Vol.2015-OS-134 No.10
2015/8/4

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

富松 将広[1]　齋藤 彰一[1]　毛利 公一[2]　松尾 啓志[1]

概要：一台の計算機に二つの OS を同時に動作させる OS 同時実行基盤 Orthros では，ActiveOS に障害が発生した際，BackupOS が ActiveOS の CPU，メモリ領域，デバイス，ファイルキャッシュ，プロセスを引き継ぐことで計算機の構成を復元して新たな ActiveOS となる．これにより，実行中のプロセスを OS の障害から保護する．本論文では，ActiveOS と BackupOS の切り替えが複数回反復可能なシステムを提案する．これには，異常停止した ActiveOS が利用していたメモリ領域の一部に新たな BackupOS のカーネルを読み込み，ActiveOS が利用していた CPU コアの一部を用いて新たな BackupOS を起動する．これらの処理によって，再びアクティブ・バックアップ構成を再構築することでフェイルオーバー後の計算機の耐障害性を向上させる．

1. はじめに

計算機の障害の一つに OS のカーネルに存在するバグが原因による故障がある．カーネルのバグは他のソフトウェアに比べ取り除くことが難しく，機能追加や既存のバグを解消するための修正により OS のコードは年々増加しており，バグの混入する可能性は増大傾向にある [1][2]．そのため，計算機に搭載されるシステムはカーネル内に存在するバグに対する耐性を持つ必要がある．

OS のカーネル内で何らかの障害が発生した場合，一般的に計算機の再起動による復旧が試行される．しかし，計算機の再起動によって障害発生前の計算機が保持していた実行状態を失う．また，再起動中は計算機は使用不可能である．さらに，再起動によって OS が正常に復帰したとしてもファイルが破損する等の問題が発生する可能性がある．

OS のカーネルで発生した障害によって影響を受けるものの一つにファイルシステムがある．もしファイルシステム操作中に OS がクラッシュした場合，主記憶上に保存されているファイルキャッシュやメタデータを損失する可能性があり，ファイルの破損につながる．ファイルシステムのデータを保護するための既存の方法としてジャーナリングファイルシステムの利用が挙げられる．Linux の標準ファイルシステムである ext3/ext4 はジャーナリングファイルシステムの実装がしてあり，ext3 では Journal,Ordered,Writeback の三種類のモードを選択できる．

また，計算機の実行状態を保護する既存手法としてチェックポイント/リスタート（C/R）[3][4] が存在する．しかし，この手法はチェックポイントを作成する実行時オーバーヘッドが大きく，リスタート時に以前のチェックポイントまで実行状態が巻戻るので完全に実行状態を保護しているとは言えない．以上より，OS の耐障害性向上には以下の各点が求められる．

- OS のバグに対するプロセスやファイルキャッシュの保護
- 高速なリカバリ
- 最小限の計算機構成による実現
- 通常実行時の最小限のオーバーヘッド

そこで我々は，一台の計算機上で二つの OS を同時に動作させアクティブ・バックアップ構成を組むことで，計算機の耐障害性を向上する手法 ORganaized Transmigratory High-Reliability OS（Orthros）[5] を提案している．Orthros は，単一計算機のハードウェアリソースを二つに分割することで，ユーザとシステムサービスの主な仕事を処理する ActiveOS と，ActiveOS に障害が発生した場合に備える待機 OS である BackupOS の二つの OS を同時実行する．障害が発生した際には，ActiveOS の実行状態を BackupOS がフェイルオーバーする．これにより，C/R のような実行時オーバーヘッドがかからず，高速にリカバリを行うことができる．また，BackupOS は必要最小限のハードウェアリソースを用い，通常実行時は ActiveOS の監視以外は何もしない．更に，要求ハードウェアはマルチ

[1] 名古屋工業大学
Nagoya Institute of Technology
[2] 立命館大学
Ritsumeikan University

コア CPU と OS 毎のディスクのみであり，高価な機器を必要としないという特徴がある．

現在の Orthros は，ただ 1 回のフェイルオーバーに対応しているのみであり，フェイルオーバー後の BackupOS における障害には対応していない．このため，2 回の障害が発生した場合には，通常と同様にリブートが必要になる．そこで，本論文ではフェイルオーバー後に計算機のハードウェアリソースを再分割し，アクティブ・バックアップ構成を復元することのできる反復可能な OS 起動方式の実装と評価について述べる．

本論文の構成は次のとおりである．まず，2 章で OS 同時実行基盤 Orthros について述べる．次に，3 章では反復可能な OS の起動方式の概要を述べ，4 章で本方式の設計と実装について述べる．5 章で，評価を行い，6 章で関連手法について述べる．最後に 7 章でまとめを述べる．

2. Orthros

本章では，OS 同時実行基盤 Orthros の概要と，Orthros に実装されている各機構について述べる．

2.1 概要

Orthros は単一計算機のハードウェアリソースをソフトウェアレベルで論理的に分割する Softwear Logical Partitioning（Software LPAR）を採用し，OS を二重に起動することでフェイルオーバーを実現する耐障害性向上を目的としたシステムである．ユーザとシステムサービスの主な仕事を処理する OS を ActiveOS とし，ActiveOS に障害が発生した場合にフェイルオーバー処理を行う待機 OS を BackupOS とする．フェイルオーバー処理は以下のような処理で行われる．

(1) あらかじめ BackupOS を起動
(2) ActiveOS に障害が発生
(3) 障害を検知した BackupOS が，ActiveOS によって使用されていたデバイス（SSD，NIC）を自 OS 管理下にマイグレーション
(4) NIC に IP を割り当て
(5) ファイルキャッシュをマイグレーション
(6) プロセスをマイグレーション

上記のフェイルオーバー処理を実現するため，Orthros では，OS 同時実行機構，デバイスマイグレーション機構，ファイルキャッシュマイグレーション機構，プロセスマイグレーション機構，死活監視機構を実装している．

2.2 OS 同時実行機構

Orthros は，SHIMOS[6] および Mint[7] を参考に x86-64 アーキテクチャ上に Software LPAR を実装している．Software LPAR によって分割された計算機上で動作する各 OS は事前に使用可能なデバイスと CPU コアとメモリ領域が指定されており，起動時に占有使用するハードウェアのみ初期化処理を行う．各 OS が占有するデバイスと CPU コアとメモリ領域は，OS 起動時のパラメータとソースコード内に設けた除外デバイスリストによって定められる．ハードウェアの識別は，CPU コアは各 OS で扱うコア ID，メモリ領域は実アドレス範囲，その他デバイスはバス番号とデバイス番号を用いる．OS の起動は，まず最初に ActiveOS を通常の OS と同様に起動する．その後，専用のブートローダによって CPU コアと使用コア数，メモリ領域を指定し BackupOS を起動する．

Orthros のメモリ領域には，各 OS が占有使用するメモリ領域の他に両 OS からアクセスが可能な OS 間共有メモリがある．この領域は，OS 間通信の他に，フェイルオーバー時に必要な ActiveOS 内の情報の保存に用いる．

2.3 死活監視機構

死活監視機構は ActiveOS の状態を監視し，ActiveOS に何らかの障害が発生したことを BackupOS へ通知する機構である．本機構は，ActiveOS が Inter-Processor Interrupt（IPI）を用いて BackupOS に生存を通知する手法（生存通知）と，ActiveOS が panic 関数を実行する際に IPI を用いて BackupOS に障害発生を通知する手法（障害発生通知）の 2 つを用いている．BackupOS は，障害発生通知を受信した場合と生存通知が一定期間受信できない場合に，ActiveOS が異常停止したと判定してフェイルオーバーを開始する．

2.4 デバイスマイグレーション機構

フェイルオーバー処理の第一段階として ActiveOS が使用していたデバイスを BackupOS にマイグレーションするための機構である．BackupOS は占有使用するデバイスのリストの他に ActiveOS が占有使用しているデバイスのリストを保持しており，デバイスマイグレーション機構はこのリストを参照して必要なデバイスを識別し BackupOS 用に初期化を行う．

2.5 ファイルキャッシュマイグレーション機構

障害発生時まで ActiveOS が使用していたファイルシステムを BackupOS が使用できるようにするために，ファイルシステムのマウントとファイルキャッシュのマイグレーションが必要となる．まず，BackupOS は，デバイスマイグレーションしたディスクドライブを，ActiveOS が使用していた時と同じパスでマウントする．これにより，プロセスマイグレーション機構によりマイグレーションされるプロセスは同じパスで同じ内容のファイルを扱うことが可能になる．

次に，ファイルキャッシュマイグレーション機構は，ActiveOS 上のディスクに書き戻されていないファイルキャッ

シュを検索し，BackupOS 上にコピーすることでファイルが破損するのを防ぐ．現在ファイルキャッシュマイグレーション機構は ext3 ファイルシステムを対象に実装済みである．

2.6 プロセスマイグレーション機構

プロセスマイグレーション機構は，ActiveOS で事前に指定したプロセス群を対象としてマイグレーションを行う．マイグレーション処理は，BackupOS が ActiveOS のメモリ領域からプロセスを管理する構造体を読み出し，BackupOS 内のメモリ領域に再構成することで行う．プロセスマイグレーションにおいて，プロセスのカーネル空間での実行状態を引き継いだ場合には OS に発生した障害も引き継ぐ可能性が高い為，障害の原因となるカーネル空間の実行状態は引き継がない．

3. 反復可能な OS 起動方式

本論文では，Orthros のフェイルオーバー後に発生する障害に対応するために，反復起動可能な OS 起動方式を提案する．本方式では，次の二つの処理を実施する．

(1) フェイルオーバー後の BackupOS に，ActiveOS 相当のハードウェアリソースを割り当てる．これにより，通常運用に耐えうる環境を整える．
(2) 新しい BackupOS 用のハードウェアリソースの分割と，新しい BackupOS の起動および死活監視を行う．

本方式により OS の起動及びフェイルオーバーを限りなく繰り返すことができるシステムになる．このために，以降，初めに起動した OS を 1stOS，2 番目に起動した OS を 2ndOS と呼び，N 番目に起動した OS を NthOS と呼ぶこととする．また，フェイルオーバー対象の OS を ActiveOS，フェイルオーバーを行う OS を BackupOS，フェイルオーバー後に新たに起動する OS を新 BackupOS と呼ぶ．

3.1 新 BackupOS 起動手順

反復起動に対応した Orthros の新 BackupOS の起動手順を示す．BackupOS でフェイルオーバー処理が終了した後，新 BackupOS 用のメモリ領域以外のユーザ用領域をメモリホットプラグを用いて BackupOS に付加する．CPU コアも同様に，新 BackupOS が使用する予定の CPU コア以外を CPU ホットプラグで BackupOS に追加する．その後，PCI ホットプラグを用いて新 BackupOS が使用する PCI デバイスを BackupOS から外す．以上により，BackupOS は新 ActiveOS としてのハードウェア環境が整う．また，新 BackupOS 用のハードウェアの確保が完了する。

次に，kexec[8] を用いて新 BackupOS のメモリ領域にカーネルイメージを読み込む．この時，新 BackupOS が何番目の OS かを表すカーネルパラメータを設定する．kexec が終了後，予め確保した CPU コアに起動用 IPI を送信して OS を起動する．

新 BackupOS の起動後，BackupOS はマイグレーション対象となるプロセス群の管理構造体のメモリアドレスを OS 間共有メモリに書き込む．新 BackupOS は，死活監視を開始して BackupOS のクラッシュに備える．以上の処理をフェイルオーバーの度に繰り返す．

3.2 既存 Orthros からの変更点

本提案を実現するため，既存 Orthros に対し，以下の変更を行う．

(1) 反復起動に対応したデバイス分割と再分割機構
(2) 専用ブートローダの反復起動対応
(3) OS 起動時パラメータの反復起動対応
(4) 論理 APIC ID の CPU コアへの割り当て方式の改良

3.3 反復起動に対応したデバイス分割と再分割機構

既存 Orthros の場合，フェイルオーバー後の BackupOS が計算機のハードウェアのほとんどをマイグレーションし，新 BackupOS を起動するためのハードウェアリソースを考慮していない．このため，本方式では，フェイルオーバー後にハードウェアを再分割し，新 BackupOS 用のハードウェアリソースを改めて割り当てる．反復起動に対応するためのハードウェア構成として，CPU コア，メモリ，ディスクドライブ，その他のデバイスに分けて述べる．

3.3.1 CPU コアの分割

CPU コアは，新 BackupOS 起動用の CPU コアを一つ決定し，それ以外の CPU コアを BackupOS のホットプラグ機能により初期化し BackupOS に付加する．これにより，BackupOS の CPU リソースが ActiveOS と同様になると同時に新 BackupOS 用の CPU コアを確保する．この時，x86-64 アーキテクチャの制約上の問題がある．CPU コアには，計算機の起動時に最初に起動する Bootstrap Processor（BSP）と OS 起動後に BSP からの特殊な起動用 IPI により起動する Application Processor（AP）の二種類がある．2ndOS 以降の OS を起動する際に CPU コアに起動用 IPI を送信するが，この IPI は AP でのみ動作して BSP では正しく動作しないため，新 BackupOS 用の CPU コアは AP の中から選択する．

3.3.2 メモリの分割

既存 Orthros では，2ndOS の起動までが可能である．また，2ndOS は，1stOS が使用していたメモリ領域を利用できない．本方式では，反復起動に対応し，かつ，各 ActiveOS が物理メモリの大部分を利用できるようにする．このために，物理メモリを次の 3 種類に分割する．

(1) 1stOS 用メモリ領域
(2) 2ndOS 以降の OS 用メモリ（3 区画）
(3) 汎用メモリ領域

（1）の 1stOS 用メモリ領域は，1stOS 専用のメモリ領

域で，2ndOS 以降が ActiveOS となった場合には未使用領域となる．1stOS のみ，2ndOS 以降と起動処理等が異なるために専用のメモリ領域を消費する．(2) は，2ndOS 以降の OS が BackupOS として活動する場合に使用する必要最低限のメモリ領域である．2ndOS 以降が ActiveOS となった場合，それまで使用していたこのメモリ領域も継続して利用する．(3) は，ActiveOS 用のメモリ領域であり，再分割機構により各時点で ActiveOS となった OS に割り当てられる．以上により，物理メモリの大部分を (3) とすることで ActiveOS が利用可能とし，ハードウェアリソースの分割による性能低下を防ぐ．また，2ndOS 以降が起動するメモリ領域を (2) のメモリ領域に固定することで，起動の反復を容易にするとともに，反復起動のためのメモリの消費を最小限に抑えている．

3.3.3 ディスクドライブの分割

既存 Orthros のディスクドライブは 2 台で構成されており，1 台目には ActiveOS 用の boot パーティションと root パーティション（以降，2 つのパーティションを合わせて起動パーティションという）とマイグレーション対象のユーザプロセスが使用するパーティションが確保され，2 台目には BackupOS 用の起動パーティションが確保されている．既存 Orthros のデバイスマイグレーションでは，ActiveOS 用のディスクドライブを BackupOS にマイグレーションすることで，ユーザプロセス用のパーティションを BackupOS が使用できるようにしている．

この状況において新 BackupOS を起動するには，BackupOS が使用していたディスクドライブを使用できることが望ましい．しかし，BackupOS 用のディスクドライブは root パーティションを含むために切り離すことができない．そこで，ActiveOS のディスクを，起動パーティション用とユーザパーティション用の 2 つに分割する．フェイルオーバー処理の再分割機構では，ユーザパーティション用ディスクのみを BackupOS に移し，起動用パーティションのディスクは移さないこととする．これにより，ActiveOS 用の起動用パーティションのディスクを用いて新 BackupOS を起動することが可能となる．以上により本方式におけるディスクドライブは，ActiveOS 用と BackupOS 用のそれぞれの起動用ディスクと，マイグレーション対象となるユーザパーティション用のディスクの 3 台構成となる．

3.3.4 その他のデバイスの分割

ディスクドライブ以外のデバイスの割り当ては，BackupOS が保持しているデバイスから新 BackupOS で使用するデバイスを予め決定しておき，フェイルオーバー処理後に OS の機能によって取り外す．

3.4 専用ブートローダの反復起動対応

反復起動に対応するための専用ブートローダへの変更点を以下に示し，それぞれについて述べる．

- 作成済み OS 間共有メモリを新 BackupOS に付加する機能の追加
- ローカル APIC ID を用いた起動コア指定方法の追加

3.4.1 OS 間共有メモリ

既存 Orthros の場合，ActiveOS から BackupOS を起動する際に，OS 間共有メモリが作成され，ActiveOS と BackupOS のメモリ領域に付加される．しかし，新 BackupOS 起動時にも同様に OS 間共有メモリを作成すると，BackupOS 側に ActiveOS との共有メモリが使用されないまま残存する．そのため，1stOS が作成した OS 間共有メモリ領域を 3rdOS 以降に引き継ぐ機能を専用ブートローダに加える．

3.4.2 OS 起動コアの指定方法

Orthros では，2ndOS 以降の OS が起動する CPU コアを指定している．既存 Orthros では，この指定に各 CPU コアが管理するコア ID を用いている．このコア ID は，OS 内で決定される ID であり起動毎に変化するため，同じコア ID を指定した場合でも起動毎に違う物理コアで BackupOS が起動することになる．しかし，前節で述べたように，OS を起動できる CPU コアは BSP 以外の AP である．このため，起動 CPU コアには AP となっている CPU コアのコア ID を指定しなければならない．なお，既存 Orthros では，この問題は発生しない．なぜなら，BSP は 1stOS が起動した CPU コアであることから，2ndOS が起動する CPU コアは，自動的に 1stOS が起動した CPU コア（BSP）以外から選択されるためである．

この問題を解決するために BSP のローカル APIC ID が 0 となることを利用し，起動時の CPU コアの指定にローカル APIC ID を用いる方法を採用した．ローカル APIC は x86-64 アーキテクチャのマルチコア CPU において，各コアに搭載されている割り込みコントローラで，CPU 外部から発せられた全割り込みを管理と，IPI の送受信を行う機構である．ローカル APIC ID は，各ローカル APIC に割り当てられた番号で計算機の電源投入時に自動で割り当てられ，変更されることが無いため，確実に BSP 以外の CPU コアを指定することができる．

3.5 OS 起動時パラメータの反復起動対応

本方式では，OS の起動が何番目かによって使用するハードウェアデバイスが決まるため，BackupOS を起動する際に，何番目に起動する OS であるかを新 BackupOS に知らせる必要がある．そのため，OS 起動時パラメータに何番目の OS であるかを表すパラメータを追加する．

3.6 論理 APIC ID の CPU コアへの割り当て方式の改良

死活監視機構が用いる IPI は，通知先の指定に各 CPU コアのローカル APIC 内の値である論理 APIC ID を用いる．論理 APIC ID は，IPI の送信先特定にのみ使用さ

れる特別な値である．IPI を送信する時は，送信先の論理 APIC ID に対応したビットを 1 にしたメッセージ・ディスティネーション・アドレス（MDA）を生成する．そして，MDA と各ローカル APIC 内の論理 APIC ID でビット単位の AND 演算を行うことで送信先 CPU コアを特定する．

既存 Orthros における論理 APIC ID の算出方法は，ActiveOS と BackupOS が起動する CPU コアが固定されているおり，2ndOS の起動までしか考慮されていない．このため，3rdOS 以降を起動すると，同じ値が複数のコアに割り当てられる問題と，OS 間での統一されていないという問題がある．したがって，どのコアで OS を起動しても，論理 APIC ID の重複が発生せず，OS 間で統一される算出方法を導入する必要がある．

4. 実装

3 章で述べた方式の詳細な実装方法について述べる．実装は Linux（version 2.6.38，processer type x86-64）上に行った．

4.1 反復起動に対応したデバイス分割

Orthros の現時点での実装では，各 OS が使用しないデバイスをソースコード内で指定している．反復起動に対応するために，OS が何番目の起動かを判断し，不必要なデバイスを初期化しない必要がある．利用しないデバイスは，1stOS 用と偶数番目用と 3rdOS 以降の奇数番目用の 3 種類に分けられる．

デバイス初期化処理は，デバイスドライバの読み込みが行われる driver_probe_device 関数内で行われる．driver_probe_device 関数内で，予め用意した除外デバイスリストを参照し，デバイスドライバが除外リスト中にあった場合，ドライバの読み込みを中断することで初期化を中止しデバイス分割を行う．自身が何番目の OS かは，後述する改変した BackupOS であることを示すカーネルパラメータにより与えられた値で判断する．

4.2 計算機再分割機構

計算機再分割機構では CPU とメモリと PCI デバイスについて再分割を行う．

4.2.1 CPU コアの再分割処理の実装

CPU コアの再分割は Linux に搭載されている CPU ホットプラグにより行う．CPU コアの再分割はプロセスマイグレーション後，sysfs 上の CPU コアを管理するファイルに特定の値を書き込むことで，対応した CPU コアの初期化が始まり，BackupOS で使用できるようになる．この時，CPU コアの内 BSP 以外の 1 つのコアを新 BackupOS 用として，初期化しない．

4.2.2 メモリの再分割処理の実装

3.3.2 で述べたように，メモリ領域は OS 起動用のメモリ領域として分割済みである．残りの汎用メモリ領域は再初期化し，BackupOS で使用する．メモリ領域の再初期化には Linux のメモリホットプラグ機能を利用する．Linux のメモリホットプラグ機能は物理メモリを 128MB のブロックに分割し管理している．1stOS で使用していたメモリブロックの内，ブロック内にハードウェアが使用するメモリが含まれるブロックは，メモリホットプラグが機能しない問題がある．このため，これらのブロックを 2ndOS 以降が利用できるようにすることができない．よって，2ndOS 以降が利用できるメモリ容量は，1stOS のメモリ容量よりも減少する．2ndOS 以降のメモリ容量には変化はない．

4.2.3 PCI デバイスの再分割処理の実装

PCI デバイスの再分割は Linux に搭載されている PCI ホットプラグ機能を用いて行う．PCI ホットプラグ機能は Linux の sysfs 上の特定のファイルに 0 以外の値を書き込むことにより，各デバイスのドライバが取り外され，デバイスの使用していた IRQ やメモリが開放される．取り外されたデバイスは，新しく起動した新 BackupOS で再初期化される．

4.3 専用ブートローダの反復起動対応

2ndOS 以降の OS を起動するための専用ブートローダの反復対応の実装について述べる．

4.3.1 OS 間共有メモリ

OS 間共有メモリは，2ndOS 起動時に作成される．反復起動に対応するには，OS 間共有メモリを 3rdOS 以降においても利用できるようにする必要がある．そこで，kexec が新 OS を起動する時に，OS 間共有メモリのアドレスとサイズを取得するように変更した．もし，OS 間共有メモリが未割り当ての場合（2ndOS 起動の場合）には新たに OS 間共有メモリを作成し，すでに作成されていた場合にはその OS 間共有メモリを利用するようにした．

4.3.2 OS 起動と起動 CPU コアの指定

kexec により新 OS を OS 起動用のメモリに書き込んだ後，起動用 CPU コアを指定して OS を起動する．この際，OS 起動用の IPI を OS が起動する CPU コアに送信する．3.4.2 で述べたように，起動用 CPU コアは AP となっている CPU コアでなければならない．そこで，ローカル APIC ID によって起動コアを指定できるように変更した．

4.4 OS 起動時パラメータの変更

起動した OS が何番目に起動された OS かを特定するため，既存 Orthros の BackupOS を表すカーネルパラメータを変更した．Linux の起動時カーネルパラメータは，カーネルソースの early_param マクロで定義される．BackupOS を表すカーネルパラメータに，何番目の OS かを表す値を追加した．なお，4thOS 以降の OS の起動処理は 2ndOS と同じ処理になるため，入力された値を 4 で割った余りを

表 1　BackupOS の論理 APIC ID 割り当ての例

ローカル APIC ID	0	2	4	6
CPU コア ID	1	2	3	0
論理 APIC ID	1	2	4	8

表 2　新 BackupOS の論理 APIC ID 割り当ての例

ローカル APIC ID	0	2	4	6
CPU コア ID	1	0	2	3
論理 APIC ID	1	2	4	8

表 3　計算機構成

CPU	Intel（R） Core（TM） i7-3770 @ 3.40GHz
Memory	8GB
Device	SATA x3，SSD x3，NIC x2

表 4　ハードウェア分割の初期状態

	ActiveOS	BackupOS
CPU	3 コア	1 コア
Memory	7392MB	352MB
Device	SATA x2，SSD x2 NIC x1	SATA x1，SSD x1 NIC x1

何番目の OS かを表す値に使用している．

4.5 論理 APIC ID の CPU コアへの割り当て方式の改良

論理 APIC ID は，IPI の送信先コアを指定するために用いられる，CPU コアを識別する ID である．Orthros では，各 CPU コアに対して IPI を送出するために，論理 APIC ID を全 OS で統一した値を割り当てる．通常の Linux では，論理 APIC ID は，1 を CPU コア ID の値だけビットシフトした値となる．このとき，論理 APIC ID 算出の基となる CPU コア ID は，OS が任意に決定できる値であるため，論理 APIC ID は各 OS で異なる値となる．よって，ローカル APIC ID も各 OS で異なる値となる．そこで，Orthros では，ローカル APIC ID の小さい順に論理 APIC ID を割り当てることで，全 OS における論理 APIC ID を統一している．

表 1 は，BackupOS がローカル APIC ID が 6 の CPU コアから起動した場合の各 ID の割り当て状況である．起動コアであるローカル APIC ID が 6 の CPU コアの CPU コア ID は 0 となるが，論理 APIC ID はローカル APIC ID に基づいて 8 が設定される．また，表 2 は，新 BackupOS の各 ID の割り当て状況であり，OS 起動コアはローカル APIC ID が 2 の CPU コアである．この場合も，BackupOS と同様に，ローカル APIC ID のみに基づいて論理 APIC ID が決定し，BackupOS の論理 APIC ID と同一の割り当てを行っている．

5. 動作確認及び評価

評価に使用する計算機の構成を表 3 に，ハードウェア分割の初期状態を表 4 に示す．本評価では，まず基本的な反復起動の動作確認を行う．次に，メモリホットプラグの制約による，2ndOS 以降のメモリ容量の損失について評価する．

5.1 動作確認

Orthros の反復起動に対する，ユーザプロセスのマイグレーションの動作確認を行った．ユーザプロセスとして，基本的なシステム機能を利用する 3 種類のアプリケーションと，実アプリケーションとして Apache2 を用いた．

5.1.1 基本アプリケーションによる確認

基本的な機能を持つアプリケーションとして，以下の 3 つの機能を持つプロセスを 4thOS までマイグレーションした．

(1) ファイルに対する読み書きがあるプロセス
(2) マルチスレッドプロセス
(3) TCP 接続と UDP 接続で外部と通信を行うプロセス

これら 3 種のプロセスを実行中に，OS をクラッシュ（panic 関数を実行）させた．OS をクラッシュさせるタイミングは以下の通りである．

- システムコールの実行中
- ファイルへの書き込み途中
- ユーザ空間のコードを実行中

実験の結果，3 種のプロセスはどのタイミングで OS をクラッシュさせた場合においても，正常に 4thOS まで実行を継続した．また，ファイル書き込みについても，ファイルの破損がないことを確認した．

5.1.2 実アプリケーションによる確認

実アプリケーションを反復起動可能にした Orthros で 4thOS までマイグレーションし，4thOS での動作を確認した．実アプリケーションとして Orthros 用に改変した Apache2 を用いた．標準の Apache2 からの変更点を以下に示す．

(1) インストール先ディレクトリをマイグレーション用ディスクドライブ（SSD）内のディレクトリへ変更した．
(2) 共有メモリの生成方法を SystemV 共有メモリから mmap で指定したファイルを共有する方法へ変更した．
(3) DocumentRoot をマイグレーション用 SSD 内のディレクトリへ変更した．

以上の変更を加えた Apache2 を 1stOS で保護対象のプロセスとして起動し，4thOS までマイグレーションしたところ，マイグレーション後も Apache2 は正常に動作することを確認した．なお，OS をクラッシュさせたタイミング

表 5　使用可能メモリ容量

	1stOS	2ndOS	3rdOS	4thOS
メモリ容量	6885MB	6480MB	6480MB	6480MB

は前節と同じである．

5.2 メモリ領域損失量

1stOS 起動時の使用可能メモリ量と，2ndOS から 4thOS までの計算機再構成後の使用可能メモリ量を比較した．4.2 節で述べた通り，Linux のメモリホットプラグでは，物理メモリをブロック（128MB）単位でしか追加できない．さらに，ハードウェア等で予約されている領域を含むブロックの場合，そのブロックは追加できない．1stOS は，メモリホットプラグを使用しないため，この制約に関係なく任意の物理メモリを利用できる．しかし，メモリホットプラグを使用する 2ndOS 以降の OS では，ハードウェアによって利用されるメモリ領域を含むブロックを追加できず，利用することができない．このため，1stOS から 2ndOS へフェイルオーバーする時にメモリ領域が減少する．しかし，3rdOS 以降へのフェイルオーバーにおけるメモリホットプラグの対象メモリ領域は 2ndOS が追加したメモリ領域となるため，ハードウェアのメモリ利用による制約は発生せず，同一のメモリ容量を利用可能となる．本評価では，このメモリ容量の変化について確認を行った．

各 OS が利用できるメモリ容量（/proc/meminfo より取得）を表 5 に示す．1stOS から 2ndOS への計算機再構成により使用可能メモリ容量が 405MB 減っているが，その後に起動される 3rdOS，4thOS が利用できるメモリ容量は 2ndOS と変化が無かった．この結果により，ハードウェアで占有されるメモリ領域によるメモリホットプラグできないブロックは 1stOS 起動後に増えることはなく，2ndOS 以降のメモリ容量は変化がないことを確認した．今後，デバイスドライバの設定変更などによりハードウェアが占有使用するメモリ領域を 1 つのブロックにまとめる方法を調査する．これにより，1stOS と 2ndOS の計算機再構成後の使用可能メモリ量の損失を少なくすることができると考える．

6. 関連研究

提案手法と関連する既存手法について述べる．

6.1 Otherworld

Otherworld[9] は，OS のカーネル中のバグによって OS が停止した場合に OS をマイクロリブートによりプロセスの状態を保護する手法である．カーネルに障害が発生した際に，kexec により新たなカーネルをウォームブートする．さらに，障害発生時に実行中だったプロセスを障害が発生したカーネルのメモリ空間から取り出し，新たに起動した OS 上で実行を再開する．これにより，カーネルのバグからプロセスを保護する方式である．しかし，この手法はファイルキャッシュの移植を行なわないため，プロセスがファイル操作を行なっていた場合は正常な動作を保証できない．

6.2 MINIX

MINIX[10] はデバイスドライバのマイクロリブート機構を有するマイクロカーネル OS である．一般にデバイスドライバは，コード数が非常に多い上に実装がハードウェア依存であるために，バグを含みやすい．MINIX では，デバイスドライバをユーザ空間のアプリケーションとしてカーネル空間から分離している．そして，何らかの障害が発生した際に，そのデバイスドライバのみリブートを行うマイクロリブートを行うことで，ドライバのバグに対する耐障害性を向上させている．しかし，ドライバ以外を原因として OS がクラッシュする場合は計算機の再起動が必要になる．そのためドライバ以外の部分のバグに対してこの手法では対処できない．

6.3 Logical Partition（LPAR）

LPAR はハードウェアの仮想化機構によりハードウェアを論理的なパーティションに分割し，各パーティションを仮想マシンとして機能させる．そのため，KVM や QEMU といった他の仮想化方法と比べてハードウェアに対するオーバーヘッドが少ないという利点がある．本提案では OS 同時実行機構に用いる LPAR のソフトウェア実装として SHIMOS，Mint を参考にしている．

6.3.1 SHIMOS

SHIMOS はそれぞれの OS が主体となり互いのハードウェアに干渉しないようにすることで，ファームウェア支援の仮想化を行うこと無く複数の OS を一つの物理計算機上で実行する．SHIMOS では各 OS が調停機構を介することなくハードウェアにアクセスできるので LPAR に比べ軽量で，LPAR が専用のプロセッサを必要とするのに対して SHIMOS は一般的なマルチコア CPU（x86）で実行可能なので，適用範囲が広い．

6.3.2 Mint

Mint は Linux ベースの OS を一つの物理計算機上で複数混載する方式である．SHIMOS と同様に，Linux ベースのそれぞれの OS が主体となって互いのハードウェアに干渉しないようにすることで，ファームウェア支援の仮想化を行うこと無く複数の Linux ベースの OS を一つの物理計算機上で実行する．Mint は SHIMOS と違い，2 つ目以降の OS の起動方法が異なる．SHIMOS では専用のカーネルモジュールを用いて OS を起動させていたが，Mint では kexec を利用して起動する．これにより，実装する際の工数削減やメンテナンス性を向上させ，個別 OS の再起動を

可能にしている．また 32bit と 64bit の Linux ベース OS の混載も可能にしている．

7. まとめ

本論文では，一台の計算機に二つの OS を同時に動作させる OS 同時実行基盤 Orthros における反復可能な OS 起動方式について述べた．Orthros のフェイルオーバー処理の後に，計算機再分割機構によって BackupOS のハードウェアリソースを ActiveOS とほぼ同じ構成にして常用に耐えうるようにする．さらに，新 BackupOS 用のハードウェアリソースを確保する．確保したハードウェアリソースを用いて新 BackupOS を起動し，BackupOS の死活監視及びフェイルオーバーを行う．これを繰り返すことで ActiveOS と BackupOS を何度でも切り替えられるシステムを構築した．

本方式を用いて基本的なアプリケーションと実アプリケーションをマイグレーションした結果，どちらも正常に動作したことを確認した．また，メモリホットプラグの制約による 2ndOS 移行時のメモリ容量の減少を確認し，3rdOS 以降においては変化しないことを確認した．

参考文献

[1] Chou, A., Yang, J., Chelf, B., Hallem, S. and Engler, D.: An empirical study of operating systems errors, *Proceedings of the 18th ACM symposium on Operating systems principles*, SOSP '01, ACM, pp. 73–88 (2001).
[2] Palix, N., Thomas, G., Saha, S., Calvès, C., Lawall, J. and Muller, G.: Faults in linux: ten years later, *Proceedings of the 16th international conference on Architectural support for programming languages and operating systems*, ASPLOS '11, ACM, pp. 305–318 (2011).
[3] Hargrove, P. H. and Duell, J. C.: Berkeley lab checkpoint/restart (BLCR) for Linux clusters, *Journal of Physics: Conference Series*, Vol. 46, No. 1, pp. 494–499 (2006).
[4] Liao, J. and Ishikawa, Y.: A New Concurrent Checkpoint Mechanism for Real-Time and Interactive Processes, *Proceedings of the 2010 IEEE 34th Annual Computer Software and Applications Conference*, COMPSAC '10, IEEE Computer Society, pp. 47–52 (2010).
[5] Yoshida, K., Saito, S., Mouri, K. and Matsuo, H.: Orthros: A High-Reliability Operating System with Transmigration of Processes, *Proceedings of the 2013 IEEE 19th Pacific Rim International Symposium on Dependable Computing*, PRDC '13, IEEE Computer Society, pp. 318–327 (2013).
[6] Shimosawa, T., Matsuba, H. and Ishikawa, Y.: Logical Partitioning without Architectural Supports, *Proceedings of the 2008 32nd Annual IEEE International Computer Software and Applications Conference*, COMPSAC '08, IEEE Computer Society, pp. 355–364 (2008).
[7] Nomura, Y., Senzaki, R., Nakahara, D., Ushio, H., Kataoka, T. and Taniguchi, H.: Mint: Booting Multiple Linux Kernels on a Multicore Processor, *Proceedings of the 2011 International Conference on Broadband and Wireless Computing, Communication and Applications*, BWCCA '11, IEEE Computer Society, pp. 555–560 (2011).
[8] Pfiffer, A.: Reducing system reboot time with kexec, http://www.osdl.org/.
[9] Depoutovitch, A. and Stumm, M.: Otherworld: giving applications a chance to survive OS kernel crashes, *Proceedings of the 5th European conference on Computer systems*, EuroSys '10, ACM, pp. 181–194 (2010).
[10] Tanenbaum, A. S.: The Minix 3 Operating System, http://www.minix3.org/.