システムソフトウェアと
オペレーティング・システム 92－7
（２００３．２．２７）

複数の計算機を対象とするプロセス管理手法

芝 公仁[†]

分散オペレーティングシステム Solelc では，資源管理を行うカーネル自体が位置透過に動作し，ネットワークで接続された複数の計算機を同時に管理する．カーネルが実現するプロセス実行環境は，すべての計算機上で同一のものであり，プログラムを実行する環境であるプロセス自体に位置透過性が実現される．実際にプロセスのコードを実行するスレッドは，任意の計算機の CPU を使用して動作することが可能である．また，スレッドが使用する CPU を動的に低コストで変更することできる．本稿では，このような複数の計算機を対象とするプロセス管理手法とその性能評価について述べる．

Process Management on Multiple Computers

Masahito Shiba[†]

In the Solelc distributed operating system, a single kernel works location-transparently and manages all resources of multiple computers on a network. The kernel provides a process execution environment, which is same on all computers. Processes, which are the environments for executing programs, are independent of locations. Threads, which actually execute codes of processes, can use arbitrary CPUs of all computers. Furthermore, it is possible to dynamically change the CPU to use with low cost. In this paper, the process management technique on multiple computers is described.

1 はじめに

分散オペレーティングシステム Solelc[1] では，単一のカーネルがネットワークで接続された複数の計算機を一括して管理する．単一カーネルによる複数計算機の管理によって，分散した資源の一元管理が実現され，効率の良い資源管理が可能となる．Solelc では，このような資源管理を実現するために，カーネルが位置透過に計算機資源を操作することを可能としている．各計算機が持つメモリやディスクなどの資源がそれぞれの計算機で抽象化され，位置透過な資源操作が可能な環境が構築される．カーネルはこの環境上で動作し，システムが持つすべての計算機を管理する．また，計算機資源の抽象化によって，カーネルと同様に，プロセスも位置透過に動作する．すなわち，ある計算機上に存在するプロセスは他の任意の計算機上でも同様に存在し，プロセスのコードの実行には，任意の計算機の CPU を使用することができる．

プロセスは，オペレーティングシステムにおいて，プログラムを実際に動作させるために必要となる非常に重要な概念である．オペレーティングシステムを分散環境に対応させるために様々な試みが行われているが，その際もプロセスにどのように位置透過性を提供するかが重要な課題となる．多くのシステムでは，プロセスを分散環境に対応させるために，メッセージを位置透過に送受信できる機能を提供している．また，計算機間で動作中のプロセスを移動させるプロセス移送の機能を持つシステムもある．

Solelc では，これらの方法とは異なる新たな手法によって，分散環境に対応したプロセスの動作を実現している．すなわち，プロセスに提供する機能に位置透過性を持たせるだけでなく，プロセスを実現する処理自体を位置透過に動作させている．Solelc では，プロセスを実現するカーネルが位置透過に動作し，複数の計算機にまたがってすべてのプロセスを一括して管理する．

このようにプロセス管理を行うことによって，以下の利点が得られる．

- すべての計算機上に同一のプロセス実行環境が構築される．
- プロセスが使用する CPU を動的に低コストで変更することが可能である．
- 単一のプロセスを複数の計算機にまたがって動作させることができる．

Solelc では，単一のカーネルによってすべての計算機を管理するため，すべての計算機のプロセス実行環境は同一のカーネルによって実現される．その結果，単一のプロセス実行環境をすべての計算機から共有する形になり，すべての計算機上で同一のプロセス実行環境が実現される．このようなプロセス実行環境の構築が可能である要因として，Solelc ではカーネルが位置透過に動作可能であるという点があげられる．また，Solelc では，計算機資源の抽象化

[†] 龍谷大学理工学部
Faculty of Science and Technology, Ryukoku University

によってカーネルの位置透過性を実現しているが，この抽象化によってプロセスの動作にも位置透過性が実現される．すなわち，プロセスは実際に動作する計算機によらず，任意の計算機のデバイスを利用することができる．また，プロセスの実行には任意の計算機の CPU を使用することができる．さらに，使用する CPU を動的に変更することや，単一のプロセスが複数計算機の CPU を同時に使用することも可能である．

本稿では，このような特徴を持つ Solelc でのプロセス管理手法について述べる．また，性能評価を行いプロセス管理機構の基本的な性能についても述べ，本手法の妥当性を示す．以下，本稿では，2 章で Solelc の概要，3 章で Solelc におけるプロセス・スレッド管理の特徴，4 章でプロセス管理，5 章でスレッド管理について述べる．次に，6 章と 7 章でプロセス・スレッド管理機構の動作の詳細について述べる．また，8 章で性能評価，9 章で関連する研究について述べ，本手法の有効性について議論する．

2 Solelc の構成

2.1 概要

Solelc では，図 1 に示すように，オペレーティングシステムが抽象化層とカーネルの 2 つに階層化されている．各々の計算機で動作する抽象化層（Abstraction Layer）は，計算機資源を抽象化する役割を持つ．各抽象化層は，他の計算機上の抽象化層と協調動作し，資源操作を行うための機能をカーネルに提供する．抽象化層が提供する機能は位置透過に使用可能であり，これを用いて単一のカーネルがすべての計算機を管理する．

カーネルが実現する機能は，従来のオペレーティングシステムのそれと同様のものである．すなわち，プロセスの実行環境を構築し，複数のプロセスに資源を適切に配分する．また，ファイル操作などのサービスを実現し，これをプロセスに提供する．従来のシステムとの違いは，カーネルが位置透過に動作し複数の計算機を同時に管理するという点である．

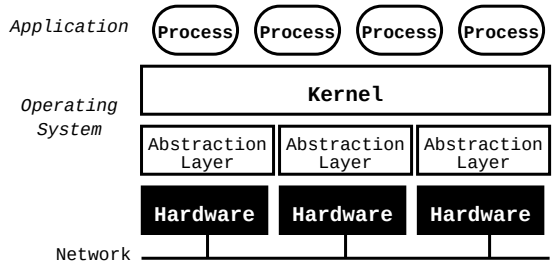

図 1　システム構成

2.2 計算機資源の抽象化

抽象化層によって抽象化される計算機資源として，CPU，メモリ，割り込み，周辺デバイスの 4 つが挙げられる．CPU はスレッドとして抽象化され，カーネルやプロセスのコードを実行するために使用される．CPU のスレッドへの抽象化，および，これを用いたプロセス管理手法については，次章以降で詳述する．

メモリは，複数の計算機から共有される単一の仮想アドレス空間として抽象化される．すなわち，抽象化層は計算機間でのメモリの一貫性制御を行い，ある計算機で行われたメモリ内容の変更を他のすべての計算機から参照可能とする．共有されるアドレス空間には順序一貫性が実現されるが，カーネルやプロセスはこのアドレス空間内で動作するため，一貫性制御を意識することなく位置透過にメモリを読み書きすることができる．

割り込みは，イベントとして抽象化される．抽象化層は，ページフォルトやユーザスレッドが発行したシステムコールなど，各計算機で発生した事象をイベントとしてカーネルに通知する．このとき，抽象化層は，イベントの種類とそれを取得するカーネルスレッドを調べ，当該スレッドが動作する計算機にイベントを転送する．そのため，カーネルは，任意の計算機上ですべての計算機で発生したイベントを取得することができる．

周辺デバイスは，抽象化層内のデバイスドライバによって抽象化される．抽象化層は，カーネルからのデバイスドライバへの要求を当該デバイスドライバを持つ計算機に転送し，さらに当該デバイスドライバの処理結果をカーネルに返す．このように，抽象化層が，デバイスドライバの位置を管理し，要求や処理結果を適切に転送するため，カーネルとデバイスドライバは互いの位置を意識することなく動作可能である．したがって，カーネルは，任意の計算機上からすべての計算機のデバイスを操作することが可能である．

3 プロセス・スレッド管理の特徴

Solelc では，ユーザが作成したプログラムをプロセス・スレッドモデルを用いて実行する．プロセスはプログラムの実行環境であり，コード，データ，スタックといったメモリの内容や，ファイルなど使用する資源に関する情報から構成される．スレッドは，プロセスが提供する環境において実際にプログラムを実行する実体である．Solelc において，スレッドは CPU を抽象化したものであり，抽象化層によって実現される．カーネルは，スレッドを管理することによって，結果として CPU 資源を管理する．

Solelc では，カーネルやプロセスのコードはスレッドによって実行されるが，抽象化層が実現する位置透過性によって，スレッドは任意の計算機が持つ資源を使用することができる．これは CPU 資源に対しても同様であり，スレッドは任意の計算機の CPU を使用して動作することができる．したがって，カーネルやプロセスは任意の計算機上で動作可能となる．このような特徴から，単一の仕事を複数に分割し，それぞれ異なる計算機の CPU を使用して処理することが容易に実現できる．単一のプロセスを複数の計算機で実行可能とすることに加えて，Solelc では，カーネルも複数の計算機で実行することが可能である．そのため，従来のシステムでは困難であったカーネル処理の負荷分散を実現することができる．

システムが複数の計算機を持つ場合，スレッドが動作する計算機を動的に変更することによって，システムが持つ資源を有効に使用できる場合がある．例えば，特定の計算機の負荷が高い場合，その計算機上で動作するスレッドを負荷の低い計算機に移送することによって，負荷の均衡を図ることができる．また，ディスクなどの資源に頻繁にアクセスするスレッドがある場合，そのスレッドを当該資源を持つ計算機に移送することによって，処理の効率化を図ることができる．Solelc では，スレッドが使用する CPU を動的に変更することが可能である．また，資源操作に対する位置透過性から，使用する CPU が変更された場合も，スレッドはそれを意識する必要がない．このように，Solelc では，スレッドの実行が資源の物理的な位置に制限されないため，システムが持つ資源を有効に使用することが可能である．

4 プロセス管理

プロセスは，コードやデータから構成されるプログラムの実行環境であり，その実体はメモリ上のデータである．Solelc では，すべての計算機が単一の仮想アドレス空間を共有し，このアドレス空間上にプロセスが配置される．したがって，ある計算機上に存在するプロセスは，他のすべての計算機上にも存在していると言える．そのため，カーネルは，1 台の計算機上にプログラムのコードやデータを配置することによって，すべての計算機から使用可能なプロセスを生成することができる．このように，カーネルはプロセス管理において，位置透過に動作することが可能である．

カーネルは，以下に示す情報を持つプロセス管理データを用いてプロセスを管理する．

- プロセス識別子
- プロセスが配置されている領域のアドレスとサイズ
- プロセスのコードを実行するスレッド
- プロセスが使用している資源

プロセス識別子は，プロセス生成時にカーネルによってプロセスごとに付与される固有の正の整数値である．プロセス生成時には，さらに，当該プロセスが使用するセグメントとして，連続した領域が確保される．カーネルは，この領域のアドレスとサイズをプロセス管理データによって管理する．

プロセスはプログラムの実行環境であり，プロセスのコードの実行はスレッドによって行われる．プロセス生成時には当該プロセスを実行するスレッドが 1 つ生成される．ただし，プロセス実行の過程においてスレッドの生成･削除が可能であり，プロセスは同時に複数のスレッドを持つことができる．カーネルは，各プロセスが持つスレッドをプロセス管理データによって管理し，あるプロセスに属するすべてのスレッドが終了した際には，当該プロセスが使用していた資源やメモリを解放し，プロセスを終了させる．プロセスが使用する資源とは，ファイル，ソケット，セマフォなどであり，これらの資源は同一プロセス内のスレッド間で共有される．

プロセス管理データは，カーネル内に保持され，その操作はカーネルによってのみ行われる．これは，プロセス管理はすべてカーネルによって実現され，抽象化層は関与しないことを意味する．すなわち，プロセスを実現する処理は位置透過に行われ，すべての計算機上に同一の環境が構築される．

5 スレッド管理

抽象化層は，各計算機の CPU 資源をスレッドとして抽象化する．スレッドを使用することによって，カーネルやプロセスは CPU 資源を使用する．スレッドは，プロセスのコードを実行するユーザスレッドと，カーネルのコードを実行するカーネルスレッドの 2 つに分けられる．カーネルスレッドはすべてのメモリ領域を読み書きでき，ユーザスレッドは自身が属するプロセスのメモリ領域のみ読み書き可能である．プロセスは複数のユーザスレッドを持つことができ，同一のプロセスに属する各々のスレッドをそれぞれ異なる計算機で動作させることも可能である．

5.1 スレッド管理データ

スレッドは，抽象化層が持つスレッド管理データによって管理される．スレッド管理データは，表 1 に示す情報から構成される．id は，スレッドを識別するためのシステム全体で一意な正の整数値であり，スレッド生成時に抽象化層によって付与される．host はスレッドに割り当てられた計算機の識別子

表 1　スレッド管理データ

Name	Description
id	スレッド識別子
register	CPU レジスタの値
state	スレッドの状態
priority	スレッドの優先度
host	スレッドに割り当てられた計算機の識別子

であり，スレッドは割り当てられた計算機の CPU を使用して動作する．スレッドの状態には，従来の OS と同様に，実行状態，実行可能状態，待ち状態がある．スレッドの状態遷移は，スレッドに割り当てられた計算機を管理する抽象化層によって行われる．すなわち，抽象化層は，実行可能なスレッドのうち最も優先度の高いスレッドを実行状態とし，当該スレッドを実行する．このように，抽象化層は，スレッド管理において，ディスパッチャとして機能する．

スレッド管理データは，各々の抽象化層が持つスレッドテーブルによって管理される．スレッドテーブルの n 番目の要素には，スレッド識別子が n のスレッドが動作する計算機の識別子が保持されている．また，自身の計算機上で動作するスレッドに関しては，そのスレッドのスレッド管理データへのポインタも保持する．抽象化層は，スレッドテーブルを用いることによって，どのスレッドがどの計算機で動作しているかを知ることができ，また，スレッド識別子からスレッド管理データを得ることができる．スレッドテーブルの内容は，スレッドの生成・削除やスレッドが動作する計算機が変更されるときに更新される．すなわち，スレッドテーブルに保持されている情報は，各抽象化層で最新の状態に保たれている．

カーネルは，スレッドの優先度を設定したりスレッドに割り当てる計算機を決定するスケジューラとして機能する．例えば，新たに生成するスレッドをアイドル状態の CPU を持つ計算機に割り当てたり，頻繁にデバイスにアクセスするスレッドをそのデバイスを持つ計算機で動作させることによって，システムが持つ資源を効率的に使用することができる．また，カーネルは，スレッド間の親子関係や，スレッドとプロセスとの関係を管理するためのデータを持つ．これらの情報の管理は単一のカーネルが一括して行うため，計算機間での情報の同期が不要であり，1 台の計算機を管理するときと同様に処理することができる．

5.2　スレッド操作機能

各スレッドのスレッド管理データは，当該スレッドが動作する計算機上の抽象化層によって管理される．カーネルは，抽象化層が持つこれらのデータを直接操作することはできないが，抽象化層が提供する以下の機能を利用することによって，スレッド管理データを操作することができる．

- thread_create(thread)
  新たなスレッドを生成し，そのスレッドのスレッド管理データを thread に得る．
- thread_set(thread)
  スレッド管理データの内容を thread とする．
- thread_get(id, thread)
  識別子 id のスレッドのスレッド管理データを thread に得る．
- thread_suspend(id, time)
  time で示される時間，識別子 id のスレッドに CPU を割り当てることを禁止する．

thread_create の要求を受けた抽象化層は，新たなスレッドを生成し，当該スレッドのスレッド管理データをカーネルに渡す．このとき，当該スレッドにはシステム全体で一意な識別子が付与され，スレッドの状態はカーネル処理待ちとなる．この状態は，スレッド管理データがカーネルに渡され，カーネルによって処理されている最中であることを表す．カーネル処理待ち状態のスレッドには CPU が割り当てられることはなく，カーネルのスレッド管理データの操作が完了するまでスレッドは実行されない．

thread_create によって新たに生成したスレッドのスレッド管理データを取得したカーネルは，register, state, priority, host を設定した後，thread_set を用いてこれを抽象化層内に設定する．thread_set の要求を受けた抽象化層は，対象となるスレッドのスレッド管理データを更新し，当該スレッドのカーネル処理待ちを解除する．さらに，当該スレッドが他の待ち要因を持たない場合，実行可能状態に遷移させる．

また，カーネルは，thread_get によって，動作中のスレッドのスレッド管理データを取得することもできる．thread_get の要求を受けた抽象化層は，対象となるスレッドをカーネル処理待ちの状態に遷移させ，スレッド管理データをカーネルに渡す．カーネルは取得したスレッド管理データを適切に設定した後，thread_set によって再び抽象化層内に書き戻す．このとき，カーネル処理待ちは解除されるため，スレッドの実行は再開される．このようにスレッド管理データを操作することによって，システム全体を停止することなく，スレッドを操作・管理することができる．

thread_set を用いることによって，カーネルは，スレッドのレジスタの値や優先度を設定するのみな

らず，スレッドそのものの操作を行うことができる．例えば，カーネルがスレッド管理データの state を終了状態に設定し，thread_set を行うことによって，当該スレッドを終了させることができる．すなわち，スレッドの状態を終了状態にすることを要求された抽象化層は，当該スレッドのスレッド管理データを解放し，また，すべての計算機上のスレッドテーブルから当該スレッドを削除する．

thread_suspend は，スレッドが実行状態になることを一時的に抑制するための機能である．thread_suspend では時間を指定することができ，これによって指定した時間スレッドの実行を中断することが可能である．カーネルは，この機能を使用することによって，sleep システムコールを実現することができる．

カーネルは，抽象化層が提供するこれらの 4 つの機能を位置透過に使用することができる．すなわち，任意の計算機上からスレッド生成の要求を発行することが可能であり，また，スレッド管理データの取得や設定を行うことができる．このとき，カーネルはスレッド識別子のみを指定すればよく，実際にそのスレッドが動作する計算機を意識する必要がない．これは，カーネルからの要求を受け取った抽象化層が，対象となるスレッドが動作している計算機を調べ，当該計算機を管理する抽象化層に要求を転送するためである．そのため，カーネルは，システム内のすべてのスレッドに対して，その位置によらず常に同一の方法で処理を行うことができる．

6 プロセス生成

図 2 は，プロセス A が発行した fork システムコールによって，新たなプロセス B が生成されたときの様子を表している．fork の要求を受け新たなプロセスを生成する際，カーネルはプロセスのコピーとスレッドの生成を行う．このときの処理の流れを以下に示す．

(1) カーネルが，プロセス A から fork システムコールを受け取る．

(2) カーネルが新たなプロセス B のための領域を確保し，プロセス A の内容をその領域にコピーする．また，プロセス B のためのプロセス管理データを作成する．

(3) カーネルが，thread_create によって新たなスレッドの生成を抽象化層に要求し，生成されたスレッドのスレッド管理データを受け取る．

(4) カーネルがスレッド管理データを設定し，これを thread_set によって抽象化層に渡す．

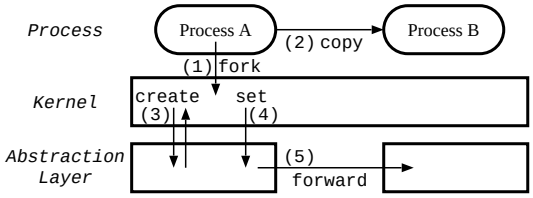

図 2 プロセスとスレッドの生成

(5) スレッド管理データを受け取った抽象化層は，当該スレッドを実行すべき計算機にスレッド管理データを転送する．

(6) スレッド管理データを受け取った抽象化層はスレッドを実行する．

(2) において，カーネルは，要求元のプロセスのメモリ内容をコピーし，これを新たなプロセスの初期状態とする．Solelc ではすべての計算機でアドレス空間が共有されているため，fork を発行したスレッドやコピーを行うスレッドが動作する計算機の位置によらず，位置透過にコピー処理が行われる．また，コピーされたプロセスは，すべての計算機から利用可能である．

(3) において，カーネルは，新たに生成したプロセスを実行するためのスレッドを生成し，スレッド管理データを設定する．具体的には，システムコールを発行したスレッドの register を新たなスレッドの register に設定する．また，カーネルは，生成したスレッドをどの計算機の CPU に割り当てるかを決定し，当該計算機の識別子をスレッド管理データの host に設定する．設定されたスレッド管理データは，thread_set によって抽象化層に渡される．

thread_set の要求を受けた抽象化層は，受け取ったデータからスレッド管理データを設定し，当該スレッドを識別子が host である計算機で実行する．すなわち，各抽象化層に通知しスレッドテーブルに当該スレッドを実行する計算機の識別子を登録する．また，スレッド管理データをこれを実行すべき計算機に送る．このとき，当該スレッドが実行するプロセスのコードやデータは任意の計算機上でアクセス可能であるため，スレッド管理データを計算機間で送受信するのみで，スレッドを目的の計算機上で開始させることができる．

7 スレッド移送

Solelc では，抽象化層による各種資源の抽象化によって，スレッドの動作は資源の物理的な位置に制約を受けない．したがって，Solelc では，スレッドは特定の計算機に固定されない．スレッドは，任意の CPU を使用することができ，また，同一アーキ

テクチャの CPU 間であれば，使用する CPU を動的に変更することも可能である．Solelc において，スレッドが使用する CPU を変更することは，計算機間でのスレッドの移送に相当する．スレッド移送を利用すると，次のような処理を実現することができる．

- 優先すべきスレッドを性能の良い計算機で実行する．
- 負荷の高い計算機上のスレッドを負荷の低い計算機に移送することによって，計算機間の負荷の均衡を図る．
- ディスクなどの周辺デバイスを頻繁に使用するスレッドを当該デバイスを持つ計算機上で動作させることによって処理の効率化を図る．

これらは，システムが複数の計算機から構成される場合，計算機が持つ資源を効率良く使用するために有効な処理である．

従来のシステムでは，移送はプロセスを単位として行われることが多い．一方，Solelc では，プロセスは，位置透過なものであり，すべての計算機上に同様に存在している．そのため，プロセスに位置の概念はなく，Solelc での移送はスレッドに対して行われる．移送の単位がスレッドであると，プロセス移送と比較して小さなコストで移送を実現することができる．また，スレッドは位置透過に動作するため，スレッドから透過的に移送を行うことが可能である．これらのことはカーネルスレッドに関しても同様であり，プロセスのみならずカーネルに対しても，上に述べたような処理を適用することができる．

スレッド移送はスレッド管理データの移送によって実現されるが，この処理は抽象化層によって行われる．抽象化層内のディスパッチャは，CPU に割り当てるスレッドを変更する際に，当該スレッドのスレッド管理データの host と自身が管理する計算機の識別子を比較する．このとき，host と当該計算機識別子が異なる場合，抽象化層は移送処理を実行する．すなわち，当該スレッドのスレッド識別子と host を他の計算機に通知し，スレッドテーブルを更新する．また，当該スレッドのスレッド管理データを計算機識別子が host の計算機に送る．

Solelc では，各々のスレッドをどの計算機で動作させるかは，カーネル内のスケジューラが決定する．すなわち，スレッド移送において，いつどのスレッドをどの計算機に移送するかを決定するのはカーネルである．抽象化層は，カーネルの決定にしたがってスレッド管理データを計算機間で送受信し，スレッド移送を実現する．図 3 は，カーネルがスレッド管理データを操作することによって，計算機 C 上で動作するスレッドを計算機 A に移送する様子を表している．このときの処理の流れを以下に示す．

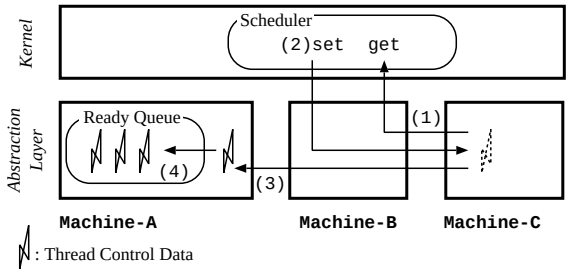

図 3　スレッド移送

(1) カーネルが計算機 C 上で動作するスレッドのスレッド識別子を指定して，thread_get を要求する．この要求を受けた計算機 C 上の抽象化層は当該スレッドをカーネル処理待ちに遷移させ，当該スレッドのスレッド管理データをカーネルに渡す．

(2) カーネルは，取得したスレッド管理データの host を移送先の計算機 A の識別子に変更し，thread_set によってこれを抽象化層に渡す．計算機 B 上の抽象化層はスレッドテーブルから当該スレッドが計算機 C 上で動作していることを調べ，スレッド管理データを計算機 C に送る．

(3) 計算機 C 上の抽象化層は，スレッド管理データの host を調べ，他の計算機にスレッドを移送することを通知する．また，計算機 A に，スレッド管理データを送る．

(4) スレッド管理データを受け取った計算機 A 上の抽象化層は，当該スレッドのカーネル処理待ちを解除し，当該スレッドを実行可能状態に遷移させる．

このように，カーネルは，スレッド管理データの host を変更するだけでよく，移送元および移送先の計算機の物理的な位置を意識しない．すなわち，カーネルの処理は，計算機の物理的な位置とは無関係に，スレッドが使用する CPU を変更するのみである．また，スレッド移送時に抽象化層間で送受信されるデータは，スレッド管理データのみである．スレッドが使用するコードやデータは，スレッドが動作する過程において，メモリの一貫性制御処理により必要に応じて計算機間で送受信される．そのため，必要なデータのみの送受信が行われ，実際には使用されない不要なデータの送受信は行われない．さらに，これらの処理はすべて抽象化層によって行われるため，プロセスやカーネルはそれらを意識する必要がないといった利点がある．

8 評価

本章では，Solelc におけるスレッド移送の評価として，スレッドが使用する CPU を変更する処理の性能評価を行う．ただし，本評価は，Celeron 1.0GHz が搭載された PC を 100Mbps のイーサネットで接続した環境で行っている．

8.1 CPU 割り当ての変更

スレッドが使用する CPU の割り当てを，以下のように，計算機 A と計算機 B 間で変更する．ただし，処理開始時には，当該スレッドが属するプロセスのすべてのメモリページは計算機 A のみによって保持されているとする．

処理 A あるスレッドの使用する CPU を計算機 A から計算機 B に変更し，その後当該スレッドの使用する CPU を計算機 B から計算機 A に戻す．

この処理は，計算機 A から計算機 B へと，計算機 B から計算機 A への 2 回スレッド移送を行う処理に相当する．

スレッドが使用する CPU を計算機 B のものに変更した直後に，計算機 A のものに戻した場合，処理 A に要する時間は 0.367ms である．この場合，計算機 B の CPU ではプロセスのコードを全く実行しない．したがって，コードの実行に必要となる，メモリ内容は計算機間で送受信されない．計算機間で送受信されるデータは，CPU の割り当て変更の通知および実際のスレッド管理データの内容のみである．

当該スレッドが計算機 B の CPU を使用してプロセスのコードを実行する場合，抽象化層は，実行に必要となるメモリページを計算機 A から計算機 B に送り，計算機 B 上で利用可能にする．計算機間で送受信されるのは，プロセス全体の内容ではなく，スレッドが動作する過程で実際にアクセスしたページのみである．したがって，処理 A において，計算機間で送受信されるメモリページは，当該スレッドが計算機 B の CPU を使用して行う処理の内容によって決まる．

図 4 は，処理 A をおいて計算機間で送受信されるメモリページの数と処理時間の関係を示している．ただし，ページサイズは 4096 バイトとする．図に示されるように，処理 A の処理時間は，計算機間で送受信されるメモリページの数に比例して大きくなる．例えば，送受信されるメモリページ数が 7 ページから 8 ページに増加すると，処理時間は 0.565ms 増加する．この時間に比べると，スレッドが使用する CPU を 2 回変更するのに要する時間である 0.367ms の方が短い時間である．したがって，実際の処理時間は，どの程度のメモリページが送受信されるかに大きく依存していると言える．

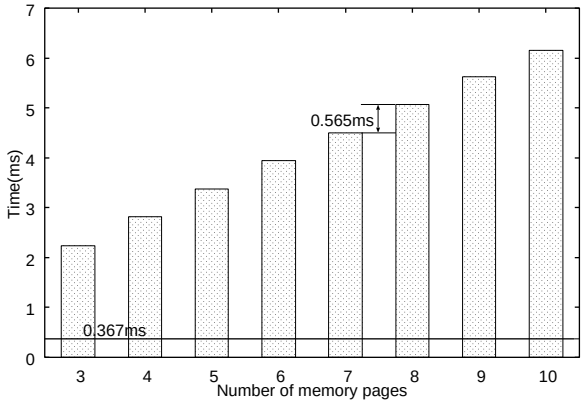

図 4　処理 A の所要時間

通常利用されるアプリケーションでは，数メガバイトから数十メガバイトのプロセスが動作することも珍しくない．このような場合，プロセス単位で移送を行うと，その処理時間は非常に長くなる．一方，Solelc では，プロセスの内容は必要なページのみが送受信されるため，無駄な通信が発生しない．したがって，使用する CPU を効率良く変更することが可能であり，また，変更時にもプロセスの処理を長時間停止することがない．

8.2 連続した移送処理

Solelc のスレッド移送では，プロセス全体ではなく，実際に必要なメモリページのみを送受信するため効率が良い．さらに，連続したスレッド移送が行われるときなどには，移送先に当該スレッドが使用するメモリページがすでに存在し，メモリページの送受信が不要な場合がある．

計算機 A，計算機 B，計算機 C の 3 台が動作する環境において，以下の処理に要する時間を測定した．

処理 B あるスレッドの使用する CPU を計算機 A から計算機 B に変更し，さらに，計算機 B から計算機 C，計算機 C から計算機 A へと順に変更する．

処理 B の開始時にはプロセスの内容は計算機 A のみが持つが，処理終了時は計算機 B と計算機 C も当該プロセスの内容の一部を取得している．そのため，処理 B を連続して行った場合，2 回目以降の処理では，1 回目に比べ送受信されるメモリページの数が減少する．

処理 B に要する処理時間を，1 回目と 2 回目の 2 つ場合について図 5 に示す．図 5 の横軸は，処理 B において各計算機で使用されるメモリページの数で

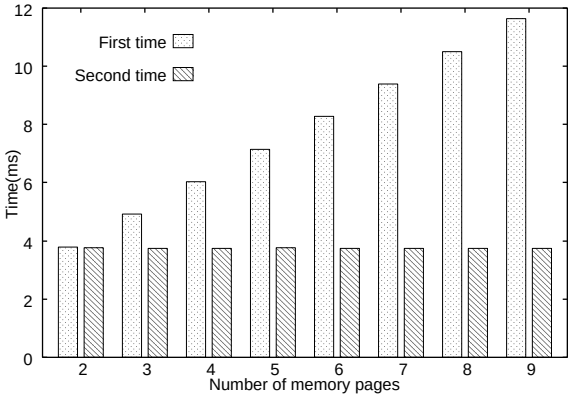

図 5　処理 B の所要時間

ある．ただし，処理 B において，変更のあるメモリページは常に 2 つであり，他のページには変更が行われない．

図 5 に示されるように，1 回目の処理では，使用するメモリページの数に比例して処理時間が長くなる．一方，2 回目の処理では，使用するメモリページの数によらず，処理時間はほぼ一定である．これは，2 回目の処理では，使用するメモリページの数によらず，計算機間で送受信されるメモリページの数が変化しないためである．1 回目の処理時に計算機 B と計算機 C が取得したメモリページのうち書き込みが行われないものは，2 回目の処理では再度取得されることなく，1 回目の処理時に取得されたものが使用される．したがって，2 回目の処理では，使用されるメモリページの数によらず，変更が行われる 2 つページのみが送受信される．そのため，2 回目の処理では，処理時間がほぼ一定の値になる．

このように，スレッドが使用する CPU を変更する場合，変更先の CPU を持つ計算機が当該スレッドが使用するメモリページをすでに保持していると，ページ内容の送受信処理が不要になり，処理コストが軽減される．このような効果が得られるのは変更されないメモリページのみであるが，ほとんどの場合，コードは読み出し専用であるためこの条件を満たす．とくに，共有ライブラリのコードは，複数のプロセスから共有されるため，CPU の割り当てを変更したとき，変更先にすでに保持されている場合があり，この場合 1 回目のメモリページの取得も不要となる．また，一部のプロセスが終了してもメモリ上に残るため，計算機間でのメモリページの送受信回数を軽減させる効果が大きい．

9　関連研究

プロセス管理はオペレーティングシステムにおいて非常に重要なものであり，システムを分散環境に対応させるために，プロセスにどのような機能を提供すべきか様々な提案がなされている．それらの機能のひとつとして，計算機間でプロセスを移動させるプロセス移送 [2] がある．

プロセス移送の機能を用いると，動作中のプロセスを他の計算機に移動させ，当該計算機上で動作を継続させることができる．これを利用すると，プロセスを複数の計算機に移動させることで，各々の計算機の資源を効率良く使用することが可能となる．しかし，移送の単位がプロセスとなっており，移送のコストが大きくなるといった問題がある．また，資源に対する操作の位置透過性が実現されていないため，プロセスは，移送後，資源に対する操作を制限される場合がある．このような制限がない場合でも，移送前に使用していた資源を移送後に使用する際には，移送元の計算機に負荷をかけることがある．これは，負荷分散を目的としたプロセス移送においては，深刻な問題となる．

Solelc での移送はスレッドに対して行われる．移送の単位がスレッドであると，プロセス移送と比較して小さなコストで移送を実現することができる．また，スレッドの実行環境であるプロセスが位置透過であり，その他の資源も位置透過に使用できることから，移送後の資源操作に制限があったり移送元に負荷をかけるといったことがない．

10　おわりに

本稿では，分散オペレーティングシステム Solelc におけるプロセス管理手法について述べた．Solelc では，単一のカーネルが実現するプロセス実行環境を複数の計算機から共有する．これによって，すべての計算機上で同一のプロセス実行環境が実現される．プロセス自体が位置透過なものになるため，実際にプロセスのコードを実行するスレッドは任意の計算機上で動作可能である．また，スレッドが使用する CPU を動的に変更することが可能であり，これによって，スレッド移送を低コストで実現することができる．今後は，スケーラビリティやスレッド移送を用いた負荷分散に関して検討する予定である．

参考文献

[1] 芝公仁, 大久保英嗣: 分散オペレーティングシステム Solelc の設計と実装, 電子情報通信学会論文誌 D-I, Vol. J84–D–I, No. 6, pp. 617–626 (2001).

[2] Milojíčić, D. S., Douglis, F., Paindaveine, Y., Wheeler, R. and Zhou, S.: Process migration, *ACM Computing Surveys*, Vol. 32, No. 3, pp. 241–299 (2000).