特 別 研 究 報 告 書

題 目

Kexecを用いたTwinOS起動方式の検討

研究室担当教員の署名

提 出 者

中原 大貴

岡山大学工学部 情報工学科

平成22年2月5日 提出

要約

計算機の性能が向上し，1台の計算機上に複数のOSを走行させ多様なサービスを実現する機能が注目されている．このような技術として，１台の計算機上で複数のOSを同時に走行させるTwinOSが研究されている．TwinOSでは，メモリ分割機能や共存OSメモリ展開機能など様々な機能を実装し，2つのLinuxの同時走行を実現している．

現在のTwinOSには，共存OSの起動処理が複雑であるという問題がある．この問題への対処として，Kexecを用いて共存OS起動処理を簡略化し汎用化する方式を提案する．Kexecは，カーネルの高速な再起動を実現する機能であり，Linux Kernelに標準搭載されている．具体的には，カーネルをメモリに展開し，カーネル起動処理の先頭アドレスにジャンプすることで高速な再起動を実現する．Kexecを用いることで，セットアップルーチンを走行せずにカーネルを起動できる．このため，セットアップルーチンを走行するために複雑な処理を行う共存OS起動処理を簡略化できる．また，KexecはLinux Kernelにより標準搭載されているため，今後もカーネルのバージョンアップに対応してKexecもバージョンアップされる．このため，Kexecを共存OS起動処理に用いることは，共存OS起動処理の汎用化に有用である．しかし，Kexecは本来カーネルを再起動する機能であり，共存OS起動処理にそのまま用いることはできない．このため，共存OS起動処理を実現するための機能をKexecに追加する必要がある．

本論文では，KexecのTwinOSへの応用方法を検討し，一部の機能を実装した結果を述べる．まず，現在のTwinOSの問題点を述べ，Kexecを用いた共存OS起動処理の目的について述べる．次に，Kexecの概要を述べ，Kexecを用いて共存OS起動処理を行う際の要求について述べ，この要求への対処方法について説明する．

目次

図 目 次

表 目 次

第 1 章

はじめに

計算機の性能が向上し，1 台の計算機上に複数の OS を走行させ多様なサービスを実現する機能が注目されている．このような技術として，1 台の計算機上で複数の OS を同時に走行させる TwinOS が研究されている．TwinOS では，メモリ分割機能や共存 OS メモリ展開機能など様々な機能を実装し，2 つの Linux の同時走行を実現している．

現在の TwinOS には，共存 OS の起動処理が複雑であるという問題がある．この問題への対処として，Kexec を用いて共存 OS 起動処理を簡略化し汎用化する方式を提案する．Kexec は，カーネルの高速な再起動を実現する機能であり，Linux Kernel に標準搭載されている．具体的には，カーネルをメモリに展開し，カーネル起動処理の先頭アドレスにジャンプすることで高速な再起動を実現する．Kexec を用いることで，セットアップルーチンを走行せずにカーネルを起動できる．このため，セットアップルーチンを走行するために複雑な処理を行う共存 OS 起動処理を簡略化できる．また，Kexec は Linux Kernel により標準搭載されているため，今後もカーネルのバージョンアップに対応して Kexec もバージョンアップされる．このため，Kexec を共存 OS 起動処理に用いることは，共存 OS 起動処理の汎用化に有用である．しかし，Kexec は本来カーネルを再起動する機能であり，共存 OS 起動処理にそのまま用いることはできない．このため，共存 OS 起動処理を実現するための機能を Kexec に追加する必要がある．

本論文では，Kexec の TwinOS への応用方法を検討した結果を述べる．

第 2 章

TwinOS

2.1 特徴

TwinOSは，1台の計算機上で2つのLinuxを独立に走行させる方式であり，以下の特徴を持つ．

(1) 両OSとも相互に処理負荷の影響を与えない

(2) 両OSとも入出力性能を十分に利用できる

このため，1台の計算機上で，CPU，メモリ，および入出力機器といったハードウェア資源を効果的に共有，または占有する必要がある．2つのOSの独立性を保つためには，共有するハードウェア資源を最小限とすることが有効であるため，CPUのみを共有する．CPU以外のハードウェアは分割し，それぞれを各OSで占有する． 図2.1にTwinOSの構成を示し，各ハードウェアの分割と共有方法を以下に示す．

(1) CPU

CPUは時分割で制御する．具体的には，起動後はタイマ割り込みを利用してOSを切替える．

(2) メモリ

上位と下位に2分割する．具体的には，最初に起動するOS(以降，先行OSと呼ぶ) にメモリの老番アドレスを，先行OSから起動するOS(以降，共存OSと呼ぶ) に若番アドレスを割り当てる．

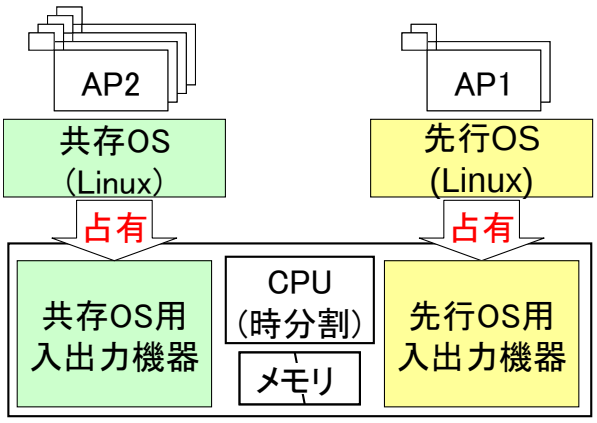

図 2.1 TwinOS の構成

(3) 入出力機器

OS ごとに指定された入出力機器のみを占有制御する．現在走行していない OS への割り込みが発生した場合，OS を切り替えた後に割り込み処理を行う．

また，各構成の実現においては，各 OS の改修量を最小化している．これにより，開発工数の削減だけではなく，他の OS への本手法適用の際の工数削減を考慮している．

2.2 起動方式

TwinOS の起動方式について以下で詳細を述べる．TwinOS は以下の順番で起動する．

(1) 先行 OS を起動

先行 OS と共存 OS は，通常の Linux と構成に関して以下の 2 点が異なっている．

(A) メモリの老番もしくは若番アドレスの部分のみを利用する

(B) あらかじめ指定された I/O 機器のみを認識する

このように改変されている先行 OS を通常の Linux 起動方式で起動させる．

(2) 共存 OS のカーネルをメモリ上に展開

先行 OS の走行環境を保存し，圧縮された共存 OS のカーネルをメモリ上に展開する．その後，共存 OS セットアップルーチンを起動し，圧縮カーネル展開ルー

チンにより，メモリ上の圧縮された共存OSカーネルをメモリの若番アドレス上に展開する．展開後，先行OSに処理を戻し，走行環境を復元する．

(3) 共存OSの起動

共存OSの起動時には，OSの初期化処理により先行OSの走行環境が壊されないように保護する必要がある．このため，共存OSの起動時に，先行OSは共存OSの起動処理を監視する．先行OSから共存OSの処理を監視することで，先行OSの環境を破壊しないように保護している．

(4) 共存状態へ移行

先行OSと共存OSが独立動作し，共存状態になる．

2.3 現在の共存OS起動処理における問題

現在のTwinOSには，共存OS起動処理において以下の2つの問題がある．

(問題1) 圧縮カーネル展開ルーチン実行前に，カーネルのセットアップルーチンを走行する．セットアップルーチンはリアルモードであるため，セットアップルーチン走行前にプロテクトモードからリアルモードへの切り替え処理が必要になる．リアルモードへの切り替え処理は，リアルモード用の走行環境を構築する必要があり，処理が複雑である．

(問題2) 起動処理の大部分を独自に追加したシステムコールにより行っている．つまり，カーネルのソースコードを大量に書き換えており，カーネルのバージョンへの依存性が高い．このため，TwinOSのカーネルのバージョンアップを行おうとすると，多くのソースコードを書きなおす必要があり，工数が非常に多くなる．

このため，簡略化と汎用化が必要となる．

第 3 章

Kexecを用いた共存OS起動処理

3.1 目的

本研究の目的は，共存OS起動処理を実現するための機能を追加したKexecを用いることで，共存OS起動処理を簡略化し，汎用化することである．

Kexecは，カーネルの高速な再起動を実現する機能であり，Linux Kernelに標準搭載されている．Kexecを用いて再起動を行うと，再起動時にセットアップルーチンを走行しない．このため，リアルモード用の走行環境を構築する必要がなくなり，2.3節で示した(問題1)に対処できる．また，Kexecは，Linux Kernelにより標準搭載されているため，今後もカーネルのバージョンアップに対応してKexecもバージョンアップされる．このため，Kexecを共存OS起動処理に用いることは，共存OS起動処理の汎用化に有用であり，2.3節で示した(問題2)に対処できる．

しかし，Kexecは，本来カーネルを再起動する機能であり，共存OS起動処理にそのまま用いることはできない．このため，共存OS起動処理を実現するための機能をKexecに追加する必要がある．

3.2 Kexec

3.2.1 特徴

Kexecとは，Linux Kernelの高速な再起動を実現する機能である．カーネルをメモリに展開し，カーネル起動処理の先頭アドレスにジャンプすることでBIOSやブー

トローダを通らずに高速な再起動を実現する．Kexecの再起動処理を利用するには，kexec-toolsというアプリケーションを用いる．カーネル内のKexecに関する機能をkexec-toolsで操作することにより，Kexecを使用できる．Kexecとカーネル内の機能について以下で説明する．

カーネル読み込み機能

カーネル読み込み機能は，kexec-toolsにおいてloadオプションをつけて実行することにより開始される．kexec-toolsは，指定されたカーネルイメージを読み込み，セグメントと呼ばれる構造体に格納する．また，起動時に必要となるinitrdのデータ，ブートパラメータの情報，およびカーネル起動時の前処理を行う部分を格納したpurgatoryをセグメントとして作成する．その後，kexec_loadシステムコールを発行することで，kexec-toolsにより指定されたメモリアドレスにセグメントを展開するためのセグメント展開マップをカーネル内で作成する．

カーネル起動機能

カーネル起動機能は，kexec-toolsにおいてexecオプションをつけて実行することにより開始される．kexec-toolsは，rebootシステムコールを発行し，kexec-toolsにより指定されたメモリアドレスにセグメントを展開する．セグメントの展開後，purgatoryのアドレスにジャンプすることで高速な再起動を実現している．

3.2.2 処理流れ

カーネル読み込み機能を実行した場合のフローチャートを 図3.1 に示し，以下で説明する．

カーネル読み込み機能の処理流れ

(1) カーネルイメージ読み込み

引数で指定されたカーネルイメージを読み込む．

(2) 各セグメントの作成

各データを管理するために，セグメントと呼ばれる構造体を作成する．bzImage形式のカーネルイメージを展開する場合のセグメントの内容を 図3.2 に示す．各セグメントは，buf，bufsz，mem，およびmemszというメンバを持つ．セグメントのメンバと役割の関係を 表3.1 に示す．

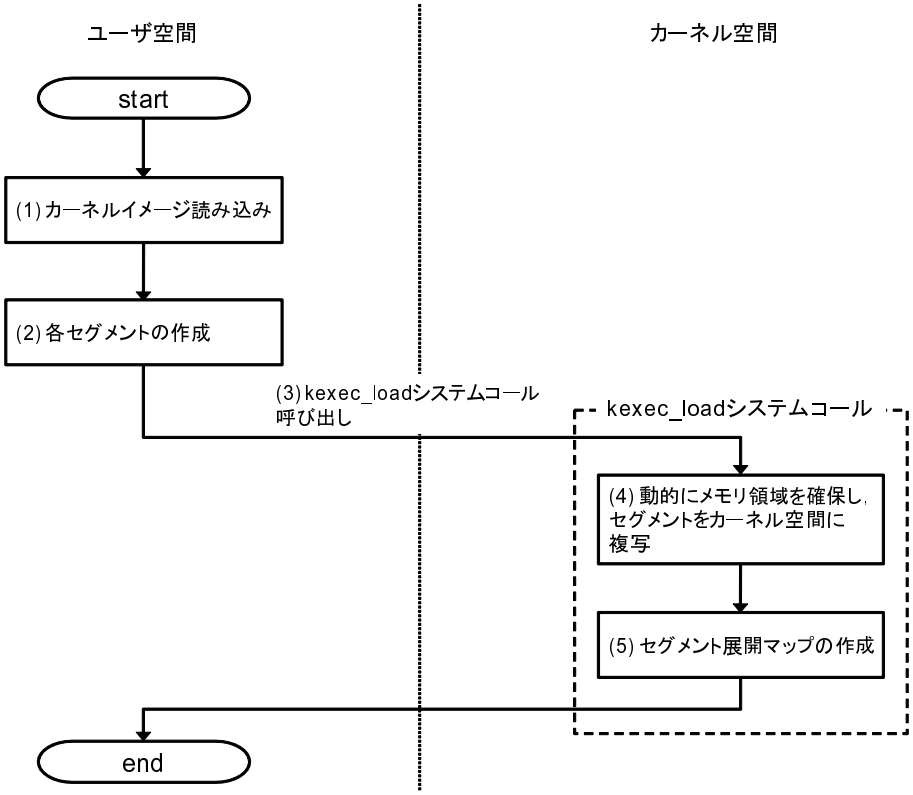

図 3.1 カーネル読み込み機能のフローチャート

(3) kexec_load システムコール呼び出し

セグメントを展開するためのセグメントマップをカーネル内で作成する kexec_load システムコールを呼び出す．ここから先はカーネル内部での処理となる．

(4) 動的なメモリ領域の確保とセグメントのカーネル空間への複写

カーネルを管理するための構造体を作成し，ユーザ空間からカーネル空間にセグメントを複写する．

(5) セグメント展開マップの作成

セグメント展開マップは，カーネル起動機能実行時にセグメントを展開するために参照されるマップである．セグメント展開マップの作成方法を 図3.3 に示す．セグメント展開マップは，(4) で作成した構造体の entry メンバが示す領域に，Src エントリと Dest エントリを追加することで作成される．セグメントの

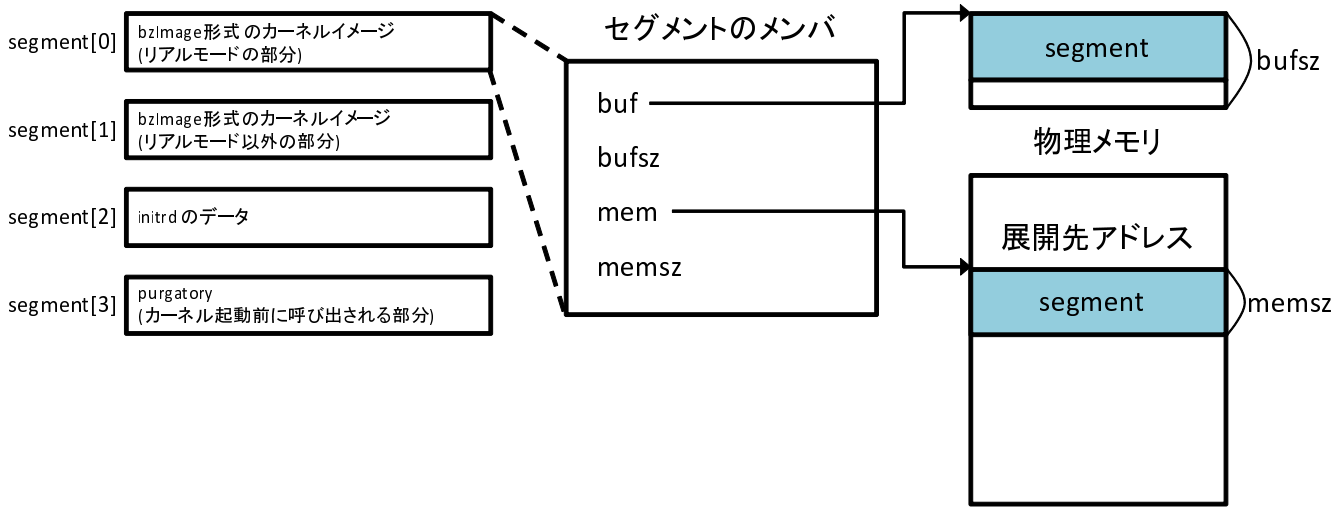

図 3.2 セグメントの内容

表 3.1 セグメントのメンバと役割の関係

メンバ	役割
buf	kexec-tools で確保したデータを一時的に格納したメモリ領域へのポインタ
bufsz	buf のサイズ
mem	最終的にデータを展開するメモリ領域へのポインタ
memsz	mem のサイズ

展開先のページの先頭アドレスを Dest エントリとして追加し，セグメントの内容が格納されたページの先頭アドレスを Src エントリとして追加する．Src エントリのページと Dest エントリのページはセグメント展開処理時に入れ替えられる．これにより，目標のメモリ領域にセグメントを展開するように処理をする．セグメントの mem をページサイズでアライメントし，Dest エントリとして entry メンバが示す領域へポインタとして追加する．Dest エントリを追加する場合は，Dest エントリであるという目印として，アライメントされたページのアドレスに 1 を加える． 図 3.3 の例では，mem が 0xNNNN1abc であり，このアドレスからセグメントを展開する．0xNNNN1abc をページサイズでアライメントし，1 を加えることで，Dest エントリは 0xNNNN1001 となる．その後，以下の (処理

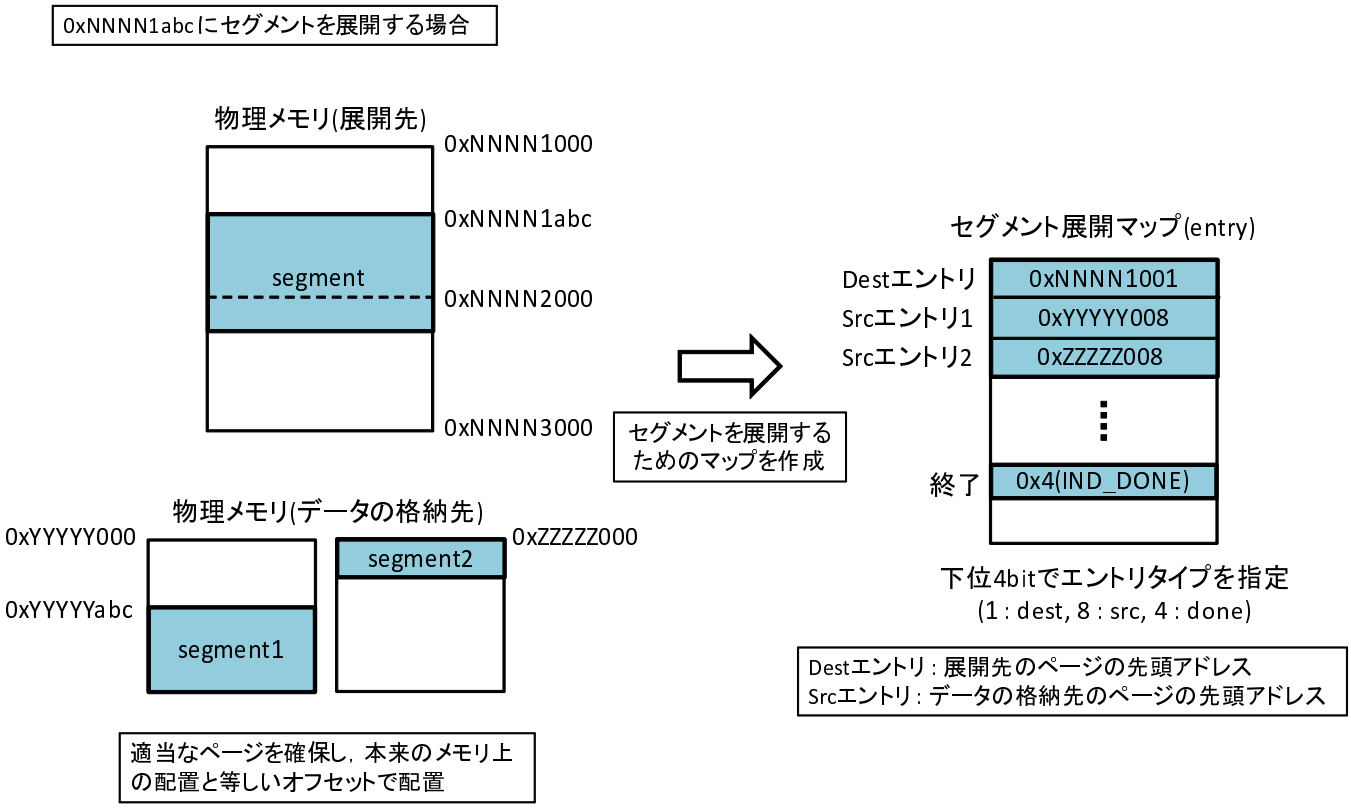

図 3.3 セグメント展開マップの作成方法

1) から (処理 3) を 1 ページずつ行う．この作業は，総ページサイズがセグメントの memsz より大きくなるまで繰り返し行う．

(処理 1) セグメントの展開先アドレスと等しいアドレスのページを確保する．確保できなければ適当なアドレスのページを確保する．

(処理 2) (処理 1) で取得したページのポインタを Src エントリとして entry メンバが示す領域へ追加する．Src エントリを追加する場合は，Src エントリであるという目印として，ページのアドレスに 8 を加える．図の例では，新しく確保したページのアドレスに 8 を加えることで，Src エントリは，0xYYYYY008 と 0xZZZZZ008 になる．

(処理 3) (処理 1) で取得したページに本来のメモリ上の配置と等しいオフセットで，セグメントの buf の内容を複写する．

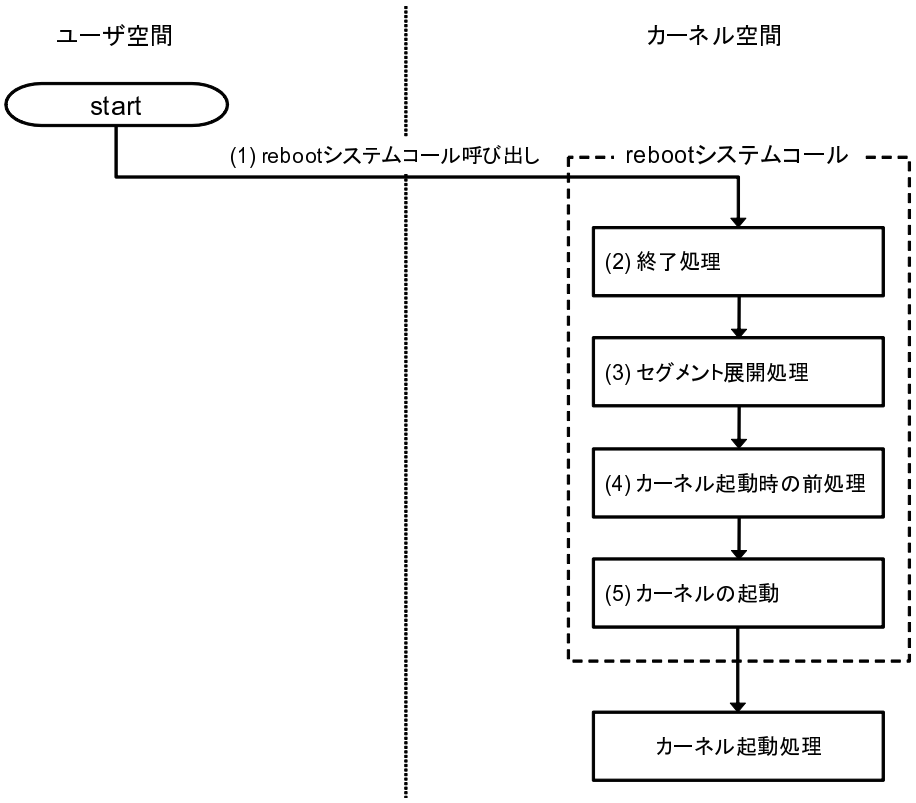

図 3.4 カーネル起動機能のフローチャート

その後, entry メンバが示す領域へ, カーネル展開処理の終了を示す IND_DONE(0x4) を格納する. 以上の処理を各セグメントに対して行う.

次に, カーネル起動機能を実行した場合のフローチャートを 図3.4 に示し, 以下で説明する.

カーネル起動機能の処理流れ

(1) reboot システムコール呼び出し

Kexec 機能を有効にした reboot であることを引数として渡し, reboot システムコールを呼び出す. reboot システムコールは, 通常の reboot の処理は行わず, カーネル起動機能の処理流れの (2) から (5) の処理を順次実行する.

(2) 終了処理

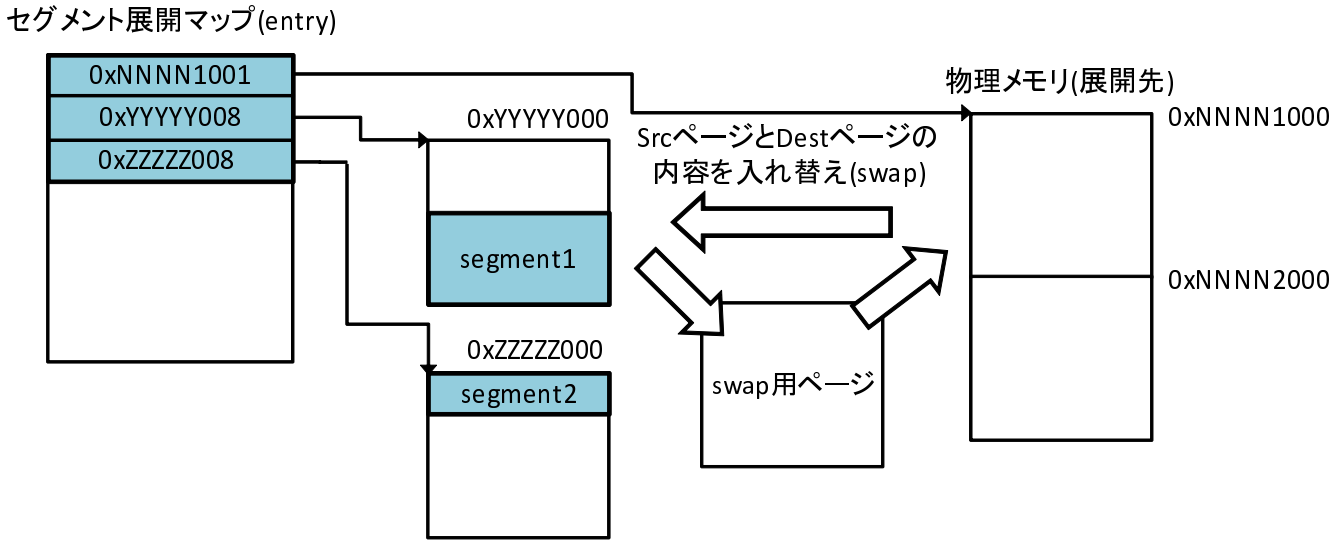

図 3.5 セグメント展開処理

終了処理一覧を 表3.2 に示す．device_shutdown 関数を呼び出すことでデバイスの終了処理を行い，sysdev_shutdown 関数を呼び出すことでシステムデバイスの終了処理を行う．その後，smp_send_stop を呼び出すことで SMP の終了処理を行い，lapic_shutdown 関数を呼び出すことで APIC の終了処理を行う．また，hpet_disable 関数を呼び出すことで，HPET(High Precision Event Timer) を無効化し，タイマ割り込みを発生させなくする．

表 3.2 終了処理一覧

ファイル	関数	対象
drivers/base/core.c	device_shutdown	デバイス
drivers/base/sys.c	sysdev_shutdown	システムデバイス
arch/x86/include/asm/smp.h	smp_send_stop	SMP
arch/x86/kernel/apic.c	lapic_shutdown	APIC
arch/x86/kernel/hpet.c	hpet_disable	HPET

(3) セグメント展開処理

カーネル読み込み処理流れの (2) で kexec-tools 内で作成し，カーネル読み込み処

理流れの (4) でカーネル空間へ複写したセグメントを，指定したメモリ上に展開する．セグメントの展開方法を 図3.5 に示す．図3.3 で説明し，カーネル読み込み処理流れの (5) で作成したセグメント展開マップを先頭から参照し，展開先のページ (Dest エントリ：0xXXXXX001) とセグメントを保存したページ (Src エントリ：0xXXXXX008) のアドレスを得る．そして，両者のページを入れ替えることで，セグメントを指定のメモリ上へ展開する．また，入れ替えに利用するためのページを別に用意している．この処理は IND_DONE のエントリが見つかるまで行う．

(4) カーネル起動時の前処理 (purgatory)

メモリ上に展開されたカーネルへジャンプするための前処理として，セグメントレジスタとその他のレジスタの値を適切なものに設定する．この処理により，ジャンプ先をカーネル起動処理の先頭に設定する．

(5) カーネルの起動

展開されたカーネルの起動処理の先頭にジャンプすることで，カーネルを起動する．

3.3 起動処理の比較

Kexec の再起動処理と TwinOS における従来の共存 OS 起動処理の比較を 図3.6 に示し，以下で説明する．

(1) 「(1-a) カーネルの読み込み」は,「(2-a) カーネル読み込み」と同じ処理を行っているため，変更を加えずに使用可能である．

(2) 「(1-b) 再起動前のカーネルの終了処理」は,「(2-b) 先行 OS の走行環境の保存」に必要ない処理であるため，省略する．

(3) 「(1-c) カーネルをメモリ上に展開」は，メモリの若番アドレスに展開するように変更することで,「(2-c) 共存 OS のカーネルをメモリの若番アドレス上に展開」に使用可能である．Kexec では，Linux Kernel のコンフィグにより指定されたメモリアドレスからカーネルを展開する．一方，共存 OS 起動処理では，メモリの若番アドレスである 1MB にカーネルの展開先を固定している．このため，Kexec でカーネルの展開先をメモリの任意アドレスにできればよい．

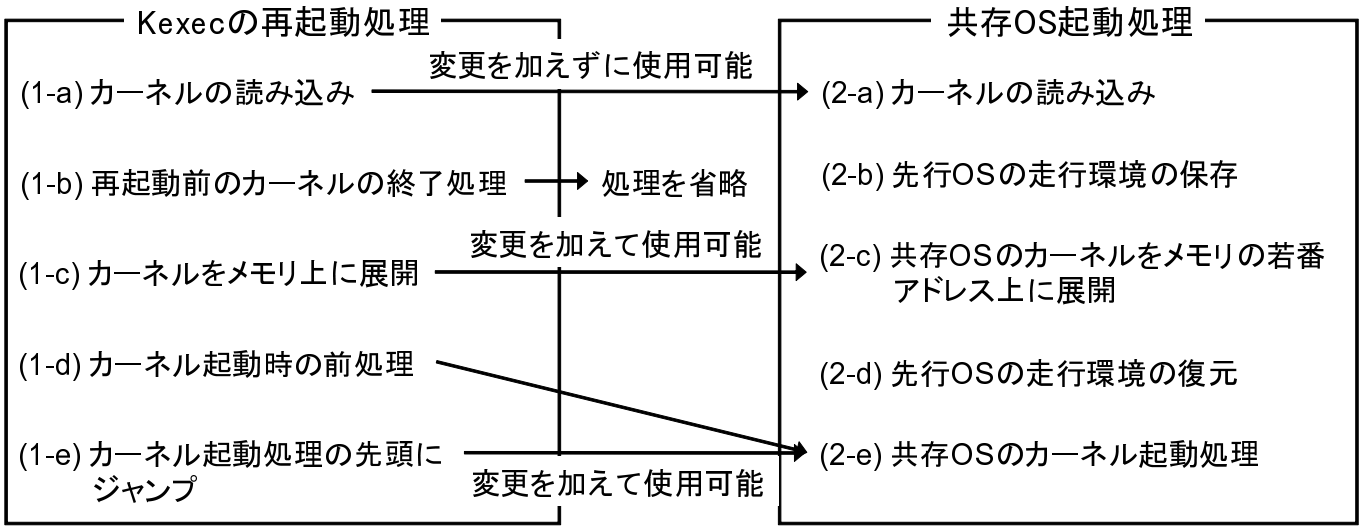

図 3.6 Kexec の再起動処理と共存 OS 起動処理の比較と応用

(4) 「(1-d) カーネル起動時の前処理 (purgatory)」と「(1-e) カーネル起動処理の先頭にジャンプ」は，purgatory に TwinOS 特有のコードを埋め込むことで,「(2-e) 共存 OS のカーネル起動処理」に使用可能である．

(5) 「(2-b) 先行 OS の走行環境の保存」と「(2-d) 先行 OS の走行環境の復元」を行う必要がある．

このように，Kexec を応用することで，共存 OS 起動処理を行える．

3.4 要求

3.3 節より，Kexec を用いた共存 OS 起動処理を実現するために，以下の 4 つの要求を導ける．

(要求 1) 先行 OS の走行環境の保存と復元

共存 OS の起動時には，OS の初期化処理により先行 OS の走行環境が破壊される．このため，Kexec のカーネル起動機能実行時 (読み込んだカーネルをメモリ上に展開する前) に先行 OS の走行環境を保存する必要がある．また，先行 OS 切

り替え時に走行環境を復元する必要がある．保存と復元が必要な対象を以下に示す.

(1) 汎用レジスタ (eax, ebx 等)

(2) セグメントレジスタ (ss, cs 等)

(3) 制御レジスタ (CR0,2,3,4)

(4) スタック (esp)

(5) GDT, IDT

(6) タスクレジスタ (TR)

(7) プロセス

(8) デバイス

(要求 2) 先行OSの走行環境の保護

3.2.2節で記述したように，Kexecのカーネル起動機能を用いて再起動を行う場合，再起動する直前に 表3.2 の対象に対して終了処理を行う．デバイスは保存対象であり，他の終了対象も先行OS走行環境保護のために終了させるべきではない．このため，これらの終了処理を行わないようする．

(要求 3) メモリマップの領域のタイプを無視してカーネルを展開

kexec-toolsは，メモリの利用可能状況を判定するために，メモリマップを/sys/firmware/memmapから取得する．このディレクトリには，メモリ領域が利用可能 (usable) か利用不可 (reserved) かという情報が格納されている．この情報をタイプと呼ぶ．kexec-toolsは，セグメント作成時に取得したメモリマップのタイプをもとにセグメントの展開先が利用可能か否かを判定する．一方，TwinOSでは，意図的にメモリの一部分のタイプを利用不可 (reserved) にすることで，メモリ分割を実現している．このため，本来はタイプが利用可能 (usable) であるが利用不可 (reserved) になっているメモリ領域に共存OSのカーネルを展開できるようにする必要がある．

(要求 4) カーネルを任意のメモリアドレスに展開

TwinOSはメモリを上位と下位で2分割して使用する．共存OSはメモリの若番アドレスである1MBから展開する．このため，共存OSのカーネルを任意のメモリアドレスから展開できるようにする必要がある．

3.5 対処

3.4 節で示した (要求 1) から (要求 4) に対して，それぞれ (対処 1) から (対処 4) を示す.

(対処 1) Kexec を用いたカーネル走行環境の保存と復元

kexec-tools には，再起動前に走行しているカーネルの走行環境を保存して再起動を行う機能がある．この機能は，--load-preserve-context オプションを用いることで利用できる.

先行 OS の走行環境の保存と復元を行うために，--load-preserve-context オプションのカーネル走行環境保存処理とカーネル走行環境復元処理をそれぞれ TwinOS 用に改変する．カーネル走行環境復元処理は，先行 OS のカーネルに戻り，実行されるべきである．しかし，--load-preserve-context オプションを用いただけでは先行 OS のカーネルに戻ることはできず，カーネル走行復元処理は呼び出されない．また，--load-preserve-context オプションを用いると，IO-APIC を無効化する．この際，先行 OS で設定した IO-APIC のエントリを削除し，TwinOS の割り込み環境を破壊する．このため，--load-preserve-context オプションを用いなくても，環境の保存と復元を行えるようにした.

先行 OS のカーネル走行環境の保存と復元を実装することにより,「(要求 1) 先行 OS の走行環境の保存」を満足できる.

--load-preserve-context オプションのカーネル走行環境保存処理とカーネル走行環境復元処理を以下に示す．なお，これらの処理は kexec_load システムコールの中で実行される.

カーネル走行環境保存処理

(1) 排他制御

(2) コンソール準備

(3) プロセスをフリーズ

(4) コンソールを停止

(5) ステートを保存してデバイス停止

(6) BSP 以外のプロセッサを停止

(7) デバイスの電源管理に関わるロックを取得

(8) ローカル割り込みを無効化

(9) デバイスをパワーオフ

(10) MTRR(メモリタイプ範囲レジスタ) を保存

(11) プリエンプションを無効化

(12) GDT, IDT, TR, タスクレジスタ, 各種セグメントレジスタ, 各種制御レジスタをそれぞれ保存

(13) 汎用レジスタを保存

(14) スタック (esp) を保存

カーネル走行環境復元処理

(1) スタック (esp) を復元

(2) 汎用レジスタを復元

(3) GDT, IDT, TR, タスクレジスタ, 各種セグメントレジスタ, 各種制御レジスタをそれぞれ復元

(4) プリエンプションを有効化

(5) MTRR(メモリタイプ範囲レジスタ) を復元

(6) デバイスをパワーオン

(7) ローカル割込みを有効化

(8) デバイスの電源管理に関わるロックを解除

(9) BSP 以外のプロセッサを起動

(10) デバイスのステートを復元

(11) コンソールのステートを復元

(12) フリーズしていたプロセスを解凍

(13) コンソールをリストア

(14) 排他制御の解除

(対処 2) 終了処理を省略

また, 表3.2 の終了処理を行わないように, 終了処理を行う関数の呼び出しを省略した. これにより, 先行 OS のカーネル走行環境の保護を行うことで,「(要求 2) 先行 OS の走行環境の保護」を満足できる.

(対処3) メモリマップのタイプによるエラー判定を省略

共存OSのカーネルは，usableなメモリ領域に配置される．この領域は，先行OSから見るとreservedである．このため，メモリマップのタイプを参照してセグメントの配置が可能か否かを判定する処理でエラーであると判定されて処理が停止する問題がある．このため，メモリマップのタイプを判定する処理を省略した．なお，カーネル内にはメモリマップを参照する処理はないため，カーネルを変更する必要はない．メモリマップのタイプによるエラー判定を省略することにより,「(要求3) メモリマップの領域のタイプを無視してカーネルを展開」を満足できる．

(対処4) Kexecを用いたカーネル展開先の制限

カーネルを任意のメモリアドレスへ展開する方法について，以下に2種類示す.

(方法1) CONFIG_PHYSICAL_STARTによるアドレス指定

Linux Kernelは，カーネル構築時にCONFIG_PHYSICAL_STARTでアドレスを指定することにより，指定したアドレスを先頭アドレスとしてカーネルを展開できる．しかし，カーネル展開先アドレスを変更したい場合，カーネルを再度コンパイルする必要がある．現在のTwinOSは，先行OSをメモリの老番アドレスに展開し，共存OSをメモリの若番アドレスに展開する必要がある．これは，各OSに対してCONFIG_PHYSICAL_STARTの値をそれぞれ指定してカーネルを構築することで実現できる．しかし，今後の研究として3つ以上のカーネルを起動させることや，先行OSと共存OSのカーネルの展開先を変更することを考える場合，毎回CONFIG_PHYSICAL_STARTの値を変えてコンパイルし直す必要があり，汎用性が低いという問題がある．このため，本方法は有用ではない．

(方法2) Kexecによるアドレス指定

kexec-toolsには，カーネル展開先アドレスの下限を指定する--mem-minオプションがある．カーネル読み込み機能実行時に，--mem-minオプションでアドレスを指定することにより，全てのセグメントは指定したアドレス以降に展開される．注意として，Linux Kernelのカーネル構築時に，CONFIG_RELOCATABLEをONにする必要がある．また，CONFIG_RELOCATABLEとCONFIG_PHYSICAL_STARTは，どちらか一方しか使用することができない．具体的には，CONFIG_RELOCATABLE

を ON にし，Kexec を用いずに起動させた場合のカーネルは，カーネル構築時のコンフィグで CONFIG_PHYSICAL_ALIGN で指定した値のアドレスから展開される．

汎用性の向上を考慮し，本論文では,「(方法 2) Kexec によるアドレス指定」により,「(要求 4) カーネルを任意のメモリアドレスに展開」に対処する．

第 4 章

実装

3.5 節の「(対処 1) Kexec を用いたカーネル走行環境の保存と復元」,「(対処 2) 終了処理を省略」，および「(対処 3) メモリマップのタイプによるエラー判定を省略」に関する実装は，3.5 節で述べた．本章では,「(対処 4) Kexec を用いたカーネル展開先の制限」の「(方法 2) Kexec によるアドレス指定」の実装について述べる．

4.1 現在の kexec-tools の問題点

kexec-tools は，bzImage 形式のカーネルをリアルモード部分とリアルモード以外の部分に分けてセグメントを作成する．具体的には，カーネル読み込み機能実行時に`--real-mode`オプションを用いるとリアルモード部分の処理を実行し，このオプションを用いない場合はリアルモード部分の処理を実行しない．また，リアルモード部分のコードは，性質上 1MB 以内にしか置くことができない．これはリアルモードの場合，全てのレジスタのアドレス幅が 16 ビットであり，セグメントレジスタの値を 4 ビット左にずらして足すアドレス変換により 20 ビット (1MB) のアドレス空間にアクセスできるためである．kexec-tools ではリアルモード部分を 640KB 以内に置くように制限している．現在の kexec-tools で`--mem-min`オプションを用いると，`--mem-min`オプションで指定したメモリアドレス以上のメモリ領域に全てのセグメントを展開する．`--mem-min`オプションで 0x8000000(128MB) を指定したとすると，リアルモード部分も 0x8000000(128MB) 以降に置こうとするため，エラーとなり，処理を終了する．

4.2 問題点への対処法

4.1 節で説明した問題への対処法として以下の 2 つの方法を検討した．

(方法 1) --real-mode を指定時のみリアルモード部分のセグメントを作成

リアルモード部分の処理を実行しないのであれば，リアルモード部分のセグメントは不要である．このため，`--real-mode` オプションを指定した場合のみリアルモード部分のセグメントを作成するように kexec-tools を変更する．

本方法の問題として，リアルモード部分のセグメントを作成せずに Kexec を用いてカーネルを起動すると，起動途中にカーネルが再起動し，カーネルを正常に起動させることができない．この原因として，カーネル起動処理において，リアルモード部分のデータ領域を使用することが挙げられる．

(方法 2) リアルモード部分を任意のメモリアドレスに展開

`--real-mode` オプションを用いずに Kexec を使用した場合は，リアルモード部分のテキスト領域は使用しないが，データ領域は使用する．リアルモード部分を走行しないのであれば，リアルモード部分を 1MB 以内に置く必要はない．このため，リアルモード部分を任意の位置に置けるように kexec-tools を変更する．具体的には，リアルモード部分のセグメントを作成する際に 640KB 以内にしか置けないようにする制限を解除する．

(方法 2) の変更を施した kexec-tools を用いて， 図 4.1 のような配置でセグメントを作成しカーネルを動作させて正常に動作することを確認した．

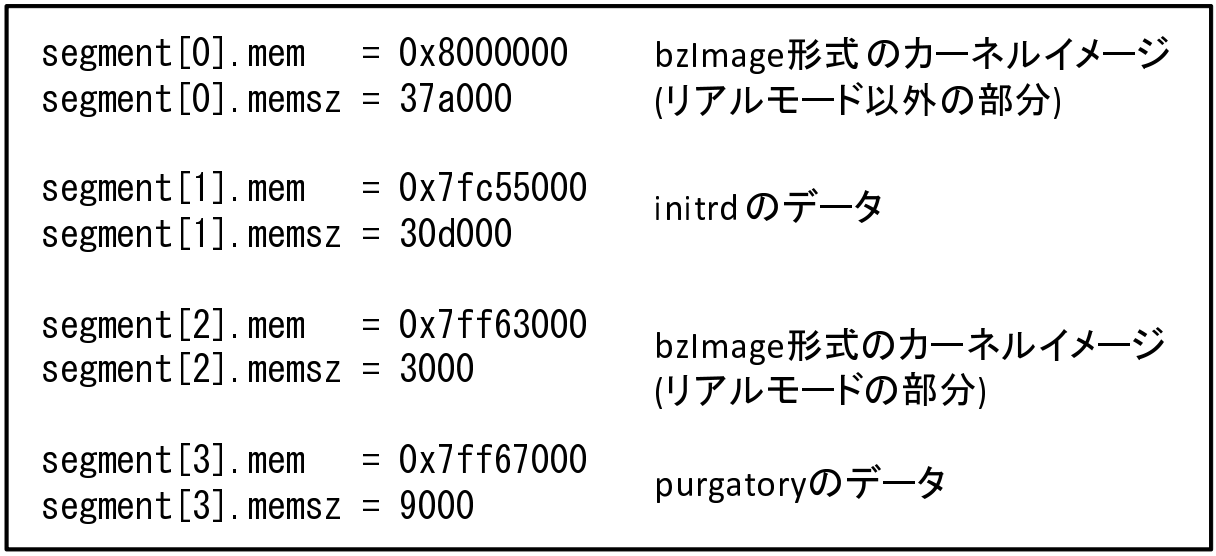

図 4.1 セグメントの展開位置の例

第 5 章

おわりに

本論文では，Kexec の TwinOS への応用方法を検討し，一部の機能を実装した結果を述べた．

まず，現在の TwinOS の問題点を述べ，Kexec を用いた共存 OS 起動処理の目的について述べた．次に，Kexec の概要を述べ，Kexec を用いて共存 OS 起動処理を行う際の要求と要求への対処について述べた．

1つ目の要求は，共存 OS のカーネル初期化処理により先行 OS の走行環境が破壊されないように，共存 OS のカーネルをメモリ上に展開する前に先行 OS の走行環境を保存し，共存 OS 起動後に復元することである．この要求への対処として，Kexec が持つ走行環境保存機能を TwinOS 用に利用できるようにした．2つ目の要求は，Kexec のカーネル起動機能で行われる終了処理を行わないようにすることである．この要求への対処として，TwinOS に影響のある終了処理の呼び出しを省略した．3つ目の要求は，メモリマップの領域のタイプによらずカーネルを展開できるようにすることである．この要求への対処として，カーネルの展開先のメモリマップの領域のタイプを判定しないようにした．4つ目の要求は，カーネルを任意のメモリアドレスに展開することである．この要求への対処として，Linux Kernel 構築時のコンフィグで CONFIG_RELOCATABLE を ON にして，任意のメモリアドレスにカーネルを展開する機能を kexec-tools に実装した．

今後の課題として，共存 OS 起動後に先行 OS に処理を戻す機能の実装がある．また，マルチコア CPU 上で，Kexec を用いて各コアを占有し，カーネルを起動する機能の実装がある．

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，数々のご指導やご助言を頂きました谷口秀夫教授，田端利宏准教授，および後藤佑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(D-I)，vol．J88-D-I，no．2，pp．251-262 (2005．02)．

[2] “The kexec Archives，”http://lists.infradead.org/pipermail/kexec/

[3] IBM，“Kexecを使ってLinuxの起動を早める，”
http://www.ibm.com/developerworks/jp/linux/library/l-kexec/index.html