システムソフトウェアと
オペレーティング・システム 92－8
（２００３．２．２７）

分散オペレーティングシステムSolelcにおける システムコール処理の分散化方式

玉田 純子 † 水口 孝夫 † 永宗 宏一 † 瀧本 栄二 †
芝 公仁 †† 毛利 公一 ††† 大久保 英嗣 †††

† 立命館大学大学院理工学研究科
†† 龍谷大学理工学部 ††† 立命館大学理工学部

現在, 我々は, 分散オペレーティングシステムSolelcの開発を行っている. Solelcでは, ネットワーク上の複数の計算機を一括して管理しており, カーネルが位置透過に動作可能である. 本稿で述べるシステムコール処理の分散化方式では, 履歴に基づいてシステムコール処理を複数の計算機上のスレッドに分散させている. このため, 計算機資源や負荷の情報を集めることなく, 効率的なシステムコール処理を行うことができる. 本稿では, Solelcにおけるシステムコール処理の分散化方式について述べる.

A Method of Spreading System Calls in Solelc Distributed Operating System

Atsuko Tamada †
Takao Mizuguchi † Koichi Nagamune † Eiji Takimoto †
Masahito Shiba †† Koichi Mouri ††† Eiji Okubo †††

† Graduate School of Science and Engineering, Ritsumeikan University
†† Faculty of Science and Technology, Ryukoku University
††† Faculty of Science and Engineering, Ritsumeikan University

We have been developing Solelc distributed operating system. Plural computers on which Solelc works are managed by an operating system, and the kernel can work location-transparently in Solelc. Therefore, system call processings are distributed to plural computers, and so operating system work efficiently. In this paper, a method of spreading system calls in Solelc are described.

1　はじめに

現在, 我々が開発を行っている分散オペレーティングシステム Solelc [1, 2] では, 単一のカーネルによって複数の計算機を管理しており, カーネルは位置透過に動作することができる. このため, カーネルの各機能をそれぞれ異なる計算機上で動作させることや, カーネルが動作する計算機を動的に変更することが可能である.

Solelc では, スレッドの生成時に, CPU, メモリ, 通信負荷に基づき算出した残存処理能力の高い計算機にスレッドを配置することで負荷分散を実現している [3]. また, 負荷の変化を考慮した負荷均衡も行っている. しかし, この機構だけでは, 負荷の増加のみを移送の契機としているため, 計算機の追加による計算機資源の変化に対応したスレッドの移送を行うことができない. また, スレッドの分散配置を行うことによって, ある計算機上のスレッドが他の計算機上のデバイスを操作する場合がある. 処理効率を向上させるためには, 負荷だけではなく, スレッドと周辺デバイスとの関係なども考慮する必要がある.

そこで, 今回, 効率的なシステムコール処理を行うために, 履歴に基づくシステムコール処理の分散化方式について検討を行った. 本方式では, 計算機ごとにシステムコール処理の履歴を取得し, それに基づいてシステムコール処理を複数の計算機上のスレッドに分散させている. 本方式を用いることで, 計算機資源や負荷の情報を集めることなく, スレッドと周辺デバイスとの関係を考慮したシステムコール処理の分散化が可能となる. したがって, 効率的なシステムコール処理が実現される.

以下, 本稿では, 2 章で Solelc の概要, 3 章でシステムコール処理の分散化方式について述べた後, 最後に 4 章で本稿のまとめを行う.

2　Solelc の概要

Solelc では, 複数の計算機が持つ資源の効率的な利用と位置透過な資源操作の実現を目的としている. これらを実現するために, Solelc では, 単一のカーネルによって複数の計算機を管理している.

Solelc では, 各計算機で抽象化機構を動作させ, 計算機資源の抽象化を行うことによって, カーネルが位置透過な資源管理を可能とする環境を構築している. カーネルは, この環境上で位置透過に動作することが可能であり, 複数の計算機資源を一括して管理できる.

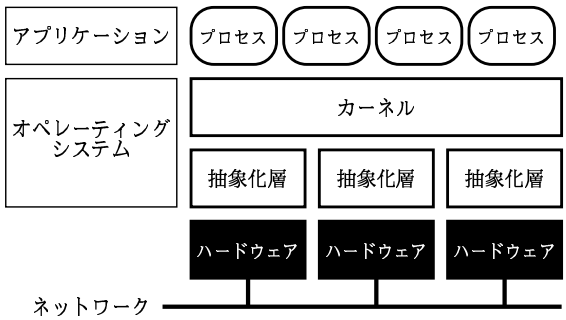

図 1　Solelc のシステム構成

Solelc 上で動作するプロセスは, システムコールを用いてカーネルの機能を使用し動作している. このため, カーネルと同様に, プロセスも位置透過に動作することが可能である.

2.1　システム構成

Solelc のシステム構成を図 1 に示す. Solelc では, オペレーティングシステム (以下 OS と記す) を, カーネルと抽象化層の 2 層に階層化している. カーネルがシステム全体の資源管理を行い, 抽象化層が個々の計算機を管理する. このような構成をとることによって, 位置透過な資源管理を実現し, 複数の計算機を効率良く管理することができる.

2.1.1　カーネル

Solelc のカーネルは, 管理対象の計算機の台数によらず 1 つであり, 抽象化層が提供する機能を使用してすべての資源を一括して管理する. 単一のカーネルですべての計算機を管理することによって, 計算機ごとにカーネルを動作させている従来の OS とは異なり, 協調動作のための処理が不要となる. このため, カーネル自体の処理は, 単純なものとなっている.

カーネルは, 従来の OS が実現しているものと同様の機能を実現する. すなわち, プロセスを実現し, これらにシステム全体が持つ計算機資源を適切に分配する. カーネルが実現する機能は, システムコー

ルによってプロセスに提供される.

一般に, カーネルの処理は, プロセスや周辺デバイスが発生させる割込みを契機として行われる. Solelc では, カーネルが割込みを直接取得するのではなく, 抽象化層が生成するイベントの取得を契機としてカーネルが動作する.

カーネルは, 複数のカーネルスレッドによって実行される. カーネルスレッドは, 抽象化層が提供する機能を使用することによって, 位置透過に動作することができる. また, おのおののカーネルスレッドを異なる計算機上で動作させることや, カーネルスレッドを他の計算機へ移送することが可能である. このような特徴を利用することによって, 効率的な資源の使用や, カーネル自体の負荷分散が可能である.

2.1.2 抽象化層

抽象化層は, すべての計算機で 1 つずつ動作し, カーネルが動作するための環境を構築する. 抽象化層は, 抽象化層間で協調処理を行うための通信機能を持ち, 専用のプロトコルを用いて通信を行う. 各抽象化層は, 互いに協調処理を行うことによってシステム全体の資源の位置を管理し, カーネルからの資源操作の要求を操作の対象となる資源を持つ計算機に転送する.

また, 抽象化層は, 自身が動作する計算機の資源を直接操作する機能を持ち, 計算機資源の抽象化を行う. 抽象化層では, OS が管理する資源であるメモリ, CPU, 割込み, 周辺デバイスを, 以下のように抽象化し管理している.

- メモリ
  各計算機の物理メモリをシステム全体で 1 つの仮想アドレス空間として抽象化する. アドレス空間は, 図 2 に示すように, 非共有領域と共有領域の 2 つに分けられる. 非共有領域は, 抽象化層のコードやデータが配置され, 計算機ごとに異なる内容を保持している. 共有領域にはカーネルやプロセスが配置され, すべての計算機がこの領域を共有する. 共有領域は, 抽象化層によって一貫性制御が行われており, 計算機間のメモリの一貫性が保たれている. このため, すべての計算機から位置透過なメモリアクセスを行うことができる.

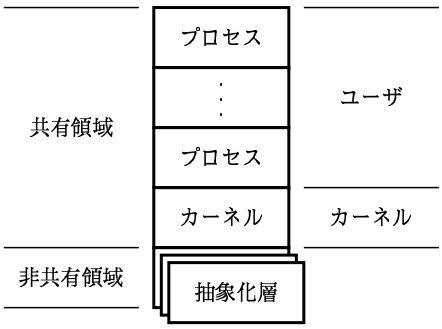

図 2 Solelc のアドレス空間

- CPU
  CPU を, カーネルやプロセスのコードを実行するスレッドとして抽象化する. すなわち, カーネルやプロセスは, スレッドを使用することによって CPU を使用する. スレッドは, 抽象化層が実現する位置透過性によって, 任意の計算機が持つ CPU を使用して動作することが可能である.

- 割込み
  タイマ割込みやシステムコールなど各計算機で発生した事象を, カーネルに通知するイベントとして抽象化する. イベントは, 発生した事象の種類を表すイベント番号とその事象に関する情報から構成される. 図 3 は, イベントがカーネルに通知される様子を表している. 抽象化層には, あらかじめ, カーネルによってイベントを取得するスレッドが登録される. 抽象化層は, このスレッドをすべての計算機に通知する. このため, 図 3 では, 計算機 B で発生した割込みがイベントとして抽象化された後, 計算機 A に通知されている. 抽象化層がこのような処理を行うことによって, カーネルは, 任意の計算機からすべてのイベントを取得することが可能となる.

- 周辺デバイス
  抽象化層は, おのおののデバイスを, デバイスドライバによって抽象化し管理している. 抽象化層は, カーネルからの要求を適切な計算機に転送し, デバイスドライバへ通知する. このため, 任意の計算機からデバイスドライバが使用可能となり, 位置透過なデバイス操作が実現できる.

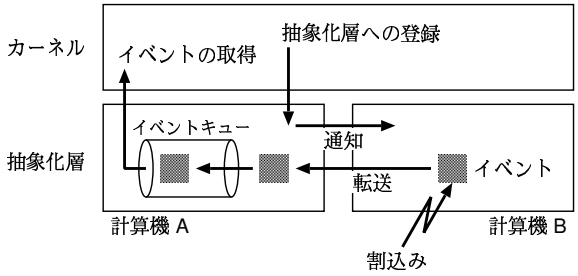

図 3　イベント処理

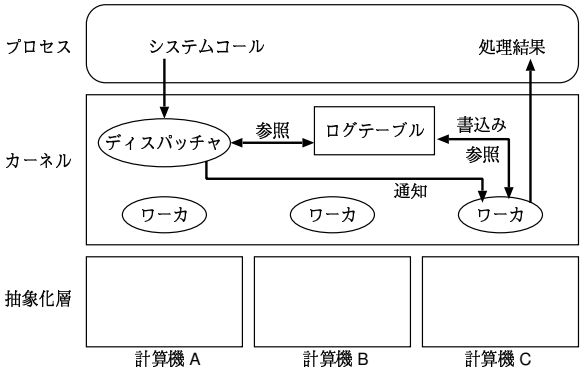

図 4　システムコール処理の流れ

2.2　システムコール

Solelc では, システムコールもイベントとして処理される. イベントは, 抽象化層によって適切な計算機に転送されるため, ユーザスレッドとカーネルスレッドは, 互いの位置を意識することなく動作可能である.

システムコール処理は, ディスパッチャ/ワーカモデルに基づき, システムコールを取得するディスパッチャと実際に処理を行うワーカの 2 種類のスレッドによって行われる. ディスパッチャの数は 1 つであるが, ワーカは複数存在する. おのおののワーカは, ディスパッチャによって取得されたすべてのシステムコールを処理することができる. このため, 複数のシステムコールが発生した場合にも, 複数の計算機上のワーカに処理を分散させることが可能である. また, システムコールの処理状態によらず, 直ちにシステムコールを取得することができる.

3　システムコール処理の分散化方式

現在, Solelc では, ディスパッチャがワーカに処理要求を通知する際, ラウンドロビンに基づいてワーカの動作する計算機を選択している. ラウンドロビンでは, 各計算機の CPU 負荷や周辺デバイスを考慮した計算機の選択が行われない.

例えば, システムコールによって要求された処理がディスプレイへの描画であれば, 描画対象のディスプレイが接続された計算機上で処理されることが望ましい. このように, システムコールは, 要求する処理の特性, 負荷などを考慮して処理されるべきである. このためには, システムコールの種類に応じて適切な計算機を選択する必要がある. しかし,

表 1　ログテーブルが保持する情報

名前	説明
syscall	システムコールの種類
host	計算機の識別子
time	システムコール処理に要した時間
average	過去 10 回分の処理時間の平均
select_host	選択された計算機の識別子
select_time	選択された計算機の平均処理時間

Solelc では, 計算機の動的な追加, 切放しによって管理する計算機資源が変化するため, あらかじめ, システムコールの種類ごとに特定の計算機を設定することは困難である.

そこで, 本方式では, 動的に変化する環境を把握するために, 計算機ごとにシステムコール処理の履歴を取得する. 履歴に基づき選択された計算機上で動作するワーカに対してディスパッチャが処理要求を通知することによって, システムコールの種類に応じて適切な環境へ処理を分散させることが可能である. したがって, システムコール処理に要する時間を短縮することができる.

3.1　分散化方式の構成

本方式は, ディスパッチャ, ワーカ, ログテーブルによって実現される (図 4 参照). ログテーブルは表 1 に示す情報を保持しており, 図 5 に示すような構成となっている. ログテーブルの保持する値は, ワーカによって書き込まれる.

ユーザスレッドによって発行されたシステムコー

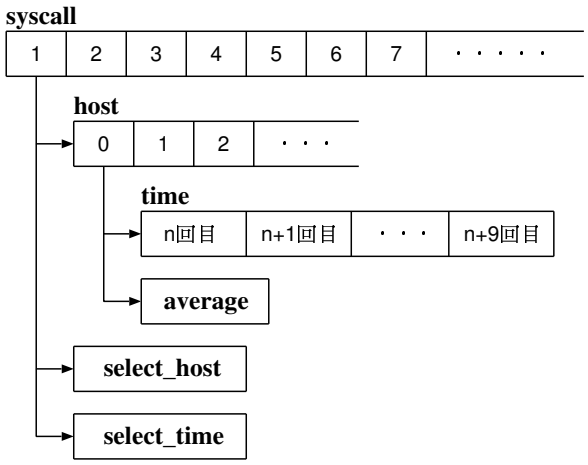

図 5　ログテーブル

ルは, ディスパッチャによって取得される. ディスパッチャは, ログテーブルを参照することによって, どの計算機上のワーカに対して処理要求を通知すべきかを知る. ディスパッチャからの処理要求を取得したワーカは, システムコールによって要求された処理を行い, 処理結果をユーザスレッドに返す. 図 4 では, 計算機 A 上で動作するディスパッチャがシステムコールを取得し, 計算機 C 上で動作する計算機がシステムコールの処理を行うものとする.

システムコールの処理時間は, ディスパッチャがワーカに処理要求を通知する直前に取得した時刻と, ワーカがシステムコールの処理結果をユーザスレッドに通知した直後に取得した時刻との差から求められる. ディスパッチャは, ワーカに対して処理要求を通知する際, 自身が取得した時刻も同時に通知する. ワーカは, 自身が取得した時刻とディスパッチャから取得した時刻との差を求め, ログテーブルに値を書き込む. 処理時間には, ディスパッチャとワーカが異なる計算機上で動作した場合や, ワーカの動作する計算機と操作対象のデバイスが接続された計算機とが異なる場合に発生する抽象化層間の通信時間が含まれている. また, CPU 利用率が高い場合の待ち時間なども含まれている. このため, 計算機資源や負荷の情報を集めることなく, 動的に変化する環境を把握することができる.

システムコール処理の履歴は, ワーカによって管理される. ワーカは, 過去 10 回分の処理時間の平均を求め, 計算機ごとの平均処理時間を比較し, 最小の値を持つ計算機を選択する. このように, ワーカが履歴の管理を行うことによって, ディスパッチャは, 取得したシステムコールの処理を直ちにワーカに対して通知することができる.

3.2　システムコール処理の流れ

システムコール処理は, ディスパッチャによるシステムコールの取得とワーカによる処理によって行われる. また, 本方式では, ワーカが履歴の管理を行い, システムコールごとに適した計算機を選択することによって, システムコール処理の分散化を行い, 処理効率を向上させている.

3.2.1　ディスパッチャの動作

ディスパッチャは, システムコールの取得を契機に動作する. ディスパッチャによる処理の流れを以下に示す.

(1) ユーザスレッドによって発行されたシステムコールを取得する.

(2) 現在の時刻を取得する.

(3) ログテーブルの select_host を参照し, システムコールを処理する計算機を確認する.

(4) (3) で確認した計算機上で動作するワーカに, システムコールの処理要求と (2) で取得した時刻を通知する.

ディスパッチャは, このように動作することによって, おのおののシステムコール処理に適した計算機上で動作するワーカに対して処理要求を通知することが可能である.

3.2.2　ワーカの動作

ワーカは, ディスパッチャからの処理要求の取得を契機に動作する. ワーカにおける処理の流れを以下に示す.

(1) ディスパッチャから取得したシステムコールの処理を行う.

(2) 処理結果をユーザスレッドに返す.

(3) 現在の時刻を取得し, ディスパッチャから取得した時刻との差を求め, ログテーブルの time に記録する.

(4) 過去10回分の処理時間の平均を求め, ログテーブルの average の値を更新する.

(5) ログテーブルの select_time の値と (4) で更新した average の値を比較する. average の値が大きければ (a) の処理を行い, 小さければ (b) の処理を行う.

(a) ログテーブルの select_host の値を参照する. 自身の動作する計算機の識別子と同じ値の場合, ログテーブルのすべての計算機の average の値を比較し, 最小の値に select_time を更新する. また, その値を保持していた計算機の識別子に select_host を更新する.

(b) select_time を (4) で更新した average の値に更新する. また, select_host を参照し, 自身の動作する計算機の識別子と異なる値の場合, select_host を自身の動作する計算機の識別子に更新する.

このように, ワーカが履歴を管理することによって, ディスパッチャは, 取得したシステムコールの処理要求を直ちにワーカに対して通知することができる.

本方式では, 履歴に基づきシステムコールの種類に適した計算機を選択することによって, システムコールの種類に応じて適切な環境で動作するワーカへ処理を分散させることが可能となる. したがって, システムコール処理に要する時間の短縮と効率化が実現される.

また, 計算機の選択を行う際に比較する値に関して, 過去 10 回分の処理時間の平均だけではなく, 最大および最小の処理時間についても検討し, これらを組み合わせることによって, 動的な環境の変化をより適切に把握することが可能となる.

4 おわりに

本稿では, Solelc におけるシステムコール処理の分散化方式について述べた. 計算機ごとにシステムコール処理の履歴を取得し, それに基づいてシステムコール処理を複数の計算機上のワーカに分散させることによって, 計算機資源や負荷の情報を集めることなく, スレッドと周辺デバイスとの関係を考慮したシステムコール処理の分散化が可能となる.

また, ワーカが履歴の管理を行い, 計算機の選択を行うことによって, ディスパッチャは, 取得したシステムコールの処理要求を直ちにワーカに対して通知することができる. したがって, システムコール処理に要する時間が短縮され, 処理効率の向上が実現できる.

今後の予定としては, 計算機の追加による計算機資源の変化に対応した, 積極的なカーネルスレッドの移送についても検討を行い, 負荷分散機構と協調動作させることによって, スレッドの動作特性と負荷状態の双方を考慮したカーネルスレッドの分散化の実現を目指す.

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 の構成, ” 情報処理学会研究報告 2000–OS–84, Vol. 2000, No. 43, pp. 237–244 (2000).

[2] 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).

[3] 永宗宏一, 芝公仁, 大久保英嗣: “分散オペレーティングシステム Solelc における負荷分散機構, ” 情報処理学会研究報告 2000–OS–85, Vol. 2000, No. 43, pp. 39–46 (2000).