1J-04

システムコールの組み合わせ発行によるオーバーヘッド削減手法

中村　公亮†　芝 公仁†
†龍谷大学理工学部

1　はじめに

プロセスがファイル I/O やメモリの確保，ソケット間通信などオペレーティングシステムの機能を使用する場合，システムコールが使用される．システムコールが発行されると CPU はカーネルモードへと切り替わり，カーネルのコードの実行を始める．このとき，ユーザプログラムの処理が中断されカーネルに処理が移ることに加え，CPU の実行モードの遷移や TLB フラッシュなどオーバーヘッドが発生する．これらを削減するために，システムコールの要求を行うスレッドと処理を行うスレッドに分ける手法が考えられる [1]．2 つのスレッド間では要求や返り値の受け渡しに共有メモリを使用する．この手法の利点は複数のシステムコールをまとめて要求できることであり，一度のシステムコール発行で複数個のシステムコールをまとめて処理できるようになる．

本稿では，メモリをプロセスとカーネルで共有し，共有しているメモリからシステムコールに必要な情報を取り出してシステムコールを処理する手法を示す．

2　提案手法

メモリ共有を用いたシステムコール発行 (以降，並列システムコールと呼ぶ) について述べる．

2.1　構成

並列システムコールのために，以下のシステムコールを Linux カーネルに追加した．

(1) 処理スレッドへ切り替わるためのシステムコール
(2) 処理スレッドを実行可能状態にするためのシステムコール
(3) 処理の完了を待つためのシステムコール

提案手法では，カーネル内でシステムコール要求を処理するための専用スレッドがプロセス毎に動作する (以降，処理スレッドと呼ぶ)．システムコールの要求を行うスレッド (以降，要求スレッドと呼ぶ) は，処理スレッドと共有メモリを介して要求の受け渡しや返り値の取得などを行う．要求の受け渡しには，システムコール発行に必要な情報をまとめた構造体 (以降，リクエスト構造体と呼ぶ) を使い，そのメンバは表 1 の通り

表 1　リクエスト構造体のメンバ

メンバ	役割
status	要求の状態を示す
sysnum	要求するシステムコール番号を格納する
args_num	引数の数を指定する
args[]	それぞれに引数を格納する
ret_val	処理が完了したときの返り値を格納する

である．要求スレッドは共有メモリ内の予め取り決めた領域（以降，リクエスト領域と呼ぶ）にそれを書き込む．また，共有メモリ内には処理スレッドに終了するよう通知するためのフラグ（以降，終了フラグと呼ぶ）がある．

並列システムコールにおけるシステムコール処理では，1 個以上のスレッドが (1) のシステムコールを用いてカーネル内で要求を処理する．要求スレッドはリクエスト構造体を使用して要求を作成し，リクエスト領域に置く．この際，要求スレッドはシステムコール処理の完了を待つために，(3) のシステムコールを使用することができる．また終了フラグを設定することで処理スレッドを (1) のシステムコールから復帰させることができる．

この手法では要求するスレッドと処理するスレッドが異なり，共有メモリに対してのアクセスが行われるだけであるためプロセスの処理が中断されない．また処理スレッドを生成するとき，複数のスレッドを生成することで複数の要求を同時に処理できる．

2.2　動作

要求スレッドはまずリクエスト領域を作成するための共有メモリを確保する．これは他のスレッドが書き込みを行えるように適切な設定を行っておく．次に，スレッドを生成し，生成されたスレッドは (1) のシステムコールを発行し処理スレッドとなる．(1) のシステムコールでは，まず，共有メモリへアクセス可能かを確認する．共有メモリ領域に対して書き込み可能であることが確認されなければその時点でエラーとして終了する．次に，要求スレッドが，実際に発行したいシステムコールの要求を作成する．これはリクエスト構造体の各メンバに値を設定することで行う．status メンバが FREE になっている構造体を探し，REQUESTING に変更する．その後，共有メモリに要求を置き，status メンバを REQUEST に設定する．要求後は (2) か (3) のシステムコールを使用して処理スレッドを実行可能状態にする．(3) の場合にはすぐに必要な処理が完了す

Compound System Calls for Decreasing the Overhead
Kousuke Nakamura†, Masahito Shiba†
†Faculty of Science and Technology, Ryukoku University

表 2　実験マシンの構成

項目	値
カーネル	Linux-4.14.9
CPU	core-i5 4570
メモリ	DDR3 1600 8 G バイト
共有メモリ領域	リクエスト数によって変化
処理スレッド数	1 から 3 まで

るまで待つ．

実行可能状態になった処理スレッドはリクエスト領域を参照しシステムコール要求が置かれているかを確認する．リクエストが格納されている場合は status メンバが REQUEST に設定されている．要求状態のリクエスト構造体を発見すれば実際に処理を行うが，このとき，処理スレッド間で競合が発生しないようにする必要がある．これを行うため，コンペア・アンド・スワップ (CAS) という CPU 命令を用いる．処理スレッドがリクエスト構造体の status メンバを，CAS 命令を用いて REQUEST から PROCESSING に更新する．その後，処理スレッドが，各メンバの値を参照してシステムコール処理を行う．システムコール処理が終了した後は，結果をリクエスト構造体の ret_val メンバに設定し，その結果に基づいて status を SUCCESS もしくは ERROR に変更する．リクエスト領域内を探索し終わった処理スレッドは，要求スレッドによって終了フラグが設定されていないかを確認する．設定されていない場合には，新たな要求が置かれていないことを確認してから，再び要求が置かれるまで待ち状態になる．

処理スレッドが要求を処理している間，要求スレッドはブロックされないため他の処理を行うことができ，結果が必要になったタイミングでリクエスト領域に完了しているかを参照しに行けば良い．全ての処理が終了して処理スレッドが不要になったときには，終了フラグを設定する．

3　評価

本章では，実際に提案手法を用いて行った実験を示す．まず，表 2 に実験を行ったマシンの構成を示す．オーバーヘッドを計測するために，/dev/zero から 4096 バイトを読み込む read システムコールを通常の手続きで発行した処理時間と，提案手法によって発行した処理時間を比較する．どちらの手法でも read の回数を変化させながら処理時間を計測した．提案手法では処理スレッド数を変化させながら計測した．また，計測プログラムをそれぞれ 500 回ずつ実行し，平均を取った．図 1 に提案手法の処理スレッド数ごとの計測結果，通常手続きでの計測結果のグラフを示す．横軸は read システムコールを要求した回数を示し，縦軸は処理時間を示す．

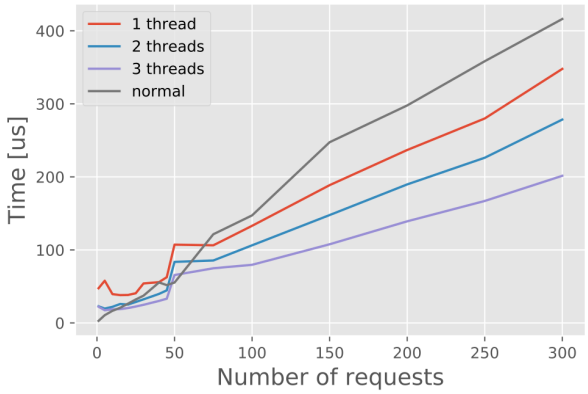

図 1　提案手法と従来型の処理時間

要求数が 50 未満のときには，提案手法より通常の手続きの方が処理時間が短くなっている．これは，要求スレッドがシステムコール要求を共有メモリに置き，処理スレッドが要求を処理し，要求スレッドが共有メモリを参照して結果を回収する一連の処理にかかる時間が，従来型の手続きと比べ大きくなったからだと考えられる．一方で，要求数が 50 を超えた後からは提案手法による処理時間が，従来型の手続きでかかる時間より小さくなった．これは，すでにカーネル内で動作している処理スレッドが複数の要求を一度に処理できることで，システムコール発行に伴う CPU の実行モードの遷移などのオーバーヘッドが低減されたからだと考えられる．

処理スレッド数ごとの処理時間を見ると，処理スレッド数が多くなるほど処理時間が短くなっているのが分かる．また，処理スレッド数が多いほど要求数に伴う処理時間の増加が緩やかである．これは，処理スレッドが並列に動くことにより，一つのシステムコール処理の間に別のシステムコール処理を行うことができるからだと考えられる．

4　おわりに

本稿では共有メモリを使用してカーネルとプロセスでシステムコール要求や結果の受け渡しを行い，システムコールを要求するスレッドと発行するスレッドを分離する手法を示した．また本手法と従来型の手続きそれぞれの処理時間を比較した．その結果，少ない要求数では本手法の処理時間の方が長くなったが，要求数が多くなるにつれ本手法の処理時間の方が短くなる．

参考文献

[1] Livio Soares and Michael Stumm.:FlexSC: Flexible System Call Scheduling with Exception-Less System Calls, *OSDI* (2010)