Vol.2017-OS-139 No.2
2017/3/1

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

松下 馨[1,a)]　岩間 響子[1]　瀧本 栄二[2,b)]　毛利 公一[2,c)]　齋藤 彰一[1]

概要：OS の障害はシステム全体に影響を及ぼす可能性があり，この障害から復帰する手段として一般的に計算機の再起動が用いられている．しかし，計算機の再起動はダウンタイムが大きく，OS 上で実行していたシステムの実行状態を消失してしまう．この問題に対し，本研究室ではプロセス耐障害性向上システム Orthros を提案している．Orthros は，単一計算機上に複数の OS を同時に実行する．本稿ではこの特徴を活かし，カーネル間でメモリを監視する障害検知機構を実装する．この機構は，監視対象となる OS のカーネルメモリ上に存在するカーネルの実行状態を示すパラメータを定期的に監視する．障害検知機構を実装した Orthros のカーネル及びデバイスドライバに対して意図的にバグを挿入することで評価を行い，定義した障害に対して異常を検知することを確認した．

1. はじめに

Operating System（OS）は計算機全体の動作に大きな影響を及ぼすため，OS に対する信頼性の向上が求められている．OS に障害が発生した場合，OS 上のプロセスはすべて停止する．しかし一般的に，OS には潜在的なバグや脆弱性が存在し，機能追加や修正が行われることにより新たなバグや脆弱性が発生する可能性がある [1][2]．これらのバグを完全に消し去ることは困難であるため，障害発生を前提とした耐障害性を持った OS が求められる．

障害の発生により OS が停止した場合，一般的に計算機の再起動による復旧が行われる．しかし，再起動は長時間計算機が使用不可能になることによる可用性の低下を招く．さらに，再起動により計算機は動作中のプロセスや未保存のファイルキャッシュ，ネットワークコネクションといった実行状態を消失する．

本研究室では，上記の問題を解決するためにプロセス耐障害性向上システム “**ORganized Transmigratory High-Reliability OS**”（**Orthros**）[3] を提案している．Orthros は，1 台の計算機上に 2 つの OS を動作させることで OS の冗長化を図り，計算機の再起動を行うことなく障害から高速に復旧する．また，障害発生時には OS のフェイルオーバー処理を行い，バックアップの OS 上で実行状態の保護を実現する．

しかし，Orthros は個別の障害に対する障害検知の機能を持たない．本稿では，個別の障害に対して検知を行う障害検知機構を Orthros に対して実装する．多様な障害に対応可能とすることで，Orthros の目的であるシステム全体の可用性向上を目指す．

本稿の構成は次のとおりである．まず第 2 章で既存手法 Orthros の構成と機能について触れ，第 3 章で OS の障害とその検知に関する関連研究について述べる．第 4 章で提案手法について述べ，第 5 章で提案を実現するために必要な実装を述べる．そして第 6 章で実装した機能の評価を行う．最後に，第 7 章でまとめと今後の課題について述べる．

2. Orthros

本章では，本提案が拡張対象とする既存手法である Orthros の概要について述べる．Orthros は，OS の高速なフェイルオーバーと OS 間でのプロセス及びファイルキャッシュの保護を実現するシステムである．高速なフェイルオーバーは，複数 OS 構成によるウォームスタンバイにより行う．プロセス及びファイルキャッシュの保護は，メモリイメージの走査によるプロセスの実行状態及びファイルキャッシュ取得により行う．

2.1 構成

Orthros のシステム構成例を図 1 に示す．高速な復旧を実現するために，Orthros は 2 つの OS を用意し，それらを

[1] 名古屋工業大学
Nagoya Institute of Technology
[2] 立命館大学
Ritsumeikan University
[a)] orthros@mail.ssn.nitech.ac.jp
[b)] takimoto@asl.cs.ritsumei.ac.jp
[c)] mouri@cs.ritsumei.ac.jp

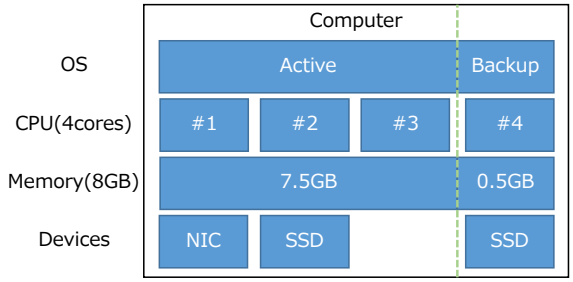

図 1　Orthros によるシステム構成例

表 1　Orthros の各機能と目的

機構	目的
複数 OS 同時実行機構	2 つの OS でアクティブとバックアップを構成
死活監視機構	ActiveOS が停止を監視
デバイスマイグレーション機構	ハードウェアの移植
ファイルキャッシュマイグレーション機構	ファイルキャッシュの移植
プロセスマイグレーション機構	指定されたプロセスの移植

アクティブとバックアップの構成で運用する．**ActiveOS** は主要な処理をすべて行い，**BackupOS** は ActiveOS に障害が発生した場合に備えて待機している．計算機構成を最小限に留めるために，これら 2 つの OS を 1 台の計算機上で同時実行する．この際，BackupOS は通常時，一切の主要な処理を行わず，割り当てた計算機資源が無駄となる．したがって，通常実行時には ActiveOS にほぼすべての計算機資源を割り当て，BackupOS には動作のために必要最低限の CPU とメモリのみを割り当てる．

BackupOS を動作させる際の性能低下を最小限とするために，SHIMOS[4] を参考にハードウェア資源の論理的分割が行われている．Linux カーネルに変更を加えることで論理的分割が行われるため，Xen[5] や KVM[6] のような仮想化システムと異なり仮想化オーバヘッドが存在しない．

2.2　動作概要

ActiveOS は複数 OS 同時実行機構を用いて BackupOS を起動する．起動した BackupOS は ActiveOS が停止していないかを監視し続ける．これを死活監視機構という．そして，停止を検知した場合には自動的にフェイルオーバーを開始し，管理者によって指定された ActiveOS のプロセス及びファイルキャッシュを保護する．プロセスの保護はプロセスマイグレーション機構，ファイルキャッシュの保護はファイルキャッシュマイグレーション機構によって実現する．ファイルキャッシュとプロセスのマイグレーションを実現するためには，ActiveOS が使用していたデバイスをマイグレーションする必要がある．これを実現する機構をデバイスマイグレーション機構と呼ぶ．

2.3　機能

2.1 節で挙げた Orthros の動作を実現するための各機構について述べる．各機構と目的を表 1 に示す．まず，機構の一部に利用される共有メモリについて述べ，その後それぞれの機構の概要を述べる．

2.3.1　共有メモリ

共有メモリは物理メモリの一部分に存在し，ActiveOS と BackupOS の双方から読み書き可能なメモリ領域である．この共有メモリは，ActiveOS 起動時に予約され，BackupOS 起動時に設定される．

2.3.2　複数 OS 同時実行機構

複数 OS 同時実行機構では，単一計算機上で 2 つの OS を起動し，アクティブとバックアップの構成を成す．また，各 OS は CPU コアや物理メモリ，ハードディスク，Network interface controller（NIC）などのデバイスを占有する．各 OS が排他的にデバイスを利用することで，一方の OS 上の障害が他方の OS へ波及することを防止している．

2 つの OS の起動は，初めに ActiveOS を起動し，その後 AcitiveOS が BackupOS を起動することで行う．BackupOS の起動は，Kexec と Kdump[7] を改良したソフトウェアを用いて行う．Kexec は，Linux に実装された OS のウォームブート機能である．Kdump は，Kexec を利用してシステム障害時に，予約されたメモリ領域に新たなカーネルをウォームブートすることで，ウォームブートしたカーネルから障害発生時点のカーネルのメモリのダンプを取得する機能である．Orthros では，Kdump を応用することで障害発生前に BackupOS をウォームブートし，障害発生時に ActiveOS のメモリ読み出しを行っている．

2.3.3　死活監視機構

死活監視機構は，BackupOS が ActiveOS の停止を検知するための機構である．BackupOS は起動後に Inter-Processor Interrupt（IPI）による ActiveOS の死活監視を行う．Orthros では，ActiveOS の使用する CPU コアから BackupOS の使用する CPU コアに対する割り込みを行うために用いている．使用する IPI は，ActiveOS に障害が発生したことを伝える異常通知と正常に動作していることを伝える生存通知の 2 種類である．ActiveOS が自身に障害を検知してカーネルパニックが発生した場合，`panic()` 関数内部で BackupOS に対して異常通知を送信する．しかし，`panic()` 関数が呼ばれないために異常通知を送ることができない場合がある．そこで，ActiveOS は各コアから定期的に BackupOS の使用している CPU コアに対して生存通知を送信する．BackupOS は一定時間生存通知を受信しない場合に ActiveOS が異常停止したと判断する．以上の死活監視によって ActiveOS の停止が検出された場合に，BackupOS はフェイルオーバー処理を開始する．

2.3.4 デバイスマイグレーション機構

デバイスマイグレーション機構は，通常時 BackupOS が使用不能である ActiveOS の物理メモリ領域とデバイスを，フェイルオーバー時に BackupOS で利用可能にする．障害発生時，BackupOS の物理メモリは実行に必要な最小限のみ割り当てられており，メモリ領域不足によりプロセス及びファイルキャッシュのマイグレーションが行えない場合がある．そこで，マイグレーションを確実に行えるようにするため，ActiveOS の物理メモリを BackupOS にマイグレーションする必要がある．また，ハードディスクはファイルキャッシュの書き戻しやプロセスが使用していたファイルを参照するために利用される．さらに，NIC もプロセスが使用していたネットワーク環境を引き継ぐために利用される．この機構は，ファイルキャッシュ及びプロセスマイグレーション開始前に実行される．

2.3.5 ファイルキャッシュマイグレーション機構

低速な I/O 処理の回数を削減するため，OS はハードディスク上のデータがアクセスされるとそのデータを含むブロックをメモリにバッファし，同一ブロックに対する読み書きはハードディスクではなくこのバッファに対して行う．このバッファをファイルキャッシュという．ファイルキャッシュマイグレーション機構は，ActiveOS が障害によりディスクに書き戻せなかったファイルキャッシュを BackupOS がディスクに書き戻す機構である．この機構は，Ext3 ファイルシステムに対して実装されている．

2.3.6 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 で動作していたプロセスを BackupOS に移植する機構である．Orthros では，ActiveOS のメモリを読み取ることでプロセスの実行状態の取得を行い，読み取った実行状態と同一の実行状態を持つプロセスを BackupOS 上に生成することでプロセスマイグレーションを実現する．この時 ActiveOS のメモリを読み取るために，BackupOS にメモリをマッピングする必要がある．マッピングするメモリ領域は ActiveOS のストレートマップ領域のみであるため，マイグレーションに利用する情報はストレートマップ領域に存在する必要がある．

この方法を用いてプロセスマイグレーションを行うことで，Checkpoint/Restart（C/R）[8][9] を用いる場合と比較して，通常実行時のオーバーヘッドを小さくすることが可能である．

3. 関連研究

本章では，OS の障害とその検知に焦点を当てる関連研究について述べる．

3.1 What is System Hang and How to Handle it[10]

OS の一部または全体が待機状態となり，ユーザアプリケーションから一切の反応がなくなる状態を Hang と定義し，この Hang を引き起こす原因を分類，調査している．分類した 6 種類の障害に対して，それぞれの障害が発生した時の Linux カーネルの実行状態を定義し，それを検知する機能を提案する．6 つの障害はデッドロックに関するものと無限ループに関するものに大別される．

- デッドロックの例：ロックの解放待ちによるビジーウェイティング
- 無限ループの例：`fork` の多発によって使用するリソースが膨大となり，他のプロセスのリソースを圧迫する

この手法では，`sar` コマンドを用いて障害時の実行状態の調査及び障害の検知を行っている．sar コマンドは Linux カーネルの実行ログを分析し，単位時間あたりのスループットを算出するコマンドである．この手法では，ユーザ空間からコマンドを介して自身の OS の障害を検知しているのに対し，本稿はカーネル空間から異なる OS の障害を検知するという点で異なる．

3.2 No PAIN, no gain?: the utility of PArallel fault INjections[11]

3.1 節の文献 [10] で提案された障害検知機能を発展させた研究である．この研究では，light detector と heavy detector の二つの障害検知機能を提案している．light detector は常時動作し，ログの収集と簡易的な検知を行う．heavy detector は light detector によって異常が検知された際に動作し，詳細な検知を行う．この手法により，検知精度の向上及びオーバヘッドの削減を実現している．この手法においても既存研究 [10] 同様，検知のために sar コマンドを実行して自身の OS を監視するため，カーネル空間から異なる OS の障害検知を行う本稿とは異なる．

3.3 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 [12]

本稿の先行研究であり，3.1 節の文献 [10] を基に自身の OS に起きた障害を検知する手法を提案する．この手法が自身の OS に起きた障害を検知するのに対し，本稿は異なる OS に起こった障害を検知するという点で異なる．

4. カーネル間メモリ監視による障害検知

本章では，まず提案により Orthros に実装する障害検知機構の概要について述べる．続いて実装する障害検知機構の動作について述べ，最後に障害検知機構が検知の対象とする障害について述べる．

4.1 概要

第 1 章で述べたように，システム全体の可用性向上のために Orthros に障害検知機構を実装する必要がある．提案する障害検知機構の概要は次の通りである．

- BackupOS が ActiveOS に起こる障害を監視
- 監視には ActiveOS のメモリ上に存在するカーネルパラメータを使用
- 定期的監視により取得したパラメータを用いて障害を検知

提案機構では，BackupOS が ActiveOS のメモリを監視することで障害の検知を行う．ActiveOS で行う処理を最小限に留めることで，ActiveOS のオーバヘッドを抑えることが可能であり，Orthros の目的であるシステムの可用性向上に寄与する．

監視対象は，既存研究 [10][12] を踏まえて ActiveOS のカーネル実行状態を示すカーネルパラメータとする．BackupOS から ActiveOS のメモリ上に存在するカーネルパラメータを監視する．カーネルパラメータを用いた障害検知を行うことで，多様な障害に対応することが可能となる．個々の障害を検知するパラメータの条件は既存研究を参考にした．この条件については 5.2.3 項で述べる．また，定期的監視における監視間隔については 5.2.2 項で触れる．

Orthros の特徴として仮想化オーバヘッドがないことを第 2 章で述べた．複数の OS を同時に実行するために仮想化技術を利用する場合，監視のためのメモリアクセスにハイパーバイザへのアクセスなどのオーバヘッドが発生する．そのため基盤として Orthros を用いることで，監視のためのメモリアクセスオーバヘッドを少なくすることが可能である．

4.2 動作

提案機構の動作を図 2 に示す．①BackupOS 起動後，ActiveOS は監視対象パラメータの物理アドレスを取得しそれを共有メモリに書き込む．続いて②BackupOS で，障害検知モジュールを起動する．このモジュールは，初回起動時，監視対象パラメータの物理アドレスを共有メモリから読み出し BackupOS の仮想アドレスに変換する．その後③変換したアドレスを用いて，設定間隔毎にパラメータを監視し，監視したパラメータを利用し障害の検知を行う．④モジュールで障害を検知した場合，フェイルオーバー及びマイグレーション処理を開始する．

4.3 想定する障害

第 1 章で述べたように，Orthros の目的はシステムの可用性向上である．Orthros の主な処理を行うのは ActiveOS であることから，Orthros における目的は ActiveOS の可用性向上と言い換えられる．そこで，本稿では ActiveOS の可用性を低下させる要因のある障害に焦点を当てる．具

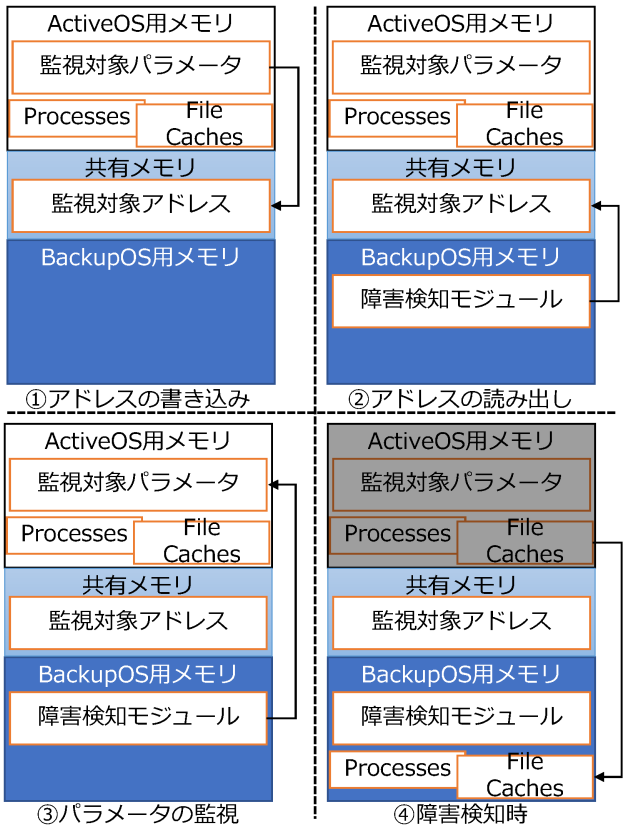

図 2 提案機構の動作

体的には，システムを停止，または著しくシステム全体に影響を与えて正常な動作を妨げる障害を想定する．ユーザレベルのプロセスやソフトウェアで発生する障害については対象とせず，Linux カーネルの実行中に発生する障害について述べる．本稿では先行研究 [12] において言及された次の障害を想定する．

- 障害 1：デッドロックによる障害
- 障害 2：メモリ確保とプロセス生成による障害
- 障害 3：バグを含むデバイスドライバによる障害

障害 1 はスピンロックによって発生する CPU のデッドロックによる障害である．Linux カーネルはスピンロックを用いて排他制御を行うことがある．スピンロックは lock と unlock のペアによって実装されており，この命令の欠損は正しい排他制御を行うプロセスの妨げとなる．スピンロックの解放を待つ間，CPU はビジーウェイティング状態となる．そのためスピンロックが永遠に解放されない場合，スピンロックの解放を待つ CPU はデッドロック状態となる．このようなデッドロック状態による CPU の障害を障害 1 とする．

障害 2 は，メモリ確保とプロセス生成による障害である．メモリの確保とプロセスの生成は，大量に繰り返すことによってコンピュータのリソースを枯渇させることが可能である．ユーザ空間でメモリの確保とプロセスの生成を大量に行うプログラムを実行することで，この障害を発生させる．この障害により，他のプログラムへのリソースの割り

当てに不具合が発生する可能性がある．

障害 3 は，バグを含んだデバイスドライバをインストールすることによって発生する障害である．この障害により，当該デバイスを利用する度に誤った処理が繰り返されることで，他のプログラムに影響を与える可能性がある．

5. 実装

本章では，第 4 章で述べた提案手法の実現方法について述べる．既存 Orthros に対し実装を追加または変更した部分を実装対象の OS 毎に明記する．

5.1 ActiveOS に対する実装

ActiveOS に追加した処理は，監視対象パラメータの物理アドレスを BackupOS に伝える処理である．この処理を実現するために，2.3.1 項で述べた共有メモリを利用した．まず監視対象となるカーネルパラメータの詳細について述べ，その後 ActiveOS に追加した詳細な実装方法を述べる．

5.1.1 監視対象パラメータ

BackupOS に実装する障害検知機構では，ActiveOS のカーネルの実行状態を示すカーネルパラメータを監視することで障害を検知する．ここで，監視対象のパラメータは既存研究 [12] を参考に次の 3 種類とした．

- CPU 使用率
- メモリ使用率
- プロセス情報

CPU 使用率は，各コア毎に単位時間あたりのカーネル処理に費やした CPU 時間を取得して計算した．この CPU 時間は，コア毎の CPU 情報を格納している `kstat_cpu` 構造体を参照し取得した．計算式は `vmstat` コマンドのソースを参考にしている．また，単位時間あたりのコンテキストスイッチ数も取得した．

メモリ使用率は，メモリの空き容量とページキャッシュの容量から計算した．容量の取得は，メモリに関する情報を格納している `si_meminfo` 構造体を参照している．

プロセス情報は，ActiveOS 上で動作するプロセスの数と状態を取得した．コア毎のプロセスのランキュー情報を持つ `cpu_rq` 構造体を参照し，コア毎の値を足し合わせることで計算した．取得した値は，ブロック状態のプロセス数と割り込み不可状態のプロセス数である．ブロック状態プロセスはメモリを確保できないプロセスであり，システムのメモリが大量に消費されている場合に発生する．割り込み不可状態プロセスはデバイスドライバによる I/O 待ち状態のプロセスであり，大量のプロセスが割り込み不可状態となるのはデバイスドライバの障害を示す．この割り込み不可状態のプロセスの使用するリソースは，プロセスが終了するまで解放されず，他のプロセスの動作の妨げとなる．更に，割り込み不可状態のプロセスはシステムが終了するまで kill されないため，常にプロセステーブルを圧迫することになる．そこで本稿はカーネル空間で，割り込み不可状態を示す `uninterruptable` フラグを持つプロセスの数を数えることで，割り込み不可状態のプロセスの数を監視した．uninterruptable フラグを持つプロセスは各コアのランキュー情報から得た．

5.1.2 監視対象パラメータアドレスの書き込み処理

本項では，ActiveOS が監視対象のパラメータのアドレスを共有メモリに書き込む処理の詳細について述べる．まず，監視対象パラメータの仮想アドレスを取得する．監視対象パラメータの詳細は前節で述べたが，全て Linux カーネル内部に存在する構造体のメンバである．2.3.6 項で述べたように，BackupOS が読み取り可能なメモリ領域は ActiveOS のストレートマップ領域のみであることから，監視対象のパラメータについてもストレートマップ領域に存在する必要がある．次に，仮想アドレスを `virt_to_phys` 関数を用いて物理アドレスに変換する．最後に，変換した物理アドレスを共有メモリに対して書き込む．書き込みは，パラメータ毎に共有メモリの決められたブロックに対して行う．このパラメータと書き込みブロックの関係は，ActiveOS と BackupOS で共有する必要がある．

ActiveOS に対して追加実装した箇所は以上である．共有メモリ設定後にこの一連の動作を実行するように Orthros の実装を変更した．上記の一連の動作が行われるのは，共有メモリ設定後一度だけであることからこの動作のオーバヘッドはほとんど存在しないことが予想される．ActiveOS のオーバヘッドについての評価は 6.2 節で述べる．

5.2 BackupOS に対する実装

BackupOS に追加した障害検知機構の処理は，次の 3 つである．

- 共有メモリからパラメータのアドレスを読み出す
- パラメータの定期的な監視
- パラメータを用いた障害の検知

障害検知機構の実装にはカーネルモジュールを用いた．BackupOS のユーザ空間から ActiveOS のカーネルパラメータを直接監視することは不可能なためである．

このカーネルモジュールを以降，障害検知モジュールとする．障害検知モジュールは初実行時，共有メモリから監視対象パラメータの物理アドレスを一度だけ読み出す．その後，読み出したアドレスを用いてパラメータの監視を定期的に行う．監視によって更新された値や前回監視時の値を用いて障害の検知を行う．以降，障害検知モジュールのそれぞれの処理の詳細について述べる．

また，BackupOS がフェイルオーバー及びマイグレーション処理を開始するタイミングを次のように変更した．

- 死活監視機構により ActiveOS の停止が確認された場合または障害検知機構により障害が検知された場合に，フェイルオーバー処理を開始する

5.2.1 監視対象パラメータアドレスの読み出し処理

監視対象パラメータのアドレスを知るために，共有メモリのあらかじめ決められたブロックからこれを読み出す必要がある．渡される物理アドレスは監視対象パラメータ毎に存在するため，これを全て読み出す．これは障害検知モジュールの初実行時に一度だけ実行される処理である．続いて，読み出した物理アドレスを `phys_to_virt` 関数を用いて BackupOS の仮想アドレスに変換する．2.3.6 項で述べたように，ActiveOS のストレートマップ領域が BackupOS にマッピングされているためこの変換が可能となる．その後，仮想アドレスを参照して ActiveOS のメモリ上に存在するパラメータの監視を定期的に行う．

5.2.2 監視対象パラメータの定期的な監視処理

障害検知モジュールは障害によって停止してはならないため，カーネルタイマーを利用してパラメータを用いた障害の検知を定期的に実行する．タイマー割り込みは Linux カーネルの制御の基本となる時間を示しており，最も優先される割り込みである．そのため障害が発生し Linux カーネルの実行状態に異常が発生した後も，タイマー割り込みが動作している間は障害検知モジュールは動作し，障害検知処理を実行可能である．さらに Orthros を基盤として採用したことで，ActiveOS のタイマーが停止しても BackupOS のタイマーが動作するため障害検知が可能となる．作成したタイマーを繰り返し実行することによって，定期的な障害検知処理の実行を可能とした．

5.2.3 監視対象パラメータを用いた障害検知処理

障害検知処理は障害検知モジュールによって繰り返される．障害検知処理は ActiveOS のカーネルの実行状態を示すパラメータを監視し，パラメータを基に障害検知を行う．取得したパラメータは直前に取得した値を記憶して比較することで障害検知に利用する．検知条件についても既存研究 [12] を参考にした．本項では障害検知処理の実装について述べる．

障害 1 はスピンロックに関する CPU の障害を想定した．そのためカーネル処理に費やされた CPU 使用率と単位時間あたりのコンテキストスイッチ数が検知に必要である．カーネル処理に費やされた CPU 使用率が 95%以上かつ単位時間あたりのコンテキストスイッチ数が 0 である時，デッドロック状態と定義する．ここにおける単位時間はパラメータの監視間隔すなわち検知モジュールのタイムアウト値である．障害検知機構は，複数のコアがデッドロック状態となって 1 秒以上が経過した時，もしくは 1 つのコアがデッドロック状態となって 5 秒以上が経過した時に，障害 1 を検知する．

障害 2 はメモリとプロセス生成による障害を想定した．メモリの空き容量とページキャッシュの合計及びブロック状態のプロセス数が検知の条件である．障害検知機構は，メモリの空き容量とページキャッシュの合計が ActiveOS の使用可能メモリ領域全体の約 15%以下となり，ブロック状態プロセス数がコア数の 3 倍を越した時，障害 2 を検知する．この状態の時，fork の多発によるリソースの枯渇とランキューの圧迫によって Linux カーネル及び他のプログラムの挙動が不安定になっていると考えられる．

障害 3 はバグを含むデバイスドライバによる障害を想定した．障害検知機能は割り込み不可状態のプロセス数が 10 以上となった時に障害 3 を検出する．デバイスドライバによってスリープされたプロセスが増加することによって．プロセスの数が増加し続ける可能性が考えられる．本稿では割り込み状態のプロセス数が 10 以上の時に障害 3 を検知する．割り込み禁止状態のプロセスはカーネルの動作を停止させることはないが，プロセステーブルには常に 0 であることが望ましいため，本稿では 10 以上で検知を行い，フェイルオーバー処理を開始する．

5.2.4 フェイルオーバー及びマイグレーション処理開始タイミングの変更

既存の Orthros では，死活監視機構により ActiveOS の使用するプロセッサが停止した際にフェイルオーバー及びマイグレーション処理を開始する．本稿では，上記処理の開始タイミングに障害検知機構により各障害が検知された場合を加えた．提案機構が障害を検知した際に，監視及び検知を終了しフェイルオーバー及びマイグレーション処理に移行する．

6. 評価

本章では，実装した障害検知機構の動作を評価する．評価環境を表 2 に示す．カーネルのバージョンは既存の Orthros と同じく 2.6.38.7 とした．ActiveOS は 4 コア中 3 コア，メモリ約 7GB を専有する．共有メモリは 256MB と設定した．

表 2 評価環境

	ActiveOS	BackupOS
OS	Arch Linux (Kernel Version 2.6.38.7)	
CPU	Intel Core i7 3770 (4cores 3.40GHz)	
	3cores	1core
Memory	8GB	
	7361MB	312MB
Device	SSD/NIC	SSD/NIC

まず，提案機構のオーバヘッドを評価する．ActiveOS におけるオーバヘッドと BackupOS におけるオーバヘッドをそれぞれ測定する．続いて，提案機構の各障害に対する有効性を評価する．この評価は，パラメータの監視間隔は 1ms，10ms，100ms，1000ms を用いる．

6.1 障害検知機構のオーバヘッド

本節では，実装した障害検知機構のオーバヘッドの測定

表 3　ActiveOS における UnixBench の実行結果

障害検知機構なし	4209.10
障害検知機構あり (1ms)	4206.58

表 4　各監視間隔における BackupOS の CPU 周波数統計

監視間隔	Idle	1.6Ghz	3.4Ghz	Turbo Mode(3.9Ghz)
1ms	23.42%	0.0%	76.58%	0.0%
10ms	51.72%	0.0%	48.28%	0.0%
100ms	99.9%	0.0%	0.1%	0.0%
1000ms	100.0%	0.0%	0.0%	0.0%

について述べる．ActiveOS ではベンチマーク実行，BackupOS では CPU 周波数の統計によりオーバヘッドを評価する．

6.1.1　ActiveOS におけるオーバヘッド

ActiveOS におけるオーバヘッドの測定のために，障害が発生しない状態において UnixBench5.1.3 を実行した．計測に用いたベンチマークは 512Byte のデータのパイプ処理を繰り返すものである．この処理にかかる時間から，CPU と OS の処理性能を算出する．

障害検知機構を動作させない状態と動作させた状態でそれぞれベンチマークを実行させ，評価を行った．評価結果を表 3 に示す．評価結果は 128MB のメモリと Solaris2.3 の OS での処理性能を 10 として正規化した数値によって表される．結果から 1ms 間隔の監視を行う障害検知機構を実装した場合にもほとんどオーバヘッドが発生しないことが確認できた．これは，ActiveOS で行う処理は監視対象パラメータのアドレスを共有メモリに一度だけ書き込むことのみであることに起因し，性能の低下は誤差の範囲内であると考えられる．

6.1.2　BackupOS におけるオーバヘッド

BackupOS におけるオーバヘッド測定のために，powertop を用いて CPU の周波数統計を計測した．powertop は Intel が提供する電力管理ツールで，CPU の wakeup 回数や CPU 周波数の統計情報などを表示する．各監視間隔における CPU 周波数統計の結果を表 4 に示す．

監視間隔が短い程，CPU は高い周波数で動作し BackupOS に負荷がかかっていることが確認できる．また，特に 10ms より短い間隔の監視においては CPU における消費電力が大きく増加していることが分かる．しかし BackupOS の目的は ActiveOS の障害に備えることであり，待機中は基本的に一切の処理を行わないためオーバヘッド自体に問題がないと言える．また 1ms 及び 10ms 間隔で監視する場合においても BackupOS においては高負荷であるが，ActiveOS での実処理には影響はないと考える．

6.2　障害に対する有効性

本節では，実装した提案機構の各障害に対する有効性について述べる．提案機構により各障害が検知可能であることと検知時のパラメータの値を確認する．

表 5　障害 2：パラメータ監視間隔と障害検知時のパラメータ

監視間隔	メモリ空き容量とページキャッシュ (KB)	ブロック状態のプロセス数
1ms	1044912.5	24940.2
10ms	996053.6	25121.0
100ms	870112.8	25803.6
1000ms	246921.6	25662.6

6.2.1　デッドロックによる障害に対する評価

障害 1 では，ActiveOS の使用するカーネルコード内の次の箇所にスピンロックを解除しないバグを挿入してデッドロックを意図的に発生させる．

- ファイルシステム
- メモリ

上記 2 箇所の障害について，どちらも提案機構で検知可能であることを確認した．また，監視間隔を変化させることで早期の障害検知を実現可能であることを確認した．しかし障害 1 の検知に 1 秒または 5 秒の時間的猶予があるため，障害検知までの時間の差は誤差の範囲内であると考える．

6.2.2　プロセスの生成とメモリの確保による障害に対する評価

障害 2 は，ActiveOS 上でプロセスの生成とメモリの確保を無限に繰り返す LKM を実行する．この障害について，提案機構で検知可能であることを確認した．パラメータの監視間隔と障害検知時のパラメータを表 5 に示す．結果から，パラメータの監視間隔が短い程使用可能メモリ量が大きく，プロセス数が少ないタイミングで検知できていることが確認できた．

6.2.3　バグを含むデバイスドライバによる障害に対する評価

障害 3 は，キャラクタ型デバイスドライバに意図的にバグを挿入することで発生させる．実際のデバイスに障害を発生させるのはリスクが高いため，本稿ではキャラクタ型デバイスとそれを利用するデバイスドライバを作成し，このデバイスドライバに対してバグを挿入する．本キャラクタ型疑似デバイスは，ユーザプロセスによるデータ書き込み機能と，そのデータの読み出し機能を提供する．また，データが書き込まれていない状態での読み出しアクセスをブロックし，当該プロセスを停止させる．このデバイスドライバに対してプロセスから読み出し要求があった場合の動作を図 3 に示す．

①プロセスからデバイスに読み出し要求があると，デバイスドライバがそれを受け付け，②デバイス内に書き込まれたデータの有無を確認する．③読み出し可能なデータがある場合は④デバイスドライバがそのデータをプロセスに返す．③’ 読み出し可能なデータがない場合，④’ デバイ

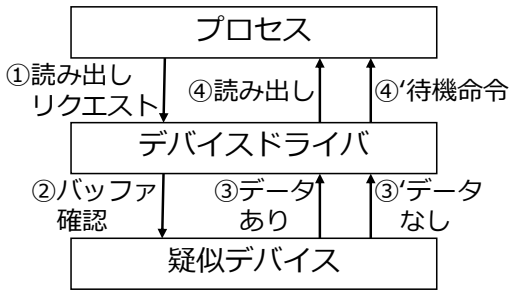

図 3 デバイスドライバの読み出し時の動作

表 6 障害 3：パラメータ監視間隔と障害検知時のパラメータ

監視間隔	割り込み不可状態のプロセス数
1ms	36.8
10ms	203.4
100ms	1761.4
1000ms	21952.0

スドライバはプロセスをブロックし，プロセスは待機状態となる．この時，待機状態となったプロセスはデバイスドライバによって uninterruptable フラグを設定されており，デバイスドライバが解除するまで待機状態が継続する．デバイスドライバは他のプロセスから書き込みリクエストがあった時，uninterruptable フラグを解除し，待機状態となっているプロセスを実行可能状態とする．

このデバイスドライバに対して，書き込みリクエストがあってもプロセスの待機状態を解除しないバグを挿入した．さらに，デバイスドライバにアクセスするプロセスを無条件ループによって無制限に生成し，待機状態となるプロセスを増加させた．

この障害について，提案機構で検知可能であることを確認した．パラメータの監視間隔と障害検知時のパラメータを表 6 に示す．結果から，監視間隔が短い程少ないプロセス数での検知が可能であることを確認した．

7. まとめ

本稿では，一台の計算機上に複数の OS を同時に実行するプロセス耐障害性向上システム Orthros における障害検知機構の実装について述べた．既存の死活監視による障害検知は，プロセッサを停止しないが ActiveOS の可用性を低下する障害が想定されておらず，対応可能な障害が少ないことが予想される．更に障害を特定できないことにより，フェイルオーバー処理によって障害を引き継ぐ可能性がある．そこで，Orthros に対して多様な障害に対応可能な障害検知機構を実装する．

提案機構では，BackupOS から ActiveOS のメモリ上に存在するカーネルの実行状態を表すパラメータを監視し，監視したパラメータを基に障害を検知する．障害の検知にカーネルパラメータを用いることで多様な障害に対応可能となる．また提案機構は，カーネル空間からの監視を行うため，ユーザ空間からの監視と比較してオーバヘッドの削減が可能である．本稿では対象とする障害として，デッドロックによる障害とプロセス生成とメモリ確保の障害，バグを含むデバイスドライバによる障害の 3 種類を選定した．評価ではこれら全ての障害に対して，提案機構により検知可能であることを確認した．また，提案機構のオーバヘッドは ActiveOS において存在せず，BackupOS においては監視間隔が短い程高負荷ではあるが，ActiveOS の実処理に影響はない範囲であることを確認した．より多様な障害に対応することが今後の課題である．

参考文献

[1] Chou, A., Yang, J., Chelf, B., Hallem, S. and Engler, D.: *An empirical study of operating systems errors*, Vol. 35, No. 5, ACM (2001).

[2] Palix, N., Thomas, G., Saha, S., Calvès, C., Lawall, J. and Muller, G.: Faults in Linux: Ten years later, *ACM SIGARCH Computer Architecture News*, Vol. 39, No. 1, ACM, pp. 305–318 (2011).

[3] 吉田健二，齋藤彰一，毛利公一，松尾啓志：プロセスの耐障害性向上のための多重 OS の開発と評価，情報処理学会論文誌. コンピューティングシステム， Vol. 7, No. 2, pp. 11–24 (2014).

[4] Shimosawa, T., Matsuba, H. and Ishikawa, Y.: Logical partitioning without architectural supports, *Computer Software and Applications, 2008. COMPSAC'08. 32nd Annual IEEE International*, IEEE, pp. 355–364 (2008).

[5] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I. and Warfield, A.: Xen and the art of virtualization, *ACM SIGOPS Operating Systems Review*, Vol. 37, No. 5, pp. 164–177 (2003).

[6] Kivity, A., Kamay, Y., Laor, D., Lublin, U. and Liguori, A.: kvm: the Linux virtual machine monitor, *Proceedings of the Linux Symposium*, Vol. 1, pp. 225–230 (2007).

[7] NetworkX, L.: Kdump, A Kexec-based Kernel Crash Dumping Mechanism, *Linux Symposium*, p. 169 (2005).

[8] Hargrove, P. H. and Duell, J. C.: Berkeley lab checkpoint/restart (blcr) for linux clusters, *Journal of Physics: Conference Series*, Vol. 46, No. 1, IOP Publishing, p. 494 (2006).

[9] Liao, J. and Ishikawa, Y.: A new concurrent checkpoint mechanism for real-time and interactive processes, *Computer Software and Applications Conference (COMPSAC), 2010 IEEE 34th Annual*, IEEE, pp. 47–52 (2010).

[10] Zhu, Y., Li, Y., Xue, J., Tan, T., Shi, J., Shen, Y. and Ma, C.: What is system hang and how to handle it, *2012 IEEE 23rd International Symposium on Software Reliability Engineering*, IEEE, pp. 141–150 (2012).

[11] Winter, S., Schwahn, O., Natella, R., Suri, N. and Cotroneo, D.: No PAIN, no gain?: the utility of PArallel fault INjections, *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, IEEE Press, pp. 494–505 (2015).

[12] 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，研究報告システムソフトウェアとオペレーティング・システム (OS)，Vol. 2016, No. 7, pp. 1–9 (2016).