B-013

SMPシステムに対応するLinux稼動中パッチ方式の検討

Linux dynamic patching method for enterprise SMP system

畑崎　恵介†　　舩生　真紀子‡
Keisuke Hatasaki　Makiko Funyu

1．はじめに

近年、企業や公共の情報システムのオープン化やコスト削減へのニーズの増大に伴って、オープンソースソフトウェア（以下 OSS）が注目を集めている。特に OSS の代表的な OS である Linux は、企業や公共の基幹業務への採用が進みつつある。

Linux のような OSS は、ソースコードと開発形態の透明性から、バグやセキュリティホールに対する迅速な対応が可能でシステムの安全性を高く保つことが期待できる。その半面、安全性を確保し続けるには、頻繁に発行される修正パッチの速やかな適用が不可欠である。しかし、システムに対するパッチ適用には、システムのリブートが必要となるため、頻繁なパッチ適用はシステムのダウンタイム増大を招くことになる。

そこで、ダウンタイムを増大させることなく Linux カーネルにパッチを適用するため、稼動中パッチ方式について検討した。本稿では、基幹業務で多く採用されている SMP システムへの適用を目的として、全プロセッサ同期式の稼動中パッチ方式の実現について述べる。

2. 稼動中パッチ方式の実現方法

Linux カーネルに対して稼動中にパッチを適用するためには、(1)Linux カーネルに対して稼動中に新しいオブジェクトコードを追加し、(2)古いコードが提供する機能を呼び出した時に、追加した新しいコードを呼び出す必要がある。これらの実現方法を以下に述べる。

2.1　稼動中カーネルへのオブジェクトコード追加

稼動中の Linux カーネルへ新規オブジェクトコードを追加するために、Linux のカーネルモジュールの機能を利用する。この機能を利用することで稼動中のカーネルに矛盾なく新規オブジェクトコードを追加することができる。このため、適用するパッチはカーネルモジュール（以下パッチモジュール）として実装する。書き換える機能はカーネル関数の単位として、パッチモジュールを以下の手順で作成し、カーネルに追加する。

① パッチを適用すべき個所を含む古い関数(パッチ適用先の関数)を元に、パッチ適用後の新しい関数を作成
② 作成した新しい関数を含むパッチモジュールを作成
③ 作成したパッチモジュールを、モジュール組み込みプログラムである insmod を使ってカーネルに組み込む

なお、モジュールを組み込む場合、モジュールのオブジェクトからモジュール組み込み先カーネル内のグローバル・カーネル・シンボル(変数および関数)に対する参照を解決するためには、対象となるシンボルはエクスポートされていなければならない。

† （株）日立製作所　中央研究所
‡ （株）日立超 LSI システムズ

2.2　古いコード実行時に新しいコードを実行

カーネルの稼動中は、すでに古いコードがメモリ上にロードされているため、新しいコードをカーネルに追加しても、プロセッサは古い命令コードの実行を続けることになる。先に述べたように、稼動中パッチ方式ではパッチモジュールの追加により、更新すべきコードを含む新しい関数をカーネルに追加する。ここで、古いコードを含む関数の先頭に分岐コードを挿入することで、古いコードを含む関数が呼び出された際に、パッチモジュール内の新しい関数へと分岐させることができる。

図１に稼動中パッチ方式の処理の流れを示す。

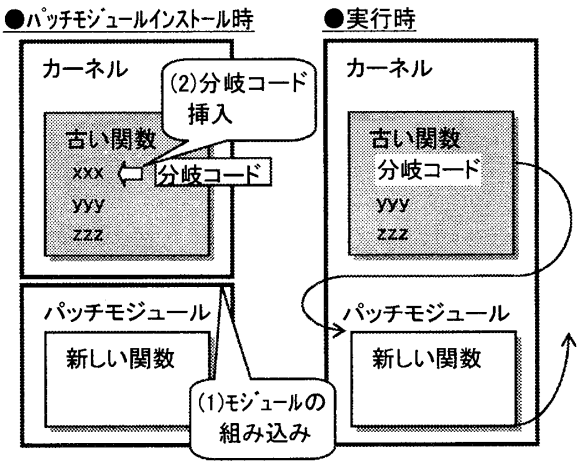

図 1. 稼動中パッチ方式

3.SMP システムにおける稼動中パッチの課題

SMP システムでは、複数のプロセッサが並列にカーネルコードを実行可能である。このため、稼動中パッチを実現するには、次のような課題が存在する。

(1) 分岐コード挿入時のプロセッサ間競合
(2) 新旧２つの関数の並列実行による不整合

3.1 分岐コード挿入時のプロセッサ間競合

先に述べたように、稼動中パッチ方式では、古い関数が呼び出された際に新しい関数が呼び出されるように、分岐コードを古い関数の先頭に挿入する。SMP システムでは、あるプロセッサがこの分岐コードを挿入する際に、別のプロセッサが書き換え対象となっているコード部分を実行中の可能性がある。このとき、書き換え対象コードを実行中のプロセッサが既に読み出したコードと、これから読み出すコードとで不整合が発生する場合があるため、システムがクラッシュする可能性がある。（図２参照）

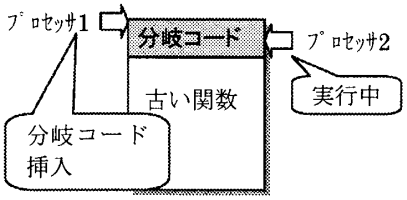

図２．分岐コード挿入時の競合

3.2 新旧２つの関数の並列実行による不整合

SMP システムでは、あるプロセッサが古い関数を実行中に分岐コードを挿入すると、一時的に古い関数を実行するプロセッサと、新しい関数を実行するプロセッサが並存する場合がある。また、複数の関数にまたがる機能を更新するため、複数の関数を同時に書き換える場合でも、同様の問題が発生する。このようなケースでは、古い関数を実行するプロセッサと、新しい関数を実行するプロセッサとでは処理の一貫性が保たれない場合がある。（図３参照）

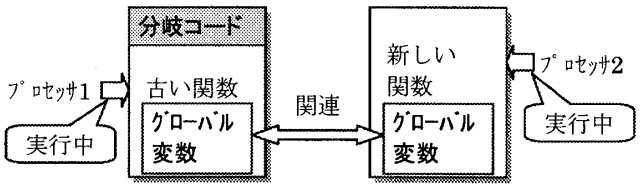

図３．古い関数と新しい関数の並存

4. プロセッサ同期式稼動中パッチ方式の開発

上記の課題を解決するには、分岐コードを挿入する際に、全てのプロセッサがクリティカルな処理を実行中でないことを保証しなければならない。そこで、全プロセッサがクリティカルな処理中でない箇所で同期をとり、同期がとれた時点でパッチ適用するプロセッサ同期式稼動中パッチ方式を実現する必要がある。以下にその実現方法について述べる。

4.1 非同期システムトラップによる同期

稼動中パッチ方式が更新の対象としているのはカーネルコードである。そこで、全てのプロセッサがカーネルモードにない場合は、全てのプロセッサがクリティカルな領域を実行中でないことを保証することができる。そこで、カーネルモードからユーザモードに切り替わる以下の箇所（カーネルの出口）でプロセッサの処理をトラップしてウェイトし、全プロセッサの同期をとる。

- 割り込み・例外の出口
- システムコールの出口

4.2 実行手順

上記方式の実現のため、プロセッサ数分の同期フラグを用意し、同期フラグが全てセットされるまでビジーウェイトすることで同期処理を実現する。図４にその実行手順を示す。ここで、割り込みハンドラも更新の対象とするため、ビジーウェイト中は割り込み禁止にする必要がある。

5.まとめ

本稿では、SMP システムでの稼動中カーネル更新方式について検討し、非同期システムトラップによるプロセッサ同期式稼動中パッチ方式の開発について述べた。この方式の実現により、基幹業務で多く採用されている SMP システムにおいて、Linux カーネルの稼動中の更新を実現し、ダウンタイムの大幅な削減を期待できる。

なお、本方式はプロトタイプとして実装済みであり、本稿で述べた想定処理の動作確認を実施済みである。今後は実業務への適用に向けて、実際に配布されているパッチを利用した稼動中パッチ適用可否の検証に取り組む所存である。

参考文献：
[1]D.P.Bovet/M.Cesati 著，高橋監訳，畑崎ら訳 「詳解 Linux カーネル第２版」 オライリー・ジャパン (2003-6)
[2]畑崎恵介 「Linux 稼動中パッチ方式の開発」 情報処理学会(2004-3)

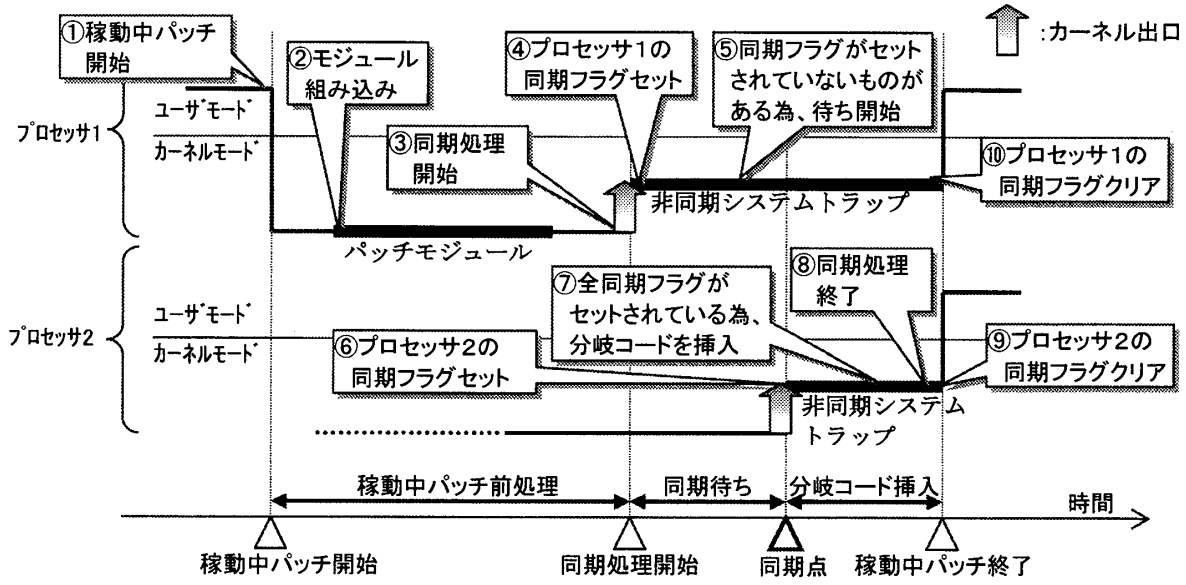

図４．プロセッサ同期式稼動中パッチ方式の動作手順