MilvusVisor
今年の進捗

森　真誠

2022/12/7　BitVisor Summit 11

背景

OSのRoot権限を渡せるような共用HPC環境を提供できるようにするプロジェクトがある

- Thin Hypervisorを用いて実現したい
- スーパーコンピュータ富岳相当の環境での実現が目標

Arm版BitVisorが欲しい！！

共用 HPC における管理者権限の利用を許す計算資源提供
https://kaken.nii.ac.jp/ja/grant/KAKENHI-PROJECT-21K17727/

MilvusVisor

RIKEN-RCCS / MilvusVisor - GitHub

Version 1.1.0 を公開中

- Intel i210のEEPROM保護機能
- Mellanox MT27800のFirmware Update阻止機能
- SMMUによるDMA Attack阻止機能
- 高速リストア機能
- PXE Boot機能

Armの概要とMilvusVisorの設計

Armの概要

- ARM Ltdによって設計されライセンスされるアーキテクチャ
- RISC系のアーキテクチャ
- アーキテクチャはARM v1からARM v9まで
- ARM v7までは32bit動作モード(AArch 32)のみ、ARM v8-Aから
  64bit動作モード(AArch64)が存在
- AArch32での命令長は32bitまたは16bit(Thumb)、AArch64では32bitのみ

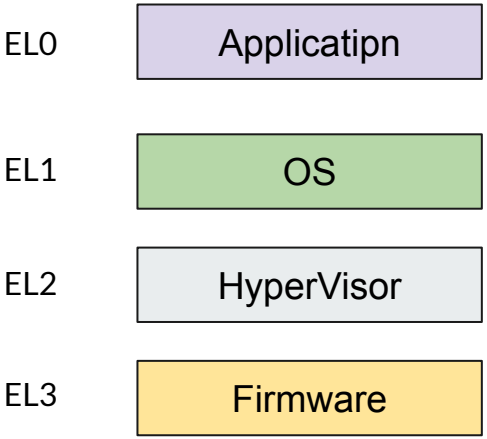
今回はArm v8.2-AのCPUであるA64FXのAArch64動作モードを対象に

AArch64の権限レベル

AArch64での権限レベルは権限が高い順に
EL3, EL2, EL1, EL0の4種類ある(x86_64と値が逆)

EL3: セキュアモニタ(ファームウェアなど)
EL2: ハイパーバイザー
EL1: スーパーバイザー(OS)
EL0: アプリケーション

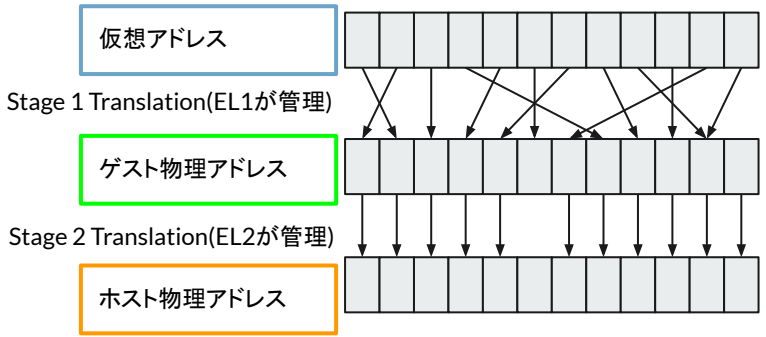
(Secure-WorldやTrusted-OSなどあるがここでは割愛)



Stage 2 Translation

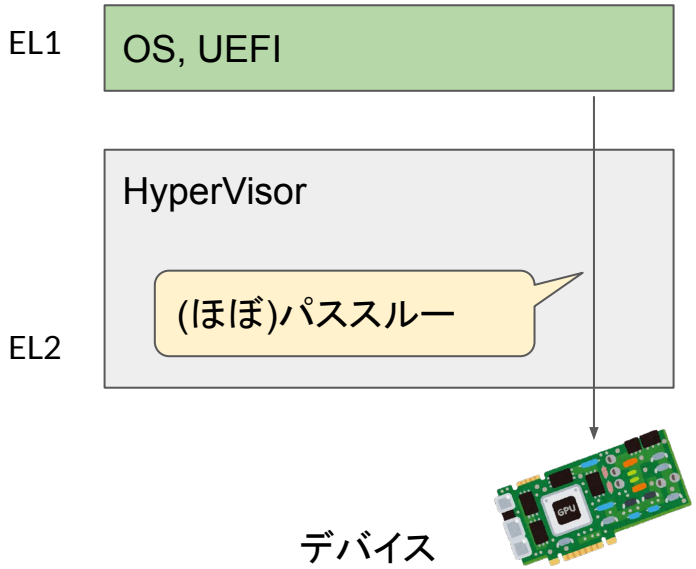
Armv8-A以降の仮想化支援機能にゲスト物理アドレスからホスト物理アドレスに変換するStage 2 Translation が存在

- Stage 1 TranslationはOSが管理
- Stage 2 Translationはハイパバイザが管理



ハイパーバイザーの設計

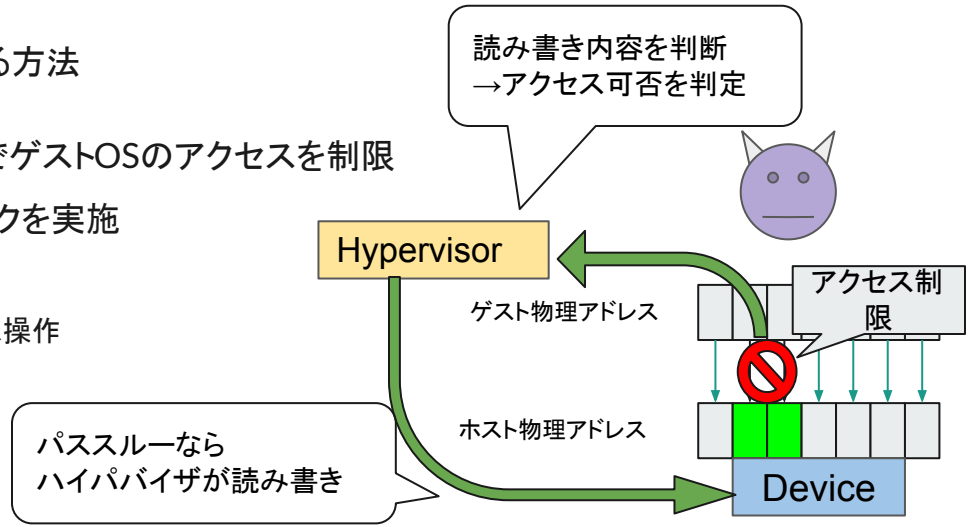
- EL2に存在し、トラップ時のみ動作
- EL0&1からのハードウェアクセスはほぼパススルー
- デバイスのアクセスを一部トラップ
    - メモリ管理機構を用いて MMIOを使ってトラップ



MMIOのフック

デバイスの保護などでMMIOのアクセスをフックする方法

- フックしたい領域を予めStage 2 TranslationでゲストOSのアクセスを制限
- フックしたアクセスごとにパススルーかブロックを実施
  - アドレスと読み書きの種類を元に判断
  - パススルーの場合、ハイパバイザがデバイス操作
  - ブロックの場合、実際にはアクセスせず、
    アクセス結果をエミュレート



MilvusVisorの
各機能の実装について

各機能の説明

以下の機能について説明

- Intel i210のEEPROM保護機能
- Mellanox MT27800のFirmware Update阻止機能
- SMMUによるDMA Attack阻止機能
  - ゲストOSとのSMMU共有機能
- 高速リストア機能
- PXE Boot機能

Intel I210の保護

I210にはEEPROMという不揮発性メモリが存在
MACアドレスやファームウェアを保持

EEPROMへのアクセスは以下のレジスタを使用して行う

- EEWR
- FLSWDATA
- iNVM

今回は上記レジスタへの書き込みアクセスをすべて遮断

Mellanox MT27800の保護

Fujitsuの公式ページでは“Flint”というツールが利用したファームウェアアップデート方法が掲載

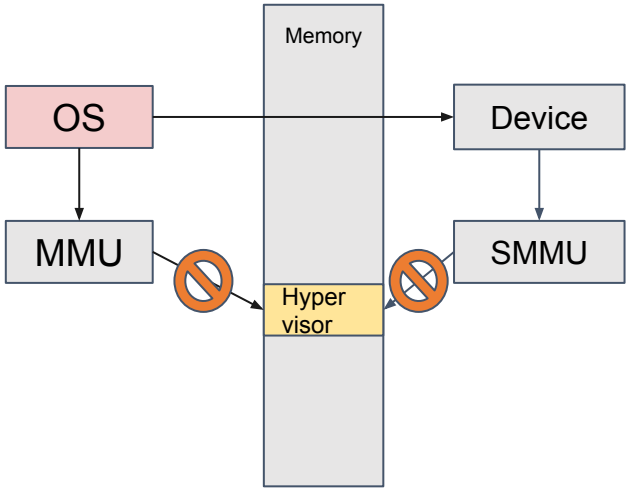
Flintはファームウェアにアクセスする際にPCI Configuration Space内に存在するセマフォを取得するために該当箇所を読み書きする

今回はセマフォへの書き込みアクセスを遮断

DMA Attackからの保護

デバイスからのメモリアクセスに対してStage 2 Translationなどを適用できるSystem Memory Management Unit(SMMU)を使用し、CPUと同じStage 2 Translation を適用

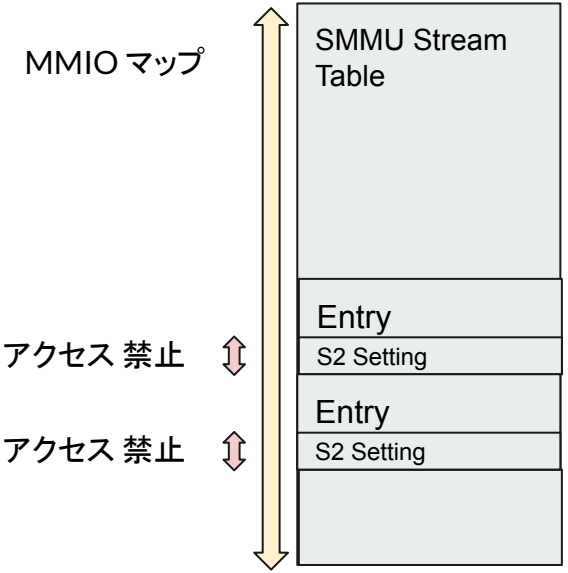
デバイスからのハイパバイザへのアクセスは不能に



SMMUの共有機能

- SMMUはStage 1/2 Translationが適用可能
- LinuxがStage 1 Translationを使用することがある

SMMUの設定をゲストOSにも開放し、Stream Tableという
各デバイス用の設定のStage 2 Translationに関する
設定のみMMIO アクセストラップで
アクセス(変更)禁止にしてそれ以外は開放し共有



高速リストア

MilvusVisorでは以下のように高速リストアを実装

復元ポイント(リストアポイント):

初回起動時のゲストOSがUEFIのExitBootServices関数を呼出し、戻ってきた時点

高速リストアを実行タイミング:

ゲストOSがシャットダウンまたはリブートを試みた時点

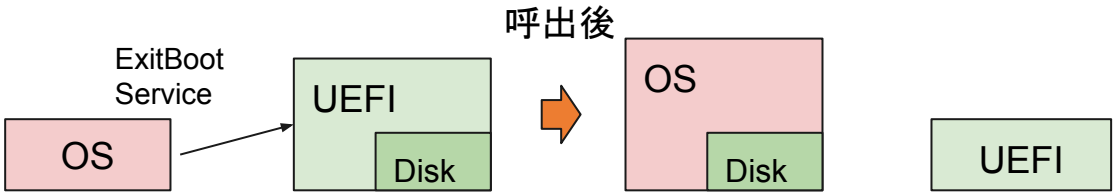
ExitBootServices関数とは

UEFIが起動時に提供しているBootServicesを終了させる関数

BootServicesでは「メモリの確保」「ファイルの読み込み」や「画面への文字列出力」を提供

そのため、ディスクやメモリ、ディスプレイはUEFIが管理

ExitBootServices関数はBootServicesを使えなくし、UEFIが管理しているデバイスを開放する



復元ポイントに ExitBootServices関数復帰時点選んだ理由

- OSが起動時に(通常)一回のみ必ず呼び出す
- 呼び出した後OSがデバイスの初期化を行う
  - MilvusVisorでデバイスの初期化をやらずに済む
- 本格的な起動処理を行う前に呼ばれる
  - メモリ使用量が少ないためスナップショット作成時に必要なメモリ量が少ない

高速リストアの実装概要

スナップショットの作成方法

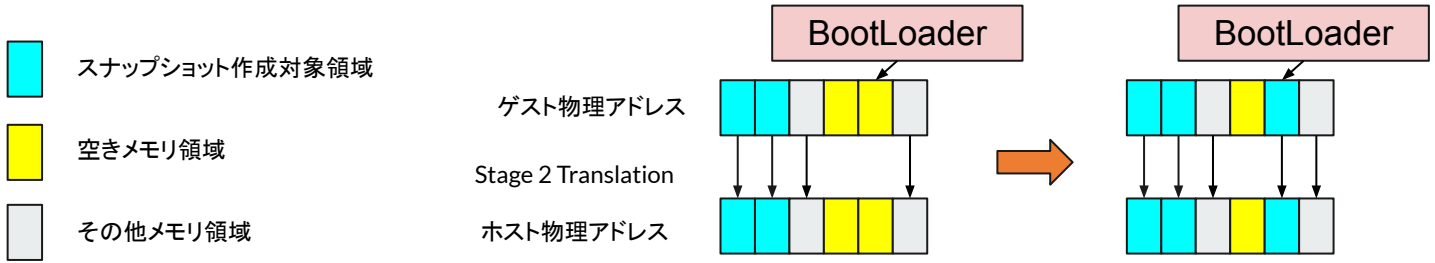
- メモリスナップショット対象領域の特定
- リストアポイントの取得
- スナップショットの作成

高速リストアの実行方法

- 再起動要求の遮断、スナップショットの復元

メモリスナップショット対象領域の特定

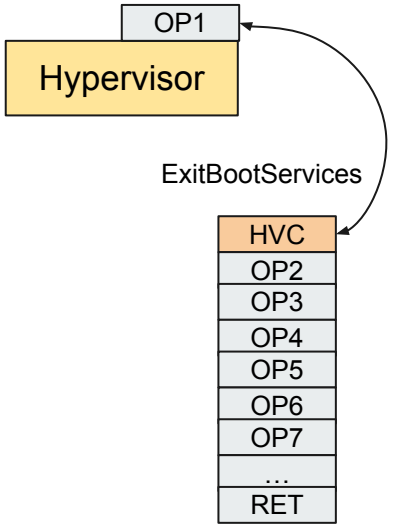
- ハイパバイザ起動時にUEFIのメモリマップを基に以下の領域を保存対象アドレスリストに追加
  - ハイパバイザが起動する前に書き込まれ、OSが起動後に利用され得る領域
- 空きメモリのうちOSのブートローダがアクセスしたメモリアドレスを記録
  - Stage 2 TranslationでOSからの空きメモリへのアクセスをトラップする用に設定
  - 記録後、該当領域 Stage 2 Translationのマップを作成し以後捕捉しないよう設定



リストアポイントの取得方法

MilvusVisor 起動時

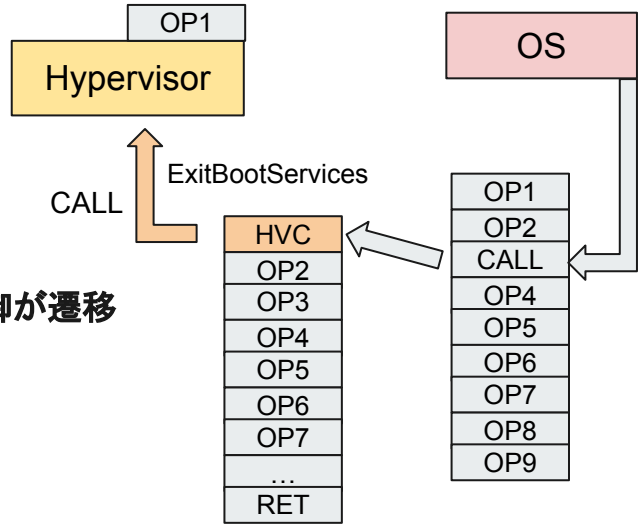
ExitBootServices関数の先頭1命令をHyperVisorCall(HVC)命令に置換しておく(元の命令は保存しておく)



リストアポイントの取得方法

ExitBootServices呼出時

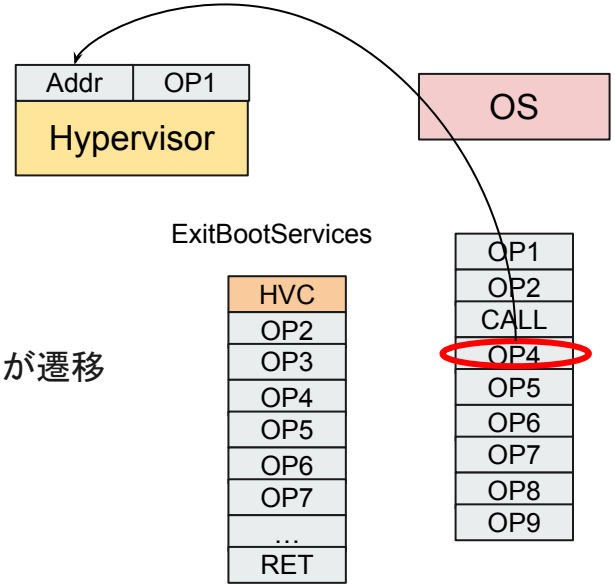
1. **ゲストがExitBootServicesを呼出・HVC命令でハイパバイザに制御が遷移**
2. 戻り先保存レジスタの値を取得し保存(リストアポイント)
3. ExitBootServices先頭1命令を復元
4. リストアポイントの命令を保存しHVC命令に置換
5. ExitBootServicesに戻る



リストアポイントの取得方法

ExitBootServices呼出時

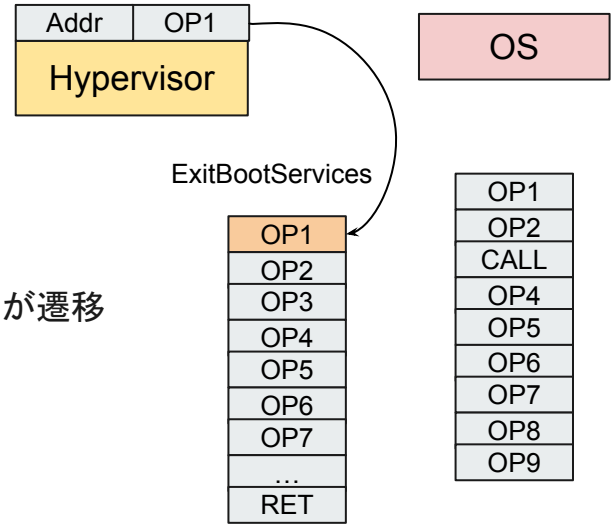
1. ゲストがExitBootServicesを呼出・HVC命令でハイパバイザに制御が遷移
2. **戻り先保存レジスタの値を取得し保存 (リストアポイント)**
3. ExitBootServices先頭1命令を復元
4. リストアポイントの命令を保存しHVC命令に置換
5. ExitBootServicesに戻る



リストアポイントの取得方法

ExitBootServices呼出時

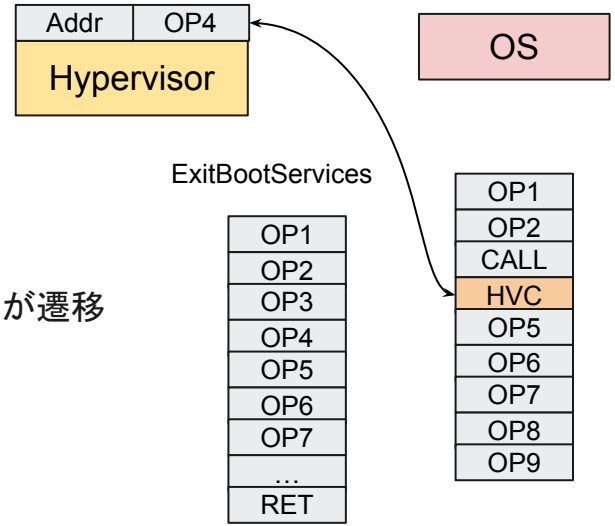
1. ゲストがExitBootServicesを呼出・HVC命令でハイパバイザに制御が遷移
2. 戻り先保存レジスタの値を取得し保存(リストアポイント)
3. **ExitBootServices先頭1命令を復元**
4. リストアポイントの命令を保存しHVC命令に置換
5. ExitBootServicesに戻る



リストアポイントの取得方法

ExitBootServices呼出時

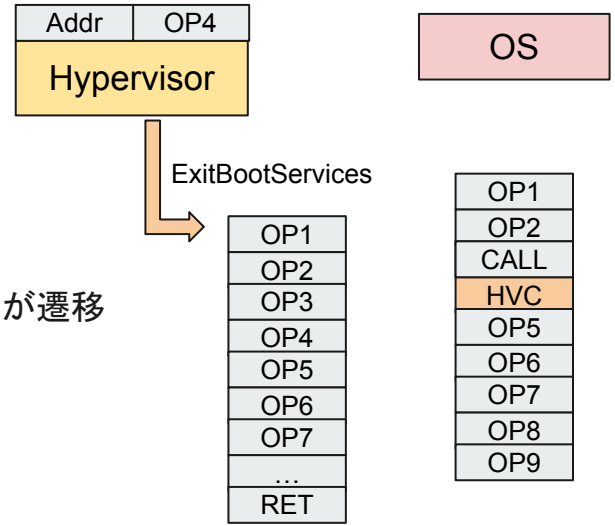
1. ゲストがExitBootServicesを呼出・HVC命令でハイパバイザに制御が遷移
2. 戻り先保存レジスタの値を取得し保存(リストアポイント)
3. ExitBootServices先頭1命令を復元
4. **リストアポイントの命令を保存しHVC命令に置換**
5. ExitBootServicesに戻る



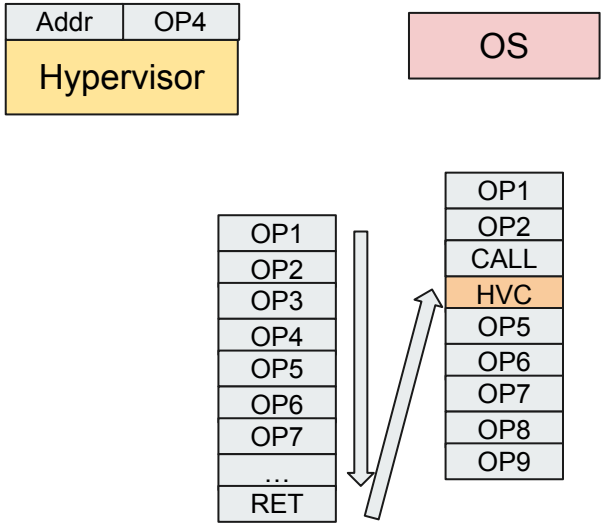
リストアポイントの取得方法

ExitBootServices呼出時

1. ゲストがExitBootServicesを呼出・HVC命令でハイパバイザに制御が遷移
2. 戻り先保存レジスタの値を取得し保存(リストアポイント)
3. ExitBootServices先頭1命令を復元
4. リストアポイントの命令を保存しHVC命令に置換
5. **ExitBootServicesに戻る**



リストアポイントの取得方法



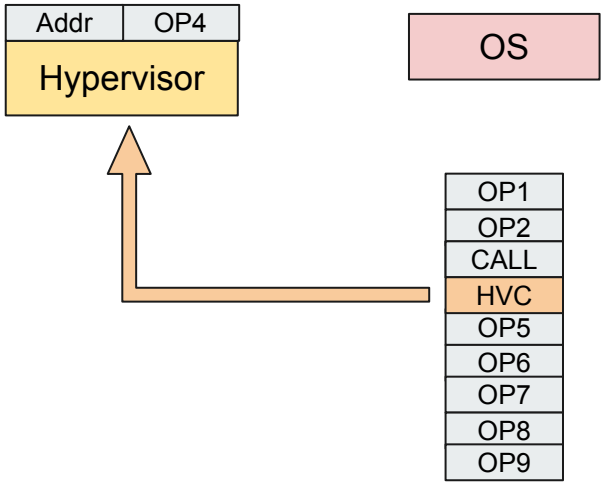
ExitBootServices終了時

1. **処理が終わりreturn命令が実行される**
2. 呼出命令の次の命令(HVC命令)によりハイパバイザに遷移
3. ExitBootServicesの結果を確認（成功していると仮定）
4. リストアポイントの命令を復元

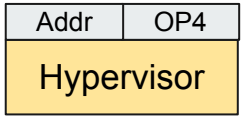
リストアポイントの取得方法

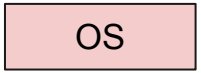
ExitBootServices終了時

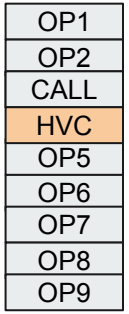
1. 処理が終わりreturn命令が実行される
2. **呼出命令の次の命令(HVC命令)によりハイパバイザに遷移**
3. ExitBootServicesの結果を確認（成功していると仮定）
4. リストアポイントの命令を復元



リストアポイントの取得方法







ExitBootServices終了時

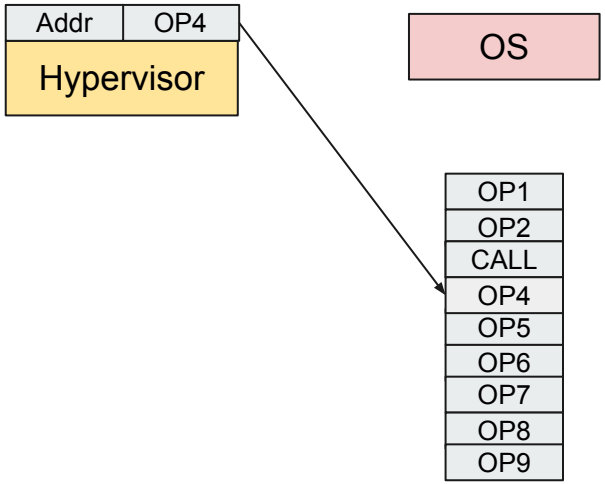
1. 処理が終わりreturn命令が実行される
2. 呼出命令の次の命令(HVC命令)によりハイパバイザに遷移
3. **ExitBootServicesの結果を確認（成功していると仮定）**
4. リストアポイントの命令を復元

リストアポイントの取得方法

ExitBootServices終了時

1. 処理が終わりreturn命令が実行される
2. 呼出命令の次の命令(HVC命令)によりハイパバイザに遷移
3. ExitBootServicesの結果を確認（成功していると仮定）
4. **リストアポイントの命令を復元**

この後、リストアポイントの状態のスナップショットを作成



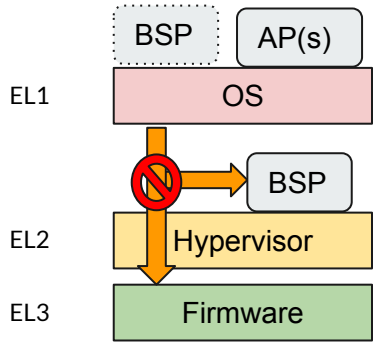
リストアの実行方法

1. **OSがファームウェアにシャットダウンを要求　→ハイパーバイザが捕捉し遮断**
2. すべてのコアの制御をハイパバイザに移動

   [コアの制御ハイパバイザに移動する方法]

   a. Stage 2 Translationのすべてのエントリを無効化

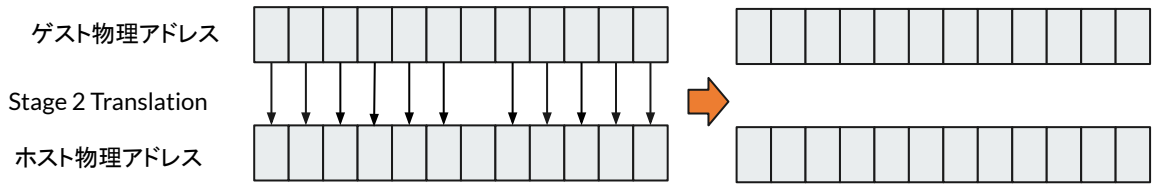
   b. キャッシュを破棄し次の命令を実行するように指示　→ページフォルトでハイパバイザに
3. スナップショットよりメモリ・レジスタの値を復元
4. EL1にてリストアポイントから実行を開始

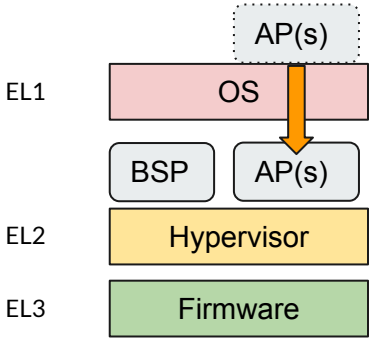


リストアの実行方法

1. OSがファームウェアにシャットダウンを要求→ハイパーバイザが捕捉し遮断
2. **すべてのコアの制御をハイパバイザに移動し起動時コア以外を停止**

   [コアの制御ハイパバイザに移動する方法
   a. Stage 2 Translationのすべてのエントリを無効化
   b. キャッシュを破棄し次の命令を実行するように指示　→ページフォルトでハイパバイザに
3. スナップショットよりメモリ・レジスタの値を復元
4. EL1にてリストアポイントから実行を開始



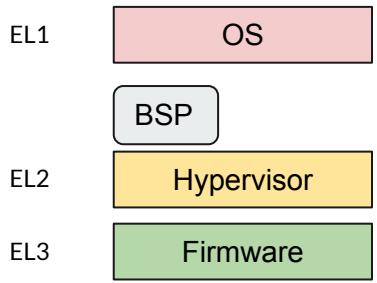


リストアの実行方法

1. OSがファームウェアにシャットダウンを要求→ハイパバイザが捕捉し遮断
2. すべてのコアの制御をハイパバイザに移動し起動時コア以外を停止

   [コアの制御ハイパバイザに移動する方法]

   a. Stage 2 Translationのすべてのエントリを無効化
   b. キャッシュを破棄し次の命令を実行するように指示　→ページフォルトでハイパバイザに
3. **スナップショットよりメモリ・レジスタの値を復元**
4. EL1にてリストアポイントから実行を開始

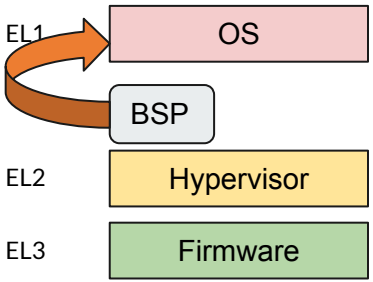


リストアの実行方法

1. OSがファームウェアにシャットダウンを要求→ハイパーバイザが捕捉し遮断
2. すべてのコアの制御をハイパバイザに移動し起動時コア以外を停止

   [コアの制御ハイパバイザに移動する方法]

   a. Stage 2 Translationのすべてのエントリを無効化
   b. キャッシュを破棄し次の命令を実行するように指示　→ページフォルトでハイパバイザに
3. スナップショットよりメモリ・レジスタの値を復元
4. **EL1にてリストアポイントから実行を開始**



PXE Boot

- MilvusVisorのカーネルをディスクからではなくTFTPを通じて取得
- サーバはDHCP Offerより自動検出
- OSのブートローダをTFTPでダウンロードしてきてUEFIのLoadImage関数に渡して実行

以上の機能を実装

FX1000の計算ノードなどDiskless環境で起動するための機能

各機能の目的や性能評価について

SIGOS 9月研究会の研究報告書を参照ください

Root権限使用可能なArmスパコン実現に向けた軽量ハイパバイザの設計と実装

http://id.nii.ac.jp/1001/00220014/

まとめ

- 産業技術総合研究所でAArch64向けのThin Hypervisor: MilvusVisorを開発
  - Arm v8.2-A CPU向けに開発
- MMIOのフックを実装
- Intel I210のEEPROMの書き込みアクセスを遮断
- Mellanox MT2780のファームウェアアップデートを阻止
- ExitBootServicesを利用した高速リストアを開発
- PXE Bootでネットワークブートにも対応