Vol.2017-OS-139 No.11
2017/3/2

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

新美 渓介[1,a)]　瀧本 栄二[2]　毛利 公一[2]　齋藤 彰一[1]

概要：ハードウェアを仮想化することなく 1 つの OS の上で複数のユーザ空間を提供するコンテナ型仮想化では，単一障害点である OS に障害が発生した場合，その上で動作するすべてのユーザ空間は破壊される．コンテナはサーバで利用されることが多く，可用性低下を抑えるためコンテナの保護が必要である．そこで本稿では，単一計算機上で動作する 2 つの OS 間でプロセスを移植し保護するシステムである Orthros を用いて，コンテナをマイグレーションすることによって保護する手法を提案する．そのため，コンテナを実現するための Linux の機能である cgroup と Namespace の，Orthros におけるマイグレーションを実装する．また，サーバで利用されることの多いプロセスの中で，プロセス間通信に用いられる UNIX ドメインソケットのマイグレーションを実装し，Orthros で保護できるプロセスの適用範囲の拡張を行う．

1. はじめに

現在，計算機上で仮想環境を実現する技術としてコンテナ型仮想化が用いられている．コンテナ型仮想化を実現するソフトウェアとして，Linux Containers(LXC[1])，Linux-VServer[2] などがある．これらはハードウェアの仮想化をせずに 1 つの OS 上で複数のプロセスを分離された環境で動作させる方法であり，この分離された環境及びこれを作る機構をコンテナと呼ぶ．コンテナ型仮想化は，Linux の標準機能である cgroup[3] や Namespace[4] によって，プロセスごとに使用できるリソースを分離し，ネットワークなどの空間を分離することによって，複数の仮想環境を実現している．コンテナ内ではホスト OS と同一のカーネルを利用してプロセスが動作するため，リソースのオーバヘッドが少なく，仮想環境を作る際に OS のブート手順が必要とならないため，高速なプロセスの起動が可能となる．

また，コンテナ型仮想化の利便性を向上させるソフトウェアとして，Docker[5]，rkt[6] などのコンテナ管理ソフトウェアがある．これらを利用することで，コンテナ内で使用するファイルや設定をイメージとして保存し再利用することができ，ソフトウェア開発などのコンテナの利便性が高まる．そのため，コンテナ管理ソフトウェアの普及と共に，コンテナ型仮想化の利用が拡大している．

仮想化によって作られた複数の仮想環境はサーバ運用やソフトウェア開発で利用されることが多いため，障害発生による仮想環境の動作停止は多大な損害が発生することが考えられる．しかし，コンテナ型仮想化は OS に対する耐障害性が備わっておらず，ホスト OS に障害が発生すると，その上で動作するすべての仮想環境は破壊されるため，このような状況は好ましくない．そこで本稿では，コンテナ型仮想化の OS 障害に対する耐障害性向上を目的とした手法を提案する．

第 2 章では本研究の対象とするコンテナ型仮想化のコンテナ作成技術について述べ，第 3 章では既存研究であるプロセス耐障害性向上システム ORganized Transmigratory High-Reliability OS(以下，Orthros[7]) について述べる．第 4 章で提案方式について述べ，第 5 章で本機構の実装を述べる．第 6 章では評価について示す．第 7 章では関連研究について述べ，最後に第 8 章でまとめを述べる．

2. コンテナ作成技術

1 台の物理計算機上で複数の仮想環境を実現する技術としてコンテナ型仮想化があり，それによって作られる仮想環境をコンテナと呼ぶ．コンテナはハードウェアの仮想化を行わず，プロセスのまとまりであるプロセスグループごとに使用できるリソースを制限し，ネットワークなどの空間を分離することで，1 つの物理計算機上での複数の仮想環境を実現している．本節では，Linux の標準機能として搭載されており，コンテナを作るために必要な機能である

[1] 名古屋工業大学
Nagoya Institute of Technology
[2] 立命館大学
Ritsumeikan Uniersity
[a)] orthros@mail.ssn.nitech.ac.jp

表 1　cgroup のサブシステム

サブシステム	概要
blkio	ブロックデバイスへの入出力を制限，監視する
cpu	使用する CPU 使用時間を制御する
cpuacct	CPU リソースの自動レポートを生成する
cpuset	使用可能な CPU コアやメモリノードを指定する
devices	デバイスファイルへのアクセスを制限する
freezer	プロセスグループの一時停止や再開を行う
memory	使用可能なメモリの上限を制限する
ns	名前空間への対応をする

表 2　名前空間

名前空間	分離するリソース
IPC	プロセス間通信リソース
MNT	ファイルシステムツリー
NET	ネットワークインタフェース
PID	PID 空間
User	UID/GID 空間
UTS	システム識別子

cgroup と Namespace について述べる．

2.1　cgroup

cgroup はプロセスグループ単位でのハードウェアのリソース制限を行う機能である．制限できるリソースのことをサブシステムと呼び，各サブシステムの中に複数のパラメータが存在する．サブシステムは 8 種類であり，概要を表 1 に示す．

cgroup の利用は，cgroup ファイルシステム (以下，cgroupfs) という仮想的なファイルシステムを用いて操作する．cgroupfs をマウントすると，そのマウントポイント以下に設定を行うための様々なテキストファイルが作られる．その設定ファイルに制限したいプロセスの PID やパラメータを書き込むことで，プロセスグループの指定やリソースの制限を行うことができる．また，プロセスグループは階層型であるためプロセスグループの下にさらにプロセスグループを作成することができ，その際，子グループは親グループからの設定を継承する．コンテナ型仮想化においては，1 つのコンテナ内で動作する複数のプロセスを 1 つのプロセスグループとし，使用するリソースの制限を行う．

2.2　Namespace

Namespace はプロセスごとにリソースの名前空間を分離する機能である．名前空間とはリソース等の名前の重複を防止するための概念であり，名前空間が分割されていると，プロセスは自身の名前空間外のリソースへの参照が制限される．この機能よって分離できるリソースは 6 種類であり，表 2 に示す．

通常，新しく生成されたプロセスは親プロセスの名前空間を引き継ぐ．新しい名前空間を作るには，clone() システムコールや unshare() システムコールを用いる．引数に分離する名前空間を指定することで，親プロセスと分離された名前空間で子プロセスが生成される．コンテナ型仮想化においては，コンテナごとにこれらすべての名前空間を分離し，個別の仮想空間を実現している．

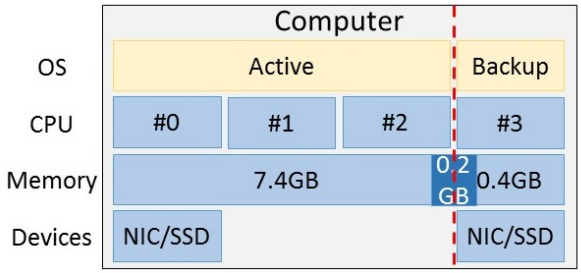

図 1　Orthors のハードウェア分割例

3.　Orthros

本章では，本研究室で提案されているプロセス耐障害性向上システム Orthros について述べる．本稿ではこの Orthros を用いたコンテナの耐障害性向上手法について述べる．Orthros は，ファイルキャッシュ及びプロセスの保護と高速なフェイルオーバーを，大きな性能低下なしに実現する手法である．ファイルキャッシュ及びプロセスの保護はメモリイメージ走査によるプロセスの実行状態取得を用いて行い，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．

3.1　構成

高速なフェイルオーバーを行うために，Orthros は 2 つの OS を用意してそれらをアクティブ，バックアップ構成で運用する．ActiveOS は主だった仕事のすべてを処理する現用系 OS であり，BackupOS は ActiveOS に障害が発生した際に使用される待機系 OS である．この 2 つの OS は 1 台の計算機上で同時に動作する．Orthros を用いた場合の計算機の構成を図 1 に示す．BackupOS は通常時は一切の処理をしないため，ActiveOS に CPU 及びメモリの大半を割り当て，デバイスは各 OS が独自に持っている．

3.2　動作概要

BackupOS は ActiveOS に障害が発生していないかを常に監視し，障害を検知した場合には，以下のようなステップで自動的にフェイルオーバーを行う．

(1) デバイスのマイグレーション
(2) ファイルキャッシュのマイグレーション
(3) プロセスのマイグレーション
(4) NIC への IP アドレス割り当て

(1) デバイスのマイグレーションは，ActiveOS が使用し

ていたデバイスを BackupOS が使用できるようにするための機構である．(2) ファイルキャッシュのマイグレーションは，ActiveOS が障害によってディスクに書き戻すことができなかったファイルキャッシュを代わりに BackupOS がディスクに書き戻すことで，ファイルキャッシュ損失によるファイルシステムの破損を防ぐための機構である．(3) プロセスのマイグレーションは， 事前に指定した ActiveOS で動作していたプロセスを BackupOS 上に復元することで継続動作を実現するための機構である．(4)NIC への IP アドレスの割り当ては，ActiveOS から BackupOS へマイグレーションされた NIC に対して，その NIC が ActiveOS 管理下で動作中に保持していた IP アドレスを再度割り当てる．

双方の OS が独立して動作することから，BackupOS は ActiveOS のメモリマップを保持しない．しかし Orthros では各 OS がソフトウェアレベルでハードウェアを占有するが，占有ハードウェア外のハードウェアにもアクセス可能であるという特徴を持つ．これを活かし，物理メモリの固定位置に共有メモリ領域 (図 1 における 0.2GB) を設け，ActiveOS によってフェイルオーバー処理で必要となる情報をこの領域に書き込むことで，BackupOS はこの情報に基づいて ActiveOS のメモリ領域の解析を行うことができる．

3.3 デバイスマイグレーション

各 OS は起動時に計算機に接続されているデバイスをすべて認識するが，双方の OS はあらかじめ占有使用するデバイスをそれぞれ指定しており，それぞれの OS が起動時に占有対象として指定されたデバイスのみを初期化し使用する．これにより，デバイスの排他利用を実現している．ActiveOS の障害発生時に，BackupOS は Linux カーネル内の ActiveOS 占有デバイスリストを参照し，マイグレーション対象のデバイスの使用するデバイスドライバを初期化することでデバイスを使用可能な状態にする．使用可能状態にした後，後続のフェイルオーバー処理を行うため，マイグレーションしたデバイス固有の処理や設定を行う．

3.4 ファイルキャッシュマイグレーション

カーネルのファイルキャッシュ機能は，ディスクから読み出したデータをメモリ上にページ単位で保存し，各メモリのページに割り当てられた page 構造体によって管理する．ファイルキャッシュマイグレーションは，ストレージに書き込まれていない ActiveOS のメモリ上のファイルキャッシュを検索し，BackupOS 上にコピーすることで行う．このファイルキャッシュをダーティページと呼ぶ．ダーティページはディスクの各ファイルシステムに対応してメモリ上に存在する super_block 構造体から辿ることができる．そして BackupOS で，ActiveOS にあるすべてのダーティページの内容をコピーすることで保護が可能である．ファイルキャッシュマイグレーションは ext3 ファイルシステムを対象に実装されている．

3.5 プロセスマイグレーション

プロセスマイグレーションは，ActiveOS で事前に指定されたプロセスを対象とする．BackupOS が ActiveOS のメモリイメージから，プロセスに関する情報が保存されている task_struct 構造体を読み出した後，BackupOS 内でプロセスの再現に必要な情報をコピーして再構成をすることで実現する．Orthros では，単一のプロセスとして動作するために最低限保護すべき状態をレジスタの値とメモリ内容と開いているファイルの管理状態とし，これらを保護する．これらをレジスタコンテキスト，メモリコンテキスト，ファイルコンテキストと呼ぶ．

レジスタコンテキストは，マイグレーションしたプロセスがコンテキストスイッチで実行可能状態から実行状態に移行する際に使用される．プロセスは，task_struct 構造体に保存されたレジスタコンテキストを CPU にセットすることで動作するため，AcitveOS 上から読み取ったレジスタコンテキストを BackupOS 上で task_struct 構造体にコピーすることで保護する．メモリコンテキストには，カーネル空間のコンテキストとユーザ空間のコンテキストの二つが存在する．コンテキストを単純にコピーすると，コピーしたコンテキスト内にポインタ変数が含まれていた場合，BackupOS のプロセスが ActiveOS のメモリを参照することになる．その ActiveOS のメモリ領域は，次の OS の障害に備えた新 BackupOS を起動する際に初期化される領域であるため，BackupOS のプロセスが ActiveOS のメモリを参照している状態は望ましくない．そのため，コピーするコンテキストがポインタ変数であった場合，その参照先の内容を取得し，コピーを行う．ユーザ空間に存在するデータはページング方式を利用した管理が行われているため，ページテーブルエントリごとにページのコピーをすることで保護する．ファイルコンテキストは，ファイルの実体に関する情報と，ファイルの扱い方に関する情報に分類される．Orthros ではファイルキャッシュマイグレーションによってファイルの実体に関する情報は保護される．プロセスマイグレーションにおいては，同じファイルを開いた後に，ファイルのパスやメモリマッピングなどファイルの扱い方に関する情報をコピーすることでファイルコンテキストを保護する．

4. 提案方式

コンテナ型仮想化によって作られる仮想環境の保護のため，Orthros を用いたコンテナマイグレーション手法を提案する．コンテナマイグレーションは，コンテナを実現するための機能である cgroup マイグレーションと Namespace

のマイグレーションに分類できる．また，コンテナ管理ソフトウェアの動作では，高速なプロセス間通信のためにUNIX ドメインソケットが用いられる．そのため，Orthros に UNIX ドメインソケットマイグレーションの機構を追加することで，コンテナ管理ソフトウェアをマイグレーション対象とする．

4.1 コンテナマイグレーション

既存の Orthros では，プロセスマイグレーションによって ActiveOS から BackupOS にプロセスを移植することによってプロセスを保護するが，コンテナの環境の保護をすることはできない．そこで，Orthros にコンテナマイグレーションの機構を追加することで，コンテナ環境を保護したままコンテナ内で動作するプロセスをマイグレーションできるようにする．そのため，コンテナを作るための機能である cgroup と Namespace の設定をマイグレーションする必要がある．

4.1.1 cgroup マイグレーション

cgroup は cgroupfs を利用し，ファイルシステム上の設定ファイルに書き込みを行うことでリソースの制限を行う．そのため cgroup マイグレーションでは，cgroupfs の内容をユーザレベルから再現する方法を採用する．まず ActiveOS の設定ファイルを保存する．その後，BackupOS からその設定ファイルを読み出して，その情報を元に BackupOS で同様の設定を再現する．保存や再現をする項目は，プロセスグループの階層構造，マイグレーション対象のプロセスが入っているグループの位置，各グループのリソース制限内容の 3 つである．

4.1.2 Namespace マイグレーション

Namespace は，プロセスごとに持つ task_struct 構造体に関連付けられた nsproxy 構造体によって管理されており，その中で名前空間ごとに管理を行うための構造体に分かれている．ただし，User 名前空間に関しては cred 構造体の中で UID/GID の管理がされている．Orthros の共有メモリ領域を利用して，ActiveOS のメモリ上に残っている各名前空間を管理する構造体を BackupOS から参照し，BackupOS で同様の名前空間を再現する．

4.2 UNIX ドメインソケットマイグレーション

ActiveOS で UNIX ドメインソケットを用いてプロセス間通信を行うプロセスが動作している場合，BackupOS にマイグレーションしたプロセスの正常な動作を保証するためには，UNIX ドメインソケットをマイグレーションする必要がある．コンテナ管理ソフトウェアのプロセスでは，プロセス間で通信を行う手段として，高速かつ安全な通信を行うことができる UNIX ドメインソケットが用いられる．そのため，Orthros に UNIX ドメインソケットマイグレーション機構を追加することで，コンテナ管理ソフトウェアを含め保護できるプロセスの範囲を拡張をすることができ，サーバ保護が可能となる．

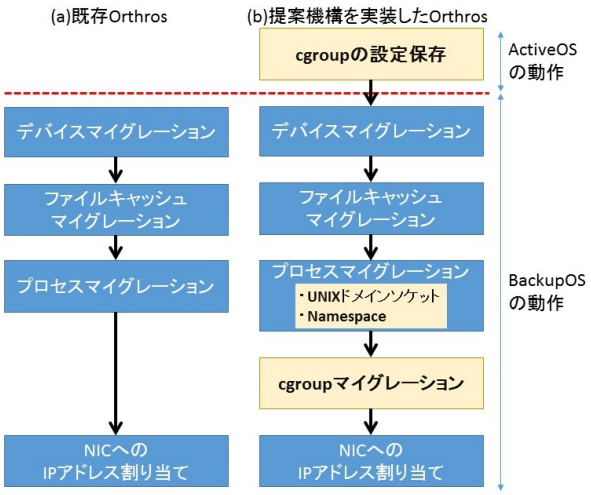

図 2 フェイルオーバー処理の流れ

5. 実装

本章では，第 4 章で述べた提案方式の実現方法について述べる．まず，コンテナマイグレーションのために必要な cgroup マイグレーションと Namespace マイグレーションの実装について述べ，次にサーバで動作するプロセス保護の範囲拡張のための UNIX ドメインソケットマイグレーションの実装について説明する．実装は既存の Orthros に追加する形式で，Linux(version 2.6.38.7) に行った．

Namespace マイグレーションと UNIX ドメインソケットマイグレーションはプロセスマイグレーションの機構の一部として実装し，cgroup マイグレーションは，動作が停止する前の ActiveOS で準備を行い，処理本体はプロセスマイグレーションが完了した後の処理として実装する．既存の Orthros と提案機構を実装した Orthros それぞれのフェイルオーバーの流れを図 2 に示す．新たに実装した提案機構は黄色で示す処理である．

5.1 コンテナマイグレーション

コンテナマイグレーションは，コンテナを作るために必要な Linux の標準機能である cgroup と Namespace のマイグレーションをそれぞれ実装することで可能となる．

5.1.1 cgroup マイグレーション

cgroup の設定は，cgroupfs をマウントしたディレクトリ以下にできた設定ファイルへ書き込みを行うことで cgroup のリソース制限を行う．ActiveOS でこの cgroup の設定ファイルの内容をすべて保存して BackupOS に渡し，BackupOS でファイルの内容を読み出して同じ設定を再現することで cgroup マイグレーションを実現する．マイグレーションはユーザ空間からシェルスクリプトによって行う．

まず，cgroup マイグレーションの準備として，ActiveOS

```
1  # プロセスグループの階層構造
2  /sys/fs/cgroup/cpu/group1
3  /sys/fs/cgroup/cpu/group2
4  /sys/fs/cgroup/cpu/group2/subgroup2−A
5  /sys/fs/cgroup/cpu/group2/subgroup2−B
6
7  # マイグレーション対象プロセスが入っているグループ
8  NAME;dummy_init
9  /sys/fs/cgroup/cpu/group1
10 NAME;target_process
11 /sys/fs/cgroup/cpu/group2/subgroup2−A
12
13 # 各プロセスグループのリソース制限内容
14 DIR;/sys/fs/cgroup/cpu/group1
15 file;cpu.shares
16 512
17 file;notify_on_release
18 0
```

図 3 cgroup マイグレーションファイルの記述例

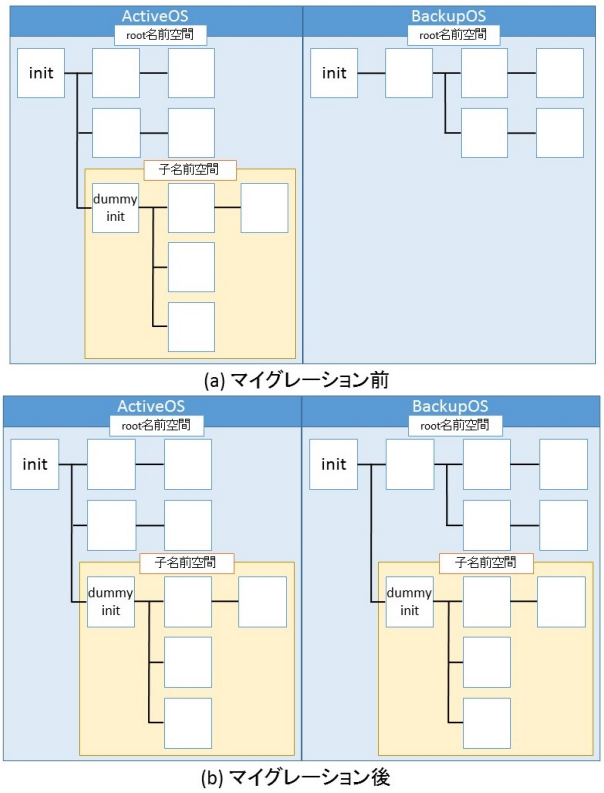

図 4 マイグレーション前後の PID 名前空間

で設定の保存をする．保存する情報は，プロセスグループの階層構造，マイグレーション対象のプロセスが入っているグループ，各グループのリソース制限内容の 3 つである．これらの情報は cgroupfs でマウントをしたディレクトリの構造やそのディレクトリ下のファイルの内容を見ることで取得することができ，それを元に BackupOS で再現をすることができる．しかし，すべてのファイルを ActiveOS から BackupOS に受け渡すことは煩雑な手順となり，処理に時間がかかることが考えられる．そのため，上記 3 つの情報すべてを 1 つのテキストファイルにまとめ，そのテキストファイルを受け渡すことで処理を簡略化し，処理時間の短縮を図る．このテキストファイルのことを今後 cgroup マイグレーションファイルと呼ぶ．

cgroup マイグレーションファイルの記述例を図 3 に示す．このファイルにはプロセスグループの階層構造，マイグレーション対象のプロセスが入っているグループ，各グループのリソース制限内容を記述する．プロセスグループの階層構造を保存するため，cgroupfs がマウントされているディレクトリの絶対パス名をすべて cgroup マイグレーションファイルに記述する (2-5 行目)．これは，プロセスグループの親子関係とそのグループに対応するディレクトリ構造の親子関係は一致しているため，ディレクトリの絶対パスがあればプロセスグループの階層構造が分かるためである．次に，マイグレーション対象のプロセスが入っているプロセスグループを保存するため，プロセス名とそのプロセスが入っているすべてのグループのディレクトリ名を一括して記述する (8-11 行目)．最後に，リソース制限内容が書き込まれている設定ファイルはプロセスグループ 1 つに対して複数あり，全サブシステムのプロセスグループ分だけ存在する．これを保存するため，ディレクトリの絶対パス名を記述し，その後にリソース制限のファイル名とパラメータを一括して記述する (14-18 行目)．

ActiveOS に障害発生後，BackupOS で cgroup マイグレーションが開始し，ActiveOS から受け取った cgroup マイグレーションファイルの情報を元に cgroup の設定の再現をする．cgroupfs をマウントし，ActiveOS と同じプロセスグループの階層構造になるようディレクトリを作成する．その後，マイグレーションされたプロセスを，元と同じプロセスグループに入れる．プロセスの PID をプロセス指定のためのファイルに記述する必要がある．しかし，ActiveOS で動作していたプロセスと BackupOS にマイグレーションされたプロセスでは PID が異なっているため，ActiveOS から PID を受け取ることは意味がない．そこで，ActiveOS からはプロセス名を受け取り，BackupOS で pidof コマンドを用いてプロセス名から PID を取得する．その際，他に同じ名前のプロセスがあった場合に，間違ったプロセスをプロセスグループに入れないようにする必要がある．これには，マイグレーション対象となるプロセス群の関係を利用する．プロセスマイグレーションは，dummy_init というプロセスを root とするツリー全体のプロセスを対象とする．図 4 の黄色で囲まれた PID 名前空間内のプロセスツリーの構造は，(a) マイグレーション前の ActiveOS と (b) マイグレーション後の BackupOS で同

じであるため，ActiveOS での dummy_init とマイグレーション対象プロセスの PID の差と，BackupOS にマイグレーションされた後の dummy_init とマイグレーションされたプロセスの PID の差は同じである．この PID の差を記憶することで，同じ名前のプロセスが複数あった場合でも，適切なプロセスをプロセスグループに入れることができる．各グループのリソース制限内容は，cgroup マイグレーションファイルを元に，各設定ファイルに対してパラメータを記述することで再現する．

5.1.2 Namespace マイグレーション

Namespace のマイグレーションは，ActiveOS 用のメモリに残っている情報を元に BackupOS で同じ名前空間を再現することで実現できる．しかし，BackupOS は ActiveOS 用のメモリ領域のメモリマップを保持しないため，マイグレーションに必要な情報を ActiveOS から直接取得することはできない．そのため，ActiveOS はマイグレーションに必要な情報が書き込まれている物理アドレスを事前に Orthros の共有メモリに書き込んでおくことで，マイグレーション時に BackupOS が必要な情報を取得できるようにする．また，Namespace は種類によって名前空間の管理構造が異なるため，マイグレーション処理を個別に行う．

UTS 名前空間のマイグレーションは，まず ActiveOS 上で動作していたプロセスの task_struct 構造体に関連付いている uts_namespace 構造体から UTS 名前空間の情報を取得する．その後，マイグレーションされたプロセスの task_struct 構造体に関連付いている uts_namespace 構造体に，取得した情報を書き込むことで UTS 名前空間のマイグレーションが完了する．PID 名前空間のマイグレーションは，最初は UTS 名前空間と同様，ActiveOS 上で動作していたプロセスの pid_namespace 構造体から PID 名前空間の情報を取得し，マイグレーションされたプロセスの pid_namespace 構造体に取得した情報を書き込む．PID 名前空間のマイグレーション前後の各 OS での PID 名前空間の状態を図 4 で示す．ただし，PID 名前空間は親の名前空間を保持する必要があるため，マイグレーションは再帰法を用いて親名前空間を先に処理するようにし，PID 名前空間全体の階層構造を保ったまま再現する．PID 名前空間作成時の最初のプロセスはその名前空間における init プロセスとなり，init プロセスが終了した時にはその名前空間以下のすべてのプロセスが終了する．そのため，マイグレーション対象のプロセスがすべて終了と同時に終了するプロセスを用意し，そのプロセスを作成する PID 名前空間の init プロセスとする．この動作を行うプロセスを dummy_init と呼ぶ．

その他，IPC，MNT，NET，User の名前空間にマイグレーションに関しては未実装であるが，task_struct 構造体から各名前空間を管理する構造体を参照することができるため，BackupOS が共有メモリを利用して ActiveOS から情報を取得し，各名前空間の構造体の再現を行うことでマイグレーションができると考える．

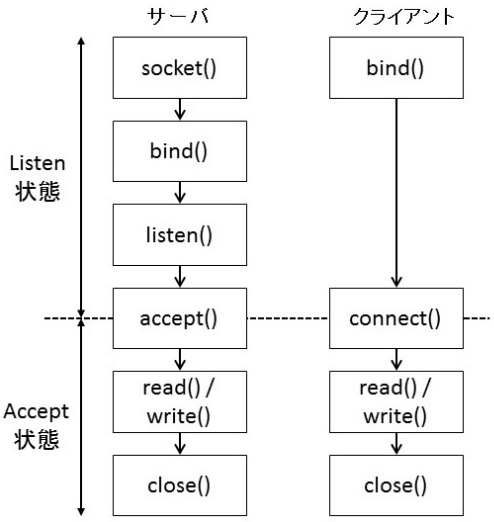

図 5 UNIX ドメインソケットの状態

5.2 UNIX ドメインソケットマイグレーション

UNIX ドメインソケットのマイグレーションは，Orthros の共有メモリを利用して BackupOS が ActiveOS のメモリを参照し，ActiveOS で動作していたソケットの情報を取得し，それを元に BackupOS でソケットの再現を行うことで実現できる．

UNIX ドメインソケットは，ソケットが生成されてから破棄されるまでのサイクルの中で，大きく 2 つの状態に分類される．1 つは Listen 状態で，ソケットが生成されてから通信相手ソケットとのコネクションが確立するまでの状態である．もう 1 つは Accept 状態で，コネクションが確立してからソケットが破棄されるまでの状態である．ソケットに関するプログラム処理の流れとソケットの状態の関係を図 5 に示す．

最初に，マイグレーションする UNIX ドメインソケットの状態を確認し，状態によって処理を変更する．Listen 状態のソケットは，ソケットを管理する socket 構造体に格納されているプロトコルファミリーや通信方式などの情報を ActiveOS からコピーし，BackupOS で新たに生成したソケットに設定する．その後，ファイルシステム上のソケットファイルのパスを取得し，socket 構造体に結びつけることで再現する．Accept 状態のソケットは通信相手の情報を相互に持つ必要があり，通信相手の pid 構造体と cred 構造体へのポインタ変数を持っている．そのため，マイグレーション処理ではまず ActiveOS で動作していた際のソケットのペアを記憶しておく．そして Listen 状態のソケットの再現と同じ処理をすべてのソケットに対して行った後，ソケットの通信相手の pid 構造体と cred 構造体へのポインタ変数を設定することで Accept 状態のソケットのマイグ

表 3 評価結果

マイグレーション対象	時間 (ミリ秒)
cgroup	287.90
Namespace	0.02
UNIX ドメインソケット	0.97
合計	288.89
フェイルオーバー全体	783.05

レーションが完了し，BackupOS でもプロセス間通信を継続することができる．

6. 評価

サーバ保護のためのコンテナマイグレーションの評価として，コンテナを作成し，その中で動作するプロセスのマイグレーションを行った．本章では，コンテナマイグレーションについて評価方法と結果について述べる．

6.1 評価方法

ActiveOS で 2 つのプロセスを動作させる．これら 2 プロセスそれぞれで 500 個の Accept 状態の UNIX ドメインソケットを作成し，プロセス間通信を行う．cgroup によって，プロセスが使用できる CPU コアと CPU 使用時間を制限する．Namespace によって，PID 名前空間と UTS 名前空間を他のプロセスと分離する．その他の名前空間に関しては分離しない．評価用コンテナ内で動作するプロセスをマイグレーションし，ActiveOS での cgroup や Namespace の設定が BackupOS で再現できているか確認する．また，コンテナマイグレーションに要する時間を計測すると同時に，フェイルオーバー処理全体に要する時間も測定し，比較を行う．

6.2 評価結果

プロセスマイグレーション後，BackupOS で cgroup の設定が再現されており，名前空間の分離がされていることを確認した．また，UNIX ドメインソケットによる通信が継続していることを確認した．

コンテナマイグレーションにかかった時間を表 3 に示す．フェイルオーバー処理全体に要した時間に対するコンテナマイグレーションに要した時間は約 37 %であった．コンテナマイグレーションのうち，cgroup マイグレーションに要する時間が大半を占めていることがわかる．これは cgroup マイグレーションの処理をユーザ空間からのコマンド実行によって行っていることが理由だと考えられる．cgroup マイグレーションの処理をカーネル空間で行うことができれば，大幅なフェイルオーバー処理の時間短縮の期待ができる．

7. 関連研究

本章では，仮想環境を保護する手法と，Orthros が対象とする OS の障害に対してプロセスの耐障害性を向上させる手法について述べる．

7.1 ライブマイグレーション

ライブマイグレーション [8] は，ある仮想環境で動作しているサービスを停止することなく別の物理マシン上へ移動する手法である．Orthros のコンテナマイグレーションは，仮想環境でのサービスを停止させることなく移動するという点でライブマイグレーションと同じ動作をする．しかし，ライブマイグレーションはマシンの管理者が能動的に行う手法であり，突然の障害時にサービスを別マシンに自動で移動させることはできない．Orthros は障害発生時に自動でマイグレーションを行うため，仮想環境のサービスを保護することができる．

7.2 Otherworld

Otherworld[9] は，OS をマイクロリブートすることによってプロセスの実行状態を保護する手法である．OS に障害が発生した際に，計算機を停止せずに新たな OS をウォームブートすることで停止した OS のメモリイメージを残存させる．新しい OS はそのメモリイメージを参照することによって，プロセスに関するデータを読み出して復元することで実行を再開する．Orthros のプロセスマイグレーションは，停止した OS のメモリを読み取りプロセスを保護するという点で Otherworld と同質の機能である．しかし，Otherworld はファイルキャッシュの保護を行わないため，それに依存するプロセスはフェイルオーバー後に正しい動作を行うことができない．また，Orthros は OS の初期化時間を必要としないため Otherworld よりも高速にフェイルオーバーを行うことができる．

7.3 Shimos2

Shimos2[10] は，OS の冗長化による高速なフェイルオーバーとチェックポイントリスタート (以下，C/R) によるプロセス保護を行う手法である．Shimos2 では CPU コアごとに OS が動作するため，1 台の OS に障害が発生しても他の OS が動作を継続でき，高速なフェイルオーバーが可能になる．しかし，プロセスの実行状態保護に C/R を用いており，実行時オーバヘッドが大きいという問題がある．また，最後のチェックポイントから障害までの最新の実行状態は保護不可能である．Orthros では Shimos2 と同じく OS の同時実行によるフェイルオーバーを行うが，実行状態保護を C/R ではなく障害が発生した OS のメモリを読み取ることによって実現する．これによって実行時オーバヘッドを発生させずにファイルキャッシュ及びプロセスの最新の実行状態を保護可能である．

8. まとめ

本稿では，仮想化技術の中でも利用が拡大しているコンテナ型仮想化を対象とし，コンテナマイグレーションによって OS 障害に対する耐障害性を向上する手法を提案した．コンテナ型仮想化はゲスト OS を使用せずにプロセスを分離された環境で動作させる技術であり，コンテナ管理ソフトウェアによって容易に利用できるようになる．提案手法のために，既存研究であるプロセス耐障害性向上システム Orthros においてコンテナマイグレーション機構を実装した．また，コンテナ管理ソフトウェアでは UNIX ドメインソケットによる通信を行っており，コンテナの環境と同時にコンテナ管理ソフトウェアも保護できるよう，UNIX ドメインソケットマイグレーションの機構も実装した．コンテナは Linux の標準機能である cgroup と Namespace を用いて仮想環境を作っているため，これらの機能をマイグレーションに対応させた．評価では評価用コンテナを作成し，コンテナ内で UNIX ドメインソケットを用いるプロセスを動作させ，プロセスマイグレーションを行った．ActiveOS で行った cgroup 及び Namespace の設定が BackupOS でも再現されていることを確認し，UNIX ドメインソケットによる通信も継続動作することを確認した．

参考文献

[1] Mircea Bardac, Razvan Deaconescu, Adina Magda Florea: Scaling Peer-to-Peer Testing using Linux Containers, *Roedunet International Conference (RoEduNet), 2010 9th*, pp. 287–292 (2010).
[2] Miguel G. Xavier, Marcelo V. Neves, Fabio D. Rossi, Tiago C. Ferreto, Timoteo Lange, Cesar A.F. De Rose: Performance Evaluation od Container-based Virtualization for High Performance Computing Environments, *2013 21th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pp. 233–240 (2013).
[3] Paul Menage: Linux kernel documents about Cgroups, https://www.kernel.org/doc/Documentation/cgroups.
[4] Rajdeep Dua, A Reddy Raja, Dharmesh Kakadia: Virtualization vs Containerization to support PaaS, *2014 IEEE International Conference on Cloud Engineering*, pp. 610–614 (2014).
[5] Docker, Inc.: Docker - Build, Ship, and Run Any App, Anywhere, https://www.docker.com.
[6] Alex Polvi: CoreOS is building a container runtime, Rocket, https://coreos.com/blog/rocket/.
[7] 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).
[8] Clark, Christopher., Fraser, Keir., Hand, Steven., Hansen, Jacob Gorm., Jul, Elic., Limpach, Christian., Pratt, Ian. and Warfield, Andrew.: Live migration of virtual machines, *Proc. 2nd conference on Symposium on Networked Systems Design and Implementation(NSDI 2005)*, pp. 273–286 (2005).
[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*, pp. 181–194 (2010).
[10] Liao, J., Shimosawa, T. and Ishikawa, Y: Configurable Reliability in Multicore Operating Systems, *Proceedings of the 2011 14th IEEE International Conference on Computational Science and Engineering*, pp. 256–262 (2011).