システムソフトウェアと
オペレーティング・システム 89－7
シ ス テ ム 評 価 2－7
（２００２．２．１４）

並列分散オペレーティングシステムCEFOSにおける一括システムコール機構の実装と評価

中山 大士†, 棚林 拓也†, 日下部 茂‡, 谷口 秀夫‡, 雨宮 真人‡

†九州大学 大学院 システム情報科学府
‡九州大学 大学院 システム情報科学研究院

nakayama@al.is.kyushu-u.ac.jp, tana@al.is.kyushu-u.ac.jp, kusakabe@csce.kyushu-u.ac.jp
tani@csce.kyushu-u.ac.jp, amamiya@al.is.kyushu-u.ac.jp

あ ら ま し　システムコールを多用する応用プログラムでは、実行時にカーネル呼び出しによるモード切り替え処理が頻発する。しかしながら、プロセッサの高周波数化に伴い、カーネル呼び出し時のモード切り替え処理は、相対的に高価なものになっている。そこで、我々は、複数のシステムコールをまとめて、一つのシステムコールとしてカーネルに処理を依頼する、一括システムコール機構について提案している。一括システムコール機構を用いることによって、システムコール多用時のカーネル呼び出しの回数を削減でき、その結果、高周波数動作のプロセッサにとって高価な処理であるモード切り替え処理を削減できる。

キーワード　オペレーティングシステム, システムコール, 並列分散処理, 細粒度マルチスレッディング

Implementation and Evaluation of Packaged System Call Mechanism in CEFOS

Hiroshi NAKAYAMA†, Takuya TANABAYASHI†, Shigeru KUSAKABE‡,
Hideo TANIGUCHI‡, and Makoto AMAMIYA‡

Graduate School of Information Science and Electrical Engineering, Kyushu University

nakayama@al.is.kyushu-u.ac.jp, tana@al.is.kyushu-u.ac.jp, kusakabe@csce.kyushu-u.ac.jp
tani@csce.kyushu-u.ac.jp, amamiya@al.is.kyushu-u.ac.jp

Abstract　Frequent transitions between user-mode and kernel-mode occur in application programs with a lot of system calls. However, for the-state-of-the-art processors of high clockrate, mode-transition is an expensiveoperation. We propose Packaged System Call Mechanism (PSCM), which packages multiple system calls into a single packaged system call and then send the request of the service of the packaged system call to the kernel. With this PSCM, we can reduce the number of system calls and reduce overhead of expensive mode-transition operations.

Key words　Operating System, System Call, Parallel and Distributed Computing, Fine-grain Multi-threading

1 はじめに

応用プログラム（以下 AP と略す）は、大きく計算主体のものと入出力主体なものに分けることができる。計算主体の AP は、プロセスとカーネルの間で連携する処理は少ない。一方、入出力主体の AP では、プロセスとカーネル間のやりとりが頻発する。計算主体の AP をより高速に実行するには、より高性能なプロセッサを用いて実行すれば良い。しかし、入出力主体の AP の場合、より高性能なプロセッサを用いて実行しても計算主体の AP ほど効果が期待できない。これは入出力処理がカーネル呼び出しを伴うものだからである。あるプロセスがカーネルを呼び出したとき、プロセッサはパイプラインをフラッシュし、走行モードを切り替えるといった処理を行う。この処理がカーネル呼び出し時のオーバヘッドになっている。AP の処理の中でカーネル呼び出し回数が増加すると、プロセッサが走行モードを切替える処理を行う時間が増え、処理効率が低下する。

そこで、我々は、カーネルの呼び出し回数を削減する一括システムコール機構を提案する。一括システムコール機構とは、システムコール発行した際、すぐにはカーネルを呼び出さず、複数のシステムコールをまとめた後にカーネルを呼び出す機構である。カーネル呼び出しの回数を削減することで、高性能プロセッサにおける処理効率の低下を抑制する。本論文では、一括システムコール機構の概要と実装内容について述べ、その評価について報告する。

2 一括システムコール

2.1 CEFOS

CEFOS[1] とは Communication-Execution Fusion Operating System の略で、従来の環境で提供されているマルチスレッド実行環境と比べて粒度の小さなスレッドを効率良く実行することで、性能面および記述面での通信と処理のより良い融合を目指している。性能面での融合とは、一つのスレッド内に閉じた処理、一つの計算機内に閉じた処理、および二つ以上の計算機またがった処理について、通信遅延を効率良く隠蔽することで処理時間の差異を短くすることである。また記述面での融合とは、スレッド間のデータ授受を通信パスを意識しない形で行えるようにし、一つの計算機内に閉じた処理と計算機間にまたがった処理とを同様に記述できることである。

CEFOS は、同期処理をプリミティブとして実現し、細粒度処理と非同期通信処理を効率良く支援する細粒度マルチスレッド実行方式を基本計算方式とするオペレーティングシステムである。CEFOS では従来の環境で提供されているマルチスレッド実行環境と比べてスレッドの粒度が小さいため、スレッド切り替え操作が多発すると考えられる。ゆえにスレッド切り替えのオーバヘッドを削減するために、スレッドを OS 核外で管理する。この OS 核外スレッド管理部を CEFOS では External Kernel と名付ける。一方、従来の OS 核のように計算機資源管理する部分を Internal Kernel と名付ける。External Kernel はユーザモードで走行し、Internal Kernel はカーネルモードで走行する。

また CEFOS では、処理の完了までの時間が長い可能性のある処理を行う場合は、処理要求を行うスレッドと、結果の受け取りを行うスレッドに分割し実行する。処理完了までの時間が長い場合、処理結果待ちの間、他のスレッド走行させることによって CPU 時間を有効に使用でき、高いスループットを得ることができる。

2.2 基本機構

一括システムコール機構の様子を図 1 に示し、以下に処理手順の概要を説明する。

(1) プロセスとカーネルは共有メモリ域を保有する。この共有メモリ域は、プロセスごとに存在し、プロセスが存在する間プロセスの状態に関わらずカーネルからは常に操作可能である。

(2) プロセスは、システムコールの実行に必要な情報を共有メモリ域に書きこむ。

(3) システムコール要求数が共有メモリ域に閾値以上書きこまれると、プロセスはカーネルを呼び出す。

(4) カーネルは、共有メモリ域に書かれてある情報を元に、システムコール処理を呼び出す。

(5) カーネルは、システムコールの実行結果を共有メモリ域に書く。

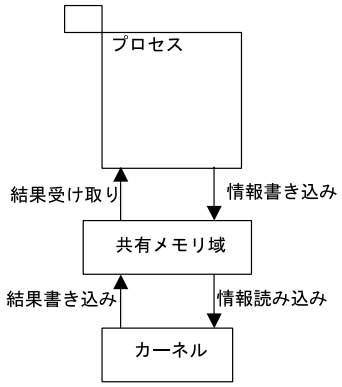


図 1: 一括システムコール

従来のシステムコールと一括システムコールの長所および短所について表 1 に示す。一括システムコールは、共有メモリ域にシステムコール要求をまとめ、要求数が

表 1: 従来のシステムコールと一括システムコールの比較

処理方式	長所	短所
従来のシステムコール	処理の開始が早い	カーネル呼び出しの回数が多い
一括システムコール	カーネル呼び出しの回数が少ない	処理の開始が遅い

閾値以上貯まるまでカーネル呼び出しを遅らせ、一括処理することによってシステムコール発行時のカーネル呼び出しの回数を削減する。これによって走行モード切り替えやスタック切り替えといった処理を削減できる。しかし、システムコール処理の開始時間を遅らせるためシステムコールの応答時間が増加する問題がある。

一方、従来のシステムコールは要求が来ると即座にカーネルを呼び出すため、応答時間が一括システムコールよりも短い。しかしシステムコール要求に対応した処理が一つ終了すると、プロセススケジューラを呼び出すかどうかチェックする。つまり N 回のシステムコールを処理したとき、最悪 N 回プロセススケジューラを呼び出すことになる。プロセスを切り替えるとメモリ空間の切り替えが起こり、キャッシュが無効になり、計算機のスループットが低下する恐れがある。

3 実現方式

我々が開発を行っているオペレーティングシステム CEFOS に一括システムコール機構を実装した。

3.1 データの管理構造

一括システムコール機構で用いられるデータについて述べる。プロセスとカーネル間の共有メモリ域は、プロセス側で確保した領域のアドレスをカーネル側に通知することで実現する。図 2 に一括システムコール機構で用いられる管理表の関係を示し、以下で説明する。一括システムコール機構では、一括システムコール情報管理表、要求情報管理表、結果情報管理表、要求内容表、要求情報表および結果情報表を用いる。これらの表はプロセスに一組ずつ存在する。

一括システムコール情報管理表は、プロセスとカーネルが共有するデータ格納域の先頭アドレスを保有している管理表で、三つの要素からなる。三つの要素は要求情報管理表の先頭へのポインタ（req_info）、要求内容表の先頭へのポインタ（res_info）、および結果情報表の先頭へのポインタ（sys_info_buf）である。

要求内容表は、システムコールを実行するために必要な情報を格納する表である。要求内容表の 1 エントリの要素を表 2 で示す。要求内容表はシステムコール番号（Syscall_NR）、要求の引数（Arg1〜arg6）および結果格納先のアドレス（Ret）を要素に持つ固定長の配列である。

表 2: 要求内容表の 1 エントリの要素

要素	説明
syscall_NR	システムコール番号
arg1〜arg6	要求の引数
ret	結果格納先のアドレス

表 3: 要求情報表の 1 エントリの要素

要素	説明
syscall_info	要求内容のエントリの先頭アドレス
priority	要求を出したスレッドの優先度

表 4: 結果情報表の 1 エントリの要素

要素	説明
syscall_info	要求内容のエントリの先頭アドレス
retval	システムコールの戻り値

要求情報表は、登録された要求内容表のエントリの中で未処理のエントリの先頭アドレスを格納する表である。要求情報表の 1 エントリの要素を表 3 に示す。要求情報表は要求内容のエントリの先頭アドレス（Syscall_info）および要求を出したスレッドの優先度（Priority）を要素に持つ固定長の配列である。要求情報表が記録できるエントリ数は、要求内容表が記録できるエントリ数と同数である。

結果情報表は、登録された要求内容表のエントリの中で処理済のエントリの先頭アドレスを格納する表である。表 4 に結果情報表の 1 エントリの要素を示す。結果情報表は要求内容のエントリの先頭アドレス（Syscall_info）およびシステムコールの戻り値（Retval）を要素にもつ固定長の配列である。結果情報表が記録できるエントリは要求内容表が記録できる数より多い。これは、結果情報表が、自プロセスが出した要求を受けとるだけでなく、他プロセスが自プロセスに対して出した情報を受けとるために用いられるからである。

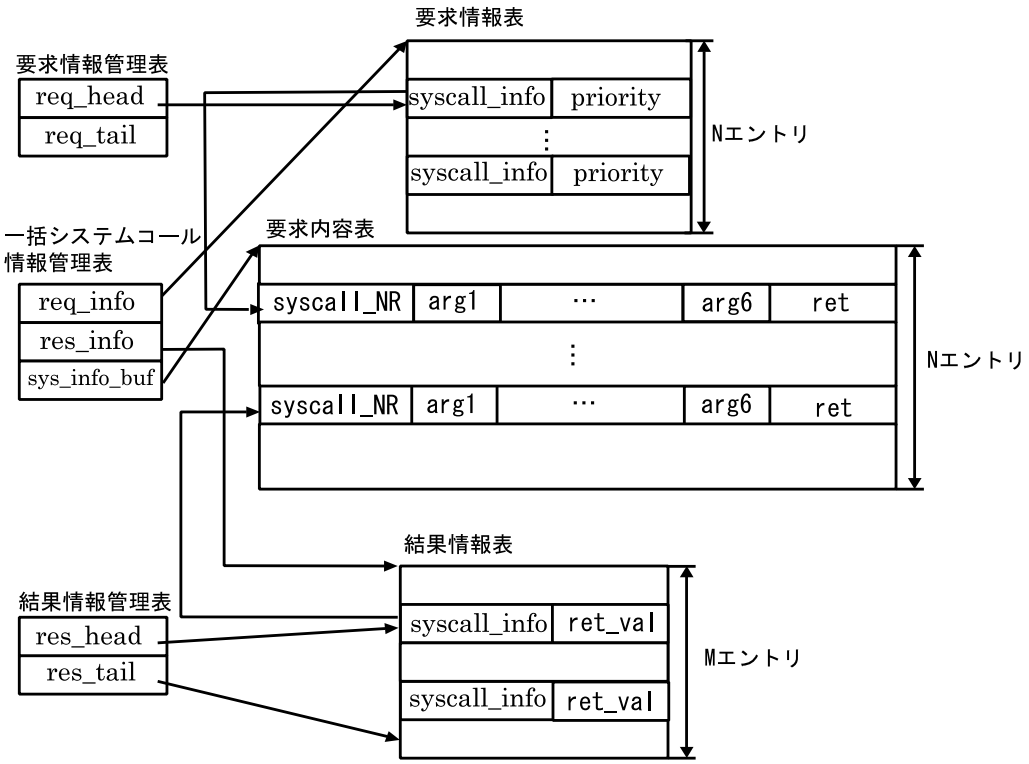


図 2: データ構造

要求情報管理表は、要求情報表を操作する際に必要な情報を保有する。要求情報管理表は二つの要素からなり、各要素は要求情報表に登録されているエントリの先頭アドレス（req_head）および要求情報表の中の次に書き込むアドレス（req_tail）を持つ。

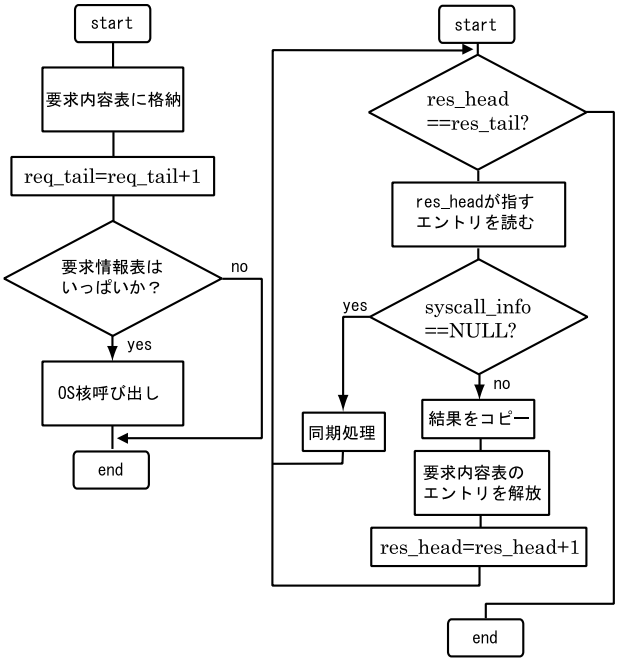
結果情報管理表は、結果情報表を操作する際に必要な情報を保有する。結果情報管理表は二つの要素からなる。各要素は結果情報表に登録されているエントリの先頭アドレス（res_head）および結果情報表の中の次に書き込むアドレス（res_tail）を持つ。

なお、req_head は Internal Kernel によってのみ操作され、req_tail は External Kernel のみが操作を行う。これによって要求情報表の排他制御を行うことができる。また、res_head は External Kernel によってのみ操作され、res_tail は Internal Kernel のみが操作することによって、結果情報表の排他制御を行う。

3.2 処理の流れ

一括システムコールの処理の流れについて述べる。図 3 に External Kernel 側での処理の流れを示す。図 3（A）はシステムコール要求を登録する処理の流れで、図 3（B）はシステムコールの結果を受け取る処理の流れである。

システムコール登録処理について述べる。



(A) システムコール登録処理　　(B) システムコール結果受け取り処理

図 3: External Kernel 側の処理の流れ

（1）要求内容表にシステムコール番号、引数、結果格納先アドレスを書き込む。

（2）要求情報表に書き込んだエントリの先頭アドレスを要求情報表に追加する。

（3）要求情報数が閾値を超えると OS 核を呼び出す。

システムコール結果受け取り処理について述べる。システムコール受け取り処理は、制御が Internal Kernel から External Kernel に移ったときに実行される。

（1）結果情報表が空かどうかを調べる。空ならばスレッドの実行を行う。

（2）結果情報表にエントリが存在する場合、res_head が指す結果情報表のエントリを得る。

（3）引いたエントリの syscall_info が指している要求内容表のエントリの ret を得る。

（4）結果情報表のエントリに書かれてあるシステムコール結果 retval を ret が指すアドレスに書きこむ。Syscall_info の値が NULL の時スレッドの同期処理を行う。

（5）参照した要求内容表のエントリを未使用にし、res_head が指す結果情報のエントリを dequeue する。

（6）（１）へ戻る。

Internal Kernel 側の処理のフローチャートを図 4 に示し説明する。

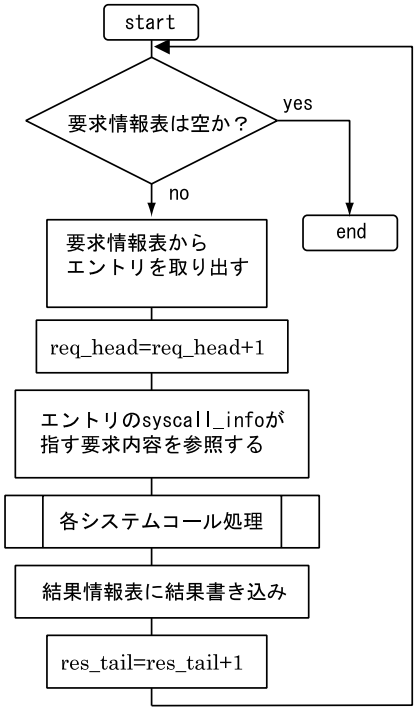


図 4: Internal Kernel 側の処理の流れ

（1）要求情報が空かどうかを調べる。空の場合プロセススケジュールを行う。

（2）要求情報表にエントリが存在する場合 req_head が指す要求情報表のエントリを得る。

（3）得たエントリの syscall_info が指してある要求内容表のエントリを得る。

（4）得た要求内容のエントリの syscall_NR に対応するシステムコール処理を呼び出す。このときシステムコールの引数に arg1〜arg6 を渡す。

（5）システムコール処理の結果を受け取ると、処理した要求内容のエントリの先頭アドレスと結果を結果情報表に書きこむ。

（6）（１）へ戻る。

4 評価

CEFOS 上で従来のシステムコール処理と一括システムコール処理の処理時間の比較を行った。測定は Celeron(300MHz) を搭載した計算機と Pentium4(1.8GHz) を搭載した計算機上で行った。評価に用いた処理の流れを図 5 に示す。図 5(A) は、従来のシステムコールを用いた処理で、図 5(B) は、一括システムコールを用いた処理である。一括システムコールの特徴を明確にするため、getpid() システムコールを用いた。getpid() はカーネル内での処理時間が短く、カーネル呼び出しの回数削減の効果が最も大きく現れるため、評価に用いるのに適していると考えられる。従来のシステムコールを用いた処理では、getpid() システムコールを 100 回発行する処理を行う。getpid() が発行される度にカーネルを呼び出す。一括システムコールを用いた処理ではシステムコール登録処理を 100 回行い、システムコール受け取り処理を行う。カーネルを呼び出すための閾値を 100 とした。つまり、システムコールが 100 個登録されるとカーネルを呼び出す。

図 5 で示した処理の流れを 1000 回実行したときの 1 回あたりの平均時間を表 5 に示す。時間の測定にはプロセッサのクロック数をカウントするカウンタを用い、2 箇所のカウンタ値の差をプロセッサの動作周波数で割ることにより算出した。

図 5 より Celeron 300MHz を用いた測定では、従来のシステムコールを用いた処理の方が、一括システムコールを用いた処理より約 1.28 倍速い。しかし、Pentium4 1.8GHz を用いた測定では、一括システムコールを用いた処理の方が、従来のシステムコールを用いた処理より約 2.10 倍速くなっている。

Celeron 300MHz を利用した測定で一括システムコールを用いた処理の方が遅いのは、システムコール登録

表 5: 処理時間の比較

使用プロセッサ	従来のシステムコール処理 (μs)	一括システムコール処理 (μs)
Celeron 300MHz	127.50	162.73
Pentium4 1.8Ghz	95.91	45.63

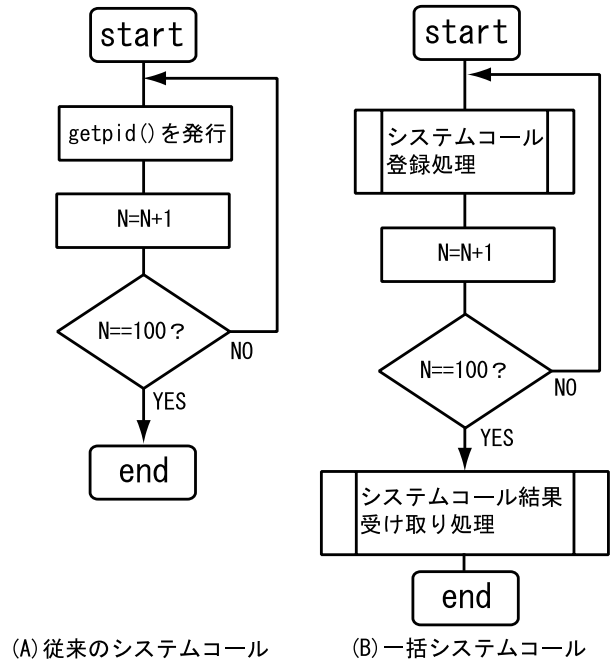


図 5: 評価プログラムの処理の流れ

処理、システムコール結果受け取り処理およびカーネル内での登録されたシステムコールに対応する処理の呼び出しにかかる時間の総和が、カーネル呼び出しによる走行モード変更にかかる時間に対して長くなったためである。

一方、Pentium4 1.8GHz を用いた測定で一括システムコールを用いた処理の方が速いのは、プロセッサの高速化によってシステムコール登録処理、システムコール結果受け取り処理およびカーネル内での登録されたシステムコールに対応する処理の呼び出しにかかる時間の総和が減少したことに対し、カーネル呼び出しによる走行モード変更にかかる時間が動作クロック数に比べ、比較的減少しなかったためである。つまり、プロセスの処理時間の中でカーネル呼び出しによる走行モード変更にかかる時間の割合が増えたためである。走行モードの変更にかかる時間の減少の割合が小さいのは、プロセッサのパイプラインの段数に原因がある。パイプラインの段数が増加すると、分岐命令や走行モード変更にかかるクロック数が増加する。Celeron ではパイプラインの段数は 10 段、Pentium4 では 20 段である。ゆえに、プロセスの全処理時間中の走行モード変更にかかる時間の割合は、Celeron 300Mhz より Pentium4 1.8GHz の方が大きくなる。

5 おわりに

カーネルの呼び出し回数を削減する一括システムコール機構について述べた。一括システムコールを利用することで、カーネル呼び出し回数を削減できる。一方、一括システムコールはシステムコールの開始時間を遅れさせるため、システムコールの応答時間が増大する問題がある。

また、一括システムコール機構の実現方式について述べた。一括システムコール処理は、システムコール登録処理、システムコール結果受け取り処理および一括システムコールから構成され、各処理の処理内容について述べた。

さらに、一括システムコール機構を CEFOS に実装し、システムコール getpid() を用いて評価を行った。その結果、Pentium4 (1.8GHz) を搭載した計算機で、100 のシステムコールを発行したとき、処理性能が 2.1 倍向上していることを示した。また、走行モード変更にかかる時間が大きなプロセッサを搭載した計算機ほど一括システムコール機構が効果的であることを述べた。

今後の課題としては、プロセスを待ち状態にするようなシステムコールに対する一括システムコール処理の評価やシステムコールを発行したスレッドの優先度やシステムコールの要求内容に基づいたシステムコール処理順序の決定法がある。

参考文献

[1] 谷口 秀夫, 日下部 茂, 棚林 拓也, 中山 大士, 雨宮 真人, “CEFOS オペレーティングシステムのスレッド管理機構,” 情処研報, 2000-OS-83, Vol.2000, No.21, pp.7-12 (2000).