想，情報処理学会オペレーティング・システム研究会，93-OS-58-4, pp.25-32 (Mar. 1993).
4) 苅部，新城，清木：オブジェクト堆積モデルに基づくファイル・サーバの実現，情報処理学会オペレーティング・システム研究会，93-OS-58-3, pp. 9-16 (Mar. 1993).
5) 前川，所，清水（編）：分散型オペレーティングシステム，共立出版 (1991).
6) 新城，清木：仮想プロセッサを提供するオペレーティング・システム・カーネルの構成法，情報処理学会論文誌，Vol. 34, No. 3, pp. 478-488 (Mar. 1993).
7) 新城：並列分散応用プログラムを対象としたオペレーティング・システムの研究，筑波大学工学研究科博士論文 (1992).
8) Shinjo, Y. and Kiyoki, Y.: The Object-Stacking Model for Structuring Object-Based Systems, Proc. 2nd International Workshop on Object Orientation in Operating Systems (IWOOOS' 92), pp. 328-340 (1992).
9) Shinjo, Y. and Kiyoki, Y.: Harmonizing a Distributed Operating System with Parallel and Distributed Applications, Proc. 1st International Symposium on High Performance Distributed Computing (HPDC-1), pp. 114-123 (1992).
10) 新城，清木：並列プログラムを対象とした軽量プロセスの実現方式，情報処理学会論文誌，Vol. 33, No. 1, pp. 64-73 (Jan. 1992).
11) Tanenbaum, A. and Renesse, R.: Distributed Operating Systems, ACM Computing Surveys, Vol. 17, No. 4, pp. 419-470 (1985).

（平成6年7月4日受付）



新城　靖（正会員）

1965年生．1988年筑波大学第三学群情報学類卒業，1993年同大学院博士課程工学研究科修了．工学博士．同年より，琉球大学工学部助手．オペレーティング・システム，データベース・システム，並列処理，分散処理，オブジェクト指向，インターネットに興味を持つ．ACM，IEEE，日本ソフトウェア科学会，電子情報通信学会各会員．



清木　康（正会員）

1956年生．1978年慶応義塾大学工学部電気工学科卒業．1983年同大学院工学研究科博士課程修了．工学博士．同年，日本電信電話公社武蔵野電気通信研究所入所．1984年より筑波大学電子・情報工学系に勤務，現在同学系助教授．データベース・システム，計算機アーキテクチャ，関数型プログラミングの研究に従事．日本ソフトウェア科学会，電子情報通信学会，ACM，IEEE各会員．