Vol.2011-ARC-197 No.33
Vol.2011-HPC-132 No.33
2011/11/29

プロセスを分散実行するためのシステムコール制御手法

三添　匠[†1]　小鍛治翔太[†1]　芝　公仁[†2]

現在，単一のプロセスを複数の計算機上にまたがって動作させることができるプロセス共有機構を開発している．近年，性能向上のため複数のコアを用いたプロセッサが普及している．これに伴い，複数のコアを使用するマルチスレッドのアプリケーションが多く開発されるようになった．我々は，このようなマルチスレッドのアプリケーションを一つの計算機で動作させるのではなく，ネットワークに接続されている複数の計算機で動作させる機構を構築している．本機構では，複数の計算機で単一のプロセスを共有することが可能で，共有されるプロセスは各計算機で分散実行される．本稿では，本機構におけるシステムコールの制御手法について述べ，評価によりその有用性を示す．

Management Method of System Call for Shared Processes

Takumi Mizoe,[†1] Shouta Kokaji[†1]
and Masahito Shiba[†2]

We have been developing a system that a process can run using two or more PC. Recently, PCs with multicore processors that are developed for performance improvement are widely used, and multi-threaded applications are developed and used to use multicore processors. These applications can use only cores of a PC. Our proposed system provides functions that enable applications to simultaneously use cores of two or more PCs connected via a network. In this system, PCs share processes and the threads of the shared processes are distributed to the PCs. This paper, describes management of system calls invoked by the shared processes.

†1 龍谷大学大学院理工学研究科
Graduate School of Science and Technology, Ryukoku University
†2 龍谷大学理工学部
Faculty of Science and Technology, Ryukoku University

1. はじめに

近年，プロセッサの性能向上に伴い複数のスレッドを使用して動作するアプリケーションが増加している．しかし，一般的なシステムでプロセスを実行した場合，単一の計算機上でしかそのプロセスを動作させることができない．本研究では，マルチスレッドのアプリケーションを単一の計算機内だけで処理させるのではなく，複数の計算機を使用して動作させるプロセス共有機構を構築している．プロセス共有機構は，複数の計算機でプロセスを共有することを可能にする．また，共有されるプロセスは任意の計算機を利用して位置透過に処理を行える．

現在，我々が開発を行っているプロセス共有機構では，単一のプロセスを複数の計算機上にまたがって動作させることができ，プロセスが持つスレッドを各計算機に分散させることが可能である．これにより，プロセスは同時に複数の計算機の資源を利用することが可能となる．これは，プロセスの持つアドレス空間を計算機間で共有することで実現される[1)]．すなわち，ある計算機でメモリへの書き込みが行われると他の計算機でもそれを読み出すことができる．アドレス空間の共有により，複数の計算機でプロセスの持つスレッドを分散させて処理させることが可能になり，システム全体で単位時間あたりの処理能力が大きくなる．また，共有プロセスが発行したシステムコールは，それを処理すべき計算機に転送され実行される．システムコールを適切な計算機に転送することにより，プロセス共有機構上で動作するアプリケーションは，どの計算機で動作していても同一のオペレーティングシステムにより処理することができる．これら，アドレス空間の共有，システムコールの転送は自動的に行われ，プロセスは分散処理を意識する必要がなく，既存のアプリケーションをそのまま複数の計算機で動作させることができる．

本稿では，特に，プロセス共有機構上でプロセスが持つスレッドを各計算機に分散させたとき，当該スレッドから発行されたシステムコールをそれを実行すべき計算機に転送し実行するシステムコールの制御手法について述べる．

2. 関連研究

システムコールをそれが発行された計算機以外の計算機で実行する提案はこれまでにも多数行われてきた．ユーザレベルでプロセスを移送した際，システムコールの発行を代行するスタブプロセスを用いることにより，システムコールの転送を実現する研究がある[2)]．これは，ユーザレベルでシステムコールにネットワーク透過性を持たせるために，C 言語

Vol.2011-ARC-197 No.33
Vol.2011-HPC-132 No.33
2011/11/29

の標準ライブラリである libc のシステムコール関数を上書きすることによりシステムコールの転送を行う．ユーザレベルでシステムコールの転送を実現し，カーネルの変更を必要としないが，C 言語におけるシステムコールのインタフェース部分を置き換える必要がある．プロセス共有機構では，カーネルモジュールにより実装しているため，カーネルの変更，既存のアプリケーションの変更は必要としない．

64 ビットの CPU やオペレーティングシステムの普及に伴って，大きなアドレス空間の利用が可能になっている．巨大なアドレス空間は，大規模なデータを扱うことが可能である．しかし，1 台の計算機で提供できる物理メモリはハードウェアの制約もあり，計算機に搭載できる物理メモリの容量には限界がある．緑川ら[4] は，遠隔計算機の物理メモリを集めて仮想的に大容量メモリを提供するシステムを構築している．これは，高速ネットワークでつながれた複数の計算機に分散する遠隔メモリを用いて，逐次プログラムのための大規模な仮想メモリを提供するものである．このようなシステムでは本システムと違い，複数の計算機を使用して分散処理させるのではなく，大規模な仮想メモリを提供するものであり，ユーザ側も大規模の仮想メモリを想定してアプリケーションを開発する必要がある．本システムでは，既存のアプリケーションを変更することなく動作させることが可能であり，遠隔の物理メモリを使用することも可能である．

アドレス空間の一貫性制御やシステムコールの転送の処理時間は，システムの性能に大きく影響する．西田ら[5] は，InfiniBand と PCI Express を組み合わせた広帯域ネットワーク上にソフトウェア分散共有メモリの環境を構築し，通信に必要なコストを削減している．通信時間を短縮することにより計算性能の向上を実現しているが，このシステムには専用のハードウェアが必要となる．

3. プロセス共有機構

本研究の目的は，単一のプロセスをネットワークに接続されている複数の計算機上にまたがって動作できるようにし，そのプロセスの持つスレッドがそれぞれの計算機の資源を効率的に利用できるようにすることである．プロセス共有機構では，計算機間でプロセスを共有するために，複数の計算機間でのアドレス空間の共有を実現する．プロセスの共有を実現するプロセス共有機構の構成を図 1 に示す．プロセス共有機構は以下の 2 つから構成される．

- メモリ制御部
  共有するプロセスのもつアドレス空間の制御
- システムコール制御部

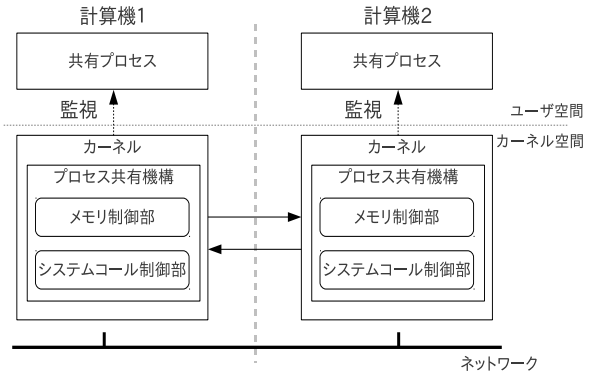

図 1 プロセス共有機構

共有するプロセスが発行するシステムコールの制御

共有プロセスは，計算機間で共有されるプロセスであり，共有プロセスの持つスレッドは，アドレス空間を共有する任意の計算機上で動作が可能である．また，プロセス共有機構上で動作する共有プロセスは，本機構用にコードを書き換えずに動作させることが可能であり，本機構上で分散実行されることを意識する必要がない．

システムコール制御部は，共有プロセスの持つスレッドが発行するシステムコールに対して，適切な計算機の転送と実行を行う．システムコール制御部は，各システムコールの実際の処理が実行される前に動作する．

メモリ制御部は，共有プロセスを監視し，必要に応じて他の計算機上のメモリ制御部と通信を行い，共有プロセスが複数の計算機上で分散実行されているのを意識することなく動作できるような環境を構築する．具体的には，計算機間で共有プロセスの持つアドレス空間の共有を実現する．これは，共有するプロセスのメモリアクセスを監視し，ある計算機でアドレス空間に書き込みが完了すると，他の計算機からでも書き込まれた値を読み出せるようにメモリの一貫性制御を行う[3]．これにより，共有プロセスはどの計算機上でも同一のメモリ内容を参照することができ，ある計算機上でメモリ内容の変更があった場合，プロセスを共有しているどの計算機上でもそれを読み出すことができる．

Linux は，アドレス空間内をページを単位として管理する．プロセスが利用可能な領域は，プログラムのコードやデータ，また，プロセスの実行に使用されるスタック，ヒープ，共有ライブラリなどが配置される．これらは，単純にプロセスによって管理されているのではな

Vol.2011-ARC-197 No.33
Vol.2011-HPC-132 No.33
2011/11/29

く，どの領域にどのファイルがマップされているか，どの領域がどのように使用されるかなど，Linux カーネルによっても管理される．複数の計算機でアドレス空間を共有するためには，単純にメモリ内容の一貫性制御を行うだけでは十分でなく，アドレス空間の情報も各計算機で同一のものとなるように制御する必要がある．

メモリ制御部では，ページ状態を管理し，他の計算機のメモリ制御部と協調し，各共有プロセスのアドレス空間に順序一貫性のメモリモデルを実現する．共有するプロセスのアドレス空間を他の計算機と共有し，プロセスがメモリへ書き込んだ値は他の計算機からでも読み出すことができ，同一計算機上でのスレッド間のメモリ共有のように計算機間でのメモリの共有が可能になる．共有プロセスのアドレス空間の順序一貫性が実現されているため，共有プロセスはメモリの一貫性制御を意識することなく動作可能である．メモリ制御部は，アドレス空間内のメモリをページ単位で管理し，ネットワーク内のどの計算機でもメモリの参照ができるように，ページの複製をそれぞれの計算機に作成する．このとき，メモリ制御部は，メモリの一貫性が保たれるように，共有プロセスのアドレス空間内のメモリページを以下の 3 つの状態で管理する．

- ページ内容を持たずに読み書きできない状態
- ページ内容を持ち読み出せるが書き込めない状態
- ページ内容を持ち読み書きできる状態

メモリの読み出しにおいては，同時に複数の計算機がメモリページを持ち参照可能であり，効率的にアクセスが行える．また，メモリの書き込みは一つの計算機のみ行え，一貫性が保たれる．このように，各計算機が持っているメモリページを管理することにより各計算機間での順序一貫性のメモリモデルが実現され，ある計算機でメモリに書き込まれた値は，他の計算機からでも参照可能となる．これにより，単一のプロセスが複数の計算機間で共有され，プロセス内の各スレッドを複数の計算機に分散させ動作させることが可能になる．

4. システムコールの制御

アドレス空間の共有によりコードを位置透過に実行を行えるように，システムコールの利用も位置透過に行えるようにする．共有プロセスのスレッドから発行されたシステムコールは，スレッドが動作する計算機上で必ず行われるのではなく，システムコールを実行すべき計算機に転送され，実行される．図 2 に，プロセスが共有される様子を示す．図 2 は，計算機 1，2，3 でプロセス 1 を共有し，計算機 3，4 でプロセス 2 を共有している．共有するプロセス内のスレッドはそれぞれの計算機に分散実行されている．このように，プロセス共

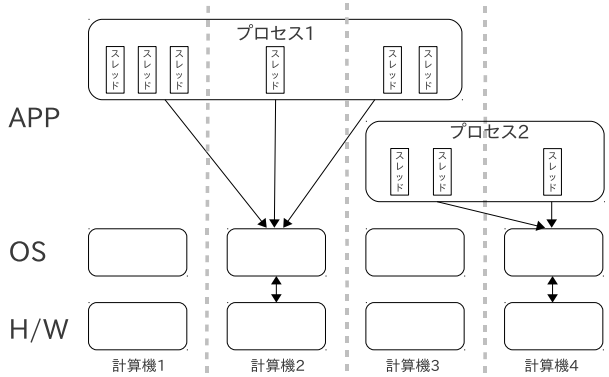

図 2 プロセスの共有

有機構内で動作するアプリケーションは，それぞれの計算機に分散され，それぞれの計算機が持つプロセッサを使用するが，利用するオペレーティングシステムは一つであり，プロセスには単一の計算機しか見えず，分散を意識したアプリケーションでなくても動作させることが可能である．システムコールを適切な計算機に転送することにより，分散されたスレッドはどの計算機からでもシステムコールを発行することが可能になる．以下，本章では，システムコールを制御するシステムコール制御部の構成について述べる．

4.1 全 体 構 成

システムコール制御部は，複数の計算機に分散した各スレッドから発行されるシステムコールを，それを実行すべき計算機に転送し，実行する機能をもつ．システムコール制御部の構成を図 3 に示す．システムコール制御部はカーネル内にあり，共有プロセスのスレッドがシステムコールを発行したときに動作する．システムコール制御部は以下の 2 つからなる．

- 制御部
  システムコールの実行を行う
- 転送部
  システムコール転送に必要な通信を行う

制御部は，転送部により転送されてくるシステムコールの実行を行う．また，制御部内には，共有プロセス内でスレッドが生成される度に当該スレッドのシステムコールを実行する実行スレッドが生成される．転送されてくるシステムコールの処理が終了すると，転送部に

Vol.2011-ARC-197 No.33
Vol.2011-HPC-132 No.33
2011/11/29

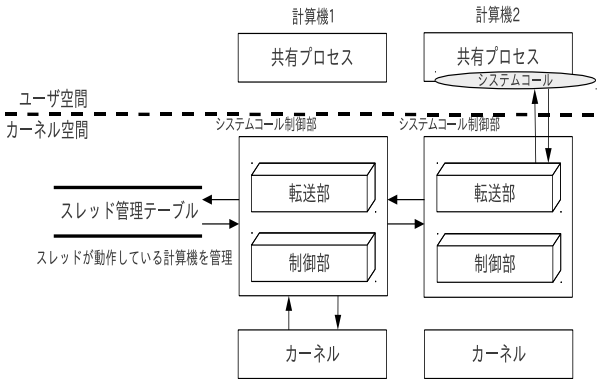

図 3 システムコール制御部

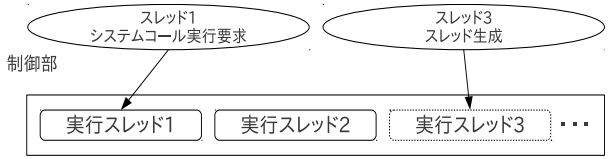

図 4 制御部

システムコール終了通知を送信するよう要求する．

転送部は，共有プロセスが動作している計算機のシステムコール制御部とそれぞれ通信を行う．転送部は，共有プロセスが発行するシステムコールの実際の処理が行われる前に実行される．システムコールを転送するか判断し，転送する場合は適切な計算機にシステムコールの実行要求を送信し，転送しない場合はシステムコールを発行した計算機の制御部で処理する．

4.2 制 御 部

制御部は，共有プロセスが発行したシステムコールを受け，それを実行する．図 4 に制御部の構成を示す．制御部には，実行スレッドがあり，転送されてきたシステムコールは実行スレッドにより処理される．この実行スレッドは共有プロセス内にスレッドが生成された時に生成され，共有プロセス内のスレッドのシステムコールを実行する．共有プロセス内でスレッドが生成される度に実行スレッドが生成され，共有プロセス内のスレッドひとつひとつにそれぞれ対応した実行スレッドがある．転送されたシステムコールは，システムコールを発行したスレッドに対応した実行スレッドにより処理される．また，共有プロセス内でスレッドが生成された際，各計算機で管理するプロセス番号とは別に，分散しているスレッドを一意に識別するためのスレッド番号を当該スレッドに割り当て，スレッド管理テーブルにより管理する．スレッド管理テーブルは，システムコールを実行する計算機の制御部により管理さる．スレッド管理テーブルにより，共有プロセス内で動作しているスレッドがどの計算機で動作しているかを調べることが可能である．

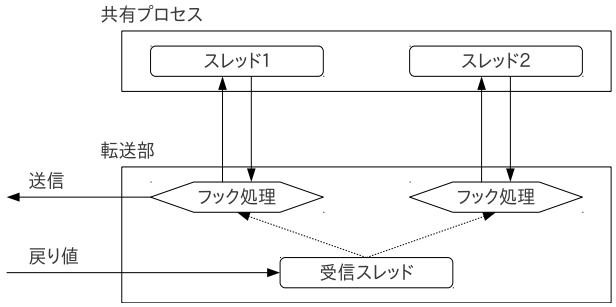

図 5 転送部

4.3 転 送 部

転送部は，共有プロセスの持つスレッドが発行するシステムコールに対して，各システムコールの実際の処理が行われる前に実行される．転送部の構成を図 5 に示す．転送部では他の計算機のプロセス共有機構と以下の通信を行う．

- システムコール実行要求
- システムコール終了通知

システムコール実行要求における通信では，他の計算機のシステムコール制御部にシステムコールを転送する．共有プロセスのスレッドからのシステムコールを取得した転送部は，当該システムコールを実行すべき計算機を調べる．システムコールを転送しない場合は，システムコールを発行した計算機で処理する．システムコールを転送する場合は，システムコールの実行に必要なシステムコール番号とシステムコールの引数を，システムコールを実行する計算機のシステムコール制御部に送信する．システムコール番号と引数は，システムコールが発行された際のレジスタの値を参照することにより取得する．システムコール実行要求を送信した後は，システムコールを発行したスレッドの状態を実行状態から待ち状態に遷移させシステムコールの終了まで停止させる．

Vol.2011-ARC-197 No.33
Vol.2011-HPC-132 No.33
2011/11/29

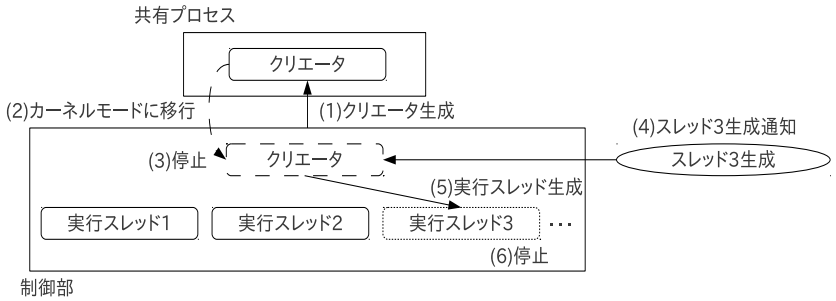

図 6 実行スレッド

システムコール終了通知では，実行が完了したシステムコールの戻り値をシステムコールを発行したスレッドが動作している計算機に送信する．また，転送されたシステムコールが無効なものであればエラーを送信する．

5. システムコール実行

システムコールは，システムコール実行要求により適切な計算機に転送され，制御部内の実行スレッドによって処理される．以下，本章では，システムコールの転送から実行までの手順を述べる．

5.1 実行スレッド

制御部は，共有プロセス内でスレッドが生成されると，カーネル内の制御部に実行スレッドを生成する．クリエータは，共有プロセスの起動時に共有プロセス内に生成されるスレッドである．生成されたクリエータは，カーネルモードに移行し，共有プロセスが終了するまでシステムコールを実行する計算機上の共有プロセスに存在し，共有プロセスでスレッドが生成されるまで待ち状態で停止する．カーネルモードに移行したクリエータは，システムコールを実行する計算機で動作している共有プロセスのタスク構造体を持つ．共有プロセス内でスレッドが生成されると，クリエータにより共有プロセスのスレッドとして実行スレッドがカーネル内に生成される．転送されたシステムコールはこの実行スレッドにより処理される．共有プロセスのスレッドである実行スレッドで処理することにより，他の計算機上のスレッドが発行したシステムコールをシステムコールを処理する計算機上のプロセスから発行されたかのように処理することができる．実行スレッドを利用することにより，どの計算機から発行されたシステムコールも，ローカルで発行されるシステムコールと同じように処理することが可能で，システムコールを発行した計算機を意識する必要はない．また，実行スレッドは共有プロセス内のスレッドに 1 対 1 に対応している．共有プロセス内でスレッドが生成されたことは，システムコールを実行する計算機の制御部に通知される．共有プロセスのスレッドが発行したシステムコールは，発行したスレッドに対応した実行スレッドが処理する．図 6 は，共有プロセス内にスレッド 3 が生成されたときに，それに対応付けられる実行スレッド 3 が生成される様子である．このときの処理の流れを以下に示す．

(1) 制御部が共有プロセスにクリエータを生成する
(2) 共有プロセスのクリエータがユーザモードからカーネルモードに移行する
(3) クリエータは共有プロセス内にスレッドが生成されるまで停止する
(4) スレッド 3 が生成され，スレッドが生成された通知を制御部が受け取る
(5) クリエータが実行スレッド 3 を生成する
(6) スレッド 3 がシステムコールを発行するまで，実行スレッド 3 を停止させる

(3) において，クリエータはカーネルモードに移行したあと，実行状態から持ち状態に移行し，共有プロセス内にスレッドが生成されるまで停止する．(5) において，他の計算機でスレッドが生成され，スレッドが生成された通知を制御部が受け取ると，クリエータは実行スレッドを生成し，生成された実行スレッドは，待ち状態に遷移し，システムコールが発行されるまで停止する．

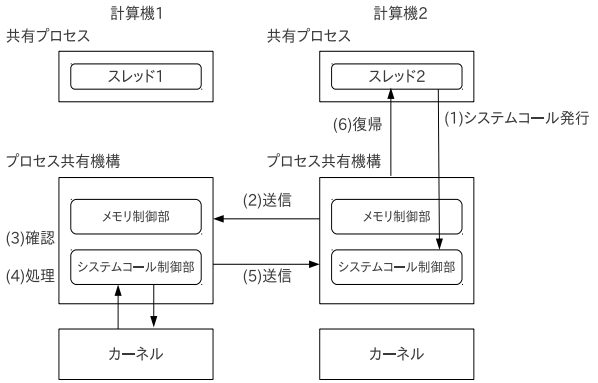

図 7 システムコール転送

5.2 システムコール転送実行

図 7 に計算機 1 と計算機 2 でプロセスを共有し，共有プロセスのスレッド 1 を計算機 1，

Vol.2011-ARC-197 No.33
Vol.2011-HPC-132 No.33
2011/11/29

表 1　システムコール処理時間

	A	B	C
open	0.353ms	0.358ms	0.717ms
close	0.368ms	0.370ms	0.460ms
mkdir	1.124ms	1.130ms	1.402ms
rmdir	1.036ms	1.037ms	1.416ms
symlink	1.521ms	1.530ms	1.884ms
flock	0.981ms	0.982ms	1.099ms

スレッド 2 を計算機 2 で実行させる様子を示す．このとき，スレッド 2 から発行されたシステムコールは，以下のように計算機 1 に転送され，実行される．

(1)　スレッド 2 がシステムコールを発行する
(2)　システムコール制御部で転送する計算機を調べ，その計算機にシステムコール実行要求を送信する
(3)　システムコール番号が有効なものか確認する
(4)　処理する実行スレッドを判断し，システムコールを処理する
(5)　戻り値を送信する
(6)　受信した戻り値をレジスタに格納し，停止していたスレッド 2 を復帰させる

(2) においてシステムコールを処理する計算機を調べた際，システムコールを転送する場合はシステムコール実行要求を送信し，転送しない場合はシステムコールが発行された計算機で処理する．また，システムコール実行要求を送信したあとは，システムコールを発行したスレッドの状態を実行状態から待ち状態に移行させる．

(3) においてシステムコールの実行要求を制御部で受け取ると，システムコールを実行するカーネルでシステムコール番号が有効なものか確認する．システムコール番号が無効なものならエラーを送信する．

(4) において転送した先でシステムコールを処理するときも，メモリ制御部により各計算機でメモリの一貫性が保証されている．システムコールを実行する計算機がメモリページを持っていないメモリ領域を参照する場合でも，他の計算機からメモリページを取得でき，矛盾なく参照が可能になる．これにより，プロセスのメモリ領域を参照するシステムコールでも転送先で処理することができる．

6. 評　　価

本章では，x86 アーキテクチャの Linux カーネル 2.6.33.7 に構築したプロセス共有機構の性能評価について述べる．本章における性能評価には，Core i7 2.8GHz のプロセッサ，2GB のメモリを搭載した複数の計算機を 1000Mbps のイーサネットで接続した環境を用いた．

6.1　システムコール転送時間

基本性能の評価として，2 台の計算機を使用し，システムコールの転送に要する時間を測定した．発行するシステムコールを open，close，mkdir，rmdir，symlink，flock とし，それぞれ以下のような条件で処理時間を測定する．

A　本機構を使用せず，システムコールをそれを発行した計算機で実行
B　本機構を使用し，システムコールをそれを発行した計算機で実行
C　本機構を使用し，システムコールを他の計算機に転送し実行

ファイルシステムには NFS を使用し，第 5 階層のディレクトリ内のファイルを操作の対象とした．処理に要した結果を表 1 に示す．

システムコール制御部では，実際にはシステムコールを転送しない場合でも，転送部によりシステムコールを実行すべき計算機を調べるため，いくらかのオーバヘッドがある．A と B の処理時間の差がシステムコール制御部を使用した際に起こるオーバヘッドである．このオーバヘッドによる処理時間の増加は全体の処理の約 0．5%であり十分小さいと考えられる．

システムコールを転送し，実行するのに要する時間は B と C との差により求められる．close，flock とそれ以外での転送時間が大きく違うことがわかる．これは，open などのシステムコールは，システムコールの実行時にプロセスのメモリ領域に参照が行われ，これに伴って，メモリページの転送が行われたためである．システムコールを実行する際に，実行する計算機に参照するメモリページがなけれは，ページ内容の送受信が発生するため処理時間が大幅に増加している．

6.2　ファイルの read 時間

ファイルの読み出し時間を測定するため，分散実行しているスレッドから read システムコールを転送処理し，他の計算機にあるファイルの read 内容を取得するまでの処理時間を評価する．これは図 8 のように，2 台の計算機を使用して，計算機 A の共有プロセスから発行した read システムコールを，計算機 B に転送処理し，ファイルが置かれたバッファのメモリページを計算機 A の共有プロセスが読み込むまでの処理時間である．処理に要した時間を図 9 に示す．

処理結果から，読み込むファイルサイズがページサイズである 4 キロバイトを越えるご

Vol.2011-ARC-197 No.33
Vol.2011-HPC-132 No.33
2011/11/29

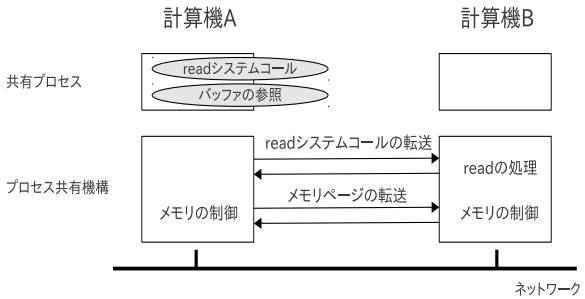

図 8 ファイル読み出し時間の評価

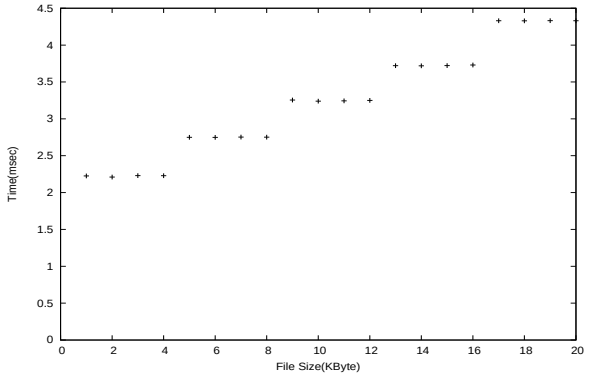

図 9 ファイルの read 時間

とに処理時間が増加していることがわかる．これは，read システムコールを転送処理した後，システムコールを発行した計算機で，ファイルが置かれたバッファのメモリページを読み出す際，メモリページは転送先の計算機が持っているためメモリページの転送が行われる．プロセス共有機構では，アドレス空間をページ単位で管理しており，4 キロバイトごとに通信を行う．8 キロバイトのファイルの読み込みでは，4 キロバイトの読み込みと比較してページ転送が 1 回多く発生するため，その分処理時間も増加する．処理結果より，ページ 1 回分の転送時間は約 0.4ms であることがわかる．読み込むファイルのサイズが大きくなるほどメモリページの転送回数も増加し，全体の処理時間も増加する．

7. おわりに

本稿では，分散されたスレッドが発行したすべてのシステムコールを適切な計算機に転送し実行するシステムコールの制御について述べた．プロセス共有機構では，共有プロセス内で生成されたスレッドに対応した実行スレッドをシステムコールを実行する計算機に生成することにより，システムコールの透過性を実現した．システムコールの透過性により，任意の計算機からシステムコールを発行することが可能になり，位置透過なスレッドの分散実行が実現され，計算機資源の効率的の活用が実現される．また，システムコール制御部の使用に対するオーバーヘッドは少なく，同時に複数の計算機からのシステムコールの要求にも十分対応できる．

参考文献

1) 小鍛治 翔太，芝公仁，岡田至弘: 分散共有メモリを用いたアドレス空間共有方式，平成 22 年度情報処理学会関西支部大会講演論文集，A-02，2010．
2) 田村 日左之, 多田 好克: プロセス移送システムにおける入出力およびシステムコールの透過性の実現, 社団法人情報処理学会, Vol.C-56,No.1,pp.70-71(1998)．
3) Lamport, L: How to Make a Multiprocessor Com-puter That Correctly Executes Multiprocess Pro-grams, IEEE Trans. Comput. , Vol. C-28, No. 9,pp.690-691 (1979).
4) 緑川博子，黒川原佳，姫野龍太郎:遠隔メモリを利用する分散大容量メモリシステム DLM の設計と 10GbEthernet における初期性能評価，情報処理学会論文誌，Vol.1,No.3,pp.136-157(2008)．
5) 西田晃:広帯域ネットワークを用いたソフトウェア分散共有メモリの実現と性能評価，情報処理学会研究報告，pp.187-192，(2006)．