社団法人 情報処理学会　研究報告
IPSJ SIG Technical Report

2004－OS－95　(4)
2004／2／26

応用プログラムの走行モード変更機構

横山和俊[†,††]　乃村能成[‡]　谷口秀夫[‡]　丸山勝巳[‡‡]

あらまし

多数のシステムコールを発行する処理のオーバヘッドを削減する新たな方式について述べる。提案する方式は、ユーザモードで実行されている応用プログラムのプロセスを、任意の時点で、スーパバイザモードで実行するプロセスへ変更する。つまり、プロセスは、応用プログラムの実行に関して、ユーザモードとスーパバイザモードの両方の走行モードを使い分けることができる。スーパバイザモードで実行される応用プログラムは、OS のシステムコール処理を関数呼出しで利用する。これにより、効率的な実行が要求される処理については、スーパバイザモードで実行しシステムコールの処理オーバヘッドを削減することが可能である。ここでは、提案方式について、検討結果を報告する。

A mechanism for switching running mode of application programs

Kazutoshi Yokoyama[†,††], Yoshinari Nomura[‡], Hideo Taniguchi[‡],
and Katsumi Maruyama[‡‡]

Abstract

This paper presents a new mechanism to reduce the overhead about processing in which a lot of system calls are issued. The proposed method changes the process of an application program running in the user mode to the process running in the supervisor mode at any time. In a word, the process can use two running modes of the user mode and the supervisor mode during executing the application program. The application program running in the supervisor mode uses the system call by the function call. As a result, the overhead of the system call processing can be reduced by executing the process in the supervisor mode for the application program to which efficient execution is required.

1. はじめに

近年の計算機性能の向上と普及により、計算機で扱うサービスの規模が増大している。また、計算機で扱うサービスの種別もトランザクション処理やマルチメディア処理を中心に多様化している。さらに、近年のインターネットの普及に伴い、計算機でサービスを実現し提供する形態も大きく変化している。すなわち、サービス毎に専用のシステムを構築する形態から、インターネットデータセンタのように、同一システム上に複数種類のサービスを実現し、負荷に応じて計算機を割り振る形態が出現している。このようなサービスの実現形態では、時刻により負荷の高いサービスが変化する。また、インターネットデータセンタの利用者のサービス品質契約に応じて、特定の応用プログラム（以降、AP と略す）や特定トランザクションのサービス品質を保証する必要がある。このように、これらの多様かつ様々な処理性能要求を持つサービスを効率良く実行するためには、サービスを実現する AP とオペレーティングシステム（以降、OS と略す）との効率的な連携が求められる。

AP は、計算機上でプロセスとして実行される。プロセスの実行状態は、プロセッサの動作モードにより、ユーザモードとスーパバイザモードの 2 つに分類できる。従来の OS では、プロセスが AP を実行する時、プロセッサの動作モードはユーザモードである。ユーザモードの場合、プロセスは、仮想空間内の AP 領域のみを用いる。一方、プロセスが OS を実行する場合、プロセッサの動作モードはスーパバイザモードである。スーパバイザモードの場合、プロセスは、AP 領域と OS 領域の両者を利用することができる。ユーザモー

†　岡山大学自然科学研究科
††　(株)NTT データ技術開発本部
‡　岡山大学工学部
‡‡　国立情報学研究所

ドからスーパバイザモードへの移行は、プロセスが、OS へシステムコールを発行することを契機に移行する。具体的には、プロセッサに対してソフトウェア割込み（trap 命令や int 命令）を発行し、プロセスからの要求を OS に通知する。スーパバイザモードからユーザモードへの移行は、システムコールからの復帰命令を OS がプロセッサに発行することにより移行する。これらの処理は、コンテキスト切り替え処理を伴う。つまり、プロセッサの動作モードの切り替え、プロセスのコンテキストの退避、および、新たなプロセスのコンテキストの復旧処理が発生する。この処理オーバヘッドは大きく、短時間で多数のシステムコール発行を伴うトランザクション処理では無視できない[1]。

AP と OS を効率的に連携される研究について、OS の構成を動的に変更する研究が行われている[2][3]。また、システムコール処理の処理オーバヘッドを削減することを目的とした研究として、AP を OS 内で走行させる研究がある[4][5]。文献[4]では、型付きアセンブリ言語を提案し、AP を安全に OS 内で動作させることを実現している。また、文献[5]の研究では、AP を OS 内で動作させるに必要な AP の改良を容易に行うことを可能にする開発環境を提案している。これらの研究では、予め決められた AP を OS 内で走行させることにより、システムコール処理のオーバヘッドの削減を実現している。

ここでは、多数のシステムコールを発行することが要求される処理を対象に、システムコール処理のオーバヘッドを削減し、効率的なプロセスの実行を可能にする新たな方式を提案する。提案方式は、AP をユーザモードで実行するプロセス（以降、ユーザモードプロセスと呼ぶ）を、任意の時点で、AP をスーパバイザモードで実行するプロセス（以降、スーパバイザモードプロセスと呼ぶ）へ変更する。つまり、プロセスは、AP の実行に関して、ユーザモードとスーパバイザモードの両方の走行モードを使い分けることができる。以降では、この方式をツインモードプロセス方式と名付ける。スーパバイザモードプロセスは、OS のシステムコール処理を関数呼出しの形態で利用できる。これにより、ユーザモードプロセスのシステムコール処理で発生したコンテキストの退避や復旧処理を不要としている。また、ツインモードプロセス方式を用いることにより、サービス品質要求の高いトランザクション処理のみを対象に、プロセス実行の効率化が図れる。さらに、提案方式は、既存の AP を変更することなく、スーパバイザモードで走行するプロセスへ変更できる特徴を有している。このため、AP を OS 内で走行させるための試験工数の削減できる。

なお、以降では、走行モードを意識しない場合、単にプロセスと呼ぶ。

2. ツインモードプロセス方式

2.1 基本方式

ユーザモードプロセスとスーパバイザモードプロセスの構成を図１に示し、以下に説明する。

図１(a)は、従来の OS のプロセスであるユーザモードプロセスである。ユーザモードプロセスでは、AP を実行する場合と OS プログラムを実行する場合で、プロセッサの動作モードと利用可能な仮想空間部が異なる。具体的には、AP 実行時には、プロセッサの動作モードはユーザモードであり、仮想空間の AP 領域のみを用いることができる。AP と OS の連携は、AP からシステムコールを発行することで実現される。システムコール xyz()が発行されると、ソフトウェア割込みを利用して OS に処理が移行する。このとき、プロセッサは、動作モードをスーパバイザモードに変更すると共に、実行プログラムやスタックを OS 領域へ切り替える。図１(b)は、スーパバイザモードプロセスの様子を示している。スーパバイザモードプロセスは、AP と OS の両方をスーパバイザモードで実行する。スーパバイザモードプロセスにおけるシステムコール処理は、AP から OS のシステムコール処理である sys_xyz()を関数呼出しすることで実現される。

ツインモードプロセス方式の概念を図２に示し、以下に説明する。

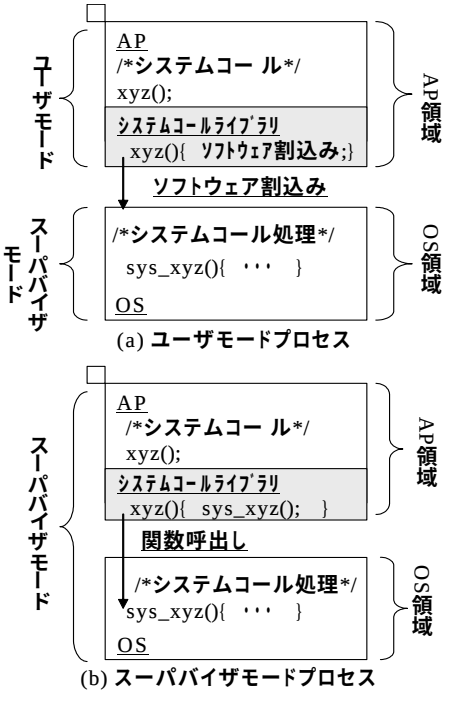


図1 プロセスの構造

(1) プロセスは、走行中にシステムコールにより、走行モードを変更する。システムコールは、プロセスが走行中の任意の時点で発行される。
(2) Change_Supervisor システムコールは、ユーザモードプロセスをスーパバイザモードプロセスに変更する。逆に、Change_User システムコールは、スーパバイザモードプロセスをユーザモードプロセスに変更する。それぞれのシステムコールでは、プロセスの走行モードを変更すると共に、AP と OS が走行する環境を設定する。具体的には、スタック部の設定や、保護領域の設定を行う。これら 2 つのシステムコールを走行モード変更システムコールと呼ぶ。
(3) プロセスの走行モードがユーザモードプロセスの場合、ソフトウェア割込みを用いて、OS へシステムコール処理を依頼する。逆に、プロセスの走行モードがスーパバイザモードプロセスの場合、関数呼出しによりシステムコール処理を呼び出す。

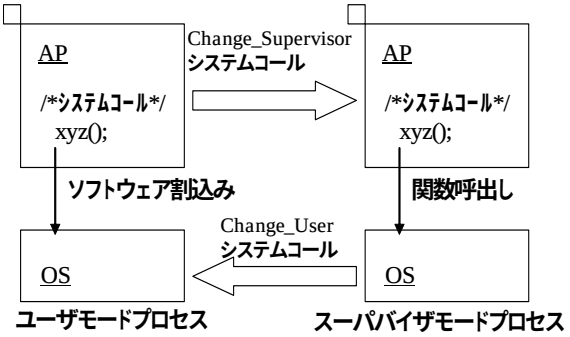


図2 ツインモードプロセスの基本方式

2.2 期待される効果と問題点

従来の OS で実現されているユーザモードプロセスは、ソフトウェア割込みを用いて、OS に処理を依頼する。ソフトウェア割込みが発生すると、OS に制御が移行すると同時に、プロセッサの動作モードをスーパバイザモードに設定する。これにより、他のプロセスが利用している計算機資源を保護すると共に、OS 自身の領域を保護している。つまり、ユーザモードプロセスの利用は、AP の不正な処理から、計算機資源や OS 領域を容易に保護できる利点がある。しかしながら、AP がソフトウェア割込みを発生させると、プロセッサの動作モードの変更、スタックの変更、プロセスコンテキストの退避と復旧処理が発生する。このため、システムコールを多数発行する AP では、ソフトウェア割込みのオーバヘッドは無視できず、処理能力の低下を招く問題がある。

一方、スーパバイザモードプロセスは、AP の実行時にも、プロセッサの動作モードがスーパバイザモードで走行する。また、OS のシステムコール処理を関数呼出しで利用する。このため、ソフトウェア割込みに伴うオーバヘッドが発生せず、処理能力の向上が期待できる。しかしながら、AP が直接 OS 機能を利用するため、他のプロセスが利用している計算機資源や OS 領域の保護が問題となる。具体的には、AP がプロセッサの特権命令（halt 命令など）を発行することや、OS 領域のメモリ内容を直接操作することが可能になる。つまり、AP の不正な処理によりシステム停止を引き起こしてしまう問題がある。

ツインモードプロセス方式は、AP により、ユーザモードプロセスとスーパバイザモードプロセスを切り替えることを可能にしている。例えば、十分に試験され、これまでの運用実績が十分である AP については、AP による不正処理の可能性が低い。このような AP については、スーパバイザモードプロセスで走行させることにより、処理の効率化を図る。一方、運用実績のない AP については、処理効率より保護を優先させ、ユーザモードプロセスとして走行させる。また、ツインモードプロセス方式は、任意の時点での走行モードの切り替えを可能にしている。これにより、例えば、特定 AP の負荷が高くなった場合にスーパバイザモードプロセスに切り替え、処理の効率化が図れる。また、サービス品質要求の高い特定のトランザクションについては、特定のトランザクション実行前にスーパバイザモードプロセスに切り替え、応答時間の短縮化を図ることができる。

3. 要求条件

ツインモードプロセス方式の実現に関して、以下の要求条件がある。

（要求１）AP の互換性

AP をスーパバイザモードで走行させるために AP のプログラム変更が発生すると、AP の試験やデバッグをやり直す必要がある。このため、既存 AP をそのまま再利用することが求められる。また、同一のプログラムで、ユーザモードプロセスとスーパバイザモードプロセスの両方で走行できることが必要である。

（要求２）OS 領域の保護

スーパバイザモードプロセスは、2.2 で述べたように、他のプロセスが利用している計算機資源や OS 自身の保護が問題となる。このため、AP の不正な処理によるシステム停止を防ぐための OS 領域の保護が必要である。この際、スーパバイザモードプロセスの利点である、処理オーバヘッドの削減効果を損なわないよう、最小限の OS 領域の保護機能を提供することが求められる。

（要求３）すみやかな走行モード切り替え
AP は、刻々と変化する環境に対処するため、処理の効率化が必要な任意の時点でプロセスの走行モードを変更する。つまり、他 AP からの走行モード変更システムコールの発行を認める必要がある。また、走行モード変更システムコールが発行された場合、即座に状態を変更することが求められる。

また、ツインモードプロセス方式を実現する際に、既存の OS への変更量の最小化と局所化を考慮する必要がある。

4. 実現方式

4.1 基本的な処理の流れ

ツインモードプロセス方式の基本的な構成を図 3 に示し、以下に説明する。

(1) プロセスは、仮想空間上にユーザモードプロセスであるか、スーパバイザモードプロセスであるかの走行モードフラグ領域を持つ。また、スーパバイザモードプロセスがシステムコール処理を関数呼出しで利用するための、システムコールアドレス領域を持つ。これら２つの領域をツインモード制御領域と呼ぶ。

(2) AP にリンクされるシステムコールライブラリは、走行モードフラグ領域を参照し、プロセスの走行モードがユーザモードプロセスであるかスーパバイザモードプロセスであるかを判定する。

(3) プロセスの走行モードがユーザモードプロセスの場合、ソフトウェア割込みを用いて、OS へシステムコール処理を依頼する。プロセスの走行モードがスーパバイザモードプロセスの場合、システムコールアドレス領域から該当するシステムコール処理のアドレスを取得し、該当アドレスへジャンプする。これら 2 つの呼出し形態を、ツインモード呼出し形態と呼ぶ。

4.2 実現上の課題

ツインモードプロセス方式を実現する基本的な課題として以下のものがある。

(a) ツインモード制御領域の実現方式
(b) ツインモード呼出しにおけるシステムコール処理の実現方式
(c) OS 保護の実現方式
(d) 走行モード変更システムコールの実現方式

以降では、各課題への対処について述べる。

4.2.1 ツインモード制御領域の実現方式

ツインモード制御領域は、ユーザモードプロセスとして動作するときには、プロセッサの動作モードがユーザモード時に参照される。一方、スーパバイザモードプロセスとして動作するときには、プロセッサの動作モードがスーパバイザモード時に参照される。つまり、プロセッサの動作モードが、ユーザモードとスーパバイザモードの両モードで参照される。一方、プロセッサの動作モードがユーザモードの場合は、AP 領域のみが参照可能である。このため、AP 領域にツインモード制御領域を確保する必要がある。

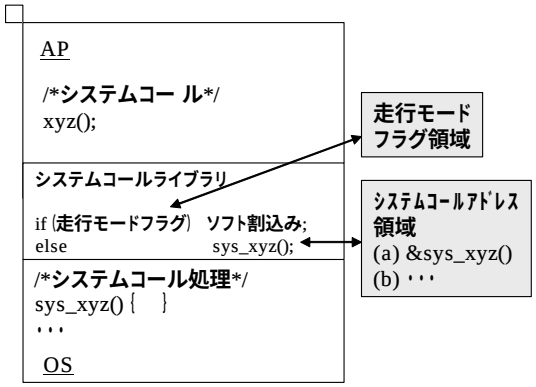


図3 ツインモードプロセス方式の基本構成

ツインモード制御領域の確保方式として、以下の２つ方式がある。

（方式 A）AP 指定方式
AP（ユーザプログラム）が明示的に、領域を確保するシステムコール（sbrk）や static 変数定義により領域を確保する。また、確保した領域のアドレスを OS に通知する。

（方式 B）OS 設定方式
プロセス生成時（fork & exec）時に、プロセスの AP 領域の固定アドレスに、OS が領域を確保する。

方式 A と方式 B を、AP の互換性（要求１）と OS の変更量の観点から比較した結果を表 1 に示す。方式 A は、AP が明示的にツインモード制御領域を確保する。このため、領域確保システムコールの呼出し処理や static 変数宣言を、応用プログラムに追加する必要があり、既存 AP の再利用ができない。また、AP が確保した制御領域のア

表1 ツインモード制御領域の確保方式の比較

	AP互換性	OS変更量
方式A	× 領域確保と領域通知システムコールの呼出しを追加	× ・領域通知システムコールの新規追加 ・プロセス毎に異なる領域管理
方式B	○ 変更なし （再コンパイルのみ）	△ fork&execの変更

ドレスを OS に通知するシステムコールを追加する必要がある。さらに、OS がプロセス毎に異なる制御領域のアドレスを管理するため、OS の制御が複雑化する。一方、方式 B は、OS が、固定領域を割り当てるため、AP の変更はない。また、固定領域の割り当ては、プロセス生成システムコール（fork & exec）を変更することで実現できる。さらに、OS は、制御領域のアドレスがすべてのプロセスの同じであるため、制御が簡単である。

以上のことより、ツインモード制御領域の実現方式は、方式 B が良いと言える。

4.2.2 ツインモード呼出しのシステムコール処理の実現方式

ツインモードプロセス方式におけるシステムコール処理は、ユーザモードプロセスとスーパバイザモードプロセスの場合で異なる。ユーザモードプロセスの場合は、ソフトウェア割り込みによりシステムコール処理を呼び出し、割り込みからの復帰によりシステムコール処理から復帰する。Linux のシステムコール処理を例に、ソフトウェア割込みによるシステムコール処理の様子を図 4 に示す。ソフトウェア割り込みが発行されると、システムコールの共通受付部であるシステムコールハンドラが最初に実行される。システムコールハンドラは、引数からシステムコール番号を取得し、該当番号のシステムコール処理を実現するシステムコール関数を呼び出す。その後、再スケジュールを行い、割り込みからの復帰命令を実行する。

一方、スーパバイザモードプロセスの場合、システムコール処理を関数呼出しで利用し、関数呼出しからの返却によりシステムコール処理から復帰する。すなわち、OS のシステムコール処理は、両方の形態の呼出しと復帰処理を実現する必要がある。ソフトウェア割込みによる呼出しと関数呼出しのツインモードでの利用を可能にする

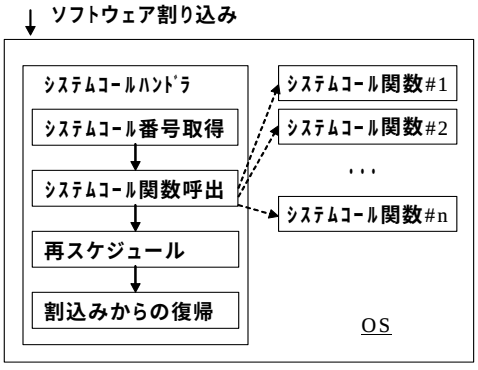


図4 ソフトウェア割り込みのシステムコール処理

システムコール処理の実現方式として、図 5 に示す 2 つの方式がある。

（方式 A）システムコール関数呼び出し

ソフトウェア割込みによるシステムコール処理は、システムコールハンドラにより受け付ける。一方、関数呼出しは、システムコールライブラリからシステムコール関数を直接呼び出す。

（方式 B）ハンドラ呼び出し

ソフトウェア割込みと関数呼出しの両方において、システムコールハンドラが受け付ける。

方式 A は、システムコール処理の処理量が少ない。具体的には、システムコールハンドラの処理を省略できると共に、システムコール関数へ渡す引数のコピー回数を削減できる。しかし、すべてのシステムコール関数のアドレスをシステムコールアドレス領域に設定し、プロセスに通知する必要がある。このため、プロセス生成時の OS 処理が複雑化する。また、システムコール関数を直接呼び出すため、システムコールからに復帰時に、プロセスの再スケジュールができない問題がある。このため、優先度の高いプロセスが存在しても、プロセス切り替えが実行されない問題が生じる。一方、方式 B は、方式 A に比べ処理量が多くなる。しかし、システムコール領域に保持するアドレスがシステムコールハンドラのアドレス

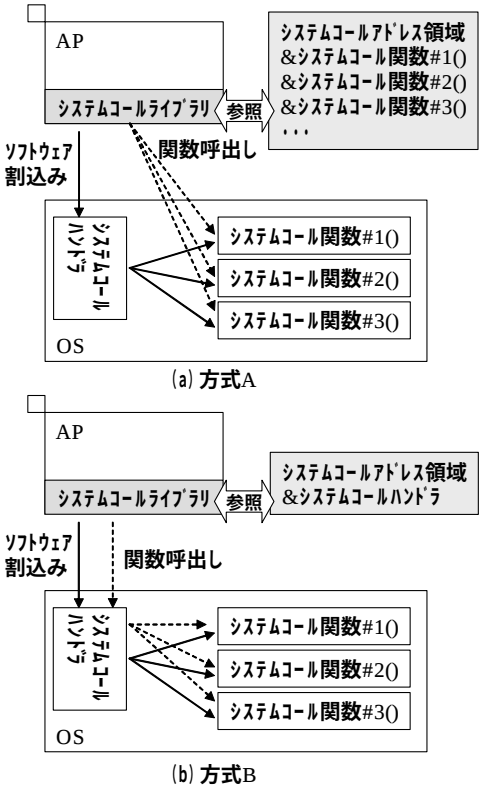


図5 システムコール処理の実現方式

のみであり、OS の制御が簡単である。

方式 A と方式 B の有効性は、ツインモードプロセス方式の適用環境に依存する。例えば、優先度の高いプロセスのみを対象にツインモードプロセス方式を適用する場合、優先度の高いプロセスが待たされることは少ない。このため、方式 A が有効となる。一方、ツインモードプロセス方式を適用するプロセスが特定できない環境では、他のプロセス処理への影響を与えないために、方式 B が有効となる。

4.2.3 OS 領域の保護

スーパバイザモードプロセスは、AP の実行時にも、プロセッサの動作モードはスーパバイザモードである。このため、不正な処理に対して保護する必要がある。AP の不正な処理は、以下の 2 つに分類できる。

(1) プロセッサの特権命令実行
AP が、プロセッサの特権命令（halt 命令など）を実行する。

(2) OS 領域への不正アクセス
AP のバグや悪意を持った AP が、OS 領域の内容を改竄し、破壊する。

(1) については、AP が利用する命令を解析することで特権命令実行を防ぐことができる。(2)については、AP の試験を十分に行うことにより、可能性を低くすることはできるが、完全に排除することはできない。このため、 (2)への対処が必要である。OS 領域の保護は、プロセッサのアドレス変換機構に依存する。通常、仮想アドレスを物理アドレスに変更するとき、ページ変換テーブルを用いる。プロセッサのアドレス変換機構は、プロセッサの動作モードと該当ページのアクセス権を比較し、OS 領域への不正アクセスを防御する。具体的には、プロセッサの動作モードがスーパバイザモードの時のみ、OS 領域へのアクセスを許可する。スーパバイザモードプロセスは、AP の実行時にも、プロセッサの動作モードはスーパバイザモードである。このため、プロセッサの動作モードがスーパバイザモードの時、AP 実行中であれば OS 領域の保護を行う制御が必要となる。

スーパバイザモードプロセスにおける、AP 実行中の OS 保護の実現方式を図 6 に示し、以下に説明する。

（方式 A）ページ変換テーブル変更方式
スーパバイザモードプロセスが AP 実行から OS 実行に移行する時に、ページ変換テーブルの OS 領域に関するアクセス権限を変更する方式である。図 6(a)に示すように、関数呼出しによりシステムコール処理を呼び出す際、OS 領域のアクセス権を Read/Write に変更する。逆に、システムコール処理から復帰する時、アクセス権を ReadOnly に設定する。つまり、AP 実行中には、アクセス権が ReadOnly のため、OS データの改竄を防止できる。

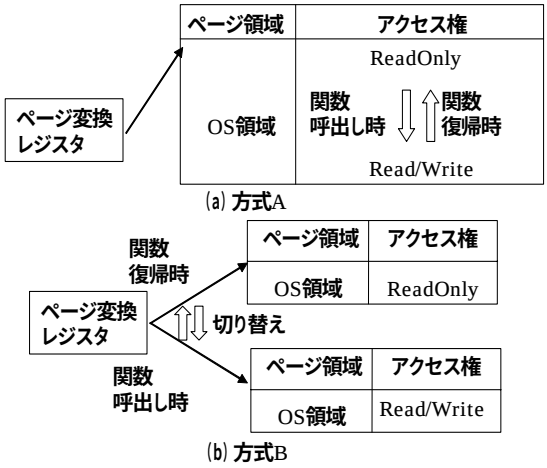


図6 OS領域の保護の実現方式

（方式 B）ページ変換テーブル切り替え方式
AP 実行中と OS 実行中で異なるページ変換テーブルを用意し、関数呼出し時と復帰時に切り替える方式である。2 つのページ変換テーブルは、OS データ部のアクセス権のみが異なる。

方式 A は、OS 領域のページ数に応じて、対応するエントリを変更する必要がある。一方、方式 B は、関数呼び出し時と復帰時にページ変換レジスタを変更するのみである。しかしながら、プロセス生成時に 2 つのページ変換テーブルを作成する必要がある。多数のシステムコールを発行する AP では、AP と OS の実行の切り替えが頻繁に発生する。このため、OS 領域の保護の実現では、処理量の少ない方式 B が有効である。

4.2.4 走行モード変更システムコールの実現方式

ツインモードプロセス方式では、走行モード変更システムコールは、システムコールが発行された時点での変更対象のプロセスの実行状態を考慮する必要がある。これは、ツインモードプロセス方式では、任意のタイミングで走行モードを変更することができ、また、他プロセスからの走行モードを変更することを許しているためである。具体的には、以下の 6 つの場合について、走行モード変更の処理を行う必要がある。

(1) 自プロセスの走行モード変更
走行モード変更システムコールを発行するプロセスが、自分自身の走行モードを変更する場合である。さらに、以下の 2 つがある。

(1a) ユーザモードプロセスからスーパバイザモードプロセスへの変更

(1b) スーパバイザモードプロセスからユーザモードプロセスへの変更

(2)他プロセスの走行モード変更

他プロセスの走行モードを変更する場合である。この変更の場合、変更対象となるプロセスの走行モードと実行しているプログラム（APかOS）により、以下の4つの場合を考慮する必要がある。

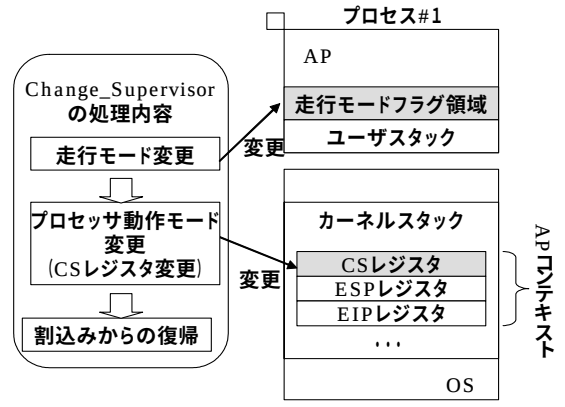
(2a) ユーザモードプロセスからスーパバイザモードプロセスの変更、かつ、AP実行中

(2b) ユーザモードプロセスからスーパバイザモードプロセスの変更、かつ、OS実行中

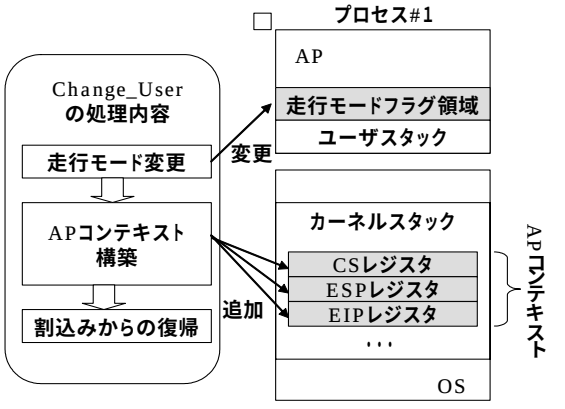
(2c) スーパバイザモードプロセスからユーザモードプロセスへの変更、かつ、AP実行中

(2d) スーパバイザモードプロセスからユーザモードプロセスへの変更、かつ、OS実行中

自プロセスの走行モード変更を例にした走行モード変更システムコールの基本的な処理を図7に示す。図7は、Pentiumプロセッサ上のLinuxでの実現を想定している。図7(a)は、Change_Supervisorシステムコールにより、ユーザモードプロセスからスーパバイザモードプロセスに変更する様子を示している。Change_Supervisorシステムコールは、主に2つの処理を行う。AP領域の走行モードフラグ領域の更新処理と、カーネルスタックのCSレジスタ領域の更新処理である。CSレジスタの値は、プロセッサの動作モードが格納されている。図7(a)のプロセス#1について、OS実行からAP実行に移行する際、プロセッサは、カーネルスタックのAPコンテキストを読み込む。つまり、CSレジスタをスーパバイザモードに設定しておくと、AP実行においてもプロセッサの動作モードがスーパバイザモードとなる。図7(b)は、Change_Userシステムコールにより、スーパバイザモードプロセスからユーザモードプロセスに変更する様子を示している。この場合、カーネルスタックにAPコンテキストが存在しないため、APコンテキストを構築する。その際、CSレジスタをユーザモードに設定することで、AP実行のプロセッサの動作モードがユーザモードに変更される。



(a) Change_Supervisorの基本処理



(b) Change_Userの基本処理

図7 走行モード変更システムコールの基本処理

他プロセスの走行モード変更機能の実現は、走行モードの変更処理を行う時期により、複雑さが異なる。走行モード変更処理を行う時期により、以下の2つの実現方式がある。

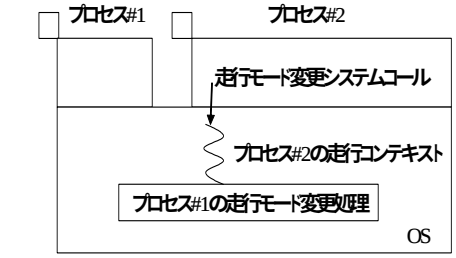
（方式A）システムコール発行プロセス方式

走行モード変更システムコールを発行したプロセスの走行コンテキストで変更対象のプロセスの走行モード変更処理を行う方式である。図8(a)にシステムコール発行プロセス方式の様子を示す。図8(a)は、他プロセスであるプロセス#2がプロセス#1の状態を切り替える場合を示している。この場合、プロセス#2の走行コンテキストで、プロセス#1の状態が変更される。

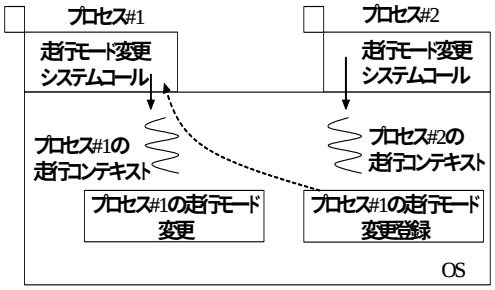
（方式B）変更対象プロセス方式

変更対象となるプロセスの走行コンテキストで変更を行う方式である。図8(b)にこの様子を示す。図8(b)は、他プロセスであるプロセス#2がプロセス#1の状態を変更する場合を示している。この場合、プロセス#2が発行したシステムコールは、プロセス#1の走行モード変更の要求を登録するのみである。プロセス切り替えにより、プロセス#1の走行が再開された場合、プロセス#1は、走行モード変更要求の有無を確認する。走行モード変更要求が登録されている場合、プロセス#1が自分自身の走行モードを変更する。

システムコール発行プロセス方式（方式A）と変更対象プロセス方式（方式B）の両者において、走行モード変更処理の実現の複雑さを比較する。比較した結果を表2に示す。方式Aの他プロセスの走行モード変更は、(2a)～(2c)までは、自プロセス変更時の基本処理を適用できる。これは、



(a) システムコール発行プロセス方式



(b) 変更対象プロセス方式

図8 他プロセスの走行モード変更の実現方式

走行モード変更システムコールが発行された時点でのカーネルスタックの内容が、空であるかAP コンテキストのみが存在しているからである。一方、(2d)の場合、カーネルスタックに、OS 実行時のコンテキストが存在する。そのため、カーネルスタックの最下位（ボトム）に、AP コンテキストを挿入し、カーネルスタックを再構築する必要がある。このため、アドレスの変換処理を行う必要があり、処理が複雑化する。一方、方式 B は、基本処理をすべての場合に適用できる。このため、走行モード変更処理の実現が容易である。

方式 A と方式 B について、走行モード変更の即時性を比較する。方式 A は、走行モード変更システムコールの処理中に走行モードが変更されるため、即時に変更される。一方、方式 B は、変更対象のプロセスが走行するまで遅延する。しかしながら、多数のシステムコールが発行される処理においては、遅延時間は十分に短い。このため、プロセスの処理の効率化に与える影響が少ないと考える。

5. おわりに

ユーザモードで応用プログラムを実行するプロセスを、任意の時点で、スーパバイザモードで実行するプロセスへ変更することで処理の効率化を図るツインモードプロセス方式を提案した。スーパバイザモードで走行するプロセスは、OS のシステムコール処理を関数呼出しで利用する。これにより、ユーザモードで応用プログラムを実行する際に発生するコンテキストの退避や復旧処理を不要にし、処理の効率化を実現している。

表2 実現方式の比較

		走行モード変更処理内容	
		方式A	方式B
自プロセス変更	(1a)	Change_Supervisor 基本処理	Change_Supervisor 基本処理
	(1b)	Change_User 基本処理	Change_User 基本処理
他プロセス変更	(2a)	(1a)と同様	・変更要求登録 ・(1a)処理
	(2b)	(1a)と同様	・変更要求登録 ・(1a)処理
	(2c)	(1b)と同様	・変更要求登録 ・(1b)処理
	(2d)	・Change_User 基本処理 ・カーネルスタック再構築	・変更要求登録 ・(1b)処理

また、ツインモードプロセス方式は、任意の時点で、走行モードを切り替えることを可能にしている。このため、ツインモードプロセス方式を用いることにより、サービス品質要求の高いトランザクション処理のみを対象に、プロセス実行の効率化が図れる。

今後は、ツインモードプロセス方式の実現の詳細化を図る。また、実装と評価を行い、有効性を検証する。

文献

[1] 箱守聰，谷口秀夫，``高負荷オンライントランザクション処理を実現するオペレーティングシステム，" 信学論 D-I, vol. J84-D-I, no.6, pp.627-638, June 2001.

[2] B . N. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. Fiuczynski, D. Becker, S. Eggers, and C. Chamvers, ``Extemsibility, safety, and performance in the SPIN operating system, " Proc. of 15th ACM Symposium on Operating systems Principles, pp.267-284, Colorado, USA, Dec. 1995.

[3] 高橋雅彦，田中淳裕，立川江介，``キャッシュサーバのカーネル内実装による高速化の実現，" 情報処理学会シンポジウムシリーズ(コンピュータシステムシンポジウム), vol.2002, no.18, pp.69-74, Dec. 2002.

[4] 前田俊行，住井英二郎，米澤明憲，``Linux/TAL:型付きアセンブリプログラムのカーネルモード実行方式，" 第 4 回プログラミングおよびプログラミング言語ワークショップ予稿集，March，2003.

[5] 佐藤喬，中村嘉志，多田好克，``アプリケーションプログラムのカーネル内実行による高速化，" 情報処理学会第 43 回プログラミング・シンポジウム報告集，pp.153-160 ，2002.