修 士 論 文

題 目

# 複数OS走行環境におけるマルチコアCPUの計算資源分割制御方法に関する研究

指導教員

乃 村 能 成

報 告 者

牛尾 裕

岡山大学大学院自然科学研究科電子情報システム工学専攻

平成23年2月9日 提出

# 要約

計算機性能の向上に伴い，1 台の計算機上に複数の OS を走行させる研究が活発に行われている．これらの研究の 1 つとして，1 台の計算機上のシングルコア CPU で 2 つの Linux を独立に走行させる TwinOS が実現されている．TwinOS は，CPU，メモリ，および入出力機器といった各資源について効果的な共有と占有を行うことにより，1 台の計算機上で 2 つの Linux の走行を実現している．また，TwinOS をマルチコア CPU に対応させ，各コア上で OS を走行できる Mint オペレーティングシステムを研究開発している．しかし，現在の Mint オペレーティングシステムは，CPU のコアの分割粒度が粗いため，コア数より多くの OS を走行させることができない．

そこで，複数 OS 走行環境において，より細かい粒度でマルチコア CPU の計算資源を分割制御する方法について提案する．提案方法は，Mint オペレーティングシステムのマルチコア CPU の分割単位としてコア単位での分割制御に加え，時間単位での分割制御を導入し，コア単位と時間単位の分割制御の併用により，各 OS に CPU の計算資源を割り当てる．この制御方法により，柔軟な計算資源の割り当てを実現できる．例えば，1 つの OS に合計 1 コア半の計算資源を割り当てるといった，変則的なマルチコア CPU の分割制御も可能となる．

本稿では，マルチコア CPU の併用分割制御法の実現に向けた課題を示した．マルチコア CPU の併用分割制御法の実現には，シングルコア時分割制御の実装，シングルコア時分割制御のマルチコア CPU 対応，およびコア分割と時分割の併用利用対応の実現が必要である．これらの内，シングルコアの時分割制御の実装における課題と設計，および実装について述べた．シングルコアの時分割制御は，TwinOS で既に実現されているが，TwinOS の実装には 3 つ以上の OS を走行できない，マルチコア上で動作できないという 2 つの問題があり，そのまま流用できない．このため，TwinOS の実装方法を見直し，新たにマルチコアに対応を見越したシングルコア時分割制御の課題を示した．更に，この課題に対処した設計，および実装法を示した．また，マルチコア CPU の併用分割制御法の実現における残された課題を示した．

# 目次

# 図 目 次

# 表 目 次

# 第 1 章

# はじめに

計算機資源を効率的に利用するため，1 台の計算機上で複数の OS を走行させる技術の研究が活発に行われている．具体的な実現方式として，仮想計算機 (以降，VM とする) を利用する方式 (以降，VM 方式とする) がある．この方式によるシステムの構成例を 図 1.1 に示す．この方式は，仮想マシンモニタ (以降，VMM とする) により，実計算機を複数の仮想的な計算機に見せかけ，この上でゲスト OS を複数動作させることが可能である．VM 方式では，入出力要求時，1 台の計算機上で 1 つの OS が動作する場合のように直接入出力機器に処理依頼を発行出来ず，VMM の仲介が必要となる．このため，VM は，VMM との依存性により，実計算機と同等の性能と環境では走行できない．

一方，より実計算機に近い環境で複数 OS を走行させる方式として，TwinOS 方式 [1] が提案されている．この方式の構成を 図 1.2 に示す．また，TwinOS 方式の実装として，シングルコア CPU 上で 2 つの Linux を走行させる TwinOS がある．TwinOS は，仮想化を用いず各 OS を実計算機上で直接走行させるため，VM 方式に比べ高性能で走行可能である．TwinOS は，1 台の計算機ハードウェアにおいて，CPU やメモリ，入出力機器といった各資源について効果的な共有と占有を行っている．例えば，メモリについては，起動時に二分割し，共有部分を有さず，入出力機器については，各 OS 毎に指定された入出力機器のみを占有制御させる．CPU については，シングルコア CPU を時分割することで，2 つの OS を交互に走行させて共有している．このため，割り込み/例外処理では，当該割り込みを発生した入出力機器を占有制御している OS の割り込み処理が呼出されるように制御している．

現在の計算機の主流は，マルチコア CPU に移行している．しかし，マルチコアを効率よく利用することは難しい．そこで，1 台の計算機上で複数の OS を動作させ，これを各コア上で走行させることにより，計算機資源を余すことなく利用可能とすることが考えられる．

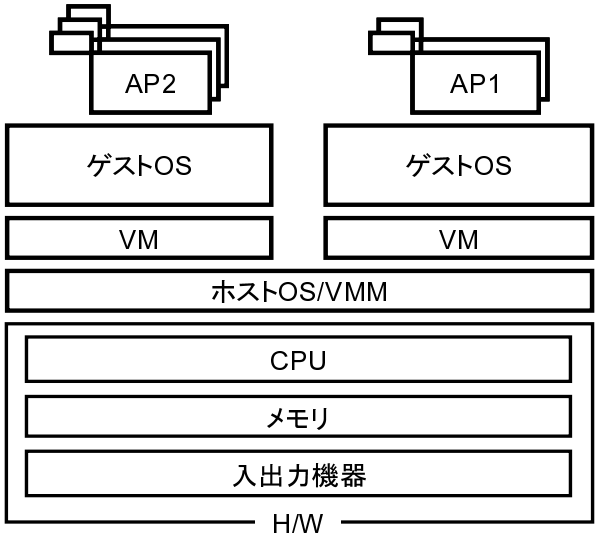


図 1.1 仮想計算機方式を用いたシステム構成例

例えば，VM 方式では，各 VM に各コアを割り当てて動作させることを可能としており，これにより高い並列性を得ている．VM 方式を用いず，複数の OS を各コア上で直接走行させる方式もすでに研究されている [2][3]．このような状況から，マルチコア CPU の計算資源を効率的に利用するため，TwinOS 方式の設計を引き継ぎ，マルチコア CPU 上で複数 OS を独立に動作させる Mint 方式 [4] が提案されている．Mint 方式の構成を 図 1.3 に示す． また，Mint 方式の実装として，マルチコア CPU 上で複数の Linux を走行させる Mint オペレーティングシステム (以降，Mint とする) がある．Mint は，マルチコア CPU をコア単位で分割し，各コア上で OS を走行させることにより，複数 OS 走行環境を実現している．しかし，現在の Mint では，CPU のコアの分割粒度が粗いため，計算機に搭載されたマルチコア CPU のコア数を超えた数の OS を走行出来ない．このため，要求としてコア単位よりも更に分割粒度の細かい制御方法の実現が挙げられている．

本稿では，TwinOS と Mint の計算資源の分割制御方法を基に，複数 OS 走行環境におけるマルチコア CPU の計算資源の効率的な分割制御方法について述べる．具体的には，マルチコア CPU のコア単位での分割制御に加え，時間単位での分割制御の導入を提案する．これにより，複数 OS 走行環境において，柔軟な計算機資源分割が可能となる．また，コア単位の分割制御と時間単位での分割制御の併用方法における課題を述べる．その後，マルチコア CPU の 1 コアを用いた時分割制御法の設計と実装について述べる．

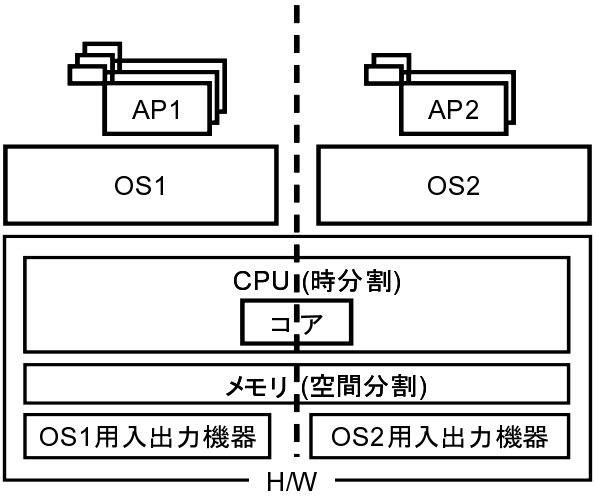


図 1.2 TwinOS 方式を用いたシステム構成例

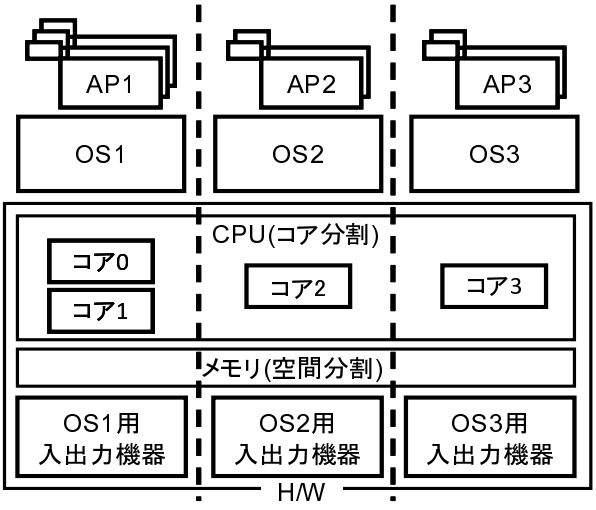


図 1.3 Mint 方式を用いたシステム構成例

# 第 2 章

# TwinOSとMint

## 2.1 特徴と構成

TwinOS と Mint は，1 台の計算機上に複数の Linux を独立に走行させることができる．また，双方ともに以下の特徴を有す．

(1) 相互に他 OS の処理負荷の影響を受けない

(2) 両 OS とも入出力性能を十分に利用できる

このために，1 台の計算機資源を CPU やメモリ，入出力機器といった各資源について効果的な共有と占有を行っている．

メモリについては，走行する OS 数に合わせ空間分割し，占有する．共有部分を有せず，各 OS は自身の占有領域を用いて走行する．

入出力機器については，各 OS 毎に指定された入出力機器のみを占有制御するようにあらかじめ環境設定しておく．このため，割り込み/例外処理では，当該割り込みを発生した入出力機器を占有制御している OS の割り込み処理が呼出されるように制御する．TwinOS の場合は，シングルコア CPU を共有しているため，割り込み発生時に走行している OS が当該割り込みを発生した入出力機器を占有制御している OS と異なる場合，走行 OS の切換 (以降，OS 切換とする) を行う．

CPU については，分割制御方法が双方で異なる．TwinOS は，シングルコア CPU を時分割し，2 つの OS で交互に占有する．一方 Mint は，マルチコア CPU をコア毎に分割し，コア単位で複数の OS が占有する．

## 2.2 CPU 分割制御法

CPU の分割制御法は，TwinOS と Mint で異なる．TwinOS は，シングルコア CPU を利用し，タイマ割り込み，または自 OS 以外の OS が占有する入出力機器からの割り込みを契機に OS 切換することで，2 つの Linux を走行させる．例えば，タイマ割り込み発生時は，まず OS 切換を行う．その後，切換後の OS で OS 切換の契機となった割り込みの処理を行うことで計算資源を割り当てる．

一方，Mint は，マルチコア CPU をコア単位で分割し，コアを各 OS に占有させることで複数の Linux を走行させる．TwinOS 方式と比較し，CPU を共有しないため，各 OS の独立性が高い．

## 2.3 各制御法の問題点

各 CPU 分割制御方法には問題点が存在する．以下でこの問題点について述べる．まず，TwinOS の CPU 分割制御法の問題点を以下に挙げる．

(1) シングルコア CPU 用に実装されているため，マルチコア CPU 上での利用に対応していない．

(2) 並行に走行可能な OS 数は 2 つである．

(3) 計算資源の共有により，Mint と比べ，各 OS の独立性が低い．更に OS 切換によるオーバヘッドが発生する．

次に，Mint の CPU 分割制御法の問題点を以下に挙げる．

(1) 並行に走行可能な OS 数は，最大でコア数と同じ数となり，コア数より多くの OS を走行できない．つまり，並行に走行可能な OS 数は，コア数に依存する．このため，計算機資源の分割粒度は粗い．

# 第 3 章

# 目的

現在の計算機は，計算性能に対して入出力性能は低い．例えば，プロセスがデータを HDD に書き込む際，HDD のシーク時間のため CPU の処理が待たされるということが挙げられる．このような場合，計算性能を最大限に利用できない．つまり，入出力性能と計算性能のバランスが悪く，入出力性能と比較し，過剰に計算性能が割り当てられていると言える．マルチコア CPU の普及により，この傾向はより顕著になっている．これに対処するため，マルチコア CPU において，1 台の計算機上で複数の OS を走行させる Mint が実現されている，Mint は，CPU のコアをコア毎に複数の OS に分割し，それぞれを各 OS で占有することで，効率的に利用できる．しかし，Mint の計算資源分割粒度は，コア単位であり，粗い．コア単位よりも細かい粒度で計算資源の分割ができる TwinOS の時分割制御法は，計算資源の効率的な利用法として妥当であるといえる．

ここで，Mint に TwinOS の時分割制御方法を取り入れ，分割粒度をより細かくしたいという要求がある．前述したように TwinOS の時分割制御法は，マルチコア CPU に対応していない．このため，TwinOS の時分割制御法をマルチコア CPU に対応させ，更に Mint のコア分割制御方式と TwinOS の時分割制御方式の組み合わせによる効率的な計算資源の利用方法を提案する．この提案方式の設計と実装を行い．計算資源の効果的な利用を実現する．

# 第 4 章

# マルチコアCPU分割制御方法

## 4.1　マルチコアCPUの分割制御方法の種類

マルチコア CPU 上で時分割制御法とコア分割制御法を用いた制御法として，以下の 2 つが挙げられる．

(1) コア分割制御法 (コア単位で分割して管理)

Mint のマルチコア CPU の分割制御法である．この制御法の例を 図 4.1 に示す．マルチコア CPU をコア単位で分割し，各 OS で占有する．図では，OS1 がコア 0 とコア 1 を，OS2 がコア 2 を占有している．

(2) 併用分割制御法 (必要なコアのみ時分割で管理)

コア分割制御法と時分割制御法を併用することにより，特定のコアのみ時分割制御する．この制御法の例を 図 4.2 に示す．OS1 がコア 0 を，OS2 がコア 2 をコア分割制御

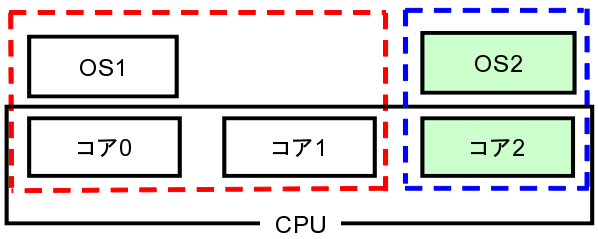


図　4.1　コア分割制御法

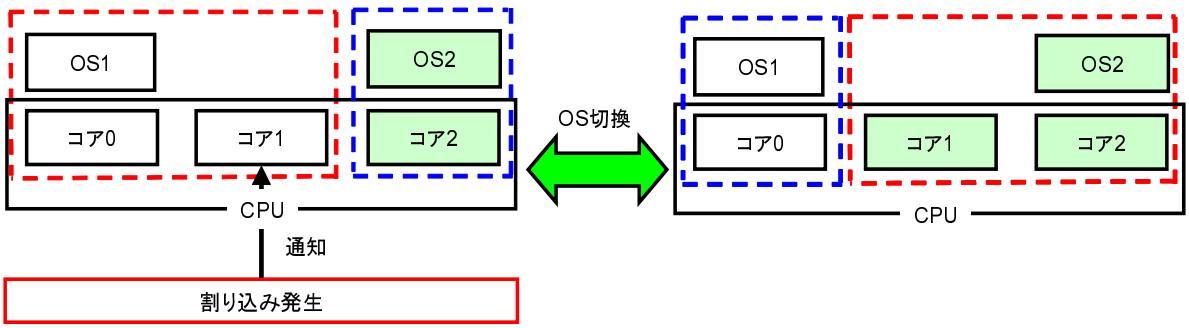


図　4.2　併用時分割制御法

している．また，コア 1 については，時分割制御し，タイマ割り込み，または走行 OS の非占有機器からの割り込みを契機に OS 切換を行うといったように特定のコアのみ時分割制御する．

## 4.2　コア分割制御法

コア分割制御法は，マルチコア CPU をコア単位で分割し，各 OS はコア上で走行する．この制御法の利点と欠点を以下に示す．

(利点 1) 各 OS の独立性が高い

計算資源を共有しないため，各 OS の独立性が高い．

(利点 2) コアの制御に必要な処理が単純である

各 OS の起動処理は，コア分割制御で必要となる処理のみでよいため，併用分割制御法と比べ単純である．また，各 OS は占有コア上で独立に走行する．このため，コアの制御に必要な処理は単純である．

(欠点 1) 計算機資源の分割粒度が粗い

計算機資源の分割粒度は，コアの分割粒度に依存する．つまり，並行に走行可能な OS 数は，実装計算機のマルチコア CPU のコア数と同じであるため，分割粒度は粗い．

表 4.1 マルチコア分割制御方法の比較

| | コア分割制御法 | 併用分割制御法 |
|---|---|---|
| コア分割方式の種類 | コア分割 | コア分割と時分割 |
| 並行に走行可能な OS 数 | コア数による制限あり | 制限なし |
| 計算資源の分割粒度 | 粗い | 細かい |
| OS 切換 | なし | あり |
| 各 OS の独立性 | 高い | 低い |

## 4.3 併用分割制御法

併用分割制御法は，コア分割制御と時分割制御を併用し，マルチコア CPU を分割制御する．この制御法における利点と欠点を以下に示す．

(利点 1) 柔軟に計算資源を分割できる

占有する計算資源の内，一部のみを共有資源とすることで，並行に走行可能な OS 数が実装計算機のマルチコア CPU のコア数に依存しない．更に，必要に応じて計算資源を増減できるため，必要な計算資源を必要なだけ割り当てることが可能である．

(欠点 1) コアの制御に必要な処理が複雑である

コア分割制御と時分割制御の 2 種類の処理を実装する必要がある．このため，処理が複雑になる．

(欠点 2) OS 切換によるオーバヘッドが発生する

時分割制御を行うには，OS 切換が必要である．このため，OS 切換によるオーバヘッドが発生する．

## 4.4 マルチコア CPU 分割制御方法の比較

コア分割制御法と併用分割制御法の利点と欠点を 表 4.1 に示す．併用分割制御法は，コアの共有を行うため，コア分割制御と比較し，各 OS の独立性が下がってしまう．しかし，並行に走行可能な OS 数の制限がなくなり，計算資源分割粒度が向上する．本稿では，計算資源の効率的な利用を目的としている．このため以降では，併用分割制御法を実現する際の課題と設計，および実装について述べる．

# 第 5 章

# 課題

## 5.1 概要

マルチコア CPU の併用分割制御法の実現には，以下の 3 つの課題がある．

(課題 1) シングルコア時分割制御の実装

(課題 2) シングルコア時分割制御のマルチコア CPU 対応

(課題 3) コア分割制御と時分割制御の併用利用対応

これらの課題の内，シングルコア時分割制御は，TwinOS で既に実現されている．しかし，TwinOS で実装されているシングルコア時分割制御には，以下の 2 つの問題点がある．

(1) 3 つ以上の OS を走行できない

TwinOS で実装されているシングルコア時分割制御は，3 つ以上の OS を走行させることができない．より柔軟な計算資源分割のためには，3 つ以上の OS を走行できなくてはならない．

(2) マルチコア上で動作できない

TwinOS は，シングルコア CPU 上でのみ動作する．コア分割制御と併用するには，マルチコア上で動作しなくてはならない．

これら 2 つの問題のため，シングルコア時分割制御についても TwinOS の実装をそのまま流用できない．以降では，これらの課題の内，シングルコア時分割制御の実装における課題の詳細と対処について述べる．

## 5.2 シングルコア時分割制御の実装における課題

シングルコア時分割制御の実装における課題を以下に示す.

(課題 1) メモリの分割と占有

メモリは，走行させる OS の数に合わせ空間分割する．メモリを空間分割するためには，各 OS が使用するメモリの範囲を制限できなければならない．占有する実メモリの終端アドレスは，Linux の既存機能により，ブートパラメータで指定可能である．このため，占有する実メモリの先頭アドレスを指定可能にする必要がある．

(課題 2) 各 OS の起動

複数 OS を同時に起動することは，入出力機器の初期化処理の競合が発生し，困難である．このため，まず 1 つの OS を起動し，この OS から順次他の OS を起動することとする (以降，先に起動する OS を先行 OS，先行 OS から起動する OS を後続 OS とする)．この際，先行 OS は，通常の Linux に近い初期化処理で起動可能である．一方，後続 OS は，先行 OS から起動処理を実行しなくてはならない．この課題の実現のため，以下の 2 つの課題への対処が必要である．

(課題 2-1) 先行 OS の走行環境の保存

後続 OS 起動後は，先行 OS の処理に復帰できなくてはならない．このために必要となる情報を保存しておく必要がある．

(課題 2-2) 後続 OS のメモリ配置と起動

先行 OS は，後続 OS 起動のため，後続 OS の圧縮カーネルイメージを配置する必要がある．圧縮カーネルイメージは，配置したメモリアドレスから展開されるため，後続 OS の占有領域に配置しなくてはならない．後続 OS 起動時は，既に起動している OS の走行環境を保護しなくてはいけない．このため，共有する機器とメモリについて対処が必要である．共有する機器については，後続 OS の初期化処理に対して対処が必要である．また，後続 OS の起動処理は，先行 OS の走行環境を破壊しないように後続 OS を起動する必要がある．

(課題 3) 割り込みのフック

OS 切換を実行する契機を決定し，契機に対応したフック方法を検討する必要がある．

(課題 4) OS 切換

OS 切換は，切換前の走行 OS(以下，切換元 OS とする) の走行環境を保存し，切換後の走行 OS(以下，切換先 OS とする) の環境を復元することにより，走行 OS を切換元 OS から切換先 OS に変更する．この課題の実現のため，以下の 2 つの課題への対処が必要である．

(課題 4-1) 切換元 OS の走行環境の保存と復元

切換元 OS の走行環境を保存し，次回 OS 切換時に切換先 OS に選ばれた場合に利用する．このため，走行環境の保存と復元に必要な情報を明確化する必要がある．

(課題 4-2) マッピングテーブル (以降，MT とする) の変更

切換元 OS と切換先 OS では，使用するマッピングテーブルが異なる．このため，MT の変更処理を行う．この際，他の OS の MT を参照する．OS は，マッピングされてない領域を参照できないため，MT の変更方法を検討する必要がある．

# 第 6 章

# シングルコア時分割制御の設計

## 6.1　設計目標

設計目標は以下の 2 つである．

(1) 各 OS は，複数あるコアのうち 1 つを共有し，2 つ以上の OS を交互に走行する

(2) 入出力機器を機器単位で分割し，非専有機器からの割り込み受付時は走行 OS を切り換える

以降では，5.2 節で示した課題への対処として，シングルコア時分割制御の設計を述べる．

## 6.2　メモリの分割と占有

OS は，使用可能なメモリ領域を起動時にメモリマップを基に認識する．メモリマップとは，BIOS コールを利用することで得られるハードウェアのメモリ実装情報である．そこで，メモリマップのエントリ内容を変更し，指定したアドレス以下の領域を使用不可能な領域とする．先行 OS は，起動時のメモリマップ取得コードを改変し，指定したアドレス以下の領域を占有しないように変更する．後続 OS は，先行 OS から後続 OS の配置時に指定したメモリマップを用いて起動するようにする．

## 6.3　各 OS の起動

計算機の電源投入から後続 OS の起動完了までの流れを 図 6.1 に示し，以下で説明する．計算機を電源投入 (1) すると BIOS(2) が起動される．BIOS(2) は，ブートローダ (3) を呼び出し，ブートローダ (3) は，先行 OS の起動処理 (4) を呼び出す．先行 OS は，自 OS が占有する CPU，メモリ，および入出力機器を設定して起動する．先行 OS 起動後，先行 OS は，後続 OS を起動する．具体的には，先行 OS は，後続 OS が占有予定であるメモリ領域内に後続 OS のカーネルイメージを配置 (5) する．カーネルイメージ配置 (5) 後，先行 OS は，次回 OS 切換時の自 OS の走行環境復元のため，自 OS の走行環境を保存 (6) する．このため，以下の 4 種類の情報を自 OS のデータ領域に保存する．

(1) OS の仮想空間情報

各 OS が独立にもつ仮想空間を構成するセグメントテーブルとページテーブル情報

(2) 割り込み管理情報

割り込み発生時にコアが参照する割り込み管理テーブル情報

(3) レジスタ情報

OS や AP 処理で用いられるコアごとに持つレジスタ値情報

(4) 走行環境復元処理の開始アドレス情報

切換先 OS の走行環境復元処理の先頭アドレス情報

走行環境の保存 (6) 後，配置したカーネルイメージの先頭に遷移 (7) することで，後続 OS に計算資源を移譲し，後続 OS を起動 (8) する．

## 6.4　割り込みのフック

後続 OS 走行中の割り込み発生時の処理流れを 図 6.2 に示し，以下で説明する．

(1) 割り込みを受付

入出力機器から割り込みを受け付ける．

(2) 割り込みフック処理

割り込みフック処理は，割り込みテーブルから IRQ 番号を取得する処理をフックし，割り込み発生源を特定する．その後は，割り込み発生源の違いにより動作が異なる．後

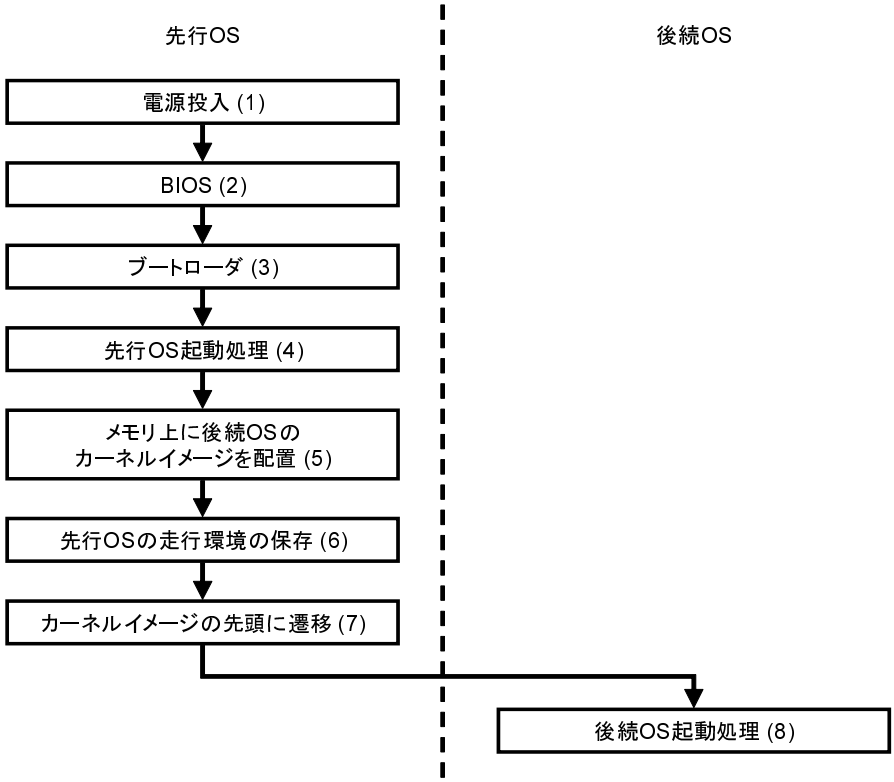


図　6.1　電源投入から後続 OS 起動完了までの流れ

続 OS の占有機器からの割り込みであった場合，割り込みフック処理は，何もしない．つまり，後続 OS の割り込み処理が実行される．後続 OS の非占有機器からの割り込みであった場合，割り込みフック処理は，OS 切換用の処理を呼び出す．

(3) OS 切換

OS 切換は，走行 OS を後続 OS から先行 OS に変更する．OS 切換については，次節で詳しく述べる．

(4) 割り込みの発生源に対応した処理

受け付けた割り込みに対応した処理を実行する．

フック対象となる割り込みについて述べる．タイマ割り込みについて，グローバルタイマ割り込みとローカルタイマ割り込みが存在する．ローカルタイマ割り込みは，グローバルタイマ割り込みと比較して，柔軟に発生頻度を設定できる．このため，ローカルタイマ割り込み

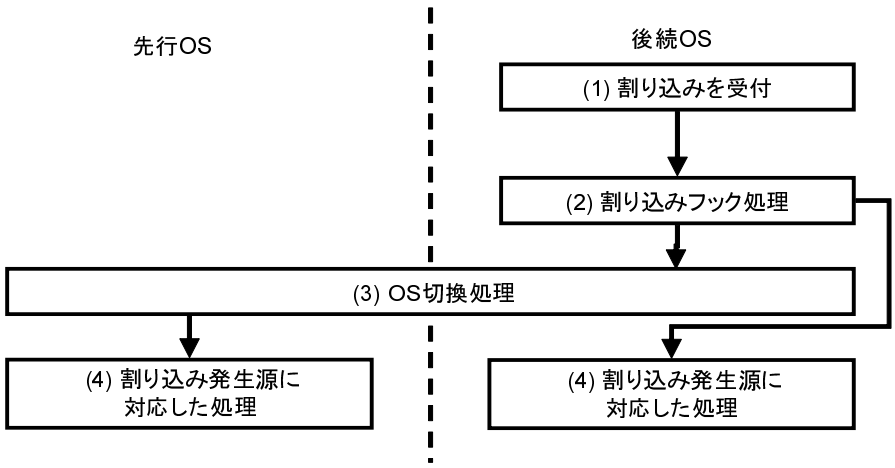


図　6.2　割り込み発生時の処理流れ

を対象とする．その他の入出力機器からの割り込みについては，自 OS の非占有機器からの割り込みを対象とする．

## 6.5　OS 切換

### 6.5.1　OS 切換の流れ

OS 切換は，割り込みのフック処理から呼び出される処理である．OS 切換の流れを 図 6.3 に示す． OS 切換は，以下の処理を順番に行う．

(1) 切換元 OS の走行環境保存

切換元 OS は，自 OS の走行環境を自 OS のデータ領域に保存する．保存する情報は，6.3 節の後続 OS 起動処理時と同様である．

(2) 切換先 OS の MT に変更

MT を切換先 OS の MT に変更する．MT の変更については，次項で詳しく述べる．

(3) 切換先 OS の走行環境復元

切換先 OS は，自 OS のデータ領域に保存して情報を基に走行環境を復元する．走行環境復元処理終了後，OS 切換契機に対応した処理を呼び出す．

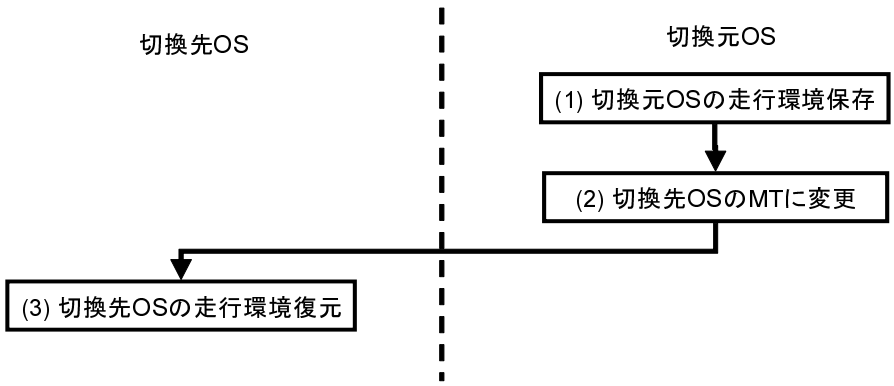


図　6.3　OS 切換の流れ

### 6.5.2　MT の変更処理

OS 切換は，切換元 OS の MT から切換先 OS の MT にページングを変更する．しかし，メモリの若番に配置している OS からは，メモリの老番に配置している OS の MT を参照出来ない．このため，ある OS から他の OS の MT を参照する方法を検討する．MT を参照する方法として，MT の配置場所の違いから以下の 2 箇所が考えられる．

(案 1) 共有領域 (各 OS の非専有領域)

各 OS がストレートマッピングしており，非占有であるメモリ領域を共有領域として利用する．各 OS は，この領域に MT を複写しておき，OS 切換時に参照する．

(案 2) 各 OS の占有領域

各 OS は，自 OS の占有領域に全ての OS の占有領域をマッピングした MT を作成しておく．各 OS は，OS 切換時にこの MT を参照する．

これらの場所に配置した場合の利点と欠点について述べる．

#### (案 1) 共有領域 (各 OS の非専有領域)

各 OS がストレートマッピングしており，非占有である領域を共有領域として利用する．MT を共有領域に配置した場合の後続 OS から先行 OS への OS 切換の流れを 図 6.4 に示し，以下で説明する．MT を共有領域に配置した場合は，以下の処理により OS 切換を行う．

(1) 走行環境保存処理の呼出

OS 切換の契機となる割り込み発生時，割り込みフック処理は，後続 OS の走行環境保存処理を呼び出す．走行環境保存処理は，後続 OS の走行環境を保存する．

(2) MT 変更コードに遷移

後続 OS の走行環境保存後，共有領域内にある先行 OS の MT 変更コードの先頭に遷移する．

(3) 複写した先行 OS MT に変更

MT 変更コードは，共有領域内に予め複写しておいた先行 OS のマッピングに変更する．予め複写しておく MT として swapper_pg_dir を用いる．swapper_pg_dir とは，各 OS が起動時に利用する MT であり，占有メモリの終端が物理メモリの 896M 未満であるなら，占有メモリの全物理ページをストレートマッピングしている．このため，複写する MT として最適である．

(4) MT 変更コードに遷移

MT を swapper_pg_dir に変更後，先行 OS の MT 変更コードの先頭に遷移する．MT 変更コードは，MT を先行 OS が前回 OS 切換時に利用していた MT に変更する，その後，走行環境復元コードを実行し，先行 OS の走行環境を復元する．

次に，MT を共有領域に配置した場合の利点と欠点を以下で述べる．

(利点 1) 特殊なマッピングに対応可能である

SHIMOS[5][6] のようにメモリの 0M からストレートマッピングを開始していない場合にも対応可能である．

(欠点 1) 共有領域が必要である

OS 切換用の領域として，並行に走行するどの OS も利用不可能な (Reserved) 領域が必要となる．また，必要な領域の大きさは，並行に走行する OS 数に比例して増大する．

(欠点 2) 共有領域に配置した MT の先頭アドレスが必要である

マッピングの切換時に共有領域に配置した MT の先頭アドレスが必要となる．

(欠点 3) 複写処理によるオーバヘッドが発生する

MT の複写処理によるオーバヘッドが発生する．複写する MT を swapper_pg_dir とした場合は，各 OS で 1 度必要である．

### (案 2) 各 OS の占有領域

MT を各 OS の占有領域に配置した場合の OS 切換の流れを 図 6.5 に示し，以下で説明する．MT を占有領域に配置した場合は，以下の処理により OS 切換を行う．

(1) 走行環境保存処理の呼出

OS 切換の契機となる割り込み発生時，割り込みフック処理は，後続 OS の走行環境保存処理を呼び出す．走行環境保存コードは，後続 OS の走行環境を保存する．

(2) OS 切換用 MT に変更

現在走行中の OS の走行環境を保存し，OS 切換用 MT に変更する．ここで用いる OS 切換用 MT は，並行に走行する全 OS の占有領域をストレートマッピングしており，事前に作成しておいたものである．

(3) MT 変更コードに遷移

先行 OS 占有領域内の MT 変更コードに遷移する．

(4) 先行 OS の MT に変更

先行 OS の MT に変更する．その後，先行 OS の走行環境を復元する．

次に，MT を各 OS の占有領域に配置した場合の利点と欠点を以下で述べる．

(利点 1) 占有領域以外の領域を利用しない

各 OS の占有領域以外の領域を用いずに OS 切換可能である．

(欠点 1) 他の OS の走行環境復元コードの先頭アドレスの情報が必要である

各 OS は，他の走行中の OS の走行環境復元コードの先頭アドレスの情報を取得できなくてはならない．

(欠点 2) OS 切換用 MT が必要である

各 OS の占有領域に OS 切換用 MT が必要となる．また，MT 作成時には，並行に走行する全 OS の占有領域中一番老番のアドレス情報が必要となる．この情報に関しては，32bit の Linux においてストレートマッピングできる領域の最大である 896M に固定する方法が考えられる．

2 つの OS を時分割制御する場合は，(案 2) が共有領域を用いない点で有用である．しかし，(案 2) では，3 つ以上の OS を並行に走行する場合，OS 終了時等に OS の切換先を変更することが困難である．Mint は，特殊なマッピングを行っていない．また，(案 2) は，実装が単純であるため，(案 2) の占有領域に配置した場合における実装法について以降で述べる．

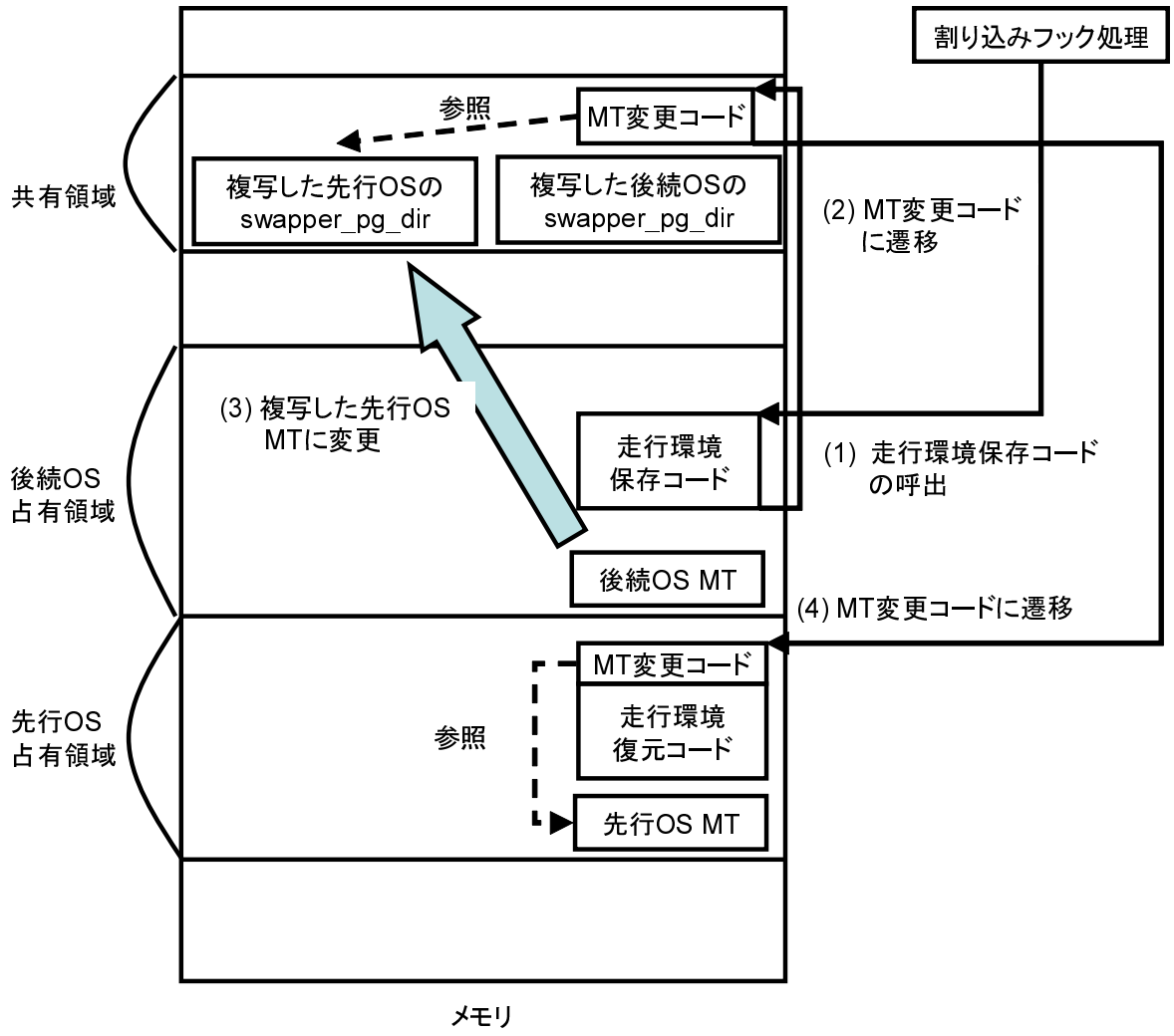


図　6.4　後続 OS から先行 OS への OS 切換 (MT を共有領域配置時)

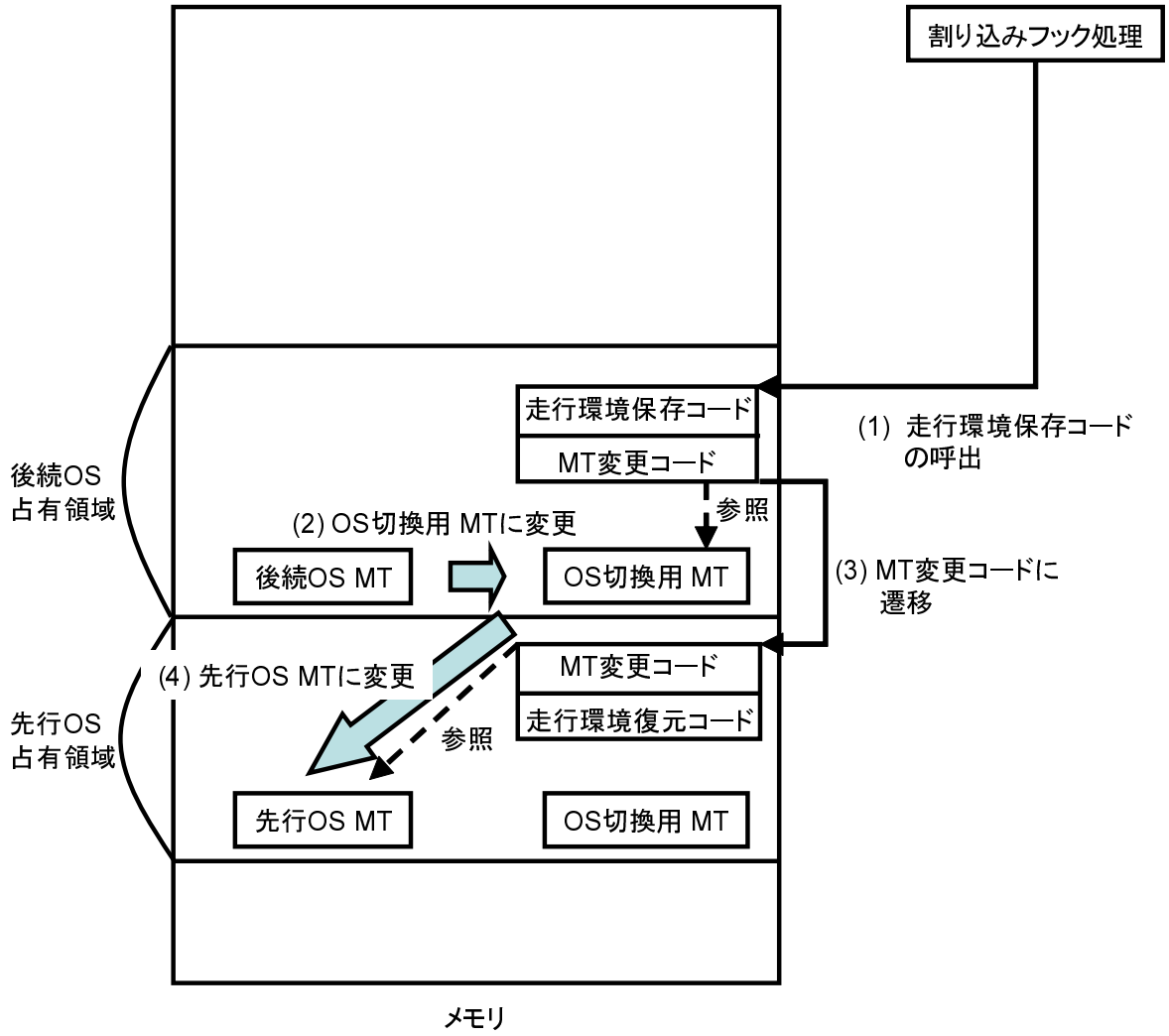


図　6.5　後続 OS から先行 OS への OS 切換 (MT を占有領域配置時)

# 第 7 章

# シングルコア時分割制御の実現

## 7.1　実装環境

シングルコアの時分割制御に用いた実装環境を 表 7.1 に示し，以下で説明する．CPU は，2 つのコアを搭載したマルチコア CPU の内，1 コアのみ利用する．メモリは，2G の内，先頭から 256M の範囲を二分割し，各 OS で占有する．

次に走行させる OS について，シングルコア上で走行する 2 つの OS の構成を 表 7.2 に示し，以下で説明する．Linux は，使用できるコア全てを利用する．このため，両 OS は，1 コアのみ利用するように制限する必要がある．ブートパラメータの変更により，2 つあるコアの内，1 コア (両 OS とも同じコア) を占有する．

## 7.2　メモリの分割と占有

各 OS のメモリマップを 図 7.1 に示し，以下で説明する．先行 OS に関しては，メモリマップを取得するコードを改変し，128M 以下の領域を使用不可能な領域として，OS に認識させる．また，メモリの上限を決定するブートパラメータ mem を 256M に指定し，256M 以上の領域を認識させない．後続 OS に関しては，先行 OS が指定したメモリマップをもとに後続 OS を起動する．具体的には，16M から 128M の領域のみ使用可能な領域とし，16M 以下の

表 7.1 実装環境

| | |
|---|---|
| CPU | Intel core 2 Duo 3.0GHz |
| メモリ | 2GB |
| 入出力機器 | HDD2 台，キーボード，VGA，シリアルポート |

表 7.2 各 OS の構成

| | | |
|---|---|---|
| OS | Linux 2.6.31 | 先行 OS と同様 |
| 占有コア | 1 つ目のコアを分割し，共有 | 先行 OS と同様 |
| 占有メモリ | 128M から 256M<br>128M から OS を配置 | 16M から 128M<br>16M から OS を配置 |
| 占有入出力機器 | HDD1，キーボード，VGA | HDD2，シリアルポート |

領域は使用不可能な領域にする．また，ブートパラメータ mem を 128M に指定し，128M 以上の領域を認識させない．

## 7.3　各 OS の起動

両 OS ともに独自のカーネルイメージを用いる．このカーネルイメージは， 表 7.2 の構成になるようにソースコードを改変する．また，以下の 2 箇所を改変し，CPU の利用制限とカーネルイメージ配置場所の指定を行う．

(1) ブートパラメータ maxcpus を 1 に指定

起動する cpu を 1 つに制限

(2) コンパイルオプション CONFIG_PHYSICAL_ADDR を変更

このオプションを先行 OS なら 128M，後続 OS なら 16M とすることでカーネルイメージの配置場所の先頭アドレスを指定

先行 OS は，ブートローダを利用して起動する．先行 OS 起動後は，後続 OS の読み込みと起動を行う．これには，実装の簡易化と後続 OS 起動の高速化のため，Kexec[7][8][9] を用いる．Kexec とは，Linux でサポートされているカーネルの高速な再起動方式である．Kexec の処理は，後続 OS 起動処理と類似している．このため，Kexec を用いることで，少ない改変量でシングルコア分割制御が実現できる．以下に Kexec の改変点を示す．

(1) OS 読み込み時に後続 OS 用のメモリマップを利用

Kexec を用いた再起動は，走行中の OS のメモリマップを用いて行う．後続 OS 起動時は，先行 OS が指定したメモリマップをもとに後続 OS のカーネルイメージの配置と起動を行う必要がある．このため，起動する OS のメモリマップを指定する部分を変更する．

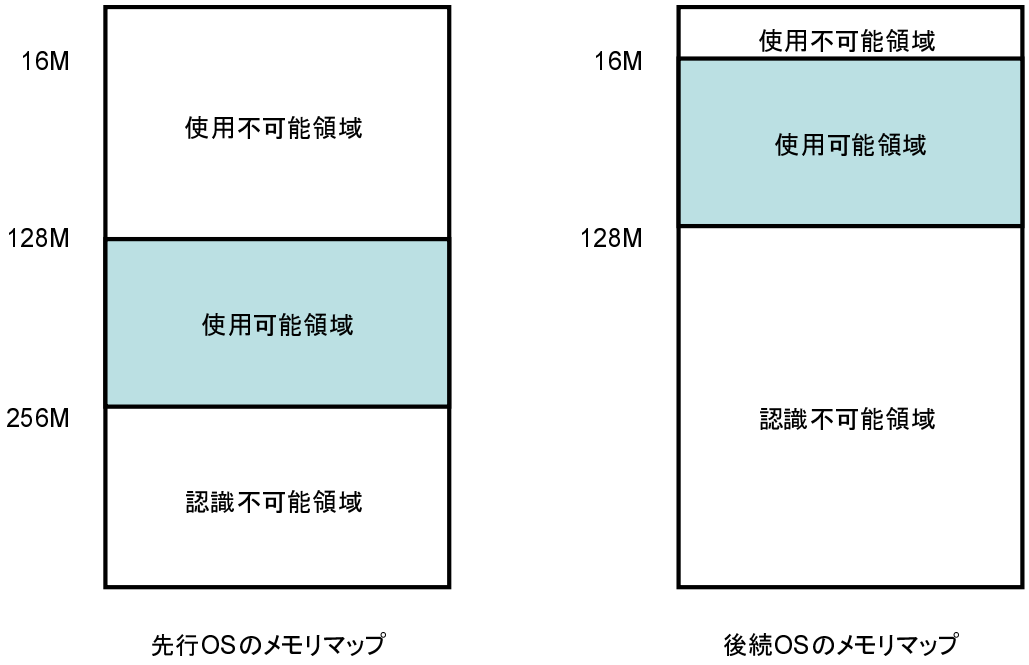


図 7.1 各 OS のメモリマップ

(2) OS 起動前に走行環境を保存

先行 OS の走行環境復元のため，後続 OS 起動前に先行 OS の走行環境を保存する．

(3) 後続 OS 起動前の終了処理を省略

再起動処理では，一度計算機の終了処理を行う．先行 OS の走行環境保護のため，この処理を省略する．

これらの変更により， 図 6.1 の流れで先行 OS と後続 OS を起動する．

## 7.4 OS 切換

OS 切換に用いる 3 種類の MT を 図 7.2 に示し，以下で説明する．ストレートマッピングされた領域に色を付けている．

(A) 先行 OS の MT

OS 切換呼び出し時に先行 OS が使用している MT である．物理メモリの 0M から 256M までを仮想メモリの 3G から 3.25G までにストレートマッピングしている．

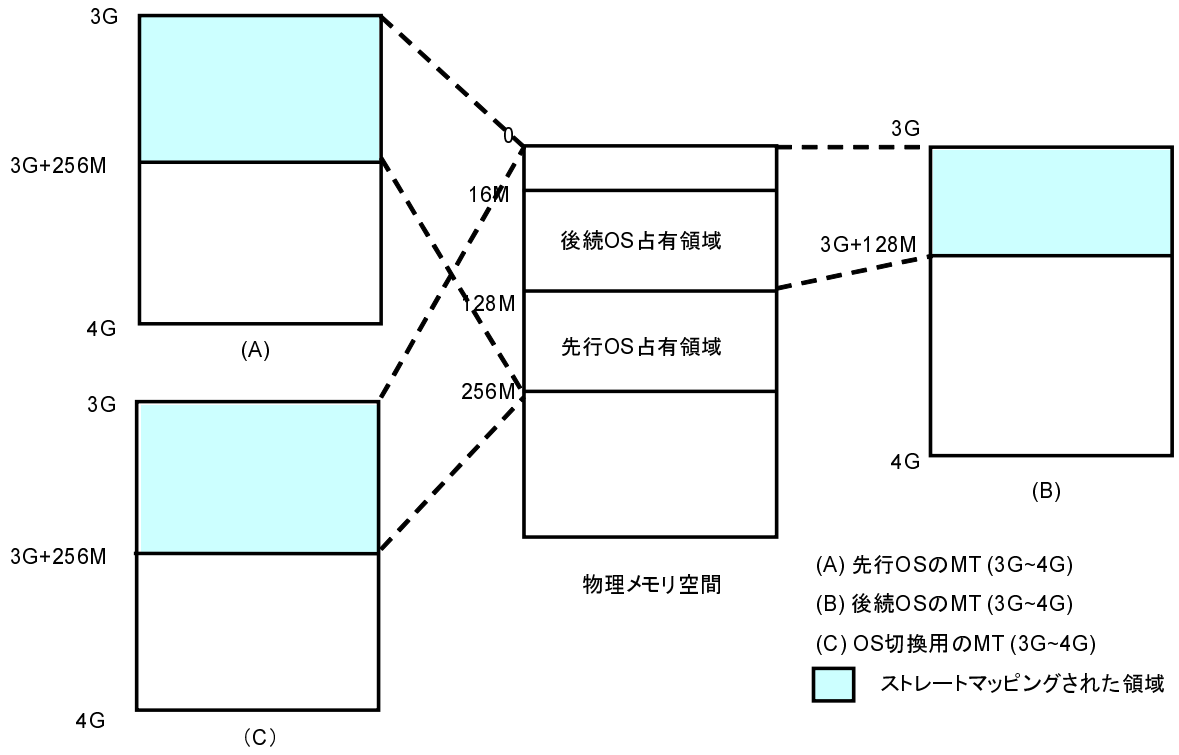


図 7.2 使用する MT

(B) 後続 OS の MT

OS 切換呼び出し時に後続 OS が使用している MT である．この MT は，物理メモリの 0M から 128M までを仮想メモリの 3G から 3.128G までにストレートマッピングしている．

(C) OS 切替用の MT

物理メモリの 0M から 256M までを仮想メモリの 3G から 3.25G までにストレートマッピングしている．この MT は，OS 切換前に作成しておく．4M バイトページ単位で管理し，全てエントリはアクセス，および読み書き可能としている．

次に，後続 OS から先行 OS への OS 切換の処理流れを 図 7.3 に示し，以下で説明する．ストレートマッピングされた領域に色を付けている．後続 OS から先行 OS への OS 切換は，以下の流れで行う．

(1) 後続 OS の走行環境保存

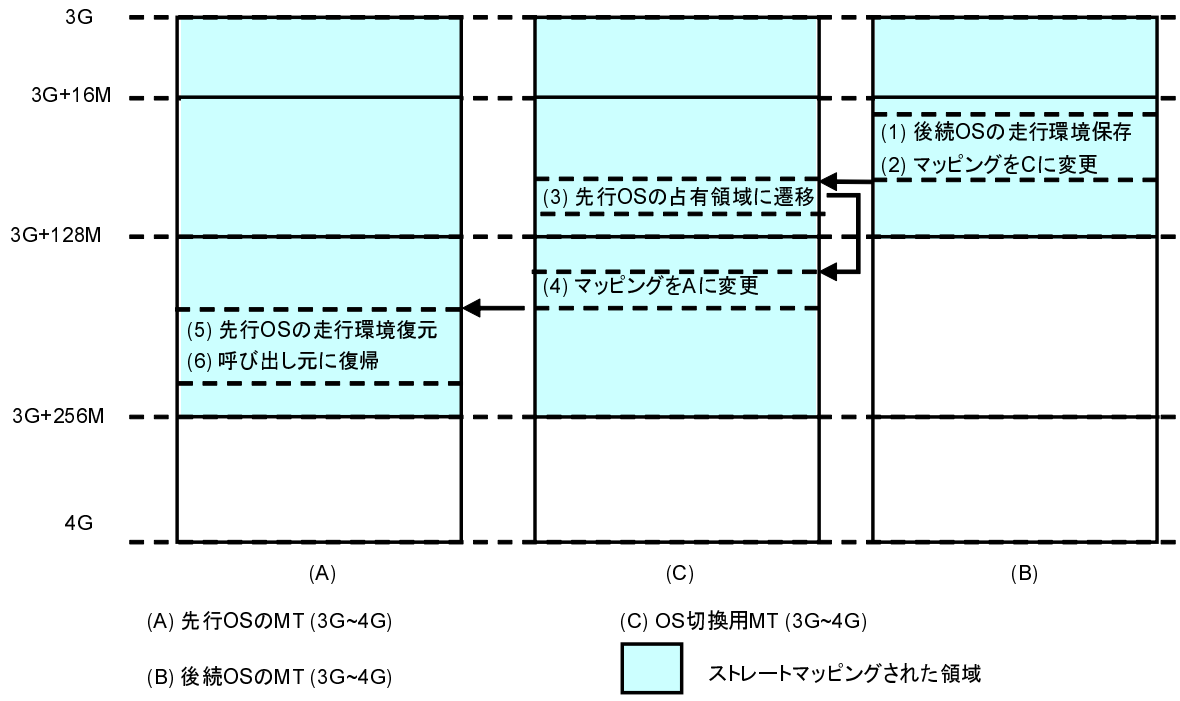


図　7.3　後続 OS から先行 OS への OS 切換の処理流れ

(2) マッピングを C に変更

(3) 先行 OS の占有領域に遷移

(4) マッピングを A に変更

(5) 先行 OS の走行環境復元

(6) 呼び出し元に復帰

以降では，切換元 OS(例では，後続 OS) の走行環境保存処理，MT の変更処理，および切換先 OS(例では，先行 OS) の走行環境復元処理の各々について説明する．

(1) 切換元 OS の走行環境保存処理

割り込みの無効化処理を行った後，切換元 OS の走行環境を保存する．具体的には，以下の情報を自 OS のデータ領域に保存する．

(A) 汎用レジスタ (eax，ebx，ecx，edx，esi，edi，ebp，esp)

(B) セグメントレジスタ (cs，ds，es，fs，gs，ss)

(C) EFLAGS レジスタ (eflags)

(D) 制御レジスタ (cr0，cr2，cr3，cr4)

(E) タスクレジスタ (tr)

(F) GDT，IDT

(G) 走行環境復元コードの先頭アドレス

ここで保存した環境は，今回の切換元 OS が切換先 OS に指定された場合に利用する．

(2) MT の変更処理

MT の変更には， 図 7.2 に示した MT を用いる．MT の変更は，cr3 レジスタに MT の先頭の物理アドレスを格納することで行う．変更手順は， 図 7.3 で示した様に，まず OS 切換用の MT に変更し，その後，切換先 OS の MT に変更という 2 段階の MT 変更を行う．

(3) 切換先 OS の走行環境復元処理

切換先 OS の走行環境を復元する．復元処理終了後，受け付けた割り込みに対応した処理が呼び出される．

# 第 8 章

# 残された課題

マルチコア CPU 分割の今後の課題を以下に示す.

(1) フック処理の実現

シングルコアの時分割制御を実現するには，キーボードやシリアルポートといった入出力機器からの割り込みについてのフック処理を実現する必要がある.

(2) 3 つ以上の OS の走行

現在シングルコア上で 2 つの Linux を走行させることに成功している．これを 3 つ以上の Linux を走行できるように拡張する．OS 切換時に切換先 OS を選択できるようにすることで対応できる.

(3) シングルコア時分割制御のマルチコア CPU 対応

現在は，マルチコア CPU の 1 コアを利用して，シングルコアの時分割制御を実現している．これをマルチコアに対応させる．コア単位での OS 切換のため，OS 切換時にコアの識別番号である Local APIC ID(以降，LAPICID とする) を参照し，自コアと同じ LAPICID の環境を復元することとする．また，切換先 OS に選択された数を保持し，OS の切換先決定時に切換回数の少ない OS に切り換える等の方法により，計算資源の柔軟な分配が行える.

(4) コア分割制御と時分割制御の併用利用対応

Mint のコア分割制御と時分割制御で占有，共有できるようにする．例えば，コア分割制御のコアと時分割制御のコアの両方を占有している場合，タイマ割り込み処理以外

の割り込み処理は，コア分割制御のコアが行う．時分割制御のコアは，タイマ割り込みのみを契機としたOS 切換により共有するといった方法が考えられる．

# 第 9 章

# おわりに

マルチコア CPU 上で動作する複数 OS 走行環境において，時分割制御とコア分割制御により，細かい計算資源の分割制御が可能である併用分割制御法を提案した．更に，併用分割制御法の課題を述べた．併用分割制御法の実現には，シングルコア時分割制御の実装，シングルコア時分割制御のマルチコア CPU 対応，およびコア分割制御と時分割制御の併用利用対応の 3 つの課題の実現が必要である．本稿では，この 3 つの課題の内，シングルコア時分割制御の実装について述べた．シングルコア時分割制御は，TwinOS で既に実装されているが，TwinOS の実装では，3 つ以上の OS を走行できない，マルチコア上で動作できない，という 2 つの問題がある．このため，TwinOS の実装方法を見直し，新たにマルチコアへの対応を見越したシングルコア時分割制御の設計を行った．設計を行うに当たり，シングルコア時分割制御の実装における課題を述べ，この課題に対処した．更に，シングルコア時分割制御を実装した．この実装において，TwinOS で用いられている実装とは異なる方法で行うことにより，シングルコアの時分割制御の簡易化を実現した．また，併用分割制御法における残された課題を述べた．

# 謝辞

本研究を進めるにあたり，懇切丁寧なご指導をして頂きました乃村能成准教授に心より感謝の意を表します．また，数々のご助言を頂きました谷口秀夫教授，山内利宏准教授，および後藤祐介助教に厚く御礼申し上げます．最後に，日頃の研究活動において，お世話になりました研究室の皆様ならびに本研究を行うにあたり，経済的，精神的な支えとなった家族に感謝いたします．

# 参考文献

[1] 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-1，no.2，pp.251-262，2005.

[2] T. Shimosawa, H. Matsuba, Y. Ishikawa,“ Logical Partitioning without ArchitecturalSupports，”Proc. of the 2008 Annual IEEE International Computer Software and Applications Conference, pp.355-364, 2008.

[3] 菅井尚人，遠藤幸典，山口義一，近藤弘郁,“ シングルチップマルチプロセッサ上のハイブリッド OS 環境の実現，” 情報処理学会第 66 回全国大会，2009.

[4] 千崎 良太, 中原 大貴, 牛尾 裕, 片岡 哲也, 粟田 祐一, 乃村 能成, 谷口 秀夫,“ マルチコアにおいて複数の Linux カーネルを走行させる Mint オペ レーティングシステムの設計と評価, ”電子情報通信学会技術研究報告, vol.110, no.278, pp.29-34 (2010.11). 電子情報通信学会 コンピュータシステム研究会 (CPSY).

[5] 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.

[6] 下沢 拓，藤田 肇，石川 裕,“ マルチコア SH における複数カーネル実行機構の設計と実装，”情報処理学会研究報告 2008-OS-109, pp.25-32, 2008.

[7] Hariprasad Nellitheertha,“ Reboot Linux faster using kexec，”IBM,

https://www.ibm.com/developerworks/linux/library/l-kexec.html

[8] Vivek Goyal, Neil Horman, Ken ichi Ohmichi, Maneesh Soni, and Ankita Garg,“ Kdump: Smarter, Easier, Trustier，”2007 Linux Symposium, Volume One, pp.167-178, June.2007.

[9] 中原 大貴, 千崎 良太, 牛尾 裕, 片岡 哲也, 乃村 能成, 谷口 秀夫, “ Kexec を利用した Mint オペレーティングシステムの起動方式, ”電子情報通信学会技術研究報告, vol.110, no.278, pp.35-40 (2010.11). 電子情報通信学会 コンピュータシステム研究会 (CPSY).

# 発表論文

[1] 千崎 良太, 中原 大貴, 牛尾 裕, 片岡 哲也, 粟田 祐一, 乃村 能成, 谷口 秀夫, “ マルチコアにおいて複数の Linux カーネルを走行させる Mint オペ レーティングシステムの設計と評価, ”電子情報通信学会技術研究報告, vol.110, no.278, pp.29-34 (2010.11). 電子情報通信学会 コンピュータシステム研究会 (CPSY).

[2] 中原 大貴, 千崎 良太, 牛尾 裕, 片岡 哲也, 乃村 能成, 谷口 秀夫, “ Kexec を利用した Mint オペレーティングシステムの起動方式, ”電子情報通信学会技術研究報告, vol.110, no.278, pp.35-40 (2010.11). 電子情報通信学会 コンピュータシステム研究会 (CPSY).