

マルチコアプロセッサにおいて 複数のLinux カーネルを 走行させる方式の設計

千崎良太[†] 乃村能成[†] 谷口秀夫[†]

[†]岡山大学大学院自然科学研究科

目次

- (1) はじめに
- (2) 設計目標
 - (A) 目標
 - (B) 提案方式の構成
- (3) 課題と設計
 - (A) メモリの分割と占有
 - (B) 各OSの起動
 - (C) CPUコアの分割と占有
 - (D) 入出力機器の分割と占有
 - (E) 割り込みの制御
 - (F) 終了処理
 - (G) 再起動処理
- (4) おわりに

はじめに

1台の計算機上で複数のOSを走行させる研究が活発

<先行研究>

(1) SHIMOS[1]

(A) マルチコアプロセッサを使用

(B) 各OSは複数コアから1つ以上を占有して独立して動作

(2) TwinOS[2]

(A) 各OSは実計算機上で直接動作

(B) ハードウェアの分割方法

(i) メモリ: 空間分割

(ii) CPU: タイマ割り込みによる時分割

(シングルコアプロセッサ)

(iii) 入出力機器: OSごとに占有

[1] T. Shimosawa, H. Matsuba, Y. Ishikawa, "Logical Partitioning without Architectural Supports," Proc. of the 2008 Annual IEEE International Computer Software and Applications Conference, pp. 355-364, 2008.

[2] 田渕正樹, 伊藤健一, 乃村能成, 谷口秀夫, "二つのLinuxを共存走行させる機能の設計と評価," 電子情報通信学会論文誌, vol. J88-D-I, no. 2, pp. 251-262, 2005.

設計目標

1台の計算機上で複数のLinuxを独立に走行させる方式を設計

- (1) 1台の計算機上で2つ以上のLinuxを動作
- (2) マルチコアプロセッサを使用
- (3) 各OSは、複数あるコアのうち1つ以上のコアを占有
- (4) 入出力機器について
 - (A) デバイス単位で分割
 - (B) 各OSが仮想化によらず直接占有制御

<特徴>

- (1) 各OS間の影響が最小になるようにそれぞれのOSが独立
- (2) 各OSが単独で動作する場合に近い環境と性能で走行

 各OSは**独自に終了と再起動**を実行可能

提案方式の構成

(1) CPU

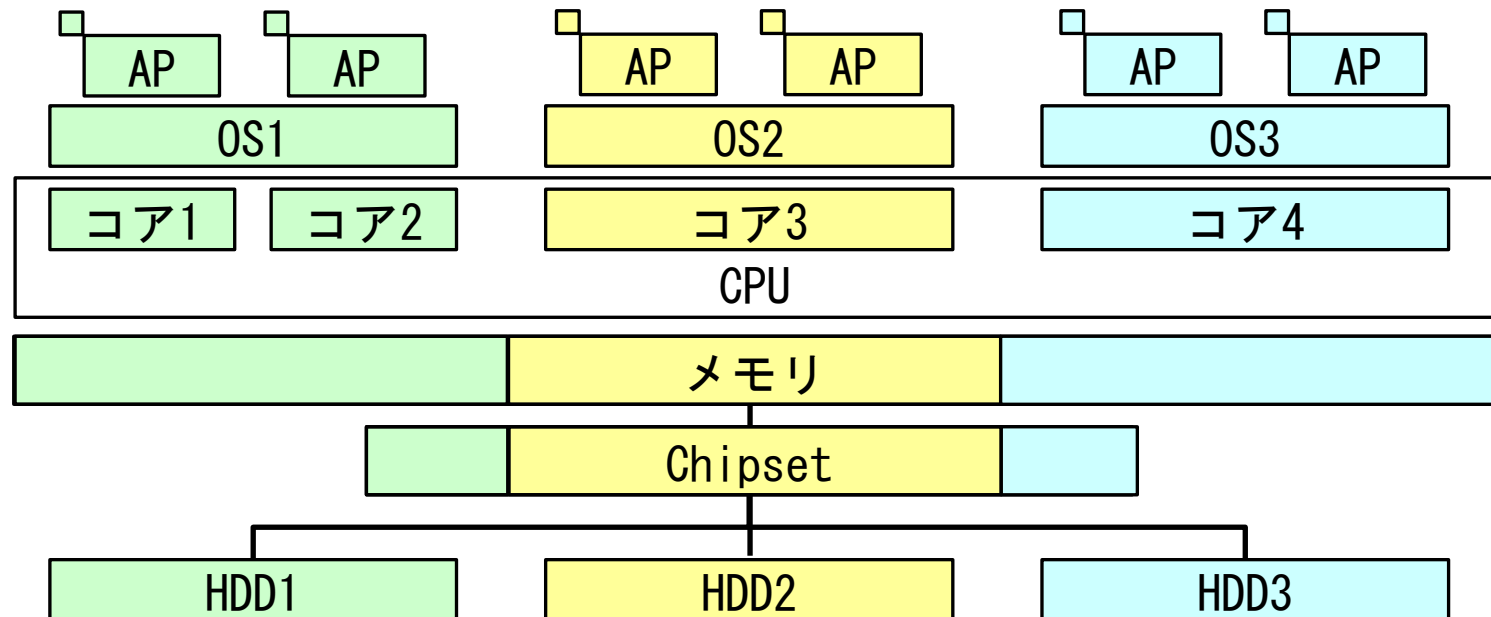
- (A) マルチコアプロセッサを使用
- (B) コアを分割し各OSに分配

(2) 実メモリ

- (A) OSの起動時に領域を分割占有

(3) 入出力機器

- (A) OS毎に指定された入出力機器のみを占有制御



提案方式の実現における課題

- (1) メモリの分割と占有
- (2) 各OSの起動
- (3) コアの分割と占有
- (4) 入出力機器の分割と占有
- (5) 割り込みの制御
- (6) 終了処理と再起動処理

以降では、課題の内容と設計について説明

メモリ分割と占有

<課題>

各カーネルが使用する物理メモリの範囲を制限

➡ 占有する領域の終端/先頭アドレスの指定が必要

(1) 終端アドレス

Linuxの既存機能によりブートパラメータで指定可能

(2) 先頭アドレス

Linuxの既存機能はないため対処が必要

<設計>

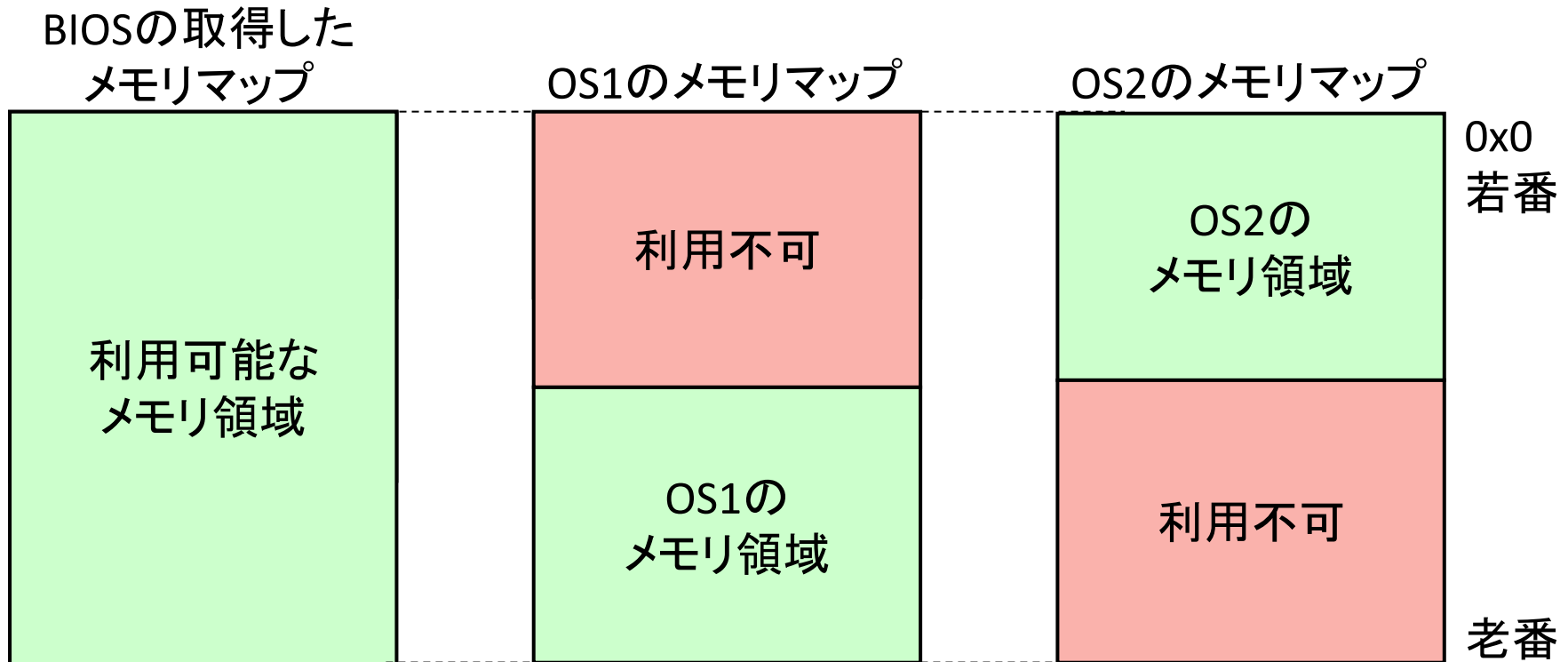
BIOSが取得したメモリマップをOSが占有するメモリ領域のみが
利用可能な領域であるよう書き換える

➡ 先頭アドレスの指定が可能

メモリアップの書き換え

<メモリ領域タイプ>

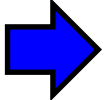
- (1) 通常のメモリとして利用可能
- (2) // 利用不可



各OSの起動

<課題>

- (1) 1番目のOSは、通常のLinuxに近い初期化処理で起動可能
- (2) 2番目以降のOSはハードウェア制御を制限
 - (A) ブートローダを介さない
 - (B) 占有するコアの初期化を行えない

 制限に合わせてカーネル起動処理を変更

<設計>

- (1) 1番目のOSから他のOSを順次起動
- (2) コアの初期化処理を追加

各OSの起動手順

(1) 1番目のOSは通常の起動プロセスでブートローダから起動

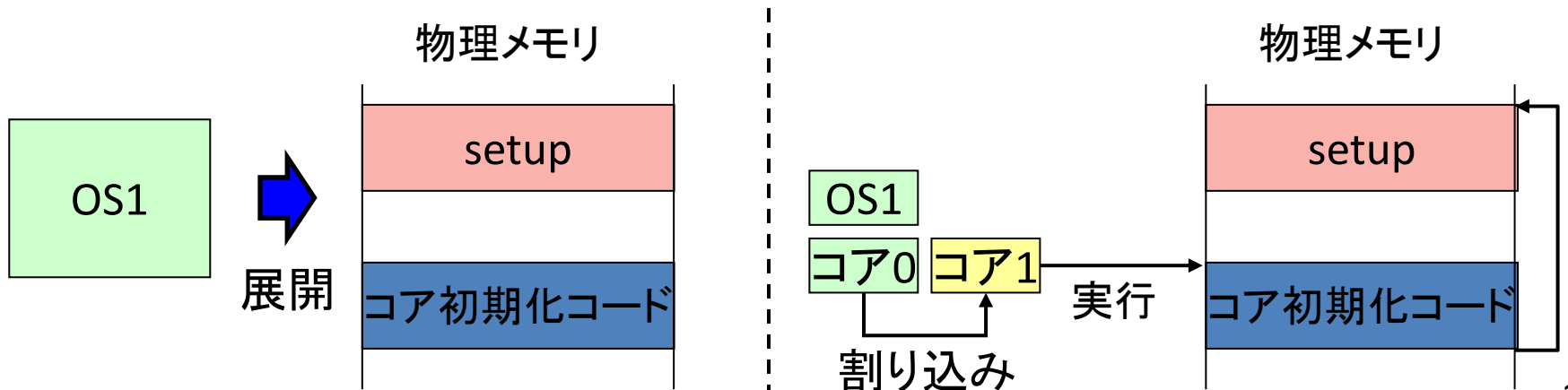
(2) 2番目以降のOSは, 1番目のOSから順次起動

(A) カーネルイメージをメモリ上に展開

(B) OSの実行

(i) コアの初期化処理

(ii) セットアップコードの実行



CPUコアの分割と占有

<課題>

Linuxの既存機能: 先頭コアからの利用コア数は指定可能

➡ 何番目のコアから何個占有させるかを指定できる必要

<設計>

(1) 占有ルールの設定

x86 SMPの各コアは, Local APICのID(LAPIC ID)により識別可能

(A) 未使用のコアの内, LAPIC IDが最小のものをBSPに指定

(B) LAPIC IDの小さいコアから順番に利用 (Boot Strap Processor)

(C) 自身のBSPよりも小さいLAPIC IDを持つコアは不使用

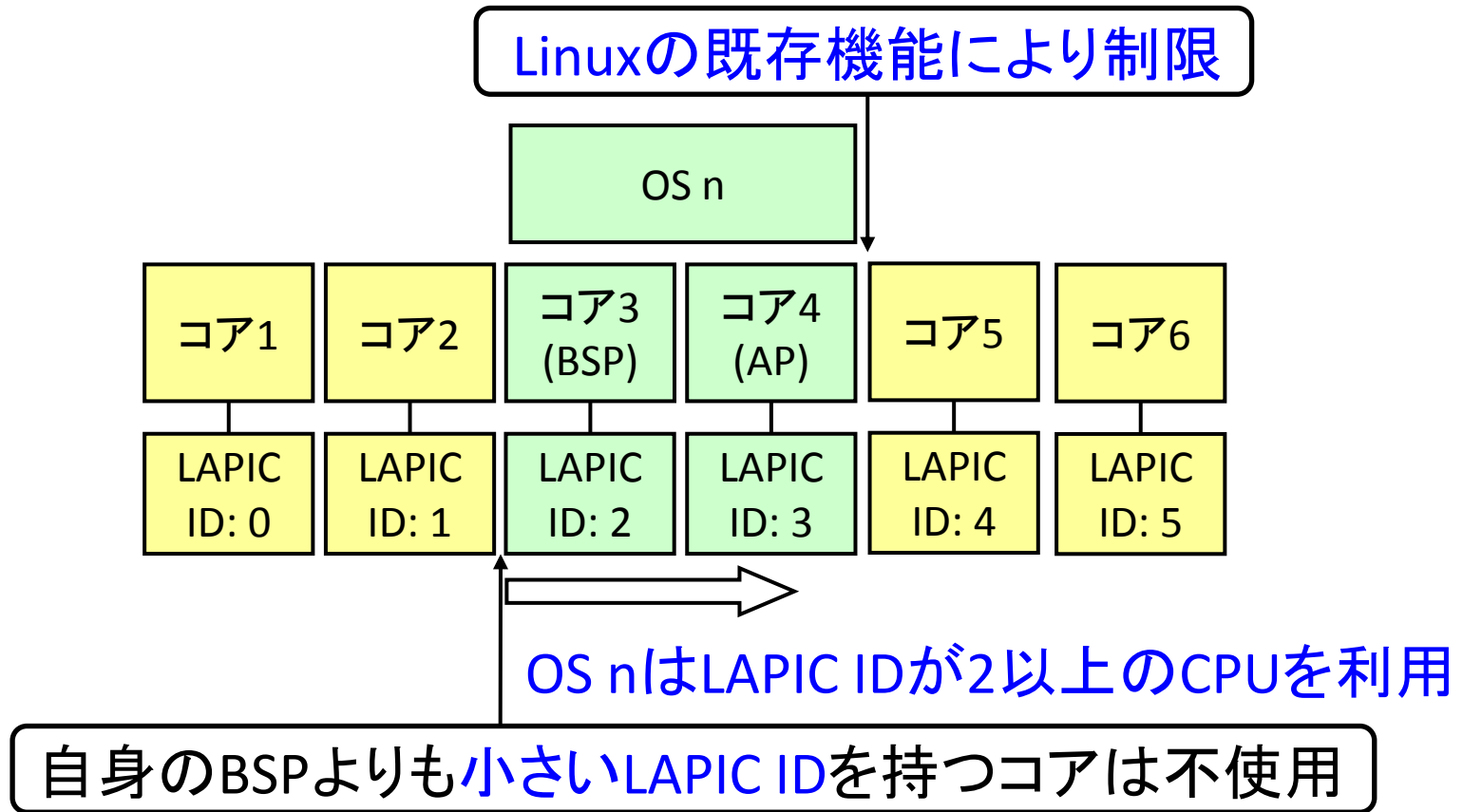
➡ 指定したコアから指定個数を占有可能

(2) 論理APIC IDの重複防止

各コアのLAPICは, 論理APIC IDにより割り込み先を決定

➡ 論理APIC IDの決定規則を操作し, 重複しないように変更

CPUコアの占有ルール



BSP: Boot Strap Processor

AP : Application Processor

入出力機器の分割と占有

<課題>

通常のLinuxは、利用可能なデバイスをすべて占有する

➡ 占有するデバイスと占有しないデバイスの仕分けが必要

<設計>

カーネルコードを変更することで、**デバイスタイプ別に分割と占有**

(1) PCI等のデバイス

デバイスドライバを使用

➡ **デバイスドライバの利用の可否により分割・占有**

(2) レガシーデバイス (PS/2キーボード, シリアルポート, VGA)

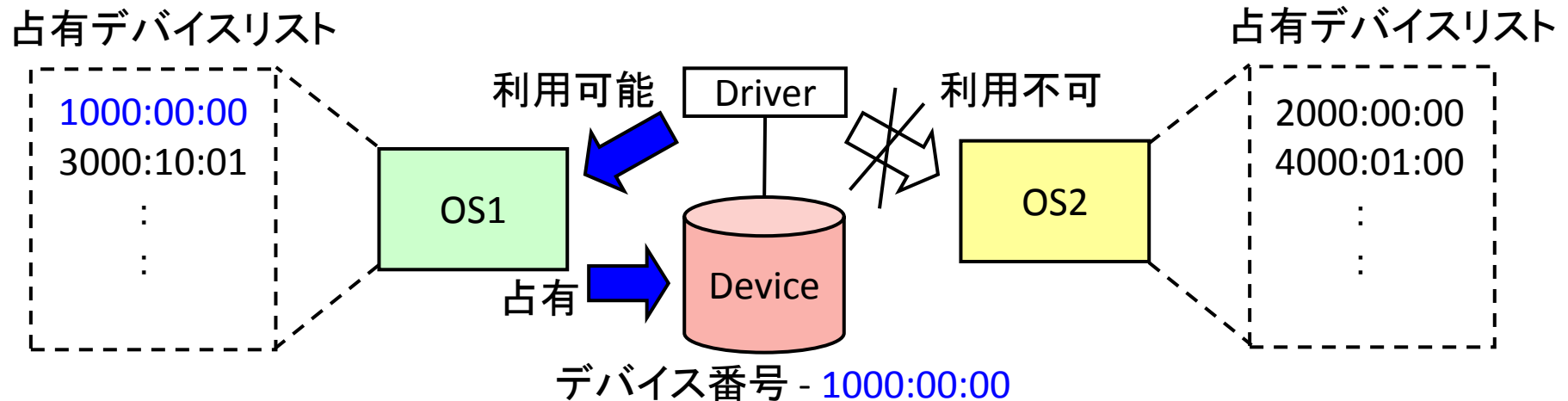
デバイスドライバを未使用

➡ **カーネルコードを直接変更することで分割・占有**

PCI等のデバイスの分割・占有方法

計算機に接続されているデバイスはデバイス番号によって識別可能

➡ デバイス番号によって占有するデバイスを識別可能



割り込みの制御

<課題>

通常のLinuxを複数走行させた場合の割り込みの設定

(A) 先に起動しているカーネルの設定を上書き

(B) 最後に起動したカーネルが割り込みを設定

 割り込みの設定の調整が各OS間で必要

<設計>

(1) I/O APICに制御される割り込み

他のカーネルのコアに関する設定内容进行操作せずに
割り込みを設定

(2) MSI(Message Signaled Interrupt)を用いる割り込み

デバイス単位で割り込みを設定可能

 特別な処理は不必要

I/O APICに制御される割り込み

割り込み別に以下のように制御

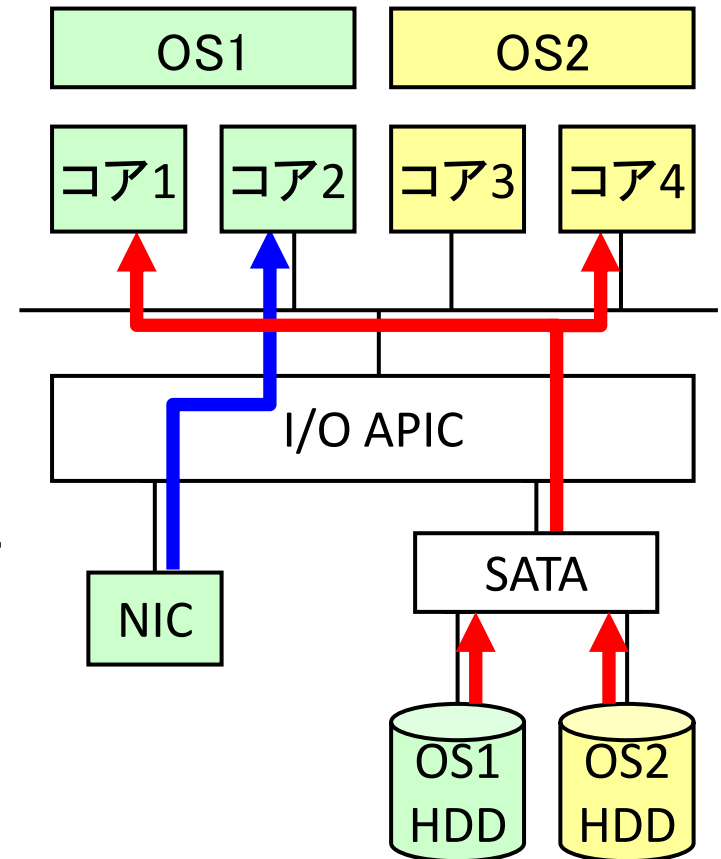
(1) 単一H/Wで占有される割り込み

(A) 対象のデバイスを占有するOSに対してのみ割り込み通知

(2) 複数H/Wで共有される割り込み

(A) 各デバイスを占有する全てのOSに割り込み通知

(B) 各OSは、占有していないデバイスからの割り込みは無視



終了処理

<課題>

他のOSに影響を与える処理には対処が必要

<設計>

(1) デバイス終了処理

(A) デバイスの終了処理を除外

(2) タイマの無効化

(A) タイマの無効化処理を除外

(3) I/O APICの無効化

(A) I/O APICの無効化処理を除外

(B) 終了するOSの割り込み設定をI/O APICから除外

(4) 計算機のシャットダウン

(A) 計算機をシャットダウンせず, コアをHALT状態に設定

再起動処理

<課題>

終了処理後, 独自にカーネルを展開し起動処理を行う必要

<設計>

独自にカーネルを展開し起動を行う処理を追加

(1) 終了処理

(2) initrdとカーネルイメージの配置

(A) initrdとカーネルイメージを指定のメモリ位置に展開する

(3) 圧縮カーネルの展開ルーチンにジャンプ

(A) BIOS, ブートローダ, セットアップルーチンの処理を行わない

(B) 圧縮カーネルの展開ルーチンから処理を開始する

おわりに

(1) まとめ

1台の計算機上で複数のLinuxを独立に走行させる方式の設計

<特徴>

- (1) 各OS間の影響が最小になるようにそれぞれのOSが独立
- (2) 各OSが単独で動作する場合に近い環境と性能で走行

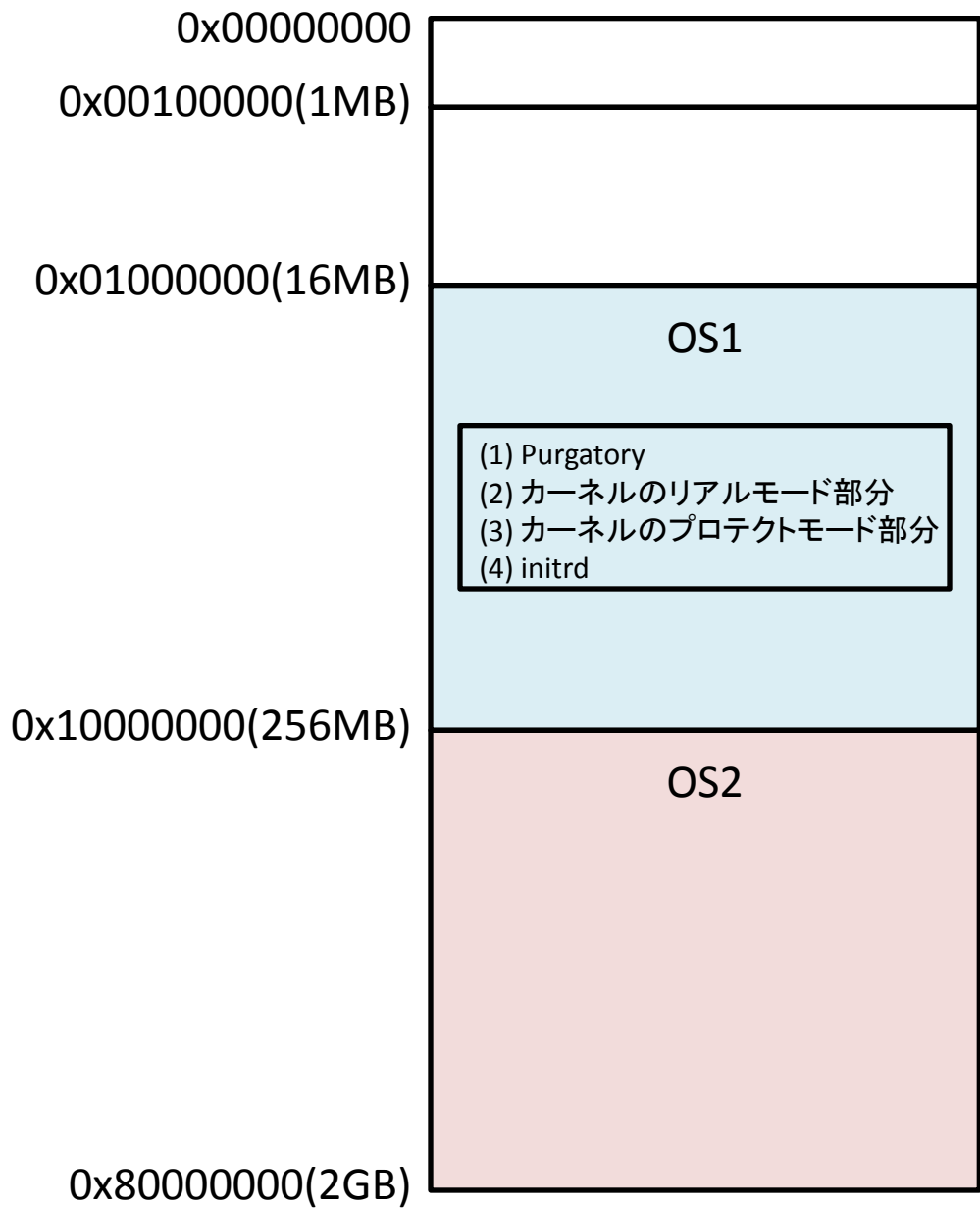
 各OSは**独自に終了と再起動**を実行可能

(2) 残された課題

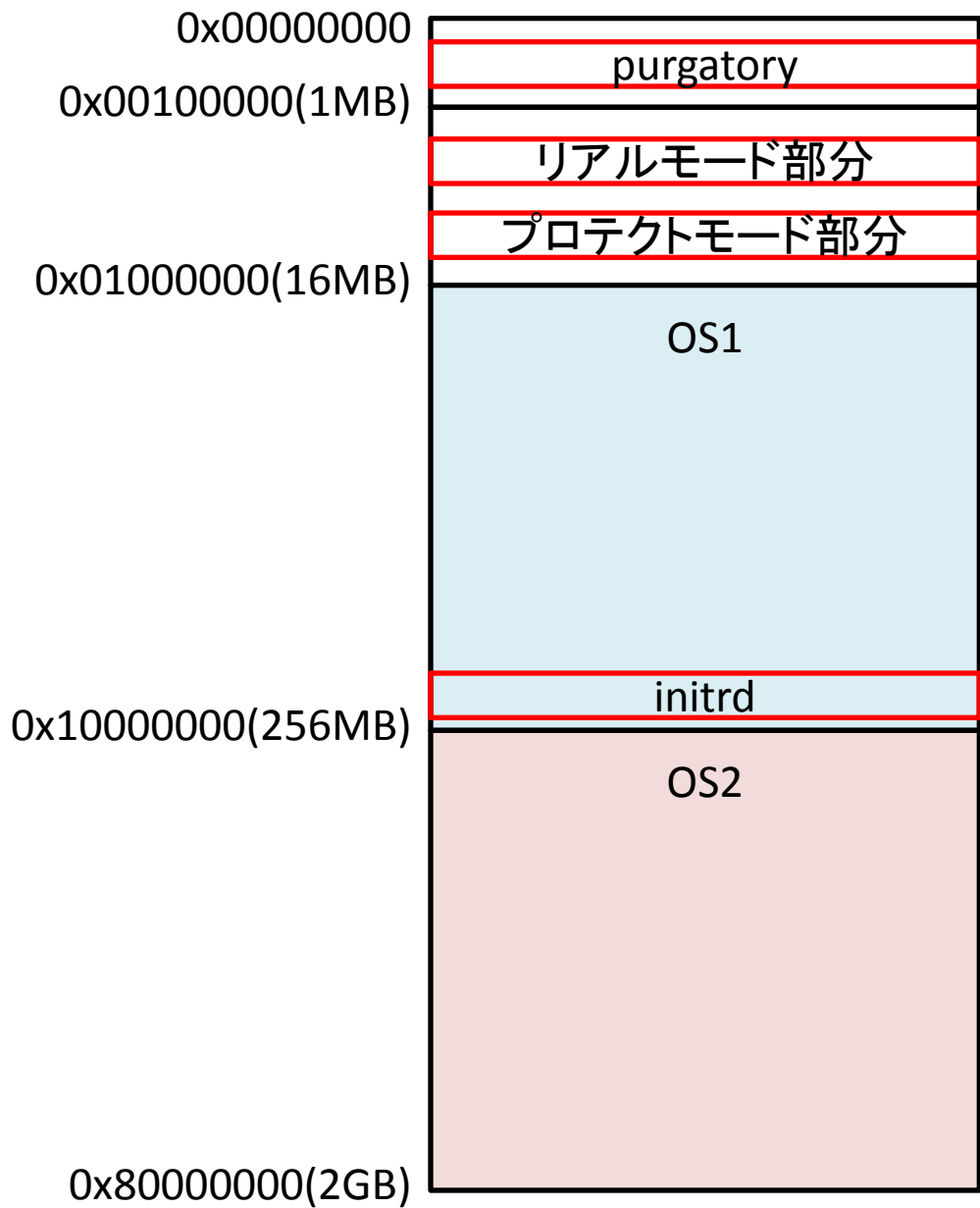
提案方式の実装と評価

Kexecの処理流れ

再起動に必要なファイルをメモリ上に確保



再起動に必要なファイルをメモリ上に展開



purgatoryにジャンプ

