Vol.2013-OS-126 No.17
2013/8/1

Mint: Linuxをベースとした複数OS混載方式の提案

北川 初音[1]　乃村 能成[1]　谷口 秀夫[1]

概要：計算機の性能の向上にともない，1台の計算機上で複数のOSを動作させる方式が研究されている．これらの研究において，複数の種類のOSを動作させることで，それぞれのOSが持つ特性を同時に利用できる．しかし，OS混載のために仮想計算機方式を用いた場合，OS間で依存関係が発生する．そこで，1台の計算機上でLinuxベースのOSを複数混載する方式を提案する．本提案方式では，Linuxの機能に制限をかけることなく動作可能である．本稿では，Linuxベースの OSを混載する場合に，改変を加える必要がある部分を明らかにする．また，本提案方式を実現するために必要な改変量は小さいことを示す．最後に本提案方式の実例として，Linuxと Androidの混載と 32/64bit Linuxの混載について述べる．

1. はじめに

1台の計算機上で複数の異なる種類のOSを動作させたいという要求がある．これは，異なる種類のOSを同時に動作させることで，それぞれのOSが持つ他にはない特性を同時に1台の計算機上で利用できるためである．例えば，既存の家電製品では，リアルタイム性の高い制御と豊富な既存アプリケーション資産の両方を得るために，CPUを複数搭載し，それぞれのCPUで異なるOSを動作させることがある．このような場合において，1つのCPUに異なる種類のOSを混載し，独立して動作させることができれば，製品内のCPUは1つで済む．

混載を実現する手段として，仮想計算機方式がある．仮想計算機方式は，混載OSの改変が不要であるものの，実計算機をそのまま利用する場合に比べて実行速度が低下したり，多くのメモリを必要としたりするため，実利用におけるオーバヘッドが大きい．また，仮想計算機間の独立性が低いために，一方のOSが他方のOSの負荷に影響されて性能が低下することがある．このような場合，リアルタイム性を要求するOSが本来の性能を発揮できないなど，混載の意義が大きく低下する．

一方，仮想化によらず混載を実現する場合には，使用環境に合わせてOSを改変する必要がある．したがって，仮想化によらない混載において重要なのは，OSのどの部分にどの程度の改変が必要になるか見積もることと，できるだけ改変量の小さい書き換え方式を確立することである．

そこで，本稿では，仮想化によらないOS混載方式の事例としてLinuxとその亜種を取り上げ，混載の手法と混載に必要なコードの改変箇所および量を明らかにする．Linuxをベースとした OSを混載対象とすることで，組み込み機器やスマートフォンなど，Linux由来の様々な用途のOSにも提案方式を適用可能であり，目的に応じたLinux亜種の混載を小さい改変量で実現できると考えられる．

調査の結果，OS混載のために必要な改変は，カーネルにおいては，主にコアの分割，メモリの分割，割り込みルーティング，レガシーデバイスの初期化，および起動終了処理の5つに分類でき，その改変量は，カーネルコード全体の1%未満であることがわかった．

また，レガシーデバイスの割り込みの扱いに関してLinux固有の問題が存在することが分かった．さらに，提案方式による改変はOSの起動処理にほぼ集約されるため，起動処理を支援するユーザコマンドを用意することで，カーネルの改変を小さく抑えられることが明らかになった．それらについて別途詳細に説明する．

提案方式では，それぞれのOSは計算機資源を仮想化せず直接占有制御する．したがって，それぞれのOSは独立に動作するため，お互いに影響を与えることがない．

以降の章では，2章で複数OS混載の要求条件と関連研究について述べ，3章でLinuxをベースとした複数OSの混載方式Mintを提案し，設計方針やOS改変の詳細について述べる．4章でMintにおける混載の実例として，LinuxとAndroidの混載と32/64bit Linuxの混載について述べる．

2. 複数OSの混載

2.1 要求条件

1台の計算機上に複数の種類のOSを混載するにあたり，

[1] 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

それぞれの OS には以下の条件が要求される．
(1) OS が単独で動作する場合に比べ，性能低下が小さい．
(2) 一方の OS の負荷が他方の OS に影響を与えないだけでなく，他方の OS の環境を破壊せずに再起動できる．
(3) 混載を実現するために必要な改変量が小さい．
仮想計算機方式を用いた場合，(1)，(2) の条件を満たさない．また，仮想化を用いない場合においても，(3) の条件を満たさない場合がある．提案方式では，上記の条件を満たし，複数の種類の OS を動作可能である．

2.2 関連研究

1 台の計算機上で複数の OS を動作させる方式について，様々な研究が行われている．これらの研究のうち，代表的なものとして仮想計算機方式がある．Xen[1] は，ハイパーバイザを用いて計算機資源を仮想化し OS を複数動作させる．KVM[2] は，Linux に VMM としての機能を加え仮想計算機を動作させる．仮想計算機方式を用いた場合，計算機の仮想化によるオーバヘッドが発生する．このため，実計算機上で動作する OS と比較して性能が低下する．

仮想化によるオーバヘッドを抑えるため，仮想化部分を小さくする研究が行われている．DARMA[3][4] は，ナノカーネル上で標準 OS と独自 OS を動作させ，OS 間の関係を希薄にしている．SPUMONE[5][6] では CPU のみを仮想化し，複数の OS を動作させる．他のデバイスは仮想化しない．また，OSV と呼ばれる軽量な VMM を用いて，プロセッサとメモリを仮想化する方式 [7] がある．これらの研究で仮想化部分を小さくしても，仮想化部分は存在するため，OS に依存関係が発生し，OS が相互に影響を与えるという問題点がある．

そこで，仮想化を用いずに，複数の OS を独立に走行させる研究が行われている．TwinOS[8] は，CPU を時分割，その他の計算機資源を分割占有することで 1 台の計算機上で 2 つの Linux を動作させる．TwinOS では各 OS はシングルコアプロセッサを時分割して使用する．一方で提案方式では各 OS はマルチコアプロセッサをコア分割して使用する．このため，提案方式では OS 切り換えのコストなしで OS を複数動作できる．OS Switching[9] は，サスペンド・レジューム機能を利用して，動作する OS を切り替えることにより複数の OS を動作させる．このため，同時に複数の OS を動作できるか否かという点で提案方式と異なっている．SHIMOS[10][11] は，計算機資源を分割し，各 OS に占有させることで複数の OS を動作させる．提案方式とは，2 番目以降の OS の起動方法が異なる．SHIMOS では専用のカーネルモジュールを用いる．一方提案方式では，専用のカーネルモジュールを作成するのではなく，Linux 既存の Kexec[12] を利用している．Kexec の本来の目的は，Linux カーネルの高速な再起動である．これを他 OS の起動に利用する．その結果，工数削減やメンテナンス性の確保といった利点とともに，Kexec が本来持つ高速再起動の機能を利用することで，個別 OS の再起動も可能になる．Twin-Linux[13] は，2 つの Linux に BSP と AP をそれぞれ占有させる．カーネルは BSP で起動したカーネルのコピーを AP にロードする．このため，異なる種類の OS を動作できないという点で提案方式と異なる．Multikernel[14] は，各 OS が CPU ドライバを用いて CPU やその他のドライバを制御する．この際，動作する OS は独自 OS である．このため，独自 OS を用いるか否かという点で提案方式と異なる．これらの研究では，OS を単独で再起動できない．一方，提案方式では他の OS の環境を破壊することなく，それぞれの OS を単独で再起動できる．

また，仮想化の利便性を追求するため，ユーザモードで Linux を動作させる User-mode Linux[15] や Cooperative Linux[16] が存在する．これらはユーザモードで動作するため，デバイスを OS が直接制御できない．

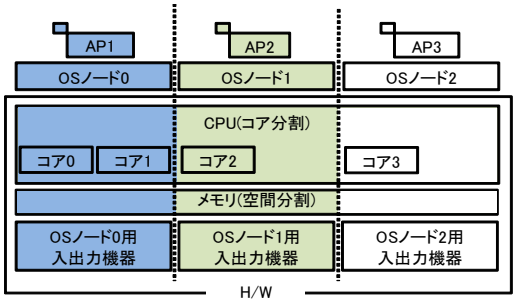

図 1 Mint の構成例

3. Linux をベースとした複数 OS の混載

3.1 設計方針

Linux をベースとした複数 OS の混載方式として Mint を提案する．Mint は 1 台の計算機上で複数の Linux を独立に走行させる方式である．本稿では Mint を構成する OS を OS ノードと呼ぶ．Mint の設計方針として，以下の 2 つがある．
(1) 全 OS ノードが相互に処理負荷の影響を与えない
(2) 全 OS ノードが入出力性能を十分に利用できる

3.2 構成

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 ノードで占有する．**図 1** に Mint の構成例を示し，CPU，メモリ，およびデバイスの分割と占有方法について以下で説明する．

CPU コア単位で分割し，各 OS ノードで占有する．
メモリ 走行させる OS ノードの数だけ実メモリを空間分割し，各 OS ノードで占有する．
デバイス デバイス単位で分割し，各 OS ノードが仮想化によらず直接占有制御する．

表 1 Mint の実現に必要な改変量の内訳

分類	改変箇所	改変対象		改変した行数		
		ファイル数	行数	追加	変更	削除
1	コア分割	13	5,007	223	7	2
2	メモリ分割	2	94	32	1	2
3	割り込みルーティング	4	4,165	122	3	23
4	レガシーデバイスの初期化	2	342	8	0	0
5	起動終了処理	14	7,158	135	9	15
1 から 5 全体		31	14,810	582		
6	デバイスドライバ	4	6,804	74	4	5
7	Kexec-tools	13	3,017	117	4	113
1 から 7 全体		48	24,631	899		

Mint において OS ノードを起動させる際，最初に 1 つの OS ノードを起動し，この OS ノードから別の OS ノードを起動させる．最初に起動する OS ノードを OS ノード 0 と呼び，後から起動する OS ノードを起動順に OS ノード 1，OS ノード 2，… と呼ぶ．

3.3 Linux からの改変箇所と改変量

3.3.1 概要

Mint は x86 アーキテクチャを対象にしている．まず，カーネルコード全体量を概算すると，Linux 3.0.8 x86 では，デバイスドライバ関連を除き 5,481 ファイル 1,427,979 行であることが分かった．このうち，95 ファイル 18,633 行はアセンブラのコードである．また，デバイスドライバ全体のコードは，11,084 ファイル，5,490,330 行である．このうち，9 ファイル 1,331 行はアセンブラのコードである．Mint における改変箇所は，大別して以下の 7 箇所に分類できる．

(1) コアの分割
(2) メモリの分割
(3) 割り込みルーティング
(4) レガシーデバイスの初期化
(5) 起動終了処理
(6) デバイスドライバ
(7) Kexec-tools

上記 (1) から (5) は，デバイスドライバを除くカーネル，(6) は，デバイスドライバのコード，(7) は，ユーザコマンドである．

本提案方式を実現するために必要な改変量について**表 1** に示す．表 1 では，上記の改変箇所のそれぞれについて，「改変対象」として，改変対象となったファイルの数とそれらの行数を示している．また，「改変した行数」として，改変対象のファイルに対する追加，変更，および削除を行った行数を示している．「1 から 5 全体」は分類 1 から 5 の「改変対象」のファイル数，行数，および改変の行数の合計を表す．

なお，「1 から 5 全体」の改変対象「ファイル数」は，分類 1 から 5 の「改変対象」のファイル数の合計と一致しない．なぜなら，ファイルによっては，複数の分類にまたがって改変対象となっている場合があるためである．例えば，`init/main.c` は分類 1，3，および 5 について改変の対象となっている．この場合，`init/main.c` はファイル数として分類 1，3，および 5 それぞれでカウントされている．このため，1 から 5 全体の「ファイル数」と「行数」は，各分類のそれらを単純に合計したものより小さくなる．「1 から 7 全体」も同様である．

各改変箇所における改変量とその詳細は，以降の項に示し，ここでは全体量の概略を示す．

まず，(1) から (5) のカーネルコードに関して説明する．Mint では，そのうち 31 ファイル 582 行に改変を加えた．これは，先に述べたカーネル 5,481 ファイル 1,427,979 行に対して，ファイル数にして 0.6%，行数にして 0.04%で，いずれも 1%を大きく下回る．改変を加えた 31 ファイル内での改変割合は，改変前の行数 14,810 行に対する 582 行 (約 3.9%) であった．また，これらのうち 3 ファイル 85 行はアセンブラのコードに対する改変であった．次に，デバイスドライバ部分の改変について述べる．Mint では，通常のデバイスドライバには手を加えずそのまま利用可能であるが，PCI バスドライバ，ATA ポートドライバに限っては，OS ノードへのデバイス振り分けのための改変が必要だった．これらは，(6) デバイスドライバとして別途集計した．改変量は，4 ファイル 83 行であった．改変を加えた 4 ファイル内での改変割合は，改変前の行数 6,804 行に対する 83 行 (約 1.2%) であった．

最後に (7)Kexec-tools のユーザコマンドの改変について説明する．Mint では，OS ノード 1 以降の起動に Linux 既存の Kexec を用いるため，Kexec のユーザコマンドである Kexec-tools にも改変を加えた．Kexec は，カーネル内の再起動部分に存在するコードおよび再起動のためのユーザコマンドからなる．前者を単に Kexec，後者を特に Kexec-tools と呼んで区別し，前者は，(5) 起動終了処理に分類した．Kexec-tools は，164 ファイル 23,775 行からなり，このうち，23 ファイル 2787 行はアセンブラのコード

であった．Mint における改変量は 13 ファイル，234 行であり，これは，Kexec-tools の 164 ファイル 23,775 行の，ファイル数にして 8%，行数にして 0.1%にあたる．また，これらのうち 3 ファイル 123 行はアセンブラのコードに対する改変であった．

また，これら改変箇所は，ほぼ起動時と終了時に実行される処理であった．唯一，分類 1 のコア分割の部分に一部，OS 走行中の処理に関わる箇所があった．具体的には，コア番号識別処理に関わる改変で，これは，表中からは読み取れないが，3 ファイルに跨る合計 6 行の追加のみであった．

以下，上記改変の詳細について説明する．

3.3.2 カーネル

カーネルについて加えた改変は，コアの分割，メモリの分割，割り込みルーティング，レガシーデバイスの初期化，および起動終了処理の 5 つに分類される．以下でこれらについて説明する．

(1) コアの分割 (改変量: 13 ファイル，232 行)
コアの分割に関して，OS ノード間で占有するコアの重複を避けるため，BSP の Local APIC ID よりも小さい Local APIC ID をもつコアは先行する OS によって占有されていると定め，これを起動しないようにする．また，Mint 全体でコアの占有状態を管理するため，共有メモリ領域を設け，コア管理部を作成している．コア管理部では各コアに関してコアを占有している OS ノードの情報を管理する．さらに，OS ノード間で論理 APIC ID の重複を防ぐため，論理 APIC ID を OS ノード間で一意に算出するように改変する．論理 APIC ID は割り込み通知先コアの指定に利用されるため，OS ノード間で重複してはならない．通常，Linux では，論理 APIC ID をコア ID から算出する．コア ID は，Linux カーネルが OS 内で独自に保持するコア識別子である．しかし，コア ID は OS ノードに一意な値にはならないため，ここでは，ハードウェアによって一意に定まる Local APIC ID を用いることで対処した．

(2) メモリの分割 (改変量: 2 ファイル，35 行)
Linux は，BIOS から取得したメモリマップをもとに，自身が利用可能な実メモリ領域を認識する．実メモリを OS ノード間で排他的に分割するため，BIOS で取得したメモリマップを OS ノード毎に書換え，各 OS ノードに個別の重複しない実メモリ領域を認識させる．

(3) 割り込みルーティング (改変量: 4 ファイル，148 行)
Linux は起動時に I/O APIC の設定をクリアし，自身の占有するデバイスに対し，コアに割り込みを通知するように設定する．このため，各 OS ノードの割り込みの設定について，先に起動した OS ノードが I/O APIC に設定した内容を後に起動した OS ノードが上書きしないようにする．具体的には，すでに I/O APIC に設定されている割り込みの通知先と新たに設定しようとする割り込みの通知先の両方に通知されるようにする．ここで，割り込みを複数のコアに通知するため，割り込みルーティングを変更する．1 つのコアにのみ割り込みが通知可能な physical flat から複数のコアに割り込みを通知可能な flat へ変更する．また，I/O APIC の各割り込みに設定するベクタ番号に関して，OS ノード間で共有する割り込みには同じベクタ番号を設定する．この際，レガシーデバイスの割り込みに割り当てられるベクタ番号である 48 から 63 番には，他のデバイスの割り込みが割り当てられないようにする．これについては，4.1.3 項で詳しく述べる．また，Linux ではタイマ割り込みの供給源として HPET を利用している．各 OS が利用する HPET の重複を避けるため，HPET の割り当てを Local APIC ID に基づいて行うように変更し，自身の利用するタイマのみ初期化する．また，コアを解放した際，割り込みが通知されないように変更する．

(4) レガシーデバイスの初期化 (改変量: 2 ファイル，8 行)
通常のデバイスは，デバイスドライバのロードを制御することで OS ノードへの割り当てを制御可能であるが，キーボード，VGA，シリアルポートのレガシーデバイスはデバイスドライバの有無にかかわらず全 OS ノードが認識しようとする．このため，これらのうち各 OS ノードで占有しないものは初期化処理を行わないことで無効化する．

(5) 起動終了処理 (改変量: 14 ファイル，159 行)
起動処理について，占有可能デバイスを指定するブートパラメータを追加した．これにより単一カーネルイメージから占有デバイスの異なる複数 OS ノードを起動できる．

また，他 OS ノードの起動処理について，Kexec のカーネル内での処理に改変を加える．Kexec は，本来自身を再起動するための仕組みであるため，通常は，カーネルイメージを実メモリ上に展開した後，各種初期化を省いたカーネル開始ルーチン (purgatory と呼ぶ) にジャンプする．Mint では，Kexec を利用して他 OS を起動するように改変した．具体的には，自身が purgatory にジャンプする代わりに指定されたコアへ IPI を送信する．この様子を**図 2** に示し，以下で説明する．

(A) ユーザプログラムである Kexec-tools により，purgatory(a)，カーネルイメージ (b)–(c)，および initrd(d) を仮想メモリ上に配置する．

(B) Kexec システムコールにより，カーネルは，Kexec-tools が用意した (a)–(d) を物理メモリ上に展開する．

(C) カーネルは，IPI を送信し，他コアを起動し，起

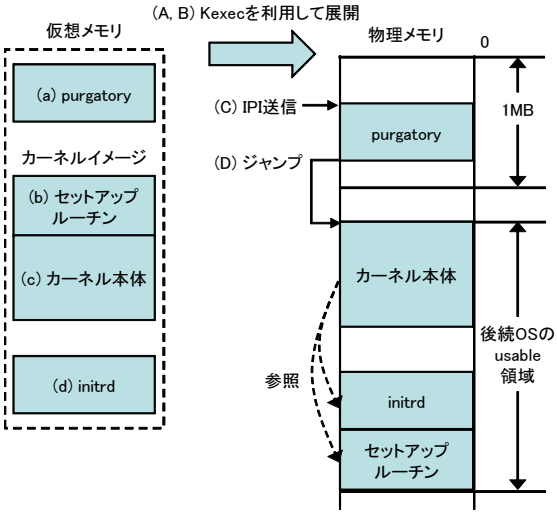

図 2 後続する OS ノード起動処理の処理流れ

動したコアに purgatory を実行させる．

(D) purgatory，セットアップルーチンの実行後，圧縮カーネル展開ルーチンにより，圧縮カーネルを展開，再配置してカーネルを起動する．

OS ノードの終了処理について，通常の終了処理では，I/O APIC とタイマを無効化する．このため，他の OS の設定を破壊しないように，I/O APIC とタイマの無効化を行わないようにする．I/O APIC の無効化を行わない代わりに，I/O APIC から終了する OS の設定した割り込みの設定を除外する．また，終了処理後に計算機をシャットダウンさせず，占有しているコアのみを HALT 状態にする．

3.3.3 デバイスドライバ

デバイス振り分けのため，PCI バスドライバに改変を加える．通常，Linux では利用可能なデバイスをすべて占有しようとする．このため，デバイスを占有制御するため，デバイスドライバを組み込むコードを改変し，占有しないデバイスのデバイスドライバを利用不可にする．これにより，各 OS ノードで占有するデバイスのみにデバイスドライバを割り当てる．

また，ハードディスクドライブ (以降，HDD) の占有に関しては，別途対処が必要であった．Mint では，HDD をそれぞれの OS ノードが占有するが，HDD を管理するコントローラ (ata_host) は共有している．そのため，同じ ata_host の管理下に 2 つの HDD があり，かつ OS ノード間で分割占有する場合，後続として起動する OS ノードによる ata_host 操作処理が，先行する OS の HDD に影響を与える．このため，ata_host の操作時に，占有する HDD に関する部分のみを操作するように変更する．

3.3.4 Kexec-tools

Kexec-tools とは，Linux 高速な再起動 (Kexec) を利用するためのユーザコマンドである．再起動用のカーネルを実メモリに配置したり再起動処理を実行するシステムコールのフロントエンドとして動作する．Kexec-tools のうち改変を加えたファイルは 13 ファイル 234 行であった．

Kexec-tools は，起動するカーネルイメージを実メモリに配置するにあたって，前処理を担う (図 2(A) 参照)．Mint では，その前処理部分を改変することで，起動後のカーネルの動きを制御する．具体的には，Kexec-tools がカーネル初期化フックとして用意するルーチン (purgatory) を改変する．改変の 1 つは，カーネルが IPI によって新規に起動された直後はリアルモードで走行することへの対処である．Kexec は，動作中の OS の再起動しか想定していないため，purgatory は一貫してプロテクトモードで動作することを前提に用意されている．このため，走行モードをリアルモードからプロテクトモードへ切り替える処理を purgatory 内に追加した．また，走行中の OS ノードの走行環境保護のため，OS の終了処理，Local APIC の無効化，セグメントレジスタ，GDT，IDT，および汎用レジスタの初期化を行わないように purgatory を書換えた．

また，通常の Kexec は，自身と同一のカーネルを再起動するために用いるので，メモリマップを走行中のカーネルから取得する．Mint では，実メモリ分配のため，起動する OS ノードが外部から与えられたメモリマップを利用する．このメモリマップを作成してカーネルの初期データとして配置する処理を追加した．

4. Mint における混載の実例

4.1 Linux と Android の混載

4.1.1 概要

Linux の亜種のうちの 1 つである Android を Mint 上で混載する．Linux と Android を混載した際の利用例として，家電製品において制御用の OS として組込用途に改変された，UI を持たない Linux に加えて，UI 用の OS として Android を動作させることが考えられる．既存の家電製品では，高度な UI を持つ一方で高いリアルタイム性を要求され，制御が複雑である．したがってこのような場合，制御用の OS と UI 用の OS を用意し，別々の CPU 上で動作させることが一般的である．Mint では独立性とオーバヘッドが発生しないという性質をいかし，1 つの CPU 上でそれぞれの OS の性能を落とすことなく，制御用 OS と UI 用 OS を動作できる．

制御用 Linux と UI 用 Android の混載を実現するため，Android に 3.3.2 項と 3.3.3 項で述べた変更を適用する．Linux と Android ではアプリケーションの実行インタフェースが異なるため，Android 上では Kexec をそのまま使用できない．このため，Linux を OS ノード 0，Android を OS ノード 1 とし，Linux から Android を起動する構成とする．これは，UI を持たない OS が先に起動した後に UI を持つ OS が起動することを意味する．Linux の割り込み

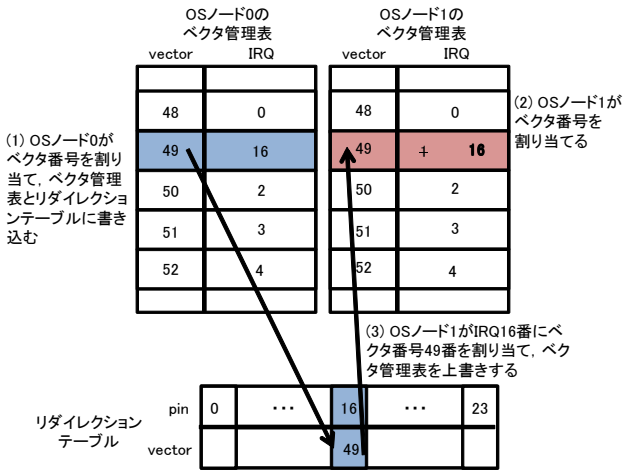

図 3 共有する IRQ 番号のベクタ番号割り当ての際にベクタ管理表が上書きされる例

ルーティング機構は，このような場合を想定していないため，Mint 実現時に問題が発生した．以降では，この問題の発生原因と対処について述べ，UI を持たない OS が先行する OS ノードである場合の混載事例として説明する．

4.1.2 問題点

ここでは，Android を混載する際に問題となる，ベクタ番号調停に関する割り込みルーティングの改変 (表 1 の分類 3) について説明する．Mint において，OS ノード間で割り込み (IRQ) を共有する場合には，OS ノード間で IRQ に対応するベクタ番号を揃えなければならないという制約がある．したがって，合わせるべきベクタ番号が既に他の割り込みのために使用されていた場合，問題が発生する．この問題の具体例として，共有する IRQ にベクタ番号を割り当てる様子を**図 3** に示し，以下で説明する．図 3 では，IRQ1 番は OS ノード 1 のみが利用する割り込みで，IRQ16 番は OS ノード間で共有する割り込みである．

(1) OS ノード 0 は，IRQ1 番は利用しないため，IRQ1 番に関しては，何もしない．次に，IRQ16 番に関してベクタ番号 49 番を割り当て，IRQ16 番とベクタ番号 49 番の対応を，ベクタ管理表と割り込みコントローラのリダイレクションテーブルに書き込む．

(2) OS ノード 1 は，IRQ1 番にベクタ番号 49 番を割り当て，IRQ1 番とベクタ番号 49 番の対応をベクタ管理表とリダイレクションテーブルに書き込む．

(3) さらに，OS ノード 1 は，IRQ16 番にベクタ番号を割り当てようとするが，既に OS ノード 0 によって IRQ16 番とベクタ番号 49 番の関係が設定されているため，これに従おうとして，IRQ16 番に OS ノード 0 と同じベクタ番号 49 番を割り当ててしまい，OS ノード 1 のベクタ管理表を上書きしてしまう．

これを回避するためには，OS1 がベクタ番号 49 番を IRQ1 番に割り当てる際に，既に 49 番が利用されていないかを確認すればよいが，ここで Linux 固有の問題がある．3.3.2(3) で述べたように，IRQ0 番から 15 番は，いわゆるレガシーデバイスと呼ばれ，ベクタ番号 48 から 63 番を優先的にレガシーデバイスに割り当てるようになっている．通常は，レガシーデバイスに対する IRQ 割り当て処理が完了した後に IRQ16 番以降の割り当てが行われるため，IRQ16 番以降がベクタ番号 48 番から 63 番と対応付けられることはない．しかし，Mint においては，レガシーデバイスを後続の OS のみが利用し，先に起動した OS は利用しない (つまり IRQ を設定しない) ため，上記の問題が発生する．

例では，Android が OS ノード 1 にあたり，後続で起動し，かつレガシーデバイスを利用している．このような場合に問題が起こる．

4.1.3 対処

以上に述べた問題が発生を避けるため，ベクタ番号を割り当てる際，以下の 2 つを守る必要がある．

(1) 共有する IRQ 番号には，同じベクタ番号を割り当てる．

(2) すでにリダイレクションテーブルで IRQ 番号に対応付けられたベクタ番号に対し，別の IRQ 番号を割り当てない．

Linux におけるベクタ番号の割り当て方法では，IRQ0 から IRQ15 番のうち使用しない割り込みがあった場合，ベクタ番号 48 から 63 番が未使用として空いてしまう．この際，ベクタ番号 48 から 63 番に IRQ16 番以降が割り当てられることがあり，4.1.2 項で述べた問題が発生する可能性がある．このため，IRQ0 から 15 番が割り当てられるベクタ番号を IRQ16 番以降に割り当てないように保護し，候補から除外する．

4.2 32/64bit Linux の混載

4.2.1 概要

CPU の走行モードは，プロセッサのコアごとに切り替え可能であることを利用して，マルチコアプロセッサを搭載した 1 台の計算機上で 32bit カーネルと 64bit カーネルを同時に走行させる [17]．これにより，32bit OS では得られない広大なメモリ領域と処理性能，64bit OS では利用不可能な 32bit OS 用のソフトウェアをシステム全体として使用可能になる．

32/64bit Linux 混載の利用例として，通常のソフトウェアは 64bit Linux 上で動作させ，対応ソフトウェアが 32 ビット用のものしかないデバイスを 32bit Linux に制御させることが考えられる．あるいは，ハードウェア制御を 32bit OS で行い，大量のメモリを必要とする処理が発生した場合に 64bit Linux を一時的に起動することがあげられる．この場合，64bit OS には，32bit OS では利用不可能な物理メモリ領域，具体的には先頭から 4GB 以降の物理メモリ領域を占有させる．そして，64bit OS に大量のメモリを必要とするプロセスを実行させる．プロセスの実行が完了した後は，OS 間通信を利用して実行結果を 32bit OS

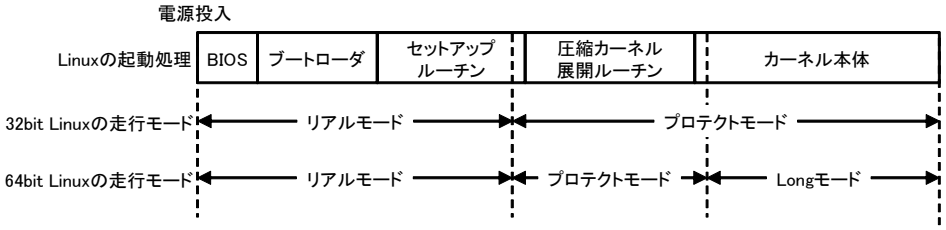

図 4 通常の Linux の起動処理と CPU の走行モードの関係

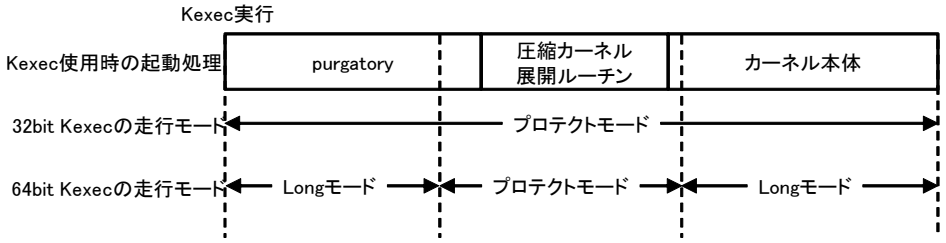

図 5 通常の Kexec 使用時の起動処理と CPU の走行モードの関係

に送信する．

32/64bit Linux の混載の実現において，課題となるのは 32bit OS と 64bit OS の走行モードの差であった．これについて，以下で説明する．

4.2.2 課題

32/64bit Linux の OS の走行モードの違いにより Kexec-tools (表 1 の分類 7) に関して発生する問題について説明する．Linux カーネルは x86 アーキテクチャと x86-64 アーキテクチャのソースコードの大部分を共有しているため，Mint カーネルは，そのまま両アーキテクチャに対応している．しかしながら，OS の走行モードが x86 と x86-64 では異なる部分で対処が必要であった．具体的には，32bit OS は Legacy モードのプロテクトモードで走行するのに対して，64bit OS は Long モードで走行し，個別のプロセスごとに 64bit モードか互換モードを選択する．また，IPI による起動直後のコアはリアルモードで走行する．これらの差を埋める処理が必要であった．以下に詳説する．

4.2.3 対処

まず，通常の Linux の起動処理と CPU の走行モードの関係を**図 4** に示す．起動直後のコアはリアルモードで走行し，BIOS とブートローダはリアルモードで走行する．そして，セットアップルーチンの後半にリアルモードからプロテクトモードに移行する．以降，32bit Linux の場合は，プロテクトモードの状態で圧縮カーネル展開ルーチンとカーネル本体を実行する．一方，64bit Linux の場合は，圧縮カーネル展開ルーチン実行後，カーネル本体の前半にプロテクトモードから 64bit モード (Long モード) に移行する．以降，プロテクトモード (Legacy モード) に戻ることは一般的にはない．

次に，通常の Kexec 使用時の起動処理と CPU の走行モードの関係を**図 5** に示す．Kexec は，BIOS，ブートローダ，およびセットアップルーチンの処理を省略する．これらの処理を実行する代わりに，Kexec は固有の前処理である purgatory を実行する．そして，purgatory の実行後はカーネル起動処理の圧縮カーネル展開ルーチンにジャンプする．

32bit Kexec の場合は，Kexec をプロテクトモードで実行し，プロテクトモードの状態で purgatory を実行する．一方，64bit Kexec の場合は，Kexec を Long モードで実行し，Long モードの状態で purgatory を実行する．purgatory 後の圧縮カーネル展開ルーチンはプロテクトモードで実行する必要がある．このため，purgatory 内で Long モードからプロテクトモードに移行する．

最後に，Mint における Kexec を利用した起動処理と CPU の走行モードの関係を**図 6** に示す．Mint における Kexec を利用した起動処理は，自コアではなく，IPI によって新しく起動した別のコアに purgatory 以降の処理を実行させる．起動直後のコアは，必ずリアルモードで走行するため，続く処理を実行させるためには，リアルモードからプロテクトモードに切り替える必要がある．このため，32bit Mint 用，64bit Mint 用の purgatory 内のいずれにもプロテクトモードへの切り替え処理を追加した．

5. おわりに

本稿では，Linux をベースとした複数の OS を混載する方式について述べた．まず，混載の目的について述べ，関連研究と比較した際の提案方式の利点について述べた．そして，提案方式を実現するために必要な改変について述べた．実現のためにカーネル部分に，資源分割，割り込み制御，および起動終了処理について改変を加えた．これに加え，デバイス振り分けに関してデバイスドライバに改変を加えた．また，他 OS ノード起動のため，Linux を高速に再

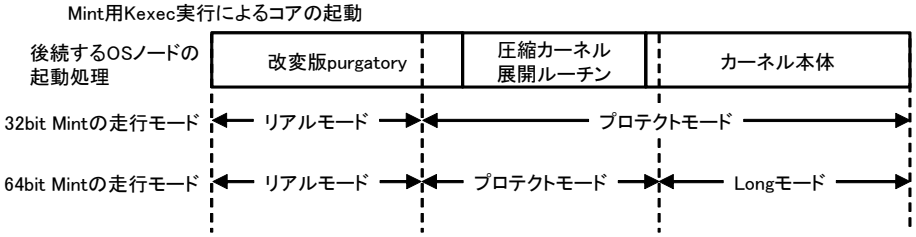

図 6 Mint における Kexec を利用した起動処理と CPU の走行モードの関係

起動する機能である Kexec を改変した．カーネルコードに対する改変量は 582 行であり，この変更はカーネルのコードの総数の 1%未満と少ないことを示した．デバイスドライバに対する改変量は 83 行であり，改変の対象となるファイルのコードの総数の 1.2%と少ない．Kexec-tools に対する改変量は 234 行であり，Kexec-tools のコードの総数の 0.1%と少ない．さらに，これらの改変は起動処理に集中していることがわかった．最後に Mint における混載の実例として，Linux と Android の混載と 32/64bit Linux の混載について述べた．Linux と Android の混載では，ベクタ番号割り当て時に OS ノード間でベクタ番号の上書きが発生するという問題について対処した．具体的にはレガシーデバイスに対応する IRQ 番号に割り当てられるベクタ番号を保護し，他の IRQ 番号に割り当てられないようにした．32/64bit Linux の混載では，32bit OS と 64bit OS の差として，CPU の走行モード切り換えについて対処した．これらの対処を行うことで，Linux と Android，32/64bit Linux の混載が実現可能であることを示した．

謝辞 本研究の一部は，科学研究費補助金基盤研究 (B)(課題番号：24300008) による．

参考文献

[1] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauery, Ian Pratt, Andrew Wareld：Xen and the Art of Virtualization，ACM SIGOPS Operating Systems Review, pp.164-177 (2003)

[2] Avi Kivity, Yaniv Kamay, Dor Laor, Uri Lublin, Anthony Liguori：kvm: the Linux Virtual Machine Monitor， Proceedings of the Linux Symposium，Vol. 1, pp.225-230 (2007)

[3] 佐藤雅英, 関口知紀, 新井利明, 井上太郎, 宮崎義弘, 中橋晃文, 梅都利和：ナノカーネル方式による異種 OS 共存技術 「DARMA」 の実装，全国大会講演論文集, 59(1) (1999)

[4] 新井利明, 関口知紀, 佐藤雅英, 井上太郎, 中村智明, 岩尾秀樹：ナノカーネル方式による異種 OS 共存技術「DARMA」の提案, 全国大会講演論文集, 59(1) (1999)

[5] Wataru Kanda, Yu Yumura, Yuki Kinebuchi Kazuo Makijima, Tatsuo Nakajima：SPUMONE: Lightweight CPU Virtualization Layer for Embedded Systems, Embedded and Ubiquitous Computing, IEEE/IFIP International Conference，Vol. 1, pp.144-151 (2008)

[6] Hitoshi Mitake, Tsung-Han Lin, Hiromasa Shimada, Yuki Kinebuchi, Ning Li, Tatsuo Nakajima：Towards Co-existing of Linux and Real-Time OSes, Linux Symposium, pp.55–68 (2011)

[7] Yuehua Dai，Yong Qi，Jianbao Ren，Yi Shi，Xiaoguang Wang，Xuan Yu：A Lightweight VMM on Many Core for High Performance Computing，Proceedings of the 9th ACM SIGPLAN/SIGOPS international conference on Virtual execution environments. ACM，pp.111–120 (2013)

[8] Masaki Tabuchi, Kenichi Itoh, Yoshinari Nomura, Hideo Taniguchi：Design and Evaluation of a System for Running Two Coexisting Linux Systems, IECIE Transactions on Information and Systems (Japanese Edition), pp.251–262 (2005)

[9] Jun Sun, Dong Zhou, Steve Longerbeam：Supporting Multiple OSes with OS Switching，USENIX Annual Technical Conference, pp. 357-362 (2007)

[10] Taku Shimosawa, Hiroya Matsuba, Yutaka Ishikawa：Logical Partitioning without Architectural Supports，Computer Software and Applications，COMPSAC'08，32nd Annual IEEE International, pp.355-364 (2008)

[11] Taku Shimosawa, Yutaka Ishikawa：Inter-kernel Communication between Multiple Kernels on Multicore Machines，IPSJ Online Transactions, 2, pp.261-279 (2009)

[12] Hariprasad Nellitheertha: Reboot Linux faster using kexec, developerWorks(online), available from 〈http:/www.ibm.com/developerworks/linux/library/l-kexec/index.html〉 (accessed 2013-02-07)

[13] Adhiraj Joshi，Swapnil Pimpale，Mandar Naik，Swapnil Rathi，Kiran Pawar：Twin-Linux: Running independent Linux Kernels simultaneously on separate cores of a multicore system，Proceedings of the Linux Symposium pp.101–108 (2010)

[14] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs,Simon Peter, Timothy Roscoe, Adrian Schüpbach, Akhilesh Singhania：The Multikernel: A New OS Architecture for Scalable Multicore Systems，Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles, pp.29–44 (2009)

[15] Dike Jeff：A user-mode port of the Linux kernel，Proceedings of the 2000 Linux Showcase and Conference, Vol. 2, No. 1 (2000)

[16] Dan Aloni：Cooperative Linux，Proceedings of the Linux Symposium，Vol. 2, pp.23-31 (2004)

[17] 中原大貴，乃村能成，谷口秀夫：32/64bit カーネル混載方式の実現，電子情報通信学会技術研究報告，vol.111，no.255，pp.25–30（2011）