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 OS環境の高速な複製手法

指導教員

報 告 者

西田 潤

岡山大学工学部 情報工学科

平成24年2月9日 提出

要約

計算機のハードウェアは高性能化が進んでおり，1 つのチップに数百個のコアの搭載されたメニーコアプロセッサが登場している．また，メモリの容量も増加し，デバイスも多様化している．この結果，1 つの OS がすべてのコアやメモリ，デバイスを管理するためには，これまで以上に OS の構造は複雑化する．このため，1 つの OS ではコア，メモリ，およびデバイスの管理が難しくなると考えられる．この問題を解決するために，複数の OS を用意し，それぞれの OS で動作させるプロセスにとって動作に適したコア数，メモリ量，およびデバイスのみを保有する．こうすることにより，プロセスの効率的な動作が期待できる．

上記の目標を実現するため，1 台の計算機上で高い独立性を持った複数の OS を走行させ，それぞれの OS に計算機資源を割り当てることができる方式として，Mint が研究開発されている．Mint では，複数の OS 間で高い独立性を実現している．

Mint は，最初に 1 つの OS(以降，先行 OS) を起動させ，先行 OS が後から複数の OS(以降，後続 OS) を順次起動させる．この際，後続 OS のカーネルイメージを HDD から読み込んで実行する．また，後続 OS を起動させる際，システムコールを複数回呼び出している．このため，後続 OS の起動に時間がかかる．Mint において多数の OS を走行させる場合，後続 OS の起動に時間がかかることよって，OS の数を高速に増やせないことが問題となる．

そこで，Mint において，先行 OS の環境を高速に複製する手法を設計する．この手法では，後続 OS を起動させる際，後続 OS のカーネルイメージを HDD から読み込まず，走行中の先行 OS をメモリから複製することで後続 OS を起動する．また，OS を起動させる際，システムコールの呼び出し回数を減らす．この結果，現行の Mint よりも高速に後続 OS を起動できる．

本論文では，OS 環境の高速な複製手法の基本設計とこれを実現するための課題を明確にした．また，複製先 OS の初期化ルーチンにおいてページテーブルの初期化に関する実装を示した．

目次

図 目 次

表 目 次

第 1 章

はじめに

計算機のハードウェアは高性能化が進んでおり，1 つのチップに数百個のコアが搭載されたメニーコアプロセッサ [1] が登場している．また，メモリの容量も増加し，デバイスも多様化している．この結果，1 つの OS がすべてのコア，メモリ，およびデバイスを管理するためには，これまで以上に OS の構造は複雑化する．このため，1 つの OS ではコア，メモリ，およびデバイスの管理が難しくなると考えられる．この問題を解決するために，複数の OS を用意し，それぞれの OS で動作させるプロセスにとって動作に適したコア数，メモリ量，およびデバイスのみを保有する．こうすることにより，プロセスの効率的な動作が期待できる．

上記の目標に対応するため，VMware[2] や Xen[3] のように，1 台の計算機上に複数の OS を走行される方式が活発に研究されている．しかし，計算機の仮想化による処理負荷のため，実計算機よりも性能が低下する．さらに，OS 間の処理負荷の影響が大きい．そこで，1 台の計算機上で高い独立性を持った複数の OS を走行させ，それぞれの OS に計算機資源を割り当てることができる方式として，Mint(Multiple Independent operating systems with New Technology)[4] が研究開発されている．Mint 方式の実装として Mint オペレーティングシステム (以降，Mint と呼ぶ) がある．Mint では，複数の OS 間で高い独立性を実現している．

Mint は，最初に 1 つの OS(以降，先行 OS) を起動させ，先行 OS が後から複数の OS(以降，後続 OS) を順次起動させる．この際，後続 OS のカーネルイメージを HDD から読み込んで実行する．また，後続 OS を起動させる際，システムコールを複数回呼び出している．このため，後続 OS の起動に時間がかかる．Mint においてメニーコアの利用を考慮すると，多数の OS を走行させることが考えられる．多数の OS を走行させる場合，後続 OS の起動に時間がかかることよって，OS の数を高速に増やせないことが問題となる．

そこで，Mint において，先行 OS の環境を高速に複製する手法を設計する．この手法では，後続 OS を起動させる際，後続 OS のカーネルイメージを HDD から読み込まず，走行中の先

行 OS をメモリから複製することで後続 OS を起動する．また，OS を起動させる際，システムコールの呼び出し回数を減らす．この結果，現行の Mint よりも高速に後続 OS を起動できる．

本論文では，まず，現行の Mint の起動方式の問題点を述べ，この問題点を解決するために OS 環境の高速な複製手法を提案する．この際，先行 OS 上で動作している AP は意識せず，カーネルのみを複製の対象とする．次に，この手法の基本設計，課題，および対処を述べる．最後に，対処のうち，ページテーブルの初期化に関する実装を示す．

第 2 章

Mint オペレーティングシステム

2.1 特徴

Mint は，VM の方式によらず，1 台の計算機上で複数の Linux カーネル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 が入出力性能を充分に利用できる．

このため，1 台の計算機上で CPU，メモリ，およびデバイスといったハードウェア資源を効果的に分割し，それぞれの OS で占有する必要がある．まず，最初に 1 つの OS を起動させ，この OS が後から複数の OS を順次起動させる．最初に起動している OS を先行 OS と呼び，後から起動させる OS を後続 OS と呼ぶ．後続 OS は起動順に後続 OS1，後続 OS2，$\cdots$ と呼ぶ．図 2.1 に Mint の構成例を示し，各ハードウェアの分割と占有方法を以下で説明する．

(1) CPU

マルチコアプロセッサを対象とし，コア毎に分割する．そして，1 つ以上のコアを各 OS が占有する．

(2) メモリ

走行させる OS の数だけ実メモリを空間分割し，各 OS で占有する．

(3) デバイス

デバイス単位で分割し，各 OS は指定されたデバイスのみ占有する．

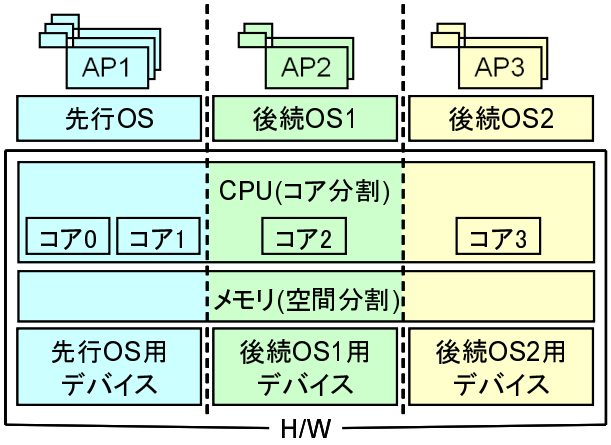

図 2.1 Mint の構成

2.2 起動方式

Mint の起動手順を図 2.2 に示し，以下で説明する．先行 OS は通常の Linux の起動手順と同様にブートローダから起動させ，後続 OS については，先行 OS から順次起動させる．先行 OS は，通常の Linux と同様に，以下の 4 つの手順で起動される．

(1) BIOS の起動

(2) ブートローダの起動

(3) 先行 OS の起動

先行 OS カーネルのセットアップルーチンを介して，先行 OS を起動する．

(4) 先行 OS のアプリケーション (以降，AP) 起動

先行 OS において，後続 OS の起動のために Kexec[5] を変更して利用する [6]．Kexec とは，Linux カーネルの高速な再起動方式である．Kexec は，指定した実メモリ上にカーネルイメージを配置する機能や，本来 BIOS やセットアップルーチンで行う初期化の結果をあらかじめ用意して再現する機能を持つ．これにより，後続 OS は BIOS とブートローダを介さずに起動できる．Kexec は，AP 部分とシステムコール部分に分かれている．これらを後続 OS 起動用に変更して使用する．システムコールの処理はロード処理 (図中 5-1) と Inter-Processor Interrupt(以降，IPI) の送信 (図中 5-2) の 2 つの手順からなる．図 2.3 で，この 2 つの処理における実メモリ上の様子を詳細に示し，以下で説明する．

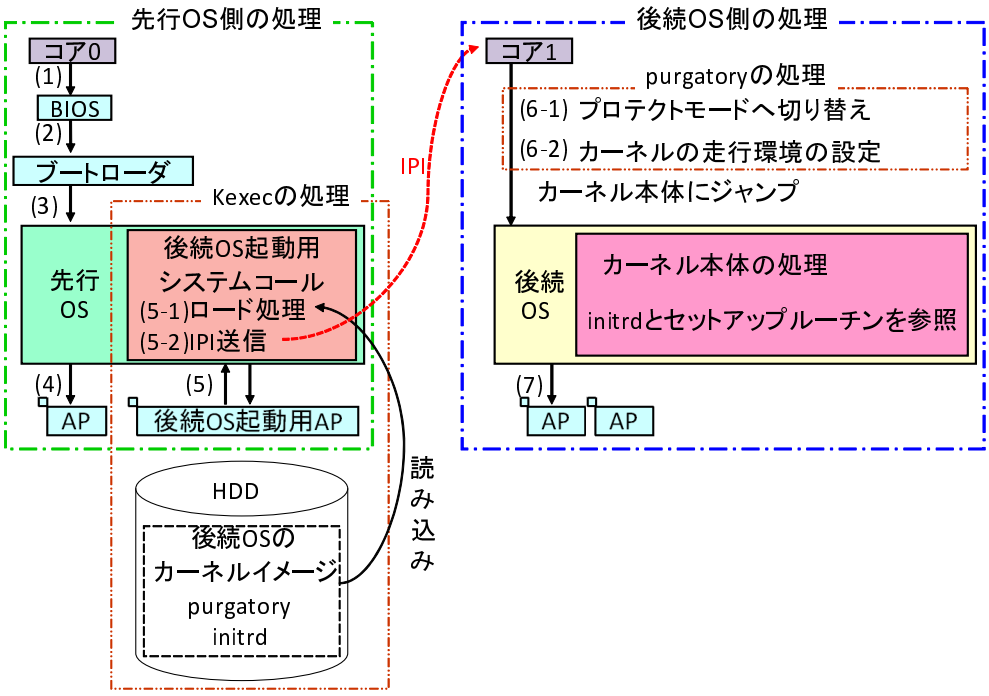

図 2.2 Mint の起動手順

(5-1) ロード処理

purgatory，セットアップルーチン，後続 OS のカーネル本体，および initrd を HDD から読み込み，先行 OS の usable 領域を避けて実メモリ上に配置する．purgatory は，Kexec 固有の前処理であり，initrd は初期 RAM ディスクである．起動直後のコアはリアルモードで purgatory の先頭アドレスから走行する．リアルモードでは，実メモリの先頭から 1MB(0x100000 番地) 以内の領域にしかアクセスできないため，purgatory の展開先は実メモリ空間の 0x100000 番地以内とする．あらかじめ，実メモリ空間に後続 OS の usable 領域を作成しておき，後続 OS のカーネルイメージと initrd は後続 OS の usable 領域に展開する．

(5-2) IPI の送信

後続 OS の起動時，未使用のコアが 1 つ以上存在することを前提としている．このコアに IPI を送信することで，このコアが起動する．この際，IPI によって purgatory の先頭アドレスの情報を送信することで，起動後のコアは，purgatory の先頭アドレスから走行する．

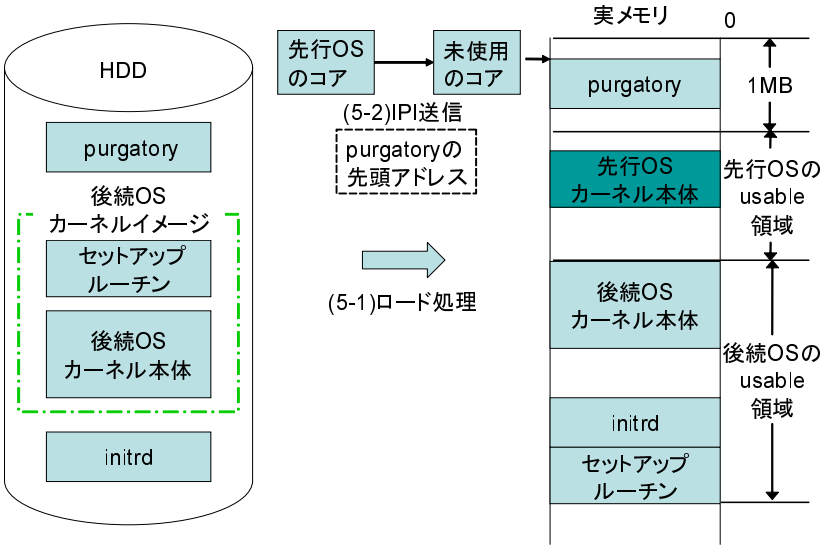

図 2.3 システムコール処理における実メモリ上の様子

以降は後続 OS 側の処理として，purgatory 内の処理が実行される．purgatory での処理 (図 2.2 の 6-1，6-2) は以下の 2 つである．

(6-1) プロテクトモードへ切り替え

リアルモードからプロテクトモードに切り替え，実メモリ空間の先頭から 1MB(0x100000 番地) 以降も扱えるようにする．

(6-2) 走行環境の設定

Interrupt Descriptor Table(以降，IDT)，Global Descriptor Table(以降，GDT)，スタック領域，および各種レジスタの設定を行う．

後続 OS を起動する際の実メモリ上の様子を図 2.4 に示し，以下で説明する．purgatory の処理が終了すると，カーネル本体にジャンプし，initrd とセットアップルーチンを参照しながら，カーネル本体が走行する．セットアップルーチンは，カーネルのリアルモードにおける初期化ルーチンであり，後続 OS の起動時は走行しない．

後続 OS は，初期化ルーチンでの処理を終えると，正常に走行を開始する．この後，図 2.2 の後続 OS 側の処理において，後続 OS の AP を起動できる．

(7) 後続 OS の AP を起動

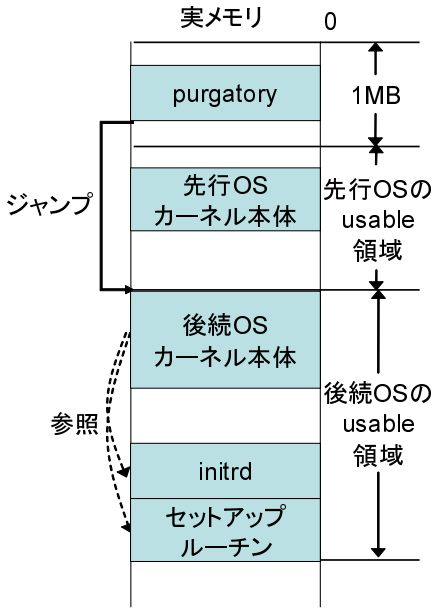

図 2.4 後続 OS を起動する際の実メモリ上の様子

2.3 後続 OS 起動における問題点

2.2 節で述べた後続 OS の起動方式において，以下に示す問題点がある．

(1) 後続 OS のカーネルイメージを HDD から読み込むための時間がかかる．

多数の OS を起動させる際，起動の度に OS のカーネルイメージを HDD から読み込むと，起動時間が I/O 処理によって左右される．このため，高速な処理が期待できないばかりか，起動時間が安定せず，システムの応答時間を保証しにくくなる．将来，メニーコア環境において多数の OS を走行させる場合に，この問題が顕著になると予想される．

(2) 空間切替処理の増加により，性能が低下する．

2.2 節で示した起動方式は，複数回にわたり先行 OS で後続 OS 起動用 AP を呼び出し，ここで後続 OS 起動用システムコールを呼び出すといった手順を介している．具体的には，HDD からの読み込みのためにシステムコールを呼び出し，メモリ上への展開，および IPI の送信のためにシステムコールを呼び出す．このように 2 段階に分けて，それぞれの段階でシステムコールを呼び出している．後続 OS の起動のために複数回にわたりカーネル空間とユーザ空間を行き来するため，空間切替処理の増加により性能が低下し，後続 OS の起動に時間を要する．

第 3 章

OS 環境の高速な複製手法

3.1　目的

2.3 節に示した問題点を解決する手法として，OS 環境の高速な複製手法を提案する．先に走行している OS を複製して，新たな OS を起動する．先に走行しており，複製対象となる OS 環境を，複製元 OS と呼ぶ．また，複製元 OS から生成した新たな OS を複製先 OS と呼ぶ．

OS 環境を複製する例を図 3.1 に示し，以下で説明する．複製元 OS は，コア 1，コア 2 を占有しており，AP1，AP2 が動作している．コア 3，コア 4 は未使用である．複製を行うと複製先 OS はコア 3，コア 4 を占有し，起動処理を行い，走行を開始する．なお，AP1，AP2 は複製せず，複製先 OS 上で AP1，AP2 は動作しない．このように，複製を行う対象は，カーネル本体のみであり，複製元 OS 上で動作している AP(プロセス) は複製の対象としない．

この手法により，期待できる効果は以下の 2 つである．

(1) HDD からの読み込み量の減少

従来の Mint では，後続 OS を起動させる際，HDD から後続 OS のカーネルイメージ，purgatory，および initrd の 3 つを読み込む必要があった．一方，OS 環境の高速な複製手法では，先行 OS のカーネル本体を複製して後続 OS とするため，HDD から後続 OS のカーネルイメージを読み込む必要がなくなる．

(2) システムコールの呼び出し回数の減少

Kexec の処理において，システムコールの呼び出し回数を減らすことができる．具体的には，HDD からの読み込みのためのシステムコールとメモリ上への展開，および IPI

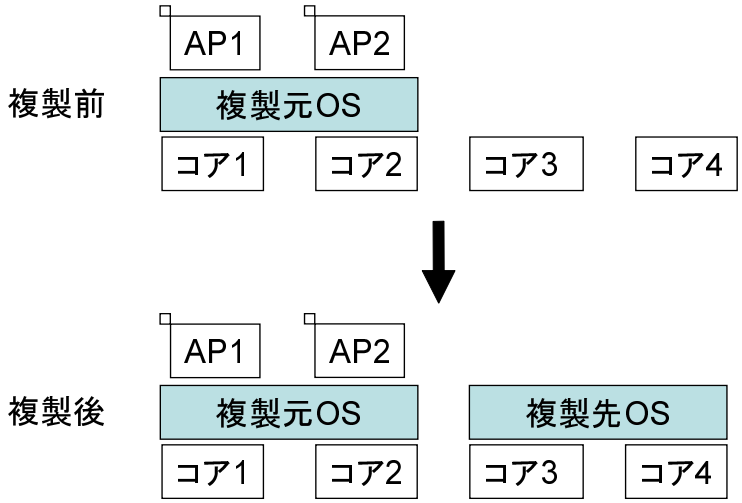

図　3.1　OS 環境を複製する例

の送信のためのシステムコールを 1 つにまとめることで，システムコールの呼び出し回数を減らす.

以上により，2.2 節で示した方法より高速に安定した速度で後続 OS を起動できる.

3.2　基本設計

先行 OS を複製元 OS として，OS 環境を複製する手順を図 3.2 に示し，以下で説明する. 複製元 OS は，2.2 節と同様に以下の 4 つの手順で起動する.

(1) BIOS の起動

(2) ブートローダの起動

(3) 複製元 OS の起動

複製元カーネルのセットアップルーチンを介して，複製元 OS を起動する.

(4) 複製元 OS の AP 起動

次に複製元 OS で，複製先 OS 起動用 AP を実行し，以下の処理を行う.

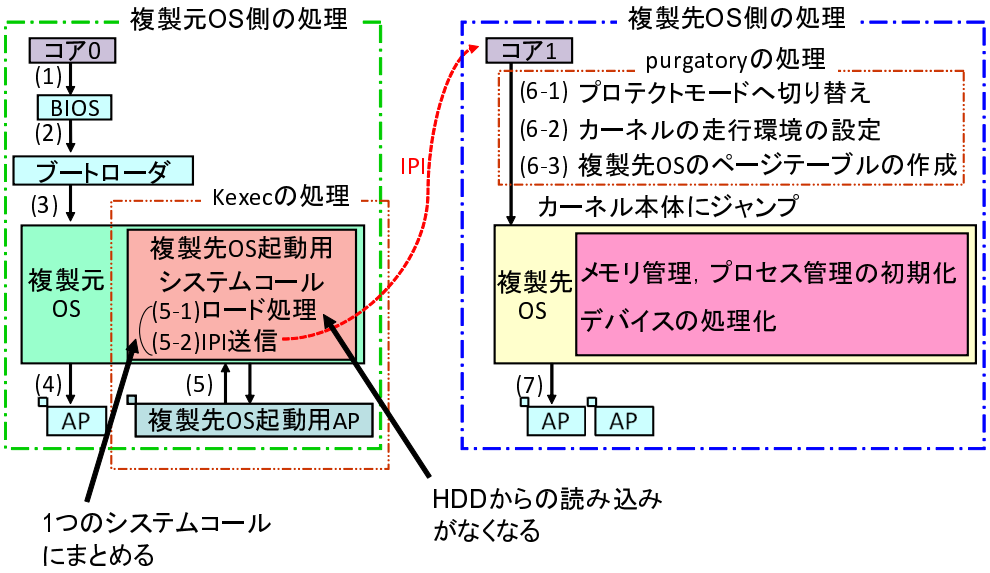

図　3.2　OS 環境の複製処理の流れ

(5-1) ロード処理

ロード処理における実メモリ上の様子を図 3.3 に示し，以下で説明する．実メモリ空間の分割方式は，先行 OS の usable 領域を複製元 OS の usable 領域，後続 OS の usable 領域を複製先 OS の usable 領域とする．2.2 節では，purgatory，カーネル本体，initrd，およびセットアップルーチンを HDD から読み込んでいた．一方ここでは，複製元 OS のカーネル本体と複製元 OS 起動時に退避しておいたセットアップルーチンを複製し，複製先 OS の usable 領域に展開する．

purgatory，および initrd は再利用可能な状態で実メモリ上に保存しておき，これを複製して利用する．purgatory は，2.2 節で示したように実メモリ空間の 0x100000 番地以内に展開する．そして，initrd は 2.2 節で示したように後続 OS の usable 領域に展開する．

(5-2) IPI の送信

IPI を送信する様子を図 3.4 に示し，以下で説明する．2.2 節と同様に，未使用のコアに IPI を送信する．そしてコア起動後，purgatory の先頭アドレスから処理を開始する．

2.2 節で示した起動手順では，上記の (5-1)，(5-2) の処理はそれぞれ別のシステムコールで実行していた．本方式では，(5-1)，(5-2) の処理を 1 つのシステムコールで実行する．

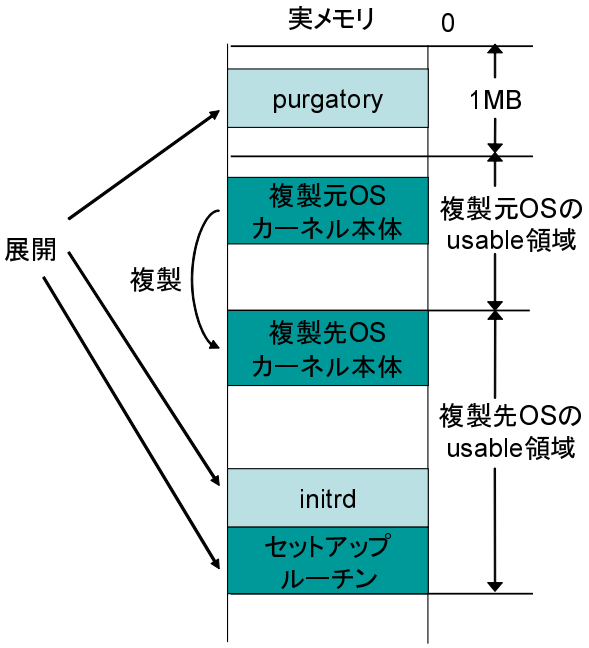

図　3.3　ロード処理における実メモリ上の様子

図 3.2 の複製先 OS の処理内の purgatory 処理において，複製先 OS の走行環境を設定する．プロテクトモードへの切り替え，および走行環境の設定は 2.2 節と同様である．この処理に加え，カーネルで処理していたページテーブルの作成を purgatory 処理において実行する．具体的な処理は，以下の 3 つである．

(6-1) プロテクトモードへ切り替え

2.2 節と同様に，リアルモードからプロテクトモードへ切り替え，実メモリ空間の 0x100000 番地以降も扱えるようにする．

(6-2) 走行環境の設定

2.2 節と同様に，IDT，GDT，スタック領域，および各種レジスタの設定を行う．

(6-3) 複製先 OS のページテーブルの作成

複製先 OS を正常に走行させるため，複製先 OS のページテーブルの作成を行う．この処理は，2.2 節では，カーネルの初期化処理で行っている．しかし，カーネル内でページテーブルの作成を行う場合，ブートオプションを使用し，複製元 OS のページテーブルか複製先 OS のページテーブルかを分岐する必要があるので，実装が難しくなることから purgatory 内でページテーブルの作成を行うことにする．

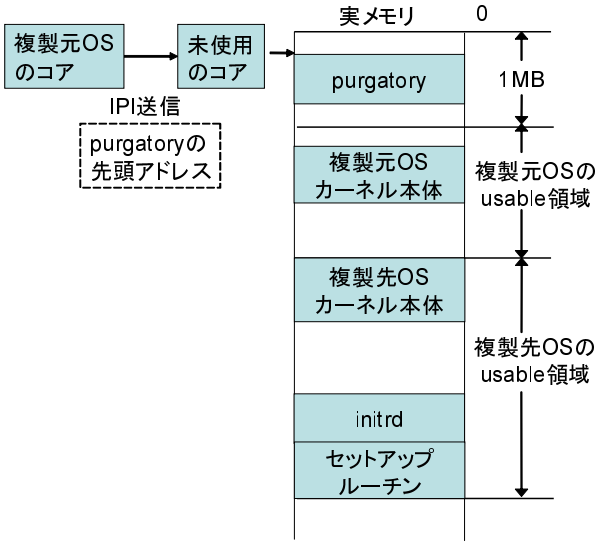

図　3.4　IPI を送信する様子

purgatory での処理を終えると複製先 OS のカーネル起動処理の途中へジャンプする．2.2 節では，カーネル本体の先頭へジャンプしていた．実メモリ上において，複製先 OS のカーネルの途中へジャンプし，カーネルで処理を行う様子を図 3.5 に示し，以下で説明する．カーネル処理開始直後は，スタック領域の設定，各種レジスタの設定，およびページテーブルの作成を行っている．これらの処理は，既に Kexec の purgatory で行うため，複製先カーネルの先頭付近の処理を省略する．複製先カーネルにおいて，BSS(Block Started by Symbol) 領域のクリア，初期化処理のためのメモリの予約，および各種初期化処理を行っている箇所から処理を開始する．複製先カーネル内では，initrd とセットアップルーチンを参照しながら処理を行う．

図 3.2 の複製先 OS 側の処理において，メモリ管理の初期化やプロセス管理の初期化，デバイスの初期化，および割り込みの初期化を行う．複製先 OS は，初期化ルーチンでの処理を終えると，正常に走行を開始する．この後，図 3.2 の後続 OS 側の処理において，複製先 OS の AP を起動できる．

(7) 複製先 OS の AP を起動

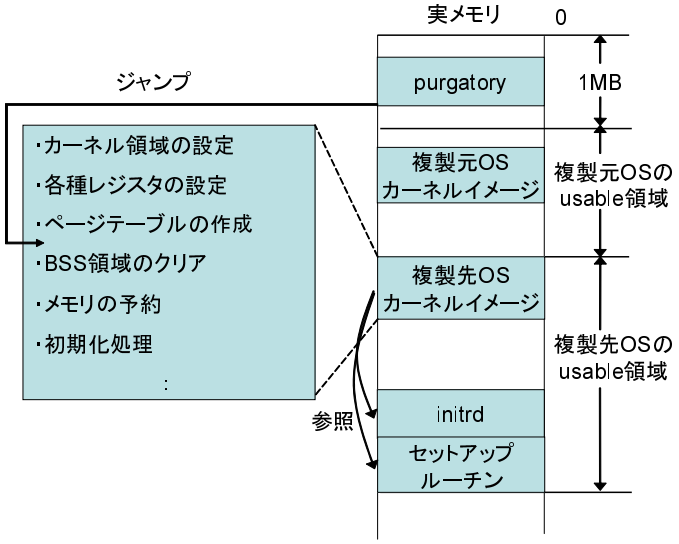

図 3.5 複製先 OS のカーネルの途中へジャンプし，カーネルで処理を行う様子

3.3 課題

3.3.1 複製したカーネルの初期化ルーチンを変更するための課題

複製元カーネルと複製先カーネルでは，カーネルを配置するアドレス，占有するコア，および占有するデバイスに違いがある．このため，複製元カーネルと全く同じコードを複製しても，複製先カーネルは正常に走行できない．これらの違いは，Mint における先行 OS と後続 OS の違いに類似しており，Mint では，これらの違いを Kexec と初期化処理に集中させている [6]．解析の結果，初期化ルーチンにおいて，具体的に変更しなければならないのは，以下の 4 項目であると分かった．

(課題 1) ページテーブルの初期化

複製元 OS と複製先 OS で同一のメモリマッピングを利用した場合に発生する問題を図 3.6 に示し，以下で説明する．

複製元カーネルは，実メモリ空間の先頭から使用可能な領域を仮想メモリ空間の 0xc0000000 番地 (3GB) 以降の領域にストレートマッピングする．ページテーブルの初期化処理を変更せずカーネルを複製すると，複製先カーネルも同様に，実メモリ空間の先頭から使用可能な領域を仮想メモリ空間の 0xc0000000 番地 (3GB) 以降の領域にストレートマッピングする．複製元カーネルが実メモリ空間の X 番地以降に配置されており，複製先カーネルが実メモリ空間の Y 番地以降に配置されているとする．すると，複製元カー

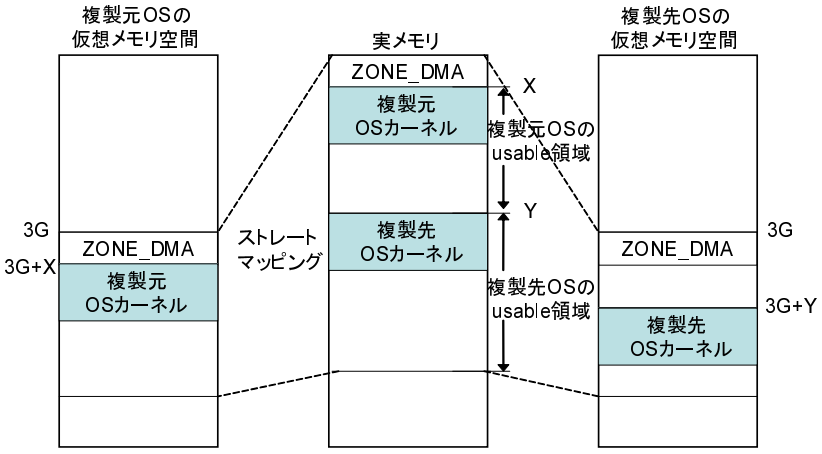

図　3.6　複製元 OS のメモリマッピングをそのまま利用した場合の問題

ネルは仮想メモリ空間の 3GB+X 番地以降にマッピングされ，複製先カーネルは仮想メモリ空間の 3GB+Y 番地以降にマッピングされる．複製元カーネルは，コンパイル時にアドレス解決され，複製元カーネルは 3GB+X 番地以降で走行することを前提としているため，複製先カーネルにおいても，マッピングされる仮想アドレスは 3GB+X 番地以降でなければならない．このため，複製先カーネルがマッピングされる仮想アドレスが 3GB+X 番地以降になるように，複製元カーネルとは異なるマッピングを与えなければならない．

また，カーネルを正常に走行させるため，以下の 4 つを考慮しなければならない．

(A) 複製先 OS は，実メモリ空間の先頭から 16MB をマッピングしなければならない．

実メモリ空間の先頭から 1MB はコアを起動させるために必要な領域である．また，実メモリ空間の先頭から 16MB(0x1000000 番地) は，ZONE_DMA と呼ばれる領域であり，古い ISA のデバイスが DMA 転送を使用する際に必要なページテーブルが格納されている．このため，全ての OS は実メモリ空間の先頭から 16MB の領域をマッピングする．

(B) 複製先カーネルは仮想メモリ空間において，ZONE_DMA の直下に配置しなければならない．

通常の Linux カーネルは，実メモリ空間上において，ZONE_DMA 直後にカーネルを配置しており，かつ，実メモリをストレートマッピングでマッピングされている．その結果，複製元 OS では，仮想空間上においても ZONE_DMA の直下にカーネルが位置している．この ZONE_DMA とカーネルとの配置関係を前提とし

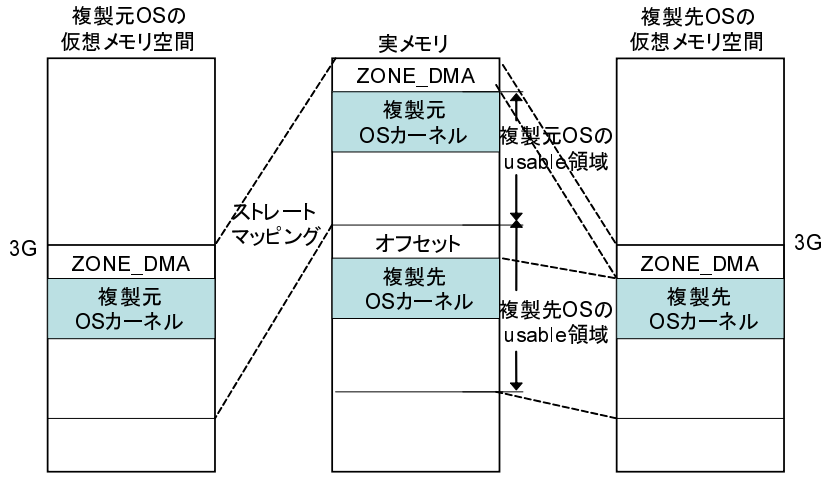

図 3.7 変更した複製元 OS と複製先 OS のマッピング

たメモリ操作がカーネル内に存在することを考慮しなければならない.

(C) 複製先カーネルを実メモリ空間上で usable 領域の先頭から 16MB 後方に配置しなければならない.

16MB のオフセットを前提としたメモリ操作がカーネル内に存在することを考慮しなければならない.

(D) 複製先 OS は，複製元 OS の usable 領域を避けてマッピングしなければならない.

図 3.6 は，複製元 OS，および複製先 OS は全ての実メモリ空間をストレートマッピングしている．複製元 OS が使用する実メモリ空間と複製先 OS が使用する実メモリ空間は互いに他の OS は使用できないようにする必要がある．このため，それぞれの OS は自分の usable 領域と ZONE_DMA のみマッピングしなければならない.

以上のことを考慮すると，複製元 OS と複製先 OS のメモリマッピングは図 3.7 のように変更しなければならない．図 3.7 では，複製元 OS と複製先 OS が走行する仮想アドレスは同じであり，先ほど述べた以下の 4 つの点が考慮されている.

(A) 複製先 OS は，実メモリ空間の先頭から 16MB をマッピングしている.

(B) 複製先カーネルは仮想メモリ空間において，ZONE_DMA の直下に配置している.

(C) 複製先カーネルを実メモリ空間上で usable 領域の先頭から 16MB 後方に配置している.

(D) 複製先 OS は，複製元 OS の usable 領域を避けてマッピングしている．

なお，2.2 節で述べた従来の Mint の起動方式では，後続 OS のアドレス解決を行うのは後続 OS 起動時である．このため，先行 OS と後続 OS は走行する実アドレス，仮想アドレスが異なり，ページテーブルの初期化処理の変更は必要ない．

また，カーネル内では，仮想アドレスと実アドレスの相互変換を随所で行っている．この相互変換処理は，前述のストレートマッピングを前提とした実装になっている．したがって，カーネル内の相互変換処理ルーチンも変更を必要とする．

(課題 2) コアの走行環境の再現

走行中の複製元 OS はプロテクトモードで走行しており，このままではコア起動直後のカーネル起動処理を実行することができない．このため，新たに起動させたコアにおいて，コアの走行環境を再現する必要がある．

(課題 3) デバイスの初期化

複製元 OS はディスプレイやキーボードといったいくつかのデバイスを占有している．複製元 OS を複製し，複製先 OS を起動させると，複製先 OS は複製元 OS と同じデバイスを占有しようとする．しかし，Mint では，1 つのデバイスは 1 つの OS しか占有できない．正常に複製元 OS と複製先 OS を走行させるには，それぞれの OS がどのようにデバイスを占有するか検討する必要がある．

(課題 4) その他の初期化関数とデータの保護

カーネル起動時に初期化処理を行う関数とデータは，初期化処理が終わると消滅してしまうものがある．OS 環境を複製する際，既に初期化処理を行う関数やデータがなくなっていると，複製先 OS を正常に起動できないため，初期化処理を行う関数やデータの消失を防ぐ必要がある．

3.3.2　OS を複製するための課題

OS を複製するためには，以下の項目を検討しなければならない．

(課題 5) カーネルを含んだ実メモリ領域を指定の領域へ複製

走行中のカーネルイメージを指定の実アドレスに複製しなければならない．また，複製元 OS が使用する実メモリ空間と複製先 OS が使用する実メモリ空間は互いに他の OS は使用できないようにする必要がある．

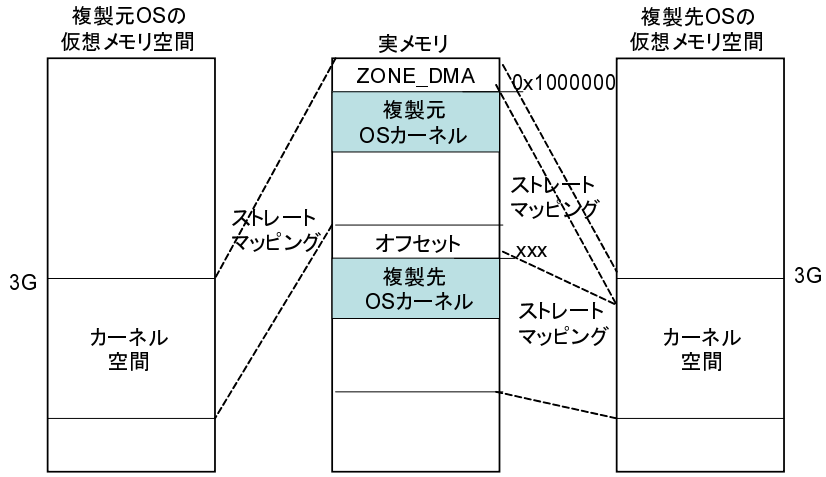

図　3.8　複製元 OS と複製先 OS のマッピング

3.4　対処

3.4.1　複製したカーネルの初期化ルーチンを変更するための対処

(対処 1)　ページテーブルの初期化

(課題 1) で述べたように，すでに走行している複製元カーネルはコンパイル時にアドレス解決がされており，走行する仮想アドレスが決定している．このため，複製先カーネルを走行させるためには，複製元カーネルと複製先カーネルをそれぞれの仮想メモリ空間の同じ仮想アドレスにマッピングする必要がある．この様子を図 3.8 に示し，以下で説明する．

複製先カーネルは，複製先 OS の usable 領域の先頭アドレスの下位 16MB(xxx 番地) から展開しているとする．マッピング先を算出する際に，xxx を足すことでマッピングする実アドレスを xxx だけずらすことができる．また，ストレートマッピングを行う実メモリ空間上の終端アドレスにも xxx を足しておく．

仮想アドレスから実アドレスに変換を行う処理は，pa と呼ばれる関数 (マクロ) であり，実アドレスから仮想アドレスに変換を行う処理は，va と呼ばれる関数 (マクロ) である．カーネル内での相互変換処理は，この 2 つの関数を通して行うように統一されている．そこで pa，va を図 3.8 のマッピングテーブルに則した処理内容に変換する．pa，va による相互変換の様子を図 3.9 に示し，以下で説明する．

複製元カーネルでは，pa は仮想アドレスから 3GB を引くことで実アドレスに，va は

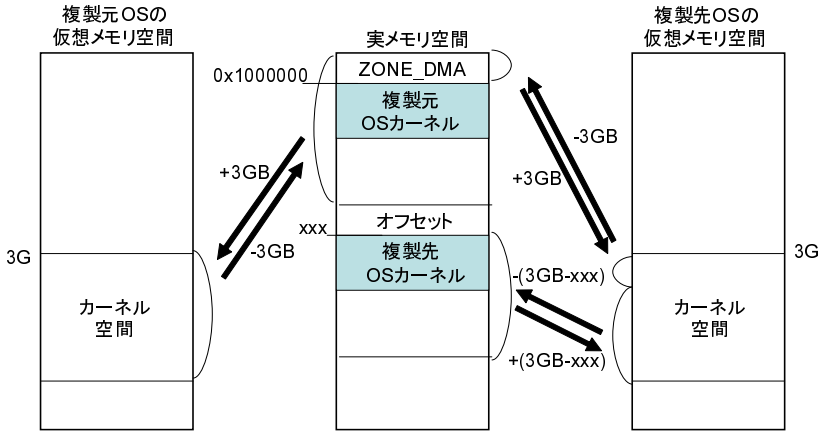

図 3.9 マクロの変換

実アドレスに 3GB を足すことで仮想アドレスに変換している．複製先カーネルを実アドレスの xxx 番地に展開した場合，ストレートマッピングを行う仮想アドレスと実アドレスの差は 3GB−xxx となる．このため，pa は仮想アドレスから 3GB−xxx を引くように，va は実アドレスに 3GB−xxx を足すように変更する．また，実メモリ空間の先頭から 16MB は変更を行わず，pa は仮想アドレスから 3GB を引き，va は実アドレスに 3GB を足す．

(対処 2) コアの走行環境の再現

複製先 OS を走行させるには，CPU の走行モードを変更しなければならない．コアは，コア起動直後はリアルモードで走行しており，実メモリ空間の先頭から 1MB 以内でしか動作できない．この制限を取り除くため，CPU の走行モードをリアルモードからプロテクトモードへ移行させる．また，コアの走行に必要な IDT，GDT，スタック領域，および各種レジスタの初期化を行う．これらの処理は，Mint で行う処理と同様である．

(対処 3) デバイスの初期化

カーネルのブートオプションにおいて複製元 OS か複製先 OS かを示すパラメータを作成する．カーネルの起動処理内でこのパラメータを用いて条件分岐を行い，それぞれの OS で占有させるデバイス処理を実行させる．

(対処 4) その他の初期化関数とデータの保護

Linux では，初期化処理のみに使用される関数は，実行後にその役目を終えると，テキスト領域を含めて解放されてしまう．これは，メモリの節約のためと考えられる．こ

れらの関数が解放されることを防ぐため，複製元 OS の起動処理において，メモリ領域を解放する処理を省略する．

初期化処理時に使用するデータは，先行 OS の起動処理中にあらかじめ退避しておき，必要に応じて保存したデータを使用する．

3.4.2　OS の複製の対処

走行中のカーネルを複製し，実メモリ上に展開したカーネルを走行させるため，以下の対処を行う．

(対処 5) カーネルを含んだ実メモリ領域を指定の領域へ複製

複製元 OS が複製先 OS のカーネルが展開される領域を使用すると，複製元 OS と複製先 OS の走行に影響を与える可能性がある．このため，複製元 OS を起動させる際，あらかじめ複製先 OS が使用する領域を reserved にしておく．そして，複製元 OS のカーネル本体，再利用可能な状態で保存しておいたセットアップルーチン，purgatory，および initrd を複製先 OS の usable 領域に複製する．

なお，セットアップルーチン，purgatory，および initrd を再利用可能な状態で保存する手法は未検討であるため，以降では，カーネル本体，セットアップルーチンのみを複製の対象とし，purgatory，および initrd は従来手法のままとする．

第 4 章

実装

4.1 ページテーブルの初期化に関する実装

複製先カーネルのページテーブルの初期化処理を実装した．しかし，カーネルの複製処理は実装できていない．このため，実験的に通常の Linux において，カーネルを配置する実アドレスを変更して実メモリ空間と仮想メモリ空間において複製先カーネルの状況を再現した．

通常の Linux のマッピングの様子を図 4.1 に，変更後の Linux のマッピングの様子を図 4.2 に示し，以下で説明する．実メモリ空間の先頭から 16MB(0x1000000 番地) は ZONE_DMA であり，通常の Linux では ZONE_DMA 以降にカーネルを配置する．そして，実メモリ空間の先頭から 896MB の領域を仮想メモリ空間の 3GB(0xc0000000 番地) 以降の領域にストレートマッピングする．変更後の Linux ではカーネルを実メモリ空間の xxx 番地以降に配置し，任意の xxx 番地以降の領域を仮想メモリ空間の 0xc1000000 番地以降の領域にストレートマッピングする．実メモリ空間の先頭から 16MB(0x1000000 番地) の領域は変更せず，仮想アドレスの 3GB(0xc0000000 番地) 以降の領域にストレートマッピングする．また，仮想アドレスと実アドレスの相互変換処理は，pa，va に集約されていることが分かったので，その箇所を変更した．

上記の方法で，実メモリ空間と仮想メモリ空間において複製先カーネルと同様の状況を再現し，複製先カーネルのページテーブルの初期化処理を実装した．表 4.1 に，変更したファイルとそれぞれのコード変更量，および hunk 数を示す．

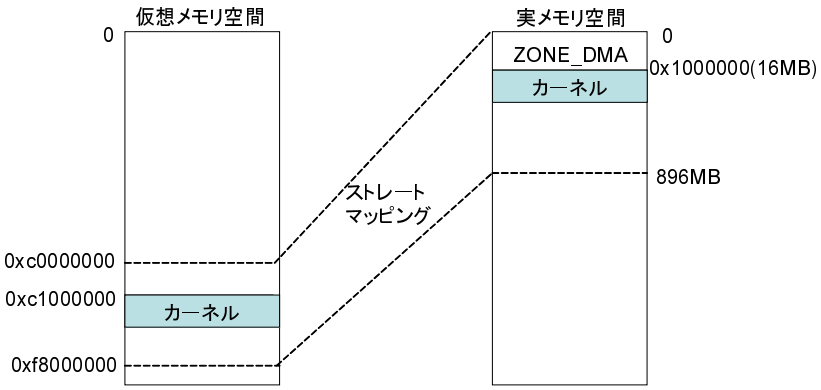

図 4.1 通常の Linux のマッピングの様子

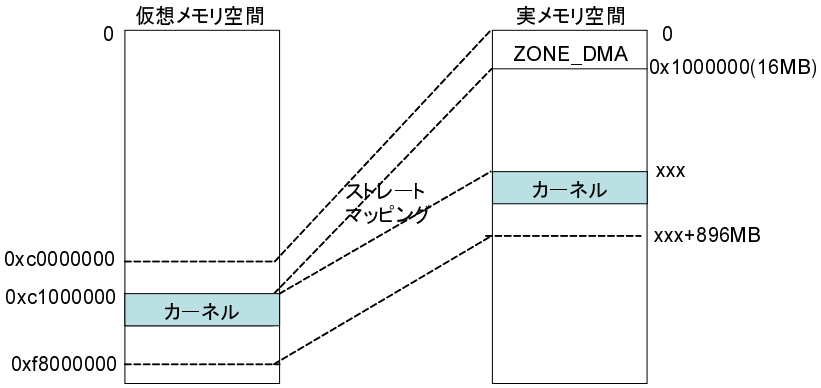

図 4.2 変更後の Linux のマッピングの様子

表 4.1 コード変更量

	コード行数	hunk 数
全体	46	6
arch/x86/kernel/head_32.S	40	2
arch/x86/configs/i386_defconfig	2	2
arch/x86/include/asm/page_32.h	2	1
arch/x86/include/asm/page.h	2	1

第 5 章

おわりに

本論文では，Mint における OS 環境の高速な複製手法を設計し，一部の内容を実装した結果を述べた．具体的には，まず，Mint の構成と起動方式を述べ，Mint の起動方式の問題点を示した．次に，この問題点の解決案として OS 環境の高速な複製手法を提案した．OS 環境の高速な複製手法の基本設計とこれを実現するための課題を述べ，その対処を述べた．また，複製先 OS の初期化ルーチンにおいてページテーブルの初期化を実装した．

今後の課題として，初期化ルーチンの変更と OS の複製の実装と 4.1 節で述べた実装内容を purgatory に移行がある．また，OS 環境を全て複製するのでなく，オンデマンドページングやコピーオンライトといった機能を使うことで複製先 OS の走行に必要な箇所のみ複製を行い，複製量を低減させる方式の設計がある．

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祐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] Shekhar Borkar, “ Thousand Core Chips - A Technology Perspective ”, In Proceedings of Annual ACM/IEEE Design Automation Conference, pp.746-749 (2007).

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

[3] 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 symposium on Operating Systems Principles，pages 164-177，October 2003.

[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).

[5] Andy Pfiffer, “ Reducing System Reboot Time with kexec, ” http://www.osdl.org/, 2003.

[6] 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).