Linux稼動中パッチ方式の開発

畑崎 恵介
(株)日立製作所 中央研究所

1．はじめに

Linux は低コストの OS として急速にシェアを拡大しており、企業基幹業務における採用のニーズも高まっている。

Linux のようなオープン・ソース・ソフトウェアは、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、安全性確保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、カーネルに対するパッチ適用にはシステムリブートを伴うため、頻繁な実施はシステムのダウンタイム増大を招くことになる。

本稿では、ダウンタイムを増大せずにカーネル更新を実現する、パッチ適用方式を提案する。

2．パッチ適用における課題

バグやセキュリティホールに対処するためのカーネル更新には大きく二つ方法がある。一つは実行形式のカーネルの一部を書き直す形でパッチを適用する方法、もう一つはカーネルのソース自体を修正して新規リビジョンとしてカーネルを作成し直す方法である。これら方法は、後者はもちろん前者でも更新後のカーネルを利用するために、一部システムを除いて、システムのリブートを必要とすることが一般的である。リブート中のサービスの中断を防ぐためには、システムを二重化することが考えられる。その場合、片系のリブート中に他方によるサービスの継続が可能であるが、コストの上昇や、システムの複雑化等が問題となる可能性がある。

上記を踏まえ、ここではリブート不要のカーネルパッチ適用を実現する稼動中パッチ方式の実現を課題として取り組む。

3．稼動中パッチ方式の概要

稼動中パッチ方式では、実行中のカーネルに動的にオブジェクトを追加し、かつ、古いコードが提供する機能を呼び出したときに新しいコードを呼び出すようにする必要がある。

Development of LINUX Dynamic Patch Application Method
Keisuke Hatasaki
HITACHI, Ltd.

従って、稼動中パッチ方式は、次の項目を実現することが不可欠である。

(a) オブジェクトコードの動的な追加
(b) 新規コード呼び出しに必要な命令列書き換えのアトミック性保証
(c) 新旧コードの関係付け

以下、本稼動中パッチ方式について、上記項目への対応方針を示す。

3.1 オブジェクトコードの動的な追加

カーネルへの動的な新規オブジェクトコード追加のために、Linux のモジュール機能を利用する。モジュール機能の利用により、動的なパッチ適用の際、実行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。パッチをモジュールとして作成するため、稼動中パッチ方式では関数単位の書き換えを行う。つまり、パッチを当てる箇所を含む関数を作成し、それをカーネルに組み込む。

稼動中パッチ用モジュールは、次の手順で作成し、カーネルに組み込む。

(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、パッチを適用した新しい関数を作成。
(2) モジュールとしてカーネルに組み込める形式で、新しい関数からオブジェクトファイル(稼動中パッチ用モジュール)を作成。
(3) 作成した稼動中パッチ用モジュールを、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み。

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。

3.2 命令列書き換えのアトミック性保証

稼動中パッチ実現のためには、古い関数を実行する代わりに、組み込んだモジュール内の新しい関数を実行するようにする必要がある。古い関数の代わりに新しい関数が実行されるようにするためには、古い関数のバイナリコードの先頭部分を書き換えて、何らかの方法(分岐命令やトラップ命令など)でモジュール内の新しい関数を呼ぶようにすればよい。ただし、この書き換えは、少なくとも書き換え対象部分をちょう

ど CPU が実行中であった場合や、書き換え中に CPU が対象部分を実行しようとした場合に不整合が生じないようにしなくてはならない。従って、書き換えはアトミックに行う必要がある。

アトミック性を確保するためには、例えば CPU の最小命令語長で分岐処理を実現できる命令へ書き換える方法や、同期点を用意して書き換えが完了するまでその先に進まないようにする方法などが考えられる。

3.3 新旧コードの対応付け

古い関数が呼ばれた場合に新しい関数を呼び出すようにするために、古い関数と新しい関数の対応付けを管理することが必要である。このため、カーネルには、古い関数のアドレスと新しい関数のアドレスを記録しておく機構と、古い関数への呼び出しがあった場合に新しい関数を呼び出す機構をあらかじめ静的に組み込んでおく。これらの機構により、稼動中パッチ用モジュールをカーネルに組み込むときに、指定された古い関数のアドレスを用いて新旧関数の対応付けを行う。

4.稼動中パッチ方式プロトタイプ

以上考察に基づき、次のような前提で稼動中パッチ方式のプロトタイプを作成した。プロトタイプの目的は稼動中パッチ方式の原理的な動作確認である。

- ・i386 アーキテクチャで実装する。
- ・新規コード呼び出しに必要な命令列書き換えのアトミック性保証には、int3 命令によるソフトウェア割り込みを利用する(int3 命令は 1Byte 長で、1cycle でアトミックに書き換え可能なため)。
- ・SMP による関数の同時実行を許すが、稼動中パッチ適用の同期点は実装しない。従って、稼動中パッチ適用の前後における新旧関数実行結果の影響が、グローバルにはないものとする。

プロトタイプの概要を図 1 に示す。具体的な手順は次のようになる。

(1) 作成した稼動中パッチ用モジュールを insmod プログラムによりカーネルに組み込み。

(2) insmod 実行に伴うモジュールの初期化処理により、モジュール内の稼動中パッチ登録関数が呼ばれ、新旧関数の対応表に新しい関数を登録。

(3) (2)と同様にモジュールの初期化処理で、古い関数のテキストの先頭に int3 命令を挿入

(4) 古い関数が呼ばれたとき、int3 命令によってソフトウェア割り込み発生。ソフトウェア割り込みハンドラは、新旧関数の対応表を参照して古い関数からの int3 割り込みであることを検知し、割り込みの戻りアドレスであるスタックのインストラクションポインタの値を登録されている新しいカーネル関数の先頭アドレスに変更。

本プロトタイプで、システム稼動中にカーネルパッチを適用可能なことが実証でき、本方式の有効性を確認した。

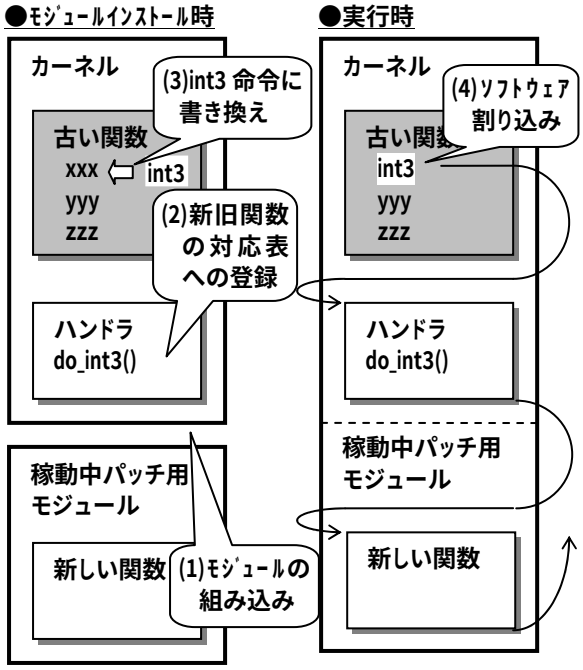

図 1. 稼動中パッチ方式プロトタイプの概要

5．まとめ

リブート不要で動的なカーネル更新を実現することを目的とした稼動中パッチ方式について検討した。検討結果に基づくプロトタイピングにより原理的な動作テストを行い、本方式の有効性を確認した。本方式により、Linux のオープン･ソース･ソフトウェアとしての透明性と品質の高さという利点を保ちつつ、ダウンタイムを増大しないカーネル更新が可能になる。

一方、本方式はプロトタイプ段階であり、実際に適用するにはいくつかの課題がある。今後、本稼動中パッチ方式の実運用への適用に向け、次のような課題に取り組んでいく。

- ・稼動中パッチ方式における命令書き換え同期点の実装。
- ・実際に配布されているパッチの分析による稼動中パッチ適用可否の検証。
- ・通常のパッチを含めたパッチの依存関係管理の実現。