様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

平成　３０　年　　５　月　２５　日現在

機関番号：　１３９０３
研究種目：　基盤研究(C)（一般）
研究期間：　2015～2017
課題番号：　１５Ｋ００９５

研究課題名（和文）仮想計算機環境におけるプロセスの継続実行が可能なカーネル更新方式

研究課題名（英文）Kernel Update by Migrating User Processes between Virtual Machines

研究代表者
斎藤　彰一（Saito, Shoichi）

名古屋工業大学・工学（系）研究科（研究院）・教授

研究者番号：７０３０４１８６

交付決定額（研究期間全体）：（直接経費）　　3,500,000円

研究成果の概要（和文）：本研究の成果としては，仮想計算機環境において，カーネル間のデータの移行によるユーザプロセスの移送が、システムコールによって作成されるデータと組み合わせることで可能であることを示したことと，複数の仮想計算機環境における障害検出方式について，カーネル間でメモリ参照を行う方式の有効性を確認したことである．しかし，申請時点で上げた課題の一部について解決することはできなかった．これは，カーネルバージョンに依存するカーネルデータ構造が，当初予想よりも複雑であったためである．特に仮想環境の構築が十分に行えなかった．

研究成果の概要（英文）：As a result of this research, I showed that it is possible to migrate user processes by combining data transfer between kernels with data created by system call, and confirmed the new fault detection method in multiple virtual machine environments. However, I could not solve some of the issues raised at the time of application, because the kernel data structure depending on the kernel version was more complicated than originally expected. Especially the virtual environment could not be constructed sufficiently.

研究分野：　情報工学

キーワード：　耐障害性向上　障害検知　仮想計算機環境　多重OS　バージョンアップ

１．研究開始当初の背景

カーネルの動的更新の研究は，関数単位で修正したコードをカーネルの実行コードに追加し，問題のある関数の代わりに追加されたコードを呼び出す方式が主流である．代表的な方式として Ksplice[1]がある．この方式は，カーネルを部分的に書き換えるため，関数単位程度の小規模な変更には適しているが，大規模な修正やバージョンアップには多数の関数の変更が必要になり適用が難しいという問題がある．本研究では，カーネルの部分的な更新ではなく，完全にカーネルを入れ替える方式を実現する．カーネルの入れ替えにより，パッチによるメモリ消費やパッチに起因する不具合の発生を防止できる．さらに，新カーネルは新規インストールしたバイナリと完全に同一であり，変更の規模に関係なく新規インストールと同等の実行環境を得ることができる．

カーネルを入れ替えることでカーネルの障害発生に対応した研究として OtherWorld[2]がある．OtherWorld は，障害により停止したカーネルの中からプロセスに関するデータを取り出し，新カーネルに移送して復元することでユーザプロセスを移送させて継続動作させる機構である．また，同様の方式をカーネルの更新に発展させた Seamless Kernel Updates[3]がある．しかし，これらの研究には，カーネルデータの変換方法が未定義であり，また仮想計算機環境を想定していない．さらに，耐障害性向上とカーネル更新を同時に対応できる方式がない．プロセスの再開までに要する時間が長いという課題がある．

本提案方式は，これらの課題を解決して，安定した計算機環境を構築することを目的とする．このために，仮想計算機環境を用いて新旧カーネルを多重実行させ，更新時には旧カーネルのメモリを走査して実行途中のユーザプロセスに関するカーネルデータを抽出して移送し，データ構造を新カーネル用に変換してメモリに配置することで，新カーネルでプロセスを継続実行する．このプロセス移送により安定した仮想計算機の実行環境と無停止によるカーネル更新を実現する．

２．研究の目的

オペレーティングシステムのカーネルは計算機ソフトウェアの基盤であるため，安定した稼働が必須である．しかし，カーネルは大きなソフトウェアであるため多数のバグが含まれており，カーネルの更新（パッチ・バージョンアップ）や障害発生による停止は避けられない．ここで，カーネルの停止は，その上で動作するユーザプロセスの停止を意味する．ユーザプロセスの停止は当該プロセスが提供するサービス（Web やデータベース等）の停止を意味し，利用者の利便性を大きく低下させる．そこで，カーネルの更新や障害発生に伴う停止に際して，カーネルを複数動作させて，ユーザプロセスをそれらカーネル間で移送することで継続動作させる．これにより，サービスの停止を伴わない計算機利用環境を構築し，安定動作の実現と管理負荷を軽減する．

３．研究の方法

ユーザプロセスを異なるカーネルバージョンのカーネル間移送には，カーネルバージョン間で異なるデータ構造がある場合にデータ構造の差異に関係なく新バージョンにデータを変換するカーネルデータ変換方法と、仮想計算機環境でのユーザプロセスの移送方法と、ユーザプロセスの移送のタイミングを決定するための障害検知方法が必要となる。本研究ではこれら 3 種類について実施した．なお，本研究に先立ち，事前研究として申請者は、ソフトウェアで計算機の論理分割を行う多重計算機実行基盤 0rthros[4]を開発している。本研究においては，この事前研究の成果を活用して行う．

○カーネルデータ変換方法

申請時点において，データ構造の変換方法をプログラム内に静的に記述した方法において一部のカーネルデータの変換には成功していた[5]．これを，様々なカーネルデータに適用する方法を検討する．

また，ユーザプロセスから見たカーネル状態の一貫性を保つ方法として，新旧カーネルの状態が同一になるようにユーザプロセスからシステムコールの発行を併用する方法を検討する．

○仮想計算機環境に関する研究

申請時点において実施していた同一バージョンでのユーザプロセス移送を応用し，仮想計算機環境に適用する．まずは，同一バージョンのカーネルを用いる 0rthros の知見を活かすために、0rthros を基盤としてカーネル間移送できるカーネルデータを増やす．ここで得た知見を基にして，仮想環境でのユーザプロセス移送を実現する．

○仮想計算機環境における障害検出方法

仮想計算機環境では，単一の物理計算機に複数の仮想計算機が動作している．通常，これらの仮想計算機間は強い保護機構が働いているために，仮想計算機間でのアクセスはできない．この仮想計算機間のメモリ保護を障害検出のために意図的に緩めることで，仮想計算機環境において効率の良い障害検出方法を実現する．また，複数のプロセッサアーキテクチャにおいて本方式を実現し，検出方式が容易に他のアーキテクチャに適用できることを示す．

４．研究成果

○「カーネルデータ変換方法」および「仮想計算機環境方法」

カーネルデータの変換は，本申請における関連研究[5]により，基本的な変換が可能と

なっていた．本研究では，この関連研究の成果に基づいてカーネルデータ変換方法の確立と、変換しなければならない対象カーネルデータの特定を行う計画であった．計画した方法は，新旧カーネルで使用されるデータ構造とは独立した中間データ構造を定義し、この中間データ構造を介して、異なるカーネルバージョン間でのデータ変換を行う方法であった。この中間データ構造は、カーネルバージョンに依存しない汎用かつ重要なデータのみを扱うことで，カーネルバージョン変更に伴うデータ構造の変更を柔軟に行う仕組みであった．しかし，カーネルデータが予想外にカーネルバージョンに依存したため，中間データ構造の作成が困難であった．

実際の研究においては，中間データ構造を構築するために，本研究の基盤となる多重 OS 実行基盤 Orthros を活用した．これを使用することで，他の課題と平行して研究を進めることができるためである．また，関連研究[5]と同じ基盤であるため，それまでの知見を活かせると判断したためである．この Orthros を用いて，ネットワーク通信を管理する socket 関係のデータ構造の変換に取り組んだ．まずは中間データ構造を構築する前段階として，人手によるデータ変換ルールを実装するプロトタイプシステムを実装した．この結果，異なるバージョンのカーネル間でユーザプロセスを移送し，移送後のユーザプロセスによる TCP 接続が可能なところまで確認できた．しかし，多様なカーネルデータ構造のすべてに対応できなかったため，TCP 通信中にカーネルパニックによる異常停止が発生し、その原因を特定することはできずに TCP 接続に完全に対応することはできなかった．このため，これ以上に対応するデータ構造の範囲を広げることができなかった．

次に，完全なカーネルデータの変換ではなく，新カーネルにおいて旧カーネルと同じ仕様となるようにシステムコールを発行する仕組みの構築を行った．カーネルレベルでのデータ変換ではなく，ユーザプロセスからみたカーネル状態の一貫性を保つことで，カーネル更新後にもユーザプロセスが動作できるようにする方法である．この方法では，主なカーネルデータは，システムコールを実行することでカーネルにおいて適切に作成される。このため，ユーザプロセス移行に伴い不足したデータを別途カーネル内に追加することで，ユーザプロセスの移行が実現できる．しかし，ユーザプロセスの移行のために，ユーザ空間から適切なシステムコールを実行する必要があり，ユーザプロセスの移行に要する時間や処理の手順が増加する問題がある．

本方式の実験として，Orthros を用いて同一バージョンのカーネル間でコンテナを移送するシステムを構築した[学会発表④]．本発表では，UNIX ドメインソケットと cgroups の移行を実現し，コンテナの移送を行った．これにより，システムコールと一部カーネルデータのカーネル間移行によるユーザプロセス移送が可能であることを示すことができた．しかし，異なるカーネルバージョン間での移行は，仮想環境基盤の構築が実現できなかったために確認できなかった．

○仮想計算機環境における障害検出方法

計算機の安定動作には，障害が発生した場合に可能な限り早期に検出する必要がある．本研究課題では，複数の仮想計算機が動作する環境におけるカーネルの障害検知機能についての研究を行った．

本研究が対象とする複数の仮想計算機が動作する環境においては，あるカーネルが他のカーネルの内部を調べることが可能という特徴がある．当然ながら，他のカーネルの内部を調べることは一般には不可能であるが，制限を意図的に緩めることで可能となる．この点を利用して，障害検知のためにあらかじめ指定されたカーネルに，他のカーネルを監視する役割を持たせることができる．

まず，本研究の基盤となった Orthros[4]における障害検知方法について述べる．Orthros では，処理を実行している ActiveOS を，監視用の BackupOS が定期的に監視する．監視方法は，Inter-Processor Interrupt(IPI)を利用した 2 種類がある．1 つ目は，ActiveOS 自身が障害を検知した場合にカーネルの panic 関数内の処理で IPI を使用して BackupOS に通知する方法である．2 つ目は，ActiveOS が定期的に BackupOS に生存通知を IPI によって送る方法である．生存通知が 1 秒以上途絶えた場合に，BackupOS は障害発生と判断する．このように，基本的にカーネル間の通知に基づく方式であるため，カーネルの動作が停止する事態になるまで障害検知ができず，プロセスの停止といった細かな障害による検知はできない．

本研究[学会発表③]では，Orthros を用いて従来研究よりも詳細な情報に基づいてカーネルとユーザプロセスの障害を検知し，ユーザプロセス移送とカーネル入れ替えに関する起点とする方式を開発した．仮想計算機環境では，カーネル間のメモリ参照制限を緩めることで，あるカーネルのメモリ内容を他のカーネルが参照することができることを利用し，監視カーネルが動作中カーネルのカーネルデータを直接参照する．これにより，監視カーネルは動作中カーネルとその上で動作するユーザプロセス群の詳細な情報を得ることができる．本研究成果は，様々な仮想環境に適用可能と考える．

提案する障害検知方式の概要を以下に示す．

- 監視には 動作中カーネルのメモリ上に存在するカーネルデータを使用
- 定期的監視により取得したカーネルデータを用いて障害を検知

提案方式では，監視カーネルが動作中カーネルのメモリを監視することで障害の検知を行う．これにより動作中カーネルが行う検知のための追加処理を最小限に留めることで，動作中カーネルのオーバヘッドを抑えることが可能となる．監視対象は，動作中カーネルのカーネル実行状態を示すカーネルデータとする．各種カーネルデータを用いた障害検知を行うことで，カーネル内で発生する多様な障害に対応することが可能となる．

本方式の評価として，障害検知の有効性と，両カーネルにおける処理オーバヘッドを測定した．一つ目の評価項目として，障害検知の有効性について述べる．有効性評価として次の 3 項目を評価した．

- カーネル内のデッドロック
- プロセス生成とメモリの大量消費
- デバイスドライバの停止

カーネル内のデッドロックは，カーネルのソースコードを一部変更し，スピンロックを解除しないバグを挿入することでデッドロックを意図的に発生させて障害を発生させた．このカーネルを動作させた結果，提案方式により監視カーネルは動作中カーネルの障害検知が可能であることを確認した．

次に，カーネル内部において無限にプロセス生成とメモリ確保を行う処理を実行することで障害を発生させた．本障害も，提案方式により検知できることを確認した．なお，監視を行う時間間隔が短いほど，利用可能として残ったメモリ容量が多いことを確認した．

最後に，デバイスドライバに意図的にバグを挿入することで，デバイスドライバが停止する障害を発生させた．デバイスドライバの通常の処理は，読み出し可能なデータがない時にユーザプロセスが読み出しアクセスを行った場合には，当該プロセスをブロックする．その後，データが追加された時にブロックを解除するものである．これに対して，データが追加された場合でも，ユーザプロセスのブロックを解除しないバグを挿入した．これを利用して，ブロック状態となるプロセスを無制限に生成する障害を発生させた．本障害に対しても，提案方式は正しく障害検知が可能であった．なお，監視間隔が短いほど，ブロックされたプロセス数が少ないことを確認した．

二つ目の評価項目として，カーネルのオーバヘッドについて述べる．オーバヘッドの計測対象として，動作中カーネルにおける監視を受けることによるオーバヘッドと，監視カーネルにおける監視間隔の違いによるオーバヘッドを計測した．まず，動作中カーネルにおけるオーバヘッドはほぼ 0 であった．これは，動作中カーネルには監視のための処理がなく，監視カーネルからのメモリ参照を受けるのみであるため，大きな負荷が発生しないことによる．

次に，監視カーネルのオーバヘッドを計測した．通常，監視カーネルは監視しか行っていないため，監視中の CPU の動作量が監視のための負荷に比例すると考え，CPU の消費電力等の詳細なデータを測定できる powertop ツールで計測した．その結果，1ms 間隔に監視を行った場合，CPU の全動作時間に対して最高動作周波数での動作が占める時間が約 76.58%，10ms 間隔で監視を行った場合で約 48.28%となった．また 100ms 間隔と 1000ms 間隔では，約 0.1%と約 0.0%であった．以上の計測結果より，監視間隔が 10ms より短い場合には，CPU の負荷が大きいことが分かった．しかし，監視専用としてカーネルと CPU を割り当てているため，動作中カーネルにおける実作業の処理にはほぼ影響がないことが分かった．

本提案方式は，他の仮想環境においても利用可能である．これは[学会発表②]と最終年度の研究において確認した．これらの研究では，仮想環境の一種である ARM 社製の CPU に搭載されたセキュリティ機構 TrustZone に利用した障害検知機構を開発して適切に検知できることを確認した．

○まとめ

本研究における成果として，仮想計算機環境において，カーネルデータの移行とシステムコールによって作成されるカーネルデータを組み合わせることで，ユーザプロセスの仮想計算機間移送が可能であることを示した．さらに，複数の仮想計算機環境における障害検出方式について，カーネル間でメモリ参照を行う方式を提案し，その有効性を確認した．しかし，申請時点で検討した課題の一部について解決することはできなかった．これは，カーネルバージョンに依存するカーネルデータ構造が，当初予想よりも複雑であったためである．特に仮想環境の構築が十分に行えなかった．

＜引用文献＞

[1] Arnold, J. and Kaashoek, M. F., “Ksplice: Automatic rebootless kernel updates”, In Proceedings of the 4th ACM European conference on Computer systems (EuroSys), pp. 187-198 (2009).
[2] Depoutovitch, A. and Stumm, M., “Otherworld: giving applications a chance to survive OS kernel crashes”, In Proceedings of the 5th ACM European conference on Computer systems (EuroSys), pp. 181-194 (2010).
[3] Siniavine, M. and Goel, A., “Seamless kernel updates”, In Proceedings of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), pp. 1-12 (2013).
[4] Yoshida, K., Saito, S., Mouri, K., and

Matsuo, H., “Orthros: A High-Reliability Operating System with Transmigration of Processes”, In Proceedings of the 19th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), pp. 318-327 (2013).
[5] 多重 OS 構成によるカーネルのライブアップデート手法の提案: 石川幸希, 安井裕亮, 齋藤彰一, 瀧本栄二, 毛利公一, 松尾啓志, 情報処理学会研究報告 2014-OS-130 (2014).

５．主な発表論文等
（研究代表者，研究分担者及び連携研究者には下線）

〔雑誌論文〕（計　0 件）

〔学会発表〕（計　4 件）
①富松 将広, 齋藤 彰一, 毛利 公一, 松尾 啓志: "OS 同時実行基盤 Orthros における反復可能な OS 起動方式の実装と評価", 情報処理学会研究報告 2015-OS-134, No. 10 (2015.8).
②岩間 響子, 毛利 公一, 齋藤 彰一："多様な障害へ対応したカーネルレベル障害検知機能の提案と実装", 情報処理学会研究報告 2016-OS-136, No. 7 (2016.2).
③松下 馨, 岩間 響子, 瀧本 栄二, 毛利 公一, 齋藤 彰一:"多重 OS 実行環境におけるカーネル間メモリ監視による障害検知機構の実装",情報処理学会研究報告 2016-OS-139, No. 2 (2017.3).
④新美 渓介, 瀧本 栄二, 毛利 公一, 齋藤 彰一:"プロセス耐障害性向上システム Orthros におけるコンテナマイグレーション手法", 情報処理学会研究報告 2016-OS-139, No. 11 (2017.3).

〔図書〕（計　0 件）

〔産業財産権〕

○出願状況（計　0 件）
○取得状況（計　0 件）

〔その他〕
ホームページ等

６．研究組織
(1)研究代表者
斎藤　彰一（SAITO, Shoichi）
名古屋工業大学・大学院工学研究科・教授
研究者番号：70304186