マイクロカーネルOSの事象トレース機能の実装と評価

2F－6

小川 武雄　遠藤 幸典　山口 義一
三菱電機（株）情報技術総合研究所

1.はじめに

システムの規模に最適なOS機能モジュールを取捨選択でき、アプリケーション開発の共通プラットフォームを提供可能なマイクロカーネル（以下μカーネル）OSの需要が高まっている。著者らはOS事象トレース機能を従来のモノリシックOSに対して開発したが、μカーネルOSに対しても、同様のOS事象トレース機能は必須である。しかし、従来の実装では、カーネルソースコードへの直接のトレースコード挿入が必要であり、事象トレース機能の保守性・移植性が低かった。そこで、この問題点を改善する、μカーネルOSの事象トレース機能の実装方法を示す。

2.課題

従来のモノリシックなOSに実装した事象トレース機能の課題を以下に挙げる。

① カーネルソースコードへの直接のトレースコード挿入が必要であり、事象トレース機能の保守性・移植性が低い。

② 事象毎に同じような事象トレース処理をソースコードの適所に手作業で埋め込む必要があり非効率である。

3.基本設計

上記の課題を解決するために、μカーネルの構造を利用して、元のカーネルソースコードに手を加えずに従来と同等の事象トレース機能を実現する設計を行った。基本的なアイデアを以下に述べる。

μカーネルOSは、OSの機能モジュールが分離された構造をもっている。このモジュールの間に事象トレースモジュールを追加し、モジュール間の呼び出しを記録することによって事象トレースを実現する（図1参照）。例えば、システムコールの事象トレースであれば、システムコールサポートモジュールが呼び出される前に、トレースモジュールを経由することにより、システムコールの事象トレースが可能である。トレース処理を別モジュールに分離することで、保守性･移植性を確保できる。

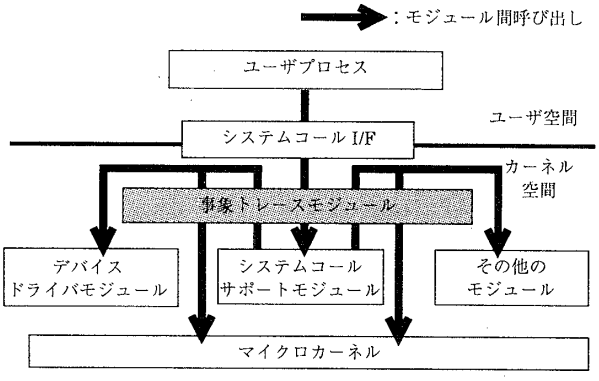

図 1 OS事象トレースの基本的アイデア

4.実装

4.1.μカーネルの構造

今回、著者らが実装の対象としたμカーネルでは、機能モジュール間の機能呼び出しが、スタブ関数を経由して呼び出す構造になっている。図2に、このμカーネルの構造を示した。機能モジュールAが機能モジュールBのfunc()を呼び出す場合を例にして、処理の流れを説明する。まず、機能モジュールAが機能モジュールBのfunc()に対応するスタブ関数を呼び出す…①。スタブ関数はジャンプテーブルからfunc()に対応するアドレスを読み込む…②。次に、読み込んだアドレスに処理をジャンプする。機能モジュールBがμカーネルOSに組み込まれていれば、ジャンプテーブルにはfunc()のアドレスが格納されているため、func()が実行される…③。一方、機能モジュールBが組み込まれていない場合は、ジャンプテーブルにはダミー関数のアドレスが格納されているため、ダミー関数が実行される。…③’ どちらの場合でも、func()の呼び出しに支障が起こらないようになっている。

また、このスタブ関数はモジュール間の依存関係を記述したモジュール間依存定義ファイルから、カーネルコンパイル時にスタブ関数生成ツールによって自動生成されてリンクされる。この様子は図2で

An Implementation of Event Tracer for Micro-Kernel OS
Takeo Ogawa, Yukinori Endo, Yoshikazu Yamaguchi
Mitsubishi Electric Corp.

は点線で表わしている。

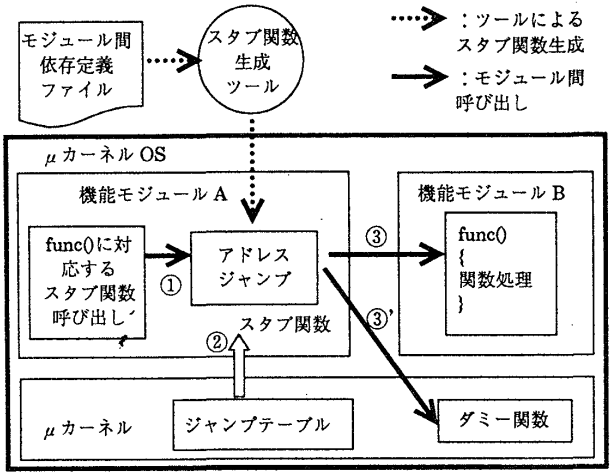

図 2 実装対象のμカーネルの構造

4.2.実装方法

機能モジュールの機能呼び出し時に、必ずスタブ関数が実行される。よって、このスタブ関数にトレースモジュールを呼び出すコードを埋め込み、モジュール間呼び出し時に、トレースモジュールを経由するようにする。図3の網がけは、図2より変更・追加されているところを示している。機能モジュールBのfunc()を呼び出す際に、分岐（A）が③の前に挿入される。トレースモジュールを呼び出すスタブ関数は、トレースモジュール呼び出しフラグを追加したモジュール間依存定義ファイルから、トレースモジュール呼び出しスタブ関数生成ツールによってカーネルコンパイル時に自動生成してリンクする。この様子を図3の点線で示す。トレースモジュールでは、トレース情報のバッファへの書き出しを行う。

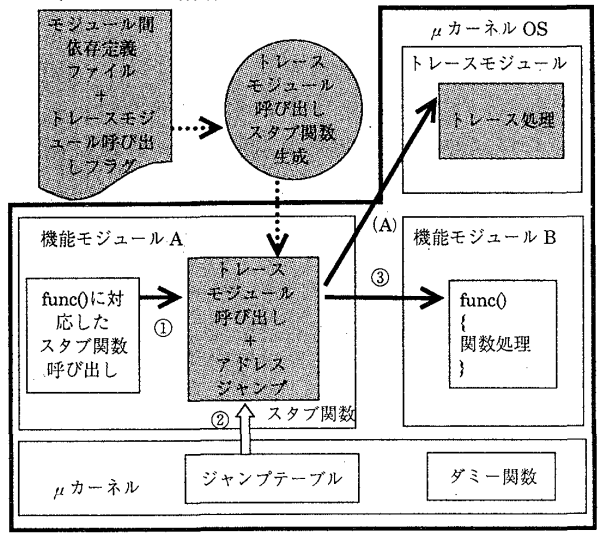

図 3 トレースモジュールを有するμカーネル

このトレースモジュール呼び出し部分は、スタブ関数生成ツールにより自動生成するため、トレースコードの埋め込みを自動化することができる。更に、この方法ではソースコードへの直接の変更は伴わない。

5.性能測定

本設計を実装したμカーネルを使って、性能測定を行った。トレース1回あたりの、オリジナルのμカーネルOSに対するオーバヘッド時間を図4に示す。“トレースON”時には、トレース情報のメモリへの書き込み処理が行われる（160バイト）。“トレースOFF”時には、書き込み処理を行わない。測定はMMX Pentium 166MHz、16Mメモリの PCで行った。

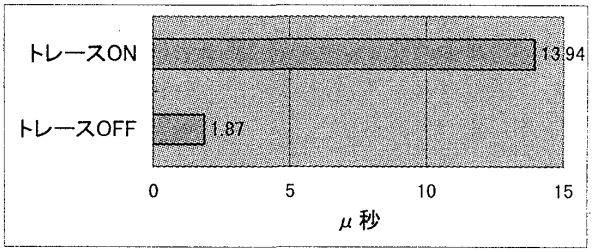

図 4 トレース処理によるオーバヘッド

“トレースON”時のオーバヘッドは約14μ秒と大きいが、トレース情報を減らせばそれだけオーバーヘッドも減少する。性能的には従来の実装と同等のオーバヘッドであり、実用領域と判断する。

6.評価・まとめ

本設計では、保守性・移植性の問題について、事象トレース処理を別モジュールに分離することで解決した。実装の作業効率については、スタブ関数の自動生成ツールの拡張により、トレースコード埋め込みの自動化により解決した。また、性能的にも十分実用可能な結果を得た。

この結果、μカーネルに追加した新しい機能モジュールであっても容易に事象トレース機能を実現できるメリットも考えられ、本設計の効果は大きい。欠点としては、本設計だけでは、事象トレースの対象がモジュールとして機能分離されていないと、事象トレースができないことが挙げられる。欠点を補うには、従来の実装を併用するしか現状はない。今後は、本設計を適用できない事象トレースについて、別の視点で移植性・作業効率の向上をはかる方法を検討していく予定である。