特 別 研 究 報 告 書

題 目

Mint オペレーティングシステムにおける コア移譲の管理機能について

指導教員

報 告 者

池田 騰

岡山大学工学部 情報工学科

平成 24 年 2 月 10 日 提出

要約

計算機の高性能化にともない，計算機資源を全て使用することは少なくなってきた．そこで，計算機資源を有効に活用するために，複数の OS を 1 台の計算機で動作させる技術として仮想計算機方式が研究されている．しかし，仮想計算機方式は，計算機の仮想化によるオーバーヘッドのため，実計算機よりも性能が低下する．また，各 OS 間での処理負荷の影響が存在する．そこで，性能の低下を抑えるため，マルチコアプロセッサ上で複数の Linux を独立に走行させる方式として Mint が研究開発されている．

OS の負荷は一定ではなく，時間とともに変化する．このため，負荷に応じて計算機資源を動的に割り当てることで計算機資源を効率的に使用できる．一方で，Mint では計算機資源を各 OS の起動時に静的に割り当てており，計算機資源を動的に割り当てることができない．そこで，コアの動的割り当てを可能とする方法として，Mint におけるコアの動的割り当て機構が提案されている．コアの動的割り当て機構を用いることで，静的に設定されている演算用のコアを指定して取り外しと取り付けができる．Mint においてコアの移譲を行う際は，コアの移譲元の OS でコアの取り外しを行い，コアの移譲先の OS でコアの取り付けを行う．

Mint では OS 間での計算機資源の管理を行っていない．また，OS がコアの識別に用いる ID が OS 間で統一されていない．このため，コア移譲の際には，ユーザが各 OS のコアの占有状況を個別に確認し，コアを識別した上で移譲するコアを指定する必要がある．そこで，OS 間でコアの識別に対する整合性を保ち，全コアの占有状況を把握することで，ユーザが手動で行っている処理を自動化したいという要求がある．この要求を満たすために，コア移譲の管理機能を実現する必要がある．

本論文では，コア移譲の管理機能の実現について述べた．まず，Mint におけるコア移譲について述べ，要求を述べた．次に，要求を実現するための課題を明確にした．そして，課題への対処方法について検討し，コア移譲の管理機能の実現について述べた．

目次

図 目 次

表 目 次

第 1 章

はじめに

計算機の高性能化にともない，計算機資源を全て使用することは少なくなってきた．そこで，計算機資源を有効に活用するために，複数の OS を仮想化して 1 台の計算機で動作させる技術として VMware[1] や Xen[2] といった仮想計算機方式が研究されている．しかし，これらの方式は，計算機の仮想化によるオーバーヘッドのため，実計算機よりも性能が低下する．また，OS 間での処理負荷の影響が存在する．そこで，性能の低下を抑えるため，マルチコアプロセッサ上で複数の Linux を独立に走行させる方式として Mint(Multiple Independent operating systems with New Technology)[3] オペレーティングシステム が研究開発されている．

OS の負荷は一定ではなく，時間とともに変化する．このため，負荷に応じて計算機資源を動的に割り当てることで計算機資源を効率的に使用できる．一方で，Mint では計算機資源を各 OS の起動時に静的に割り当てている．このため，計算機資源を動的に割り当てることができない．この問題を解決するために，コアの動的割り当てを可能とする方法として，Mint におけるコアの動的割り当て機構 [4] が提案されている．コアの動的割り当て機構を用いることで，静的に設定されている演算用のコアを指定して取り外し (以下，解放) と取り付け (以下，占有) ができる．Mint においてコアの移譲を行う際は，まずコアの移譲元の OS でコアの解放を行う．その後，コアの移譲先の OS でコアの占有を行う．

Mint では OS 間での計算機資源の管理を行っていない．また，OS がコアの識別に用いる ID が OS 間で統一されていない．このため，コア移譲の際には，ユーザが各 OS のコアの占有状況を個別に確認し，コアを識別した上で移譲するコアを指定する必要がある．そこで，OS 間でコアの識別に対する整合性を保ち，全コアの占有状況を把握することで，ユーザが手動で行っている処理を自動化したいという要求がある．この要求を満たすために，コア移譲の管理機能を実現する必要がある．

本論文では，コア移譲の管理機能の実現について述べる．まず，Mint におけるコア移譲について述べ，要求を述べる．次に，要求を実現するための課題を明確にする．そして，課題への対処方法について検討し，コア移譲の管理機能の実現について述べる．

第 2 章

Mint オペレーティングシステム

2.1 設計方針

Mint は，マルチコアプロセッサ上で複数の Linux を独立に走行させる方式である．Mint は以下の設計方針に基づき実現されている．

(1) 全ての OS が相互に処理負荷の影響を与えない．

(2) 全ての OS が入出力性能を十分に利用できる．

2.2 構成

Mint の構成例を図 2.1 に示し，CPU，メモリ，および入出力機器についての分割方法を以下で説明する．Mint では，最初に 1 つの OS を起動し，後から別の OS を起動する．最初に起動する OS を OS1 とし，後から起動する OS を起動順に OS2，OS3 とする．

(1) CPU

Mint では，各 OS の起動時にそれぞれの OS が占有するコア数を静的に決定することで OS 間でのコアの分割を行っている．この際，最初に起動する OS はコア 0 を Boot Strap Processor(以下，BSP) とし，最初に起動したコアとして扱う．後に起動する OS は起動時にコア 0 以外の指定したコアを BSP として扱う．

(2) メモリ

OS ごとに使用するメモリマップを改変し，実メモリ領域を分割する．

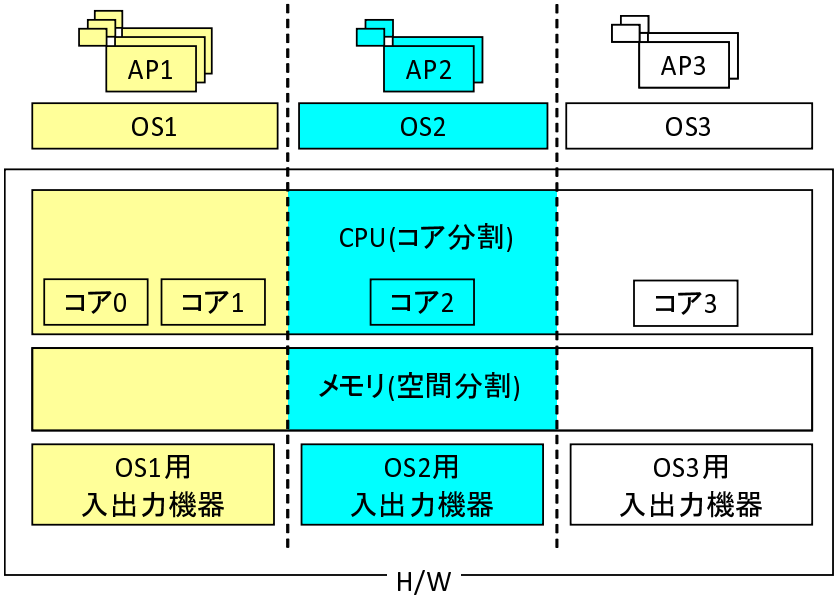


図　2.1　Mint の構成例

(3) 入出力機器

デバイス単位で分割し，各 OS は指定された入出力機器のみを占有する．

第 3 章

Mint におけるコアの移譲

3.1 コアの動的割り当て機構の目的

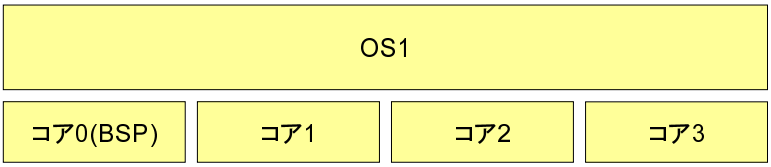
OS の負荷は一定ではなく，時間とともに変化する．このため，負荷に応じて計算機資源を動的に割り当てることで計算機資源を効率的に使用できる．一方で，Mint では計算機資源を各 OS の起動時に静的に割り当てている．このため，計算機資源を動的に割り当てることができない．この問題を解決するために，コアの動的割り当てを可能とする方法として，Mint におけるコアの動的割り当て機構が提案されている．コアの動的割り当て機構を用いることで，静的に設定されている演算用のコアを指定して解放と占有ができる．Mint においてコアの移譲を行う際は，まずコアの移譲元の OS で解放を行う．その後，コアの移譲先の OS でコアの占有を行う．

3.2 Linux におけるコアの識別方法

図 3.1 に Linux におけるコア識別子の割り当て例を示し，以下で説明する．コア識別子とはコアの識別に用いる ID である．Linux のコア識別子は，以下の 3 つがある．

(1) コアの識別用 ID(以下，コア ID)

コア ID は，Linux が自身の占有するコアの識別に使用する ID である．このコア ID について，Linux は BSP をコア ID 0 として設定し，以降コアの検出順に 1,2,··· と設定する．



OS1のコアID	コアID:0	コアID:1	コアID:2	コアID:3
LAPIC ID	LAPIC ID: 0	LAPIC ID: 2	LAPIC ID: 4	LAPIC ID: 6
論理APIC ID	論理 APIC ID: 1	論理 APIC ID: 2	論理 APIC ID: 4	論理 APIC ID: 8

図 3.1　Linux におけるコア識別子の割り当て例

(2) Local APIC ID(以下，LAPIC ID)

電源投入時にハードウェアによって各コアに割り当てられる ID である．

(3) 論理 APIC ID

割り込みの通知先を指定する際に用いられる ID である．Linux ではコア ID をもとに作成される ID であり，2 の N 乗 (N は対応するコア ID) で表される．

3.3　Mint におけるコアの識別の問題点

図 3.2 に Linux におけるコア識別子の割り当てをそのまま複数 OS に割り当てた例を示し，以下で説明する．Mint では，コア識別に関して以下の 2 つの問題がある．

(1) OS 間でコア ID の整合性が保てない．

Mint において，コア ID は各 OS が自身の占有するコアの識別に使用する ID である．コア ID について，Linux は BSP をコア ID 0 として設定し，以降コアの検出順に 1,2,･･･と設定する．Linux をベースとする Mint では，各 OS の BSP が異なるため，複数の OS 間で同一のコアに割り振られるコア ID が異なることがある．図 3.2 では，OS1 はコア 0 を BSP として起動しており，OS2 と OS3 はそれぞれコア 2 とコア 3 を BSP として起動している．OS1 はコア 0 を BSP としているため，コアの検出順にコア ID が 0 から順番に割り当てられる．OS2 はコア 2 を BSP としているため，コア 2 にコア ID 0 が割り当てられる．コア ID 1 以降はコアの検出順にコア ID 1 から割り振られる．このため，コア 3 以外は，OS1 と OS2 で割り当てられるコア ID は異なる．OS3 はコア 3 を BSP としているため，コア 3 にコア ID 0 が割り当てられる．コア ID 1 以降

	OS1		OS2	OS3
	コア0(BSP)	コア1	コア2(BSP)	コア3(BSP)
LAPIC ID	LAPIC ID: 0	LAPIC ID: 2	LAPIC ID: 4	LAPIC ID: 6
OS1のID	コアID: 0	コアID: 1	コアID: 2	コアID: 3
	論理APIC ID: 1	論理APIC ID: 2	論理APIC ID: 4	論理APIC ID: 8
OS2のID	コアID: 1	コアID: 2	コアID: 0	コアID: 3
	論理APIC ID: 2	論理APIC ID: 4	論理APIC ID: 1	論理APIC ID: 8
OS3のID	コアID: 1	コアID: 2	コアID: 3	コアID: 0
	論理APIC ID: 2	論理APIC ID: 4	論理APIC ID: 8	論理APIC ID: 1

図 3.2 Linux におけるコア識別子の割り当てをそのまま複数 OS に割り当てた例

はコアの検出順にコア ID 1 から割り振られる．このため，OS3 は OS1 と比較すると，全てのコアに割り当てられるコア ID が異なる．また，OS3 と OS2 を比較すると，コア 2 とコア 3 のコア ID が異なる．

(2) 複数の OS 間で論理 APIC ID の整合性が保てない．

論理 APIC ID は，割り込みの通知先を指定する際に用いられる ID である．Mint では，複数の OS 間で同一のコアに割り当てられるコア ID が異なることがある．このため，Linux と同じように，2 の N 乗 (N はコア ID) で論理 APIC ID を算出すると，複数の OS 間で論理 APIC ID の整合性が保てない．

図 3.3 に，Mint におけるコア識別子の割り当て例を示し，以下で説明する．複数の OS 間で論理 APIC ID の整合性が保てない問題について，論理 APIC ID は割り込みの通知先の指定に使用されるため，複数の OS 間で整合性が保てないことが問題となる．複数の OS 間で論理 APIC ID の整合性が保てないのは，論理 APIC ID の算出元であるコア ID が複数の OS 間で整合性を保てないためである．このため，複数の OS 間でのコア ID の整合性を保つように改変することで，複数の OS 間で論理 APIC ID の整合性を保つことができる．ただし，もともと Linux では BSP の異なる複数の OS が同時に走行することを想定していない．このため，Linux が内部で使用するコア ID は，BSP を 0 とする実装となっている．コア ID の割り当てを変更するためには，コア ID 0 を BSP と想定して処理している部分を全て改変しなければならない．このため，カーネルに対する改変量が多くなると予想され，実装が困難であると考えられる．そこで，Mint では論理 APIC ID の算出方式で，コア ID の代わりに OS 間で一意な値を取る LAPIC ID を用いるよう改変している．これにより，複数

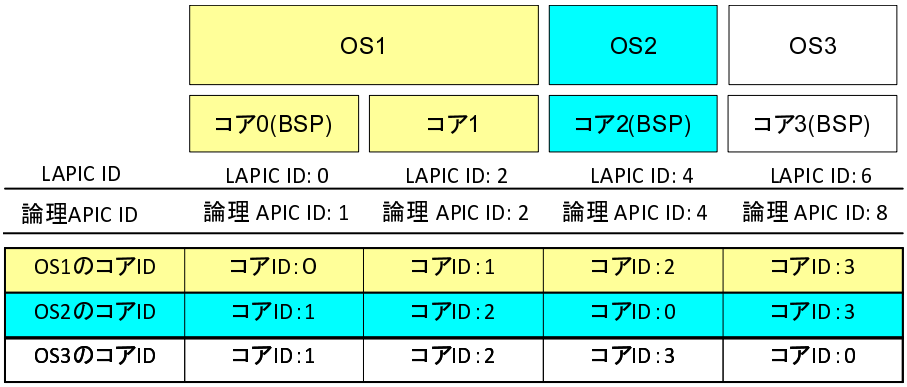


図　3.3　Mint におけるコア識別子の割り当て例

の OS 間で論理 APIC ID の整合性を保っている．

複数の OS 間でコア ID の整合性が保てない問題について，コア ID は 1 つの OS 内のみで使用されるコア識別子である．このため，OS 間でコア ID に関する整合性を保てないことが Mint 全体に与える影響は小さい．また，OS 間でコア ID の整合性を保つための実装は困難である．このため，コア ID の整合性が保てない問題については，対処を行っていない．

3.4　CPU ホットプラグ機能

コアの動的割り当て機構では，OS 間でのコアの移譲に Linux の既存機能である CPU ホットプラグ機能を用いている．CPU ホットプラグ機能とは，Linux において動的にコアの占有と解放を行うための機能である．

CPU ホットプラグ機能を用いてコアの移譲を行う利点として，Linux の既存機能を用いてコアの移譲を実装できるため，Linux のコードを変更する量を抑える事ができる．また，Linux がバージョンアップした際に Mint を Linux のバージョンアップに容易に対応させることができる..

3.5　コア移譲の管理機能における要求

Mint では OS 間での計算機資源の管理を行っていない．また，OS がコアの識別に用いる ID が OS 間で統一されていない．このため，コア移譲の際には，ユーザが各 OS のコアの占有状況を個別に確認し，コアを識別した上で移譲するコアを指定する必要がある．そこで，OS 間でコアの識別に対する整合性を保ち，全コアの占有状況を把握することで，ユーザが

手動で行っている処理を自動化したいという要求がある．以下で，これらの要求について説明する．

(要求 1) 全てのコアの占有状況を把握したい．

Mint は占有する計算機資源を起動時に静的に決定し，OS 間で計算機資源の管理を行なっていない．このため，各 OS は自身の占有する資源は確認できるが，他の OS が占有する資源を確認できない．例として，図 3.3 では，コア 0 とコア 1 を占有する OS1 は自身がコア 0 とコア 1 を占有していることは確認できる．しかし，OS1 は OS2 と OS3 がどのコアを占有しているか，あるいは未使用のコアが存在するか否かを確認できず，走行している OS の個数も確認できない．コアの移譲を行うためには全ての OS のコアの占有状況を把握した上でコアの解放とコアの占有を行う必要がある．このため，全てのコアの占有状況を把握可能にしたいという要求がある．

(要求 2) OS 間でコアの識別に関する整合性を保ちたい．

Mint では，同一のコアであっても OS ごと割り振られるコア ID が異なることがあるため，OS 間でコアの識別に関する整合性を保つことができない．このため，コアの解放を行う OS とコアの占有を行う OS で，同一のコアに対して別々のコア ID を指定する必要がある．異なる OS 間であっても同一のコアに対して同じ ID を使用したいため，OS 間でのコアの識別に関する整合性を保ちたいという要求がある．

(要求 3) 手動で行っている処理を自動化したい．

Mint では，OS 間の協調をしていない．しかし，コア移譲において，移譲元 OS と移譲先 OS の 2 つ OS の協調が必要となる．このため，コア移譲の際にユーザが OS 間の協調を行わなければならない．図 3.3 の構成の Mint を例に，Mint におけるコア移譲の手順を以下に示し，説明する．

(1) 各 OS の起動時にどのコアを BSP としたかを予め記録しておく．

OS 間でのコア ID の違いは BSP とするコアの違いによって発生する．このため，OS 間でのコア ID の違いを意識するためには，各 OS がどのコアを BSP としているかを記録しておかなければならない．例として，図 3.3 では，OS2 の起動時に，OS2 の BSP はコア 2 であることを記録しておき，OS3 の起動時に OS3 の BSP はコア 3 であることを記録しておく．

(2) 移譲元の OS が占有しているコアの確認を行う．

解放するコアを決定するため，移譲元の OS でコアの占有状況の確認を行う．OS の占有しているコアは，/sys/devices/system/cpu/ 以下にあるファイル online の中身を確認することで分かる．この際に表示されるコアはコア ID によって示される．例として，図 3.3 のように OS1 がコア 0 とコア 1 を占有している状態で，OS1 がコアの占有状況を確認すると以下のようになる．

```
# cat /sys/devices/system/cpu/online


0-1
```

これは，OS1 がコア ID 0 のコアとコア ID 1 のコアを占有していることを示している．

(3) 移譲元 OS で CPU ホットプラグ機能を用いて移譲するコアを解放する．

具体的には/sys/devices/system/cpu 以下にあるファイル cpuX/online(X は解放するコアのコア ID) に 0 を書き込むことでコアを解放する．例として，OS1 からコア 1 を解放する場合は，OS1 で以下のコマンドを入力する．

```
# sudo sh -c 'echo 0 > /sys/devices/system/cpu/cpu1/online'
```

(4) (1) で各 OS の BSP がどのコアであるかを記録した情報から，解放したコアが移譲先の OS のどのコア ID に当たるかを計算する．

ユーザは各 OS が BSP としたコアの情報から，図 3.3 のような OS ごとの対応を計算し，解放したコアが移譲先 OS で割り振られているコア ID を求める．例として，図 3.3 の構成の Mint で OS1 がコア ID 1 のコアを OS2 に移譲する場合を想定する．まず，OS1 の BSP はコア 0 であることから，OS1 において，コア ID 1 がコア 1 を指していることを計算する．次に，OS2 の BSP はコア 2 であることから，OS2 において，コア 1 には コア ID 2 が割り当てられていることを計算する．

(5) 移譲先 OS において，(4) で求めたコア ID のコアを CPU ホットプラグ機能を用いて占有する．

具体的には/sys/devices/system/cpu 以下にあるファイル cpuX/online(X は占有するコアのコア ID) に 1 を書き込むことでコアを占有する．例として，OS2 がコア 1 を占有する場合は，OS2 で以下のコマンドを実行する．

```
# sudo sh -c 'echo 1 > /sys/devices/system/cpu/cpu2/online'
```

以上が Mint におけるコア移譲の手順である．OS 間の協調を可能にすることで上記の手順の自動化ができる．このため，コア移譲に関する OS 間の協調を可能にし，これらの手順を自動化したいという要求がある．

第 4 章

コア移譲の管理機能を実現するための課題と対処

4.1　課題

3.5 節で述べた 3 つの要求を満たすため，コア移譲の管理機能を実現する．コア移譲の管理機能の実現において，以下の 3 つの課題が存在する．

(課題 1) コアの占有状況を管理する機能の導入

OS が全コアの占有状況を把握可能にするために，コアの占有状況について管理するための機能を導入する．

(課題 2) コア識別子変換機能の導入

OS 間でコアの識別に対する整合性を保つために，コア識別子変換方式を導入する．

(課題 3) コアの自動指定機能の導入

コア移譲の際にユーザがコア ID を指定する手順を自動化するために，コアの自動指定機能を導入する．ユーザはコア移譲の際にコア ID の指定ではなく，移譲するコア数を指定する．

4.2　コアの占有状況を管理する機能の導入

4.1 節で述べた (課題 1) への対処として，以下の 2 つの対処案が考えられる．

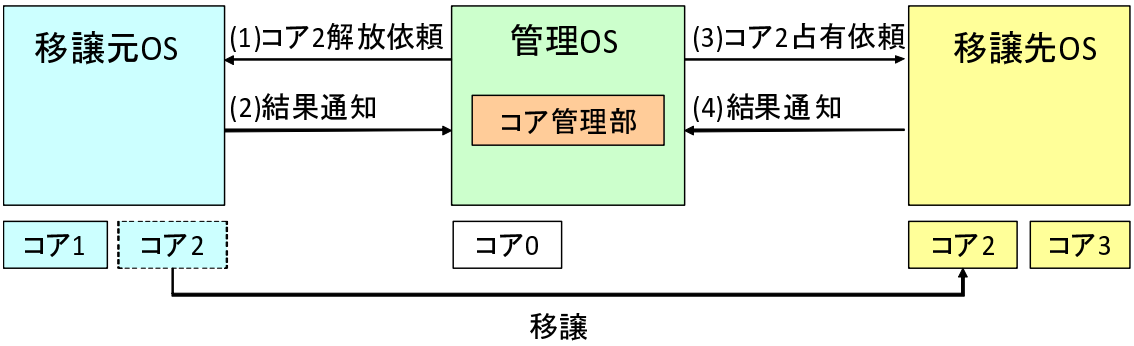


図　4.1　管理 OS によるコアの集中管理を行う場合のコアの移譲機能の実現例

(対処案 1) 特定の OS によるコアの集中管理

特定の OS(以下，管理 OS) がコアの占有状況の管理を行い，各 OS に指定したコアの占有と解放を命令する．

図 4.1 に管理 OS によるコアの集中管理を行う場合のコアの移譲機能の実現例を示し，以下で説明する．図 4.1 では移譲元 OS からコア 2 を移譲先 OS に移譲している．この際の手順を以下に示す．

(1) 管理 OS が移譲元 OS にコア 2 の解放依頼を送る．

(2) 移譲元 OS がコア 2 の解放結果を管理 OS に返す．

(3) 管理 OS が移譲先 OS にコア 2 の占有依頼を送る．

(4) 移譲先 OS がコア 2 の占有結果を管理 OS に返す．

以下で，管理 OS によるコアの集中管理を行う場合の利点と欠点について説明する．

(1) 利点

(A) 管理 OS 以外の OS はコアの把握が不要

コアの管理は管理 OS のみが行うため，管理 OS 以外の OS はコアの把握が必要ない．また，コア管理部に各 OS の BSP の情報を格納しておくことで，管理 OS 内で各 OS のコア ID とコア識別子の変換が可能である．これにより，管理 OS は自身のコア ID を用いてコアの管理を行い，他の OS は管理 OS が変換したコア識別子を用いて命令を行う．

(B) 1 つの OS の操作でコアを移譲可能

特定の OS が全 OS のコアの解放と占有を集中管理するため，ユーザは集中管理を行う OS のみの操作でコアの移譲ができる．

(2) 欠点

(A) OS 間に依存関係が存在

管理 OS がコアの占有状況の管理とコアの移譲の管理を行い，他の OS は管理 OS に従うため，管理 OS に不具合が生じた場合にはコア移譲ができなくなる．また，コア管理部を管理 OS のみが持つため，管理 OS に不具合が生じた場合にコアの管理ができなくなる．このように管理 OS が単一障害点となる．

(対処案 2) 各 OS によるコアの分散管理

コアの占有状況を格納したコア管理部を各 OS とは独立に作成し，OS ごとにコア管理部を用いてコアを管理する．まず，コア管理部専用の実メモリ領域を用意し，各 OS から操作可能な共有メモリ領域とする．この共有メモリ領域にコア管理部を作成する．コア移譲では移譲元 OS と移譲先 OS がそれぞれ以下の処理を行う．

(移譲元 OS) コアを解放して占有可能なコアとして管理部に登録

(移譲先 OS) 占有可能なコアを管理部から確認して占有

図 4.2 に各 OS によるコアの分散管理を行う場合の移譲機能の実現例を示し，以下で説明する．図 4.2 は移譲元 OS がコア 0 とコア 1 を占有し，移譲先 OS がコア 2 を占有している状態から，移譲元 OS から移譲先 OS にコア 1 を移譲している．この際の手順を以下に示す．

(1) 移譲元 OS のコアの解放処理

(A) 移譲元 OS がコアを解放

(B) 移譲元 OS がコア管理部をロック

(C) 移譲元 OS がコア管理部を更新

(D) 移譲元 OS がコア管理部のロックを解除

(2) 移譲先 OS のコアの占有処理

(A) 移譲先 OS がコア管理部を参照し，占有可能なコアを確認

(B) 移譲先 OS がコア管理部をロック

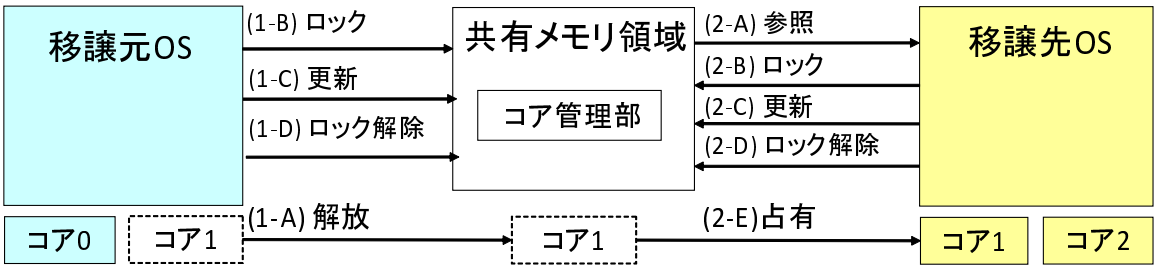


図　4.2　各 OS によるコアの分散管理を行う場合のコア移譲の実現例

(C) 移譲先 OS がコア管理部を更新

(D) 移譲先 OS がコア管理部のロックを解除

(E) 移譲先 OS が占有可能なコアを占有

以下で，各 OS によるコアの分散管理を行う場合の利点と欠点について説明する．

(1) 利点

(A) OS 間に依存関係が存在しない

各 OS はそれぞれコアの解放または占有を行う際に，共有メモリ領域にあるコア管理部を参照と更新をする．この際，コアの解放と占有を行う OS はコア管理部には干渉するが，他の OS には干渉しない．このように，特定の OS に依存しないため，単一障害点が存在しない．ただし，管理部に対してロックをかけている OS に不具合が生じた場合は問題となるため，一定時間同じ OS がロックをかけている場合にはロックを解除するといった機能を作成する必要がある．

(2) 欠点

(A) 各 OS でコアの把握が必要

コア ID は OS ごとに異なるため，コア管理部ではコア ID を使用してコアの管理が行えない．このため全てのコアに OS 間で一意な値を設定する必要がある．その上で，設定した一意な値とコア ID を変換する機能を全ての OS に作成する必要がある．また，コア管理部を共有メモリ領域に置くため，共有メモリ領域を作成する必要がある．

(B) OS 間で排他制御が必要

共有メモリ領域にコア管理部を作成し，全ての OS から更新を可能とするため，コア管理部に排他制御が必要となる．

表 4.1 各管理方式の利点と欠点

管理方式	利点	欠点
管理 OS によるコアの集中管理	(1) 管理 OS 以外の OS はコアの把握が不要 (2) 管理 OS から全操作が可能	(1) OS 間に依存関係が存在
各 OS によるコアの分散管理	(1) 他の OS に依存しない	(1) 全 OS でコアの把握が必要 (2) OS 間で排他制御が必要 (3) コア移譲に 2 度の操作が必要

(C) コア移譲の際に 2 つの OS による操作が必要
コア移譲の際にはコアの解放と占有を別々の OS がそれぞれ実行する必要がある．このため，ある OS から別の OS へコアを移譲する場合には，移譲元の OS でコアを解放し，移譲先の OS でコアを占有するという 2 つの OS による操作を行う必要がある．

以上 2 つの対処案の利点と欠点を表 4.1 にまとめ，以下で説明する．管理 OS よるコアの集中管理では，共有メモリ領域を作成する必要がなく，OS 間でのコア管理部の排他制御を行う必要がない．ただし，OS 間に依存関係があり，単一障害点が存在する．これに対し，各 OS によるコアの分散管理では，OS 間に依存関係がなく，単一障害点が存在しない．ただし，OS 間で共通で使用する領域があるため OS 間で排他制御が必要となる．

OS 間で依存関係がある場合，依存先の OS が不具合を起こした場合に全ての OS に影響を与えることになる．Mint では全ての OS が相互に処理負荷の影響を与えないという方針がある．このため，1 つの OS の不具合が全体に影響を与えることは問題となる．一方，各 OS によるコアの分散管理では，全ての OS でコアの占有状況を把握する必要があり，OS 間での排他制御によるオーバヘッドも問題となる．しかし，コア管理部を使用する頻度は低いと想定されるため，コア管理部に対する排他制御の回数は多くならないと予想される．このため，排他制御によるオーバーヘッドは大きくならない．これらの点を踏まえると，各 OS によるコアの分散管理は，全ての OS でコアの把握が必要となるものの，OS 間で依存関係が存在しないため，この課題の対処に適している．よって，各 OS によるコアの分散管理を行う方式を採用する．

4.3 コア識別子変換機能の導入

4.1 節で述べた (課題 2) への対処として，以下の 2 つの対処案が考えられる．

(対処案 1) 全ての OS で同じコア ID を用いるように改変する．

Linux のコア ID の割り当て方式を変更し，全ての OS で同じコア ID を用いるように改変する．以下で利点と欠点を説明する．

(1) 利点

全ての OS で同じコア ID を用いることができるため，OS 間で ID の変換なしにコアの識別ができる．

(2) 欠点

Linux において，BSP はコア ID 0 と決められている．このコア ID の割り当てを変更するためには，コア 0 を BSP として想定して処理している部分を全て改変しなければならない．このため，カーネルに対する改変量が多くなると予想され，実装が困難であると考えられる．

(対処案 2) 全ての OS で一意に設定されるコア識別子を用意し，コア ID の代わりにコア移譲の際にのみ用いる．

全ての OS で一意に設定されるコアの識別子を用意し，Linux 本来のコア ID と相互に変換する変換部 (以下，ID 変換部) を OS ごとに持たせ，コア移譲の際には変換してコアを特定する．以下で利点と欠点について説明する．

(1) 利点

OS の改変箇所を局所化でき，改変量を小さく抑えることができる．

(2) 欠点

コアの移譲を行うたびにコア ID とコア移譲用のコア識別子との変換を行う必要がある．

OS 間でコアの識別に関する整合性を保つ方法としては，OS 間で ID の変換なしにコアの識別ができる (対処案 1) の方が望ましい．しかし，(対処案 1) を実装するのはカーネルに対する改変量が多くなると予想されるため，実装が困難である．そこで，OS の改変箇所を局所化でき，改変量を小さく抑えることができる (対処案 2) を採用する．

OS 間でコアの識別に使用する共通の ID として LAPIC ID を利用する．LAPIC ID とはハードウェアによってコアごとに割り振られる値であり，全 OS を通して一意である．コア ID を LAPIC ID に変換して OS 間でのコアの管理に利用することで，OS 間のコアの識別に対する整合性を保つことができる．

4.4　コアの自動指定機能の導入

4.4.1　コア移譲 AP

4.1 節で述べた (課題 3) への対処として，以下の 4 つの機能を作成する．

(1) 占有コア自動決定機能

占有可能なコアをコア管理部から検索し，占有するコアを決定する機能である．

(2) 解放コア自動決定機能

解放可能なコアを自身の占有するコアから検索し，解放するコアを決定する機能である．

(3) OS 間 CPU ホットプラグ機能

指定したコアについて，コアの占有と解放を行う機能である．

(4) コア管理部の更新機能

コア管理部の内容を更新する機能である．コアの解放を行った際にコア管理部の内容を更新するために必要となる．コアの占有の際は，占有可能なコアの確保時にコア管理部を更新するため，コア移譲 AP でコア管理部の更新をする必要はない．

4.4.2　コア移譲インタフェース部

コア移譲の際のコア指定を自動化するために，コア管理部と ID 変換部の機能をコア移譲 AP に提供する必要がある．コア移譲 AP が占有可能なコアを確認するためにはコア管理部を利用し，コアの占有状況を確認しなければならない．また，コア移譲 AP が使用する Linux の CPU ホットプラグ機能はコア ID でコアの指定を行う必要があり，コア管理部は LAPIC ID を用いてコアの管理を行う．このため，コア移譲 AP がコア管理部を利用するためにはコア ID 変換部も利用しなければならない．このように，コア移譲の際のコア指定を自動化するためには，コア移譲 AP がコア管理部と ID 変換部の機能を利用しなければならない．コア移譲 AP にこれらの機能を提供するため，コア移譲インタフェース部を作成する．

第 5 章

実現

5.1 コア移譲の管理機能の基本構成

図 5.1 に Mint におけるコア移譲の例を示し，以下で Mint におけるコア移譲の管理機能の構成要素とコア移譲の手順を説明する．コア移譲 AP は，コア解放 AP とコア占有 AP に分かれており，これらの AP はコア移譲インタフェース部を呼び出す．まず，コアを解放する際の手順を以下で示し，説明する．

(A) コア解放 AP

(1) 解放するコア数を引数として，ユーザがコア解放 AP を実行する．

(2) コア移譲インタフェース部を呼び出し，解放するコアの検索を依頼する．

(B) コア移譲インタフェース部

(3) 解放可能なコアを検索し，検索結果から解放するコアを決定する．解放するコアのコア ID をコア解放 AP に返す．

(A) コア解放 AP

(4) 解放するコアに対し，OS 間 CPU ホットプラグ機能を用いてコアを解放する．

(5) コア移譲インタフェース部に解放したコアについてコア管理部の更新を依頼する．

(B) コア移譲インタフェース部

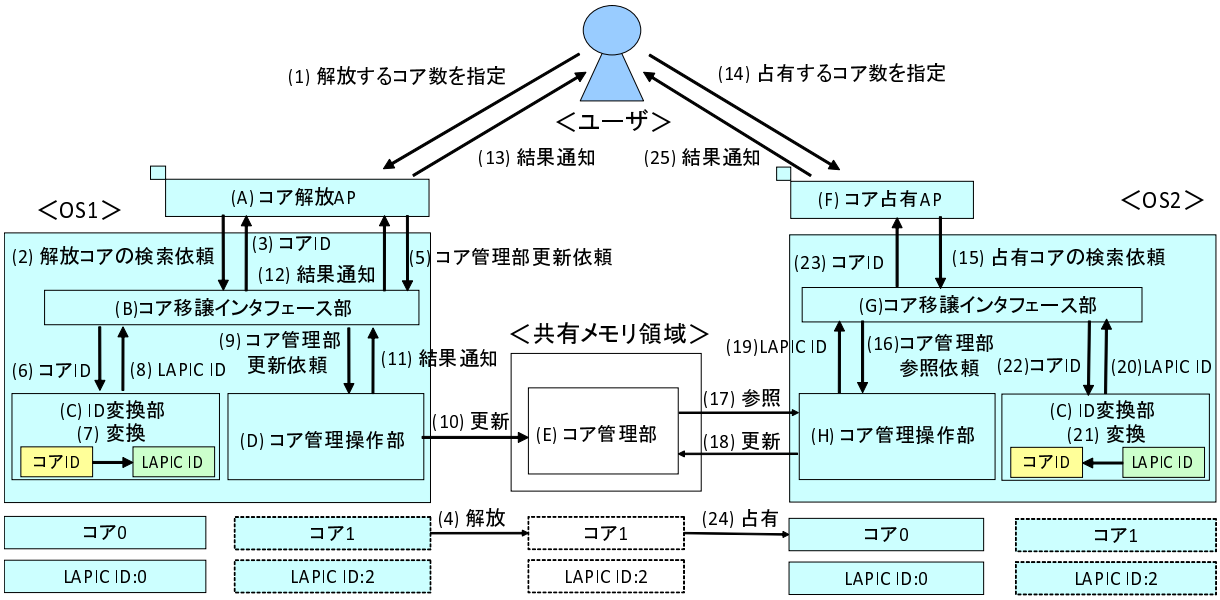


図　5.1　Mint におけるコア移譲の例

(6) コア管理部の更新依頼をされたコアのコア ID について，ID 変換部に LAPIC ID への変換を依頼する．

(C) ID 変換部

(7) コア ID を LAPIC ID に変換する．

(8) LAPIC ID をコア移譲インタフェース部に返す．

(B) コア移譲インタフェース部

(9) 更新するコアの LAPIC ID を用いてコア管理操作部にコア管理部の更新を依頼する．

(D) コア管理操作部

(10) コア管理部の更新を行う．

(11) コア管理部の更新結果をコア移譲インタフェース部に返す．

(B) コア移譲インタフェース部

(12) コア解放 AP にコア管理部の更新結果を返す．

(A) コア解放 AP

(13) ユーザにコア解放の結果を通知する.

次に，コアの占有を行う際の手順を以下に示し，説明する.

(F) コア占有 AP

(14) 占有するコア数を引数として，ユーザがコア占有 AP を実行する.

(15) コア移譲インタフェース部を呼び出し，占有するコアの検索を依頼する.

(B) コア移譲インタフェース部

(16) コア管理操作部に占有するコアの検索を依頼する.

(H) コア管理操作部

(17) コア管理部を参照し，占有可能なコアを検索する．検索結果から占有するコアを決定する.

(18) 占有するコアについてコア管理部の内容を占有済みの状態に更新する.

(19) コア移譲インタフェース部に占有するコアの LAPIC ID を返す.

(G) コア移譲インタフェース部

(20) 占有するコアの LAPIC ID のコア ID への変換をコア ID 変換部に依頼する.

(I) ID 変換部

(21) LAPIC ID をコア ID に変換する.

(22) コア ID をコア移譲インタフェース部に返す.

(G) コア移譲インタフェース部

(23) 占有するコアの LAPIC ID をコア占有 AP に返す.

(F) コア占有 AP

(24) 占有するコアに対し，OS 間 CPU ホットプラグ機能を用いてコアを占有する.

(25) ユーザにコアの占有結果を通知する.

表 5.1 コア管理部の格納データ

コア	占有 OS を示す値
コア 0	0 (OS1 の BSP:コア 0 の LAPIC ID)
コア 1	2 (OS2 の BSP:コア 1 の LAPIC ID)
コア 2	2 (OS2 の BSP:コア 1 の LAPIC ID)
コア 3	255 (占有されていない状態)

5.2 コア管理部の設計

5.2.1 コア管理部の構成

コア管理部で格納するデータは，各コアがどの OS に占有されているかという情報である．コアごとに占有している OS を識別する値を格納する．この OS を識別する値には各 OS の BSP の LAPIC ID を用いる．BSP は OS ごとに 1 つずつ存在し，LAPIC ID は OS 間で一意な値である．このため，BSP の LAPIC ID は OS の識別に利用できる．どの OS にも占有されていないコアに関しては 255 を格納する．255 を占有されていないコアに割り振る理由は，1 Byte で表される LAPIC ID に関するエラーコードが 255 であるため，255 は LAPIC ID として割り振られないからである．

例として，OS1 がコア 0(BSP) を占有し，OS2 がコア 1(BSP) とコア 2 を占有している場合のコア管理部の格納データを表 5.1 に示し，以下で説明する．コア 0 の占有 OS を示す値は 0 である．これはコア 0 が LAPIC ID 0 のコアを BSP とした OS に占有されていることを示す．同様にコア 1 とコア 2 は LAPIC ID 2 のコアを BSP とした OS に占有されていることを示している．コア 3 の占有 OS を示す値は 255 であり，これは占有されていない状態を示している．このように，占有 OS を識別する際に占有 OS の BSP の LAPIC ID を用いることで，OS 間でのコア ID の違いに左右されずに OS を識別可能である．

コア管理部を作成するにあたり，以下の 4 つの項目について設計する必要がある．

(1) コア管理部を作成する OS と作成タイミング

コア管理部を作成する OS とコア管理部を作成するタイミングを設定する必要がある．これについて，最初に起動する OS が起動時にコア管理部を作成する．

(2) 後から起動した OS にコア管理部の存在する位置を伝える方法

最初に起動した OS が作成したコア管理部について，後から起動した OS にコア管理部の存在する実メモリアドレスを伝える方法を設定する必要がある．これについて，管

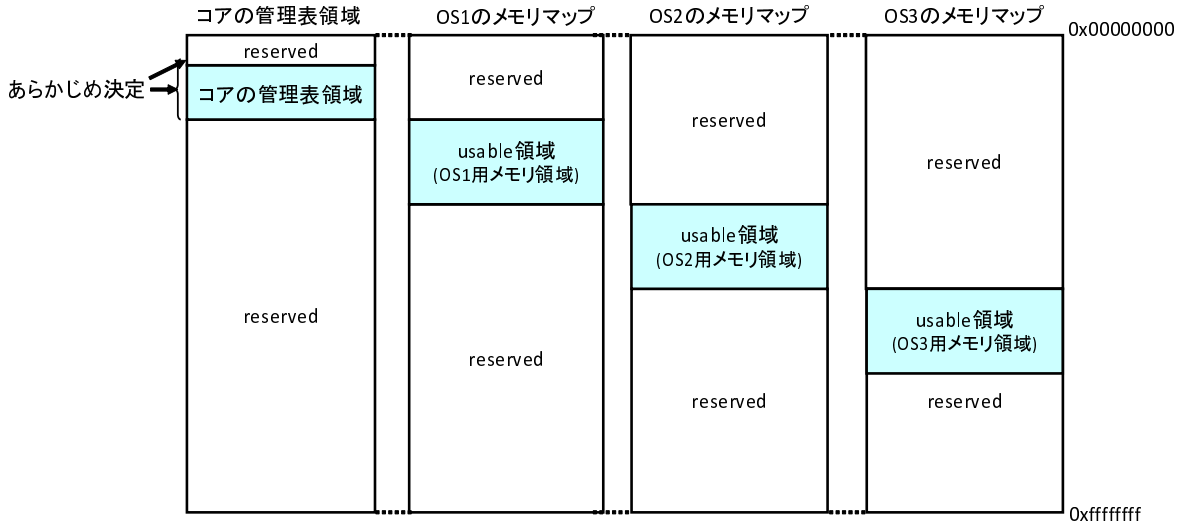


図 5.2 メモリマップの変更

理部を作成するアドレスを静的に設定し，各 OS は静的に設定したアドレスに管理部があるものとする．

(3) 共有メモリ領域の確保

各 OS が利用する実メモリ領域として，コア管理部の領域を使用しないように，各 OS のメモリマップを変更する．メモリマップを変更する様子は図 5.2 で示す．ここでは，OS1，OS2，および OS3 のメモリマップを実メモリ領域の終端側にずらし，先頭側に空いた領域にコア管理部の領域を配置する．これにより，起動する OS を増やした場合でもコア管理部の領域に OS の利用可能 (usable) 領域が重複することは無くなる．

(4) 排他制御の設計

(A) ロックを掛ける範囲の検討

表 5.1 のコア管理部の格納データに対して，ロックを掛ける範囲を検討する．ロックを掛ける範囲としては以下の 2 つがある．

(a) 管理部全体

(b) 更新する項目のみ

通常はロックの範囲が小さい (b) の方が望ましい．ただし，(b) の範囲でロックをかけると，複数の項目に対して同時にロックをかける必要がある場合に，ロックを掛ける順番によってはデッドロックが発生する可能性がある．このため，複数

の項目に対して同時にロックを掛ける必要がある場合には (a) の範囲でロックを行い，ロックする資源を 1 つにすることが考えられる．今回のコア管理部では，コア管理部の更新は 1 度に 1 つの項目についてのみ行うためデッドロックは発生しない．このため，(b) の更新する項目のみをロックする．

(B) 排他制御の手順

排他制御には Compare And Swap(CAS) 命令を用いる．CAS 命令とはあるメモリ位置の内容と指定された値を比較し，等しければそのメモリ位置に別の指定された値を書き込む処理を 1 クロックサイクルで行う命令である．CAS を用いて，コア管理部の更新前の内容の確認と更新を 1 クロックサイクルで行う．コアの解放後に解放したコアの項目の更新をする場合とコアの占有前に占有するコアの項目を更新する場合がある．

CAS による排他制御の手順を以下で説明する．

(a) 更新する項目の値について CAS を用い，想定している更新前の値と同じかどうか確認

解放後の更新の場合は，更新前の値が自身の OS を指す内容かどうかを確認する．占有前の更新の場合は更新する前の値が 255(どの OS も占有していない状態) であるかをどうかを確認する．

(b) 確認した値が想定している値と一致した場合，CAS は項目に指定した値を格納

確認した内容が想定していた内容と一致した場合，解放後の更新の場合は 255 を格納し，占有前の更新の場合は自身の OS を指す内容を格納する．確認した内容が想定していた内容と異なる場合は更新の失敗となる．占有前の更新を失敗した場合は，占有する予定のコアを既に他の OS に占有されていることが予想されるため，占有可能なコアの検索から処理を繰り返す．解放後の更新を失敗した場合は，更新の失敗をユーザに通知して処理を終える．

5.2.2 コア管理操作部の設計

コア管理操作部として以下の 3 つの機能を実現する．

(1) 占有可能なコアのコア ID を返す機能

占有可能なコアを検索し，最初に見つかったコアを占有できる状態にした上で，そのコア ID を返す機能である．手順として，まずコア管理部を参照し，どの OS にも占有

されていないコアを検索する．次に，コア管理部の最初に見つかったコアの項目を占有された状態に更新する．最後にコア管理部を更新したコアのコア ID を返す．

(2) 解放可能なコアのコア ID を返す機能

解放可能なコアを検索し，最初に見つかったコアのコア ID を返す機能である．

(3) コア管理部の更新を行う機能

コア管理部の更新を行う機能である．

5.3 ID 変換部の設計

5.3.1 LAPIC ID を格納する配列について

Linux では，各コアの LAPIC ID は OS の起動時に配列 x86_bios_cpu_apicid に格納されている．各 OS の x86_bios_cpu_apicid の格納内容を表 5.2 に示し，以下で説明する．表 5.2 に示すように，配列 x86_bios_cpu_apicid には，コア ID 0 であるコアの LAPIC ID から，コア ID 1 であるコアの LAPIC ID ，コア ID 2 であるコアの LAPIC ID，･･･と順番に格納されている．

5.3.2 コア ID と LAPIC ID を相互に変換する方法

(1) コア ID を LAPIC ID に変換する方法

配列 x86_bios_cpu_apicid のコア ID 番目の要素内容が対応する LAPIC ID の値となる．

(2) LAPIC ID をコア ID に変換する方法

LAPIC ID からコア ID の変換方法は以下の通りである．配列 x86_bios_cpu_apicid の中身を順番に変換元の LAPIC ID と比較していき，一致した際の配列 x86_bios_cpu_apicid の要素番号がコア ID となる．

5.4 コア選択自動化

5.4.1 コア移譲 AP

コア移譲の際のコアの選択を自動化するために，以下の 2 つの AP を作成する．

表 5.2 各 OS の配列 x86_bios_cpu_apicid の格納内容

	配列 x86_bios_cpu_apicid			
配列の要素番号	0	1	2	3
OS1 (BSP:コア 0)	LAPIC ID:0 (コア 0)	LAPIC ID:2 (コア 1)	LAPIC ID:4 (コア 2)	LAPIC ID:6 (コア 3)
OS2 (BSP:コア 2)	LAPIC ID:4 (コア 2)	LAPIC ID:0 (コア 0)	LAPIC ID:2 (コア 1)	LAPIC ID:6 (コア 3)
OS3 (BSP:コア 3)	LAPIC ID:6 (コア 3)	LAPIC ID:0 (コア 0)	LAPIC ID:2 (コア 1)	LAPIC ID:4 (コア 2)

(1) コア占有 AP

コアの占有を行う AP である．占有するコアの数を引数とし，指定された数のコアを占有する．この際のコアの占有手順を以下に示し，説明する．

(A) コア移譲インタフェース部の占有可能なコアを返す機能を用いて占有可能なコアのコア ID を得る．

(B) 占有可能なコアのコア ID に対して CPU ホットプラグ機能を用いてコアの占有を行う．

(C) 指定されたコア数を占有するか，占有できるコアがなくなるまで (A) と (B) を繰り返す．

(2) コア解放 AP

コアの解放を行う AP である．解放するコアの数を引数とし，指定された数のコアを解放する．コアの解放手順を以下に示し，説明する．

(A) コア移譲インタフェース部の占有可能なコアを返す機能を用いて解放可能なコアのコア ID を得る．

(B) 解放可能なコアのコア ID に対して CPU ホットプラグ機能を用いてコアの解放を行う．

(C) コア移譲インタフェース部のコアの管理部の更新機能を用いてコアの管理部の更新を行う．

(D) 指定されたコア数を解放するか，解放できるコアがなくなるまで (A) から (C) を繰り返す.

5.4.2 コア移譲インタフェース部の設計

コア移譲インタフェース部は，コア移譲の際の占有するコアの選択と解放するコアの選択を自動化するため，コア占有 AP とコア解放 AP にコア管理操作部と ID 変換部の機能を提供する．コア移譲インタフェース部の機能は以下の 3 つである．

(1) 占有可能なコアのコア ID を返す機能

この機能が呼び出されると，まず，占有可能なコアについてコア管理操作部にコア管理部の参照依頼をする．コア管理操作部から占有可能なコアの LAPIC ID が返ってくるため，ID 変換部を呼び出し，LAPIC ID をコア ID に変換する．その後，変換したコア ID を呼び出し元に返す.

(2) 解放可能なコアのコア ID を返す機能

この機能を使用した OS 自身が占有しているコアから解放可能なコアを検索し，発見したコアのコア ID を返す機能である．解放可能なコアについてはコア管理部の参照なしに検索できるため，この機能についてはコア移譲インタフェース部の内部で処理を行う．解放可能なコアの検索方法を以下で具体的に説明する．

CPU ホットプラグ機能では BSP は特殊なコアであるため，解放することができない．このため，BSP を除いたコアから解放するコアを決定する必要がある．OS は自身の占有するコアを確認できる．BSP であるコア 0 を除いて自身の占有するコアを確認し，最初に見つかった占有しているコアを解放するコアとする．

(3) コア管理部の更新機能

指定したコアについてコア管理操作部にコア管理部の更新を依頼する．

第 6 章

おわりに

本論文では，コア移譲の管理機能について述べた．まず，Mint におけるコア移譲について述べ，要求を述べた．次に，要求を実現するための課題を明確にした．そして，課題の対処案について検討し，コア移譲の管理機能を設計した．

Mint におけるコアの移譲について，1 つ目の要求は，全てのコアの占有状況を把握することである．この要求を実現するために，コア占有状況を管理する機能を導入した．コアの占有状況を管理する機能の実現方式として，管理 OS によるコアの集中管理と各 OS によるコアの分散管理を検討した．検討の結果，各 OS によるコアの分散管理を採用し，コア管理部を実現した．2 つ目の要求は，OS 間でコアの識別に関する整合性を保つことである．この要求を実現するために，コア識別子変換機能を導入した．この実現方法を検討した結果，全 OS で一意に設定されるコア識別子を用意し，コア ID の代わりにコア移譲の際に用いる方式を採用し，ID 変換部を実現した．3 つ目の要求は，手動で行っている手順を自動化することである．この要求を実現するために，コアの自動指定機能を導入した．コアの自動指定機能として，コア移譲 AP とコア移譲インタフェース部を実現した．

本論文で設計した Mint におけるコア移譲の管理機能では，演算用のコアについての移譲を行った．残された課題として，コア移譲の際の割り込み通知先の変更処理の実現と CPU の使用率により自動でコアの解放と占有を行う方式の検討がある．

謝辞

本研究を進めるにあたり，懇切丁寧なご指導をしていただ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，研究活動において，数々のご指導やご助言を与えていただいた谷口秀夫教授，山内利宏准教授ならびに後藤佑介助教に心から感謝申し上げます．

また，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に感謝いたします．

最後に，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] Jeremy Sugerman，Ganesh Venkitachalam，and Beng-Hong Lim,“ Virtualizing I/O De-vices on VMware Workstation’s Hosted Virtual Machine Monitor，”Proc. of the GeneralTrack: 2002 USENIX Annual Technical Conference，pp.1-14，2001.

[2] P.Barham，B.Dragovic，K.Fraser，S.Hand，T.Harris，A.Ho，R.Neugebauer，I.Pratt，and A. Warfield. Xen and the art of virtualization. In Proceedings of the ACM sympo-siumon Operating Systems Principles，pages 164-177，October 2003.

[3] 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)

[4] 片岡 哲也,“ Mint オペレーティングシステムにおけるコアの動的割り当て機構, ”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 修士論文,2011