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 Linux と Android の共存制御

指導教員

報 告 者

北川 初音

岡山大学工学部 情報工学科

平成 25 年 2 月 8 日 提出

要約

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 を走行させる方式が研究されている．この方式の 1 つに仮想計算機方式がある．しかし，仮想計算機方式では，計算機の仮想化によるオーバーヘッドが発生し，実計算機と比較して性能が低下する．

そこで，仮想計算機方式に対して，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 Mint 方式が研究開発されている．Mint では，CPU，メモリ，および入出力機器を分割し，仮想化によらず直接占有制御する．これにより，OS の独立性を実現し，仮想化によるオーバヘッドの問題を解決できる．

Mint において，用途の異なる OS を混載することで，OS 固有の特性が同時に使用可能になる．この一例として，本研究では，Mint において Linux と Android を混載する方式を提案する．Linux と Android を混載する際の利用例として，家電製品において制御用の OS として Linux，ユーザインタフェース用の OS として Android を動作させることが考えられる．Mint では OS 間の独立性とオーバヘッドが発生しないという性質を生かし，1 つの CPU 上でそれぞれの OS の性能を低下させることなく，制御用 OS とユーザインタフェース用 OS を動作できる．

本論文では，Mint における Linux と Android の混載について述べた．まず，Android と Linux を混載する方法について示した．次に，Linux と Android の混載時における割り込み制御の問題点について示した．具体的には，OS の起動後に割り込みが利用できなくなる問題について示し，対処としてレガシーデバイスを考慮した新たな割り込みベクタ調停方式を提案した．最後に評価により，混載した Android は，コードを改変していない未改変の Android と比較し，性能が低下しないことを示した．また，割り込みベクタ調停方式に関するコード変更量は，Mint 全体のコード変更量の 1%未満と小さいことを示した．

目次

1 はじめに **1**

2 Mint オペレーティングシステム **3**

2.1 設計方針 3

2.2 構成 3

2.3 実現方式 4

3 Linux と Android の混載 **6**

3.1 Android 6

3.2 混載の目的 6

3.3 構成 7

3.4 混載方法 7

4 割り込みベクタ調停方式 **9**

4.1 割り込み処理 9

4.1.1 APIC 9

4.1.2 割り込みがコアに通知されるまでの処理の流れ 9

4.1.3 割り込み処理を呼び出すまでの処理の流れ 10

4.2 Linux におけるベクタ番号割り当て方法 11

4.3 Mint におけるベクタ番号の割り当て方法 13

4.4 問題点 14

4.5 対処 15

4.5.1 対処の方針 15

4.5.2 OS ノード 0 のベクタ番号の割り当て方法の変更 16

4.5.3 OS ノード 1 以降のベクタ番号割り当て方法の変更 17

5 評価 **20**
5.1 評価項目 20
5.2 ベンチマークの測定 20
5.3 コード変更量 21

6 おわりに **23**

謝辞 **24**

参考文献 **25**

図 目 次

2.1 Mint の構成例 4

3.1 Android の構成 7
3.2 混載時の Linux と Android の構成 7

4.1 割り込みがコアに通知されるまでの処理の流れ 10
4.2 割り込み処理が呼び出されるまでの処理の流れ 11
4.3 Linux におけるベクタ番号の割り当て例 12
4.4 共有する IRQ 番号のベクタ番号割り当て例 14
4.5 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 16
4.6 対処後の OS ノード 0 のベクタ番号割り当て例 17
4.7 対処後の OS ノード 1 のベクタ番号割り当て例 19

表 目 次

5.1 評価環境 21
5.2 1 コア時の測定結果 21
5.3 2 コア時の測定結果 21
5.4 カーネルの変更量 22

第 1 章

はじめに

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 を走行させる方式が研究されている．この方式の 1 つに仮想計算機方式がある．しかし，仮想計算機方式では，計算機の仮想化によるオーバーヘッドが発生し，実計算機と比較して性能が低下する．

そこで，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 Multiple Independent operating systems with New Technology(以下，Mint) 方式 [1] が研究開発されている．Mint では，CPU，メモリ，および入出力機器を分割し，仮想化によらず直接占有制御する．これにより，OS 間の独立性を実現し，仮想化によるオーバヘッドの問題を解決できる．

Mint では，32/64bit Linux の混載を実現 [2] している．用途の異なる OS を混載することで，OS 固有の特性が同時に使用可能になる．これに対する更なる試みとして，本研究では，Mint において Linux と Android を混載する方式を提案する．Linux と Android を混載する際の利用例として，家電製品への適用が考えられる．この具体例として，制御用の OS として Linux，ユーザインタフェース (以下，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，OS と CPU を 2 つ用意することが一般的である．1 つの CPU 上でそれぞれの OS を動作させるため，仮想計算機方式を用いた場合，仮想化によるオーバヘッドが発生する．一方 Mint では，OS 間の独立性とオーバヘッドが発生しないという性質を生かし，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．

本論文では，Mint における Linux と Android の混載について述べる．まず，Android を混載する目的と方法について述べる．次に，Linux と Android の混載時における割り込み制御の問題点について示す．具体的には，OS の起動後に割り込みが利用できなくなる問題につ

いて示し，対処としてレガシーデバイスを考慮した新たな割り込みベクタ調停方式を提案する．最後に混載した Android の評価について述べる．

第 2 章

Mintオペレーティングシステム

2.1 設計方針

Mint は 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．本論文では 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) すべての OS ノードが相互に処理負荷の影響を与えない．

(2) すべての OS ノードが入出力性能を十分に利用できる．

2.2 構成

Mint では，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 2.1 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．

(1) CPU
コア単位で分割し，各 OS ノードで占有する．

(2) 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．

(3) 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．

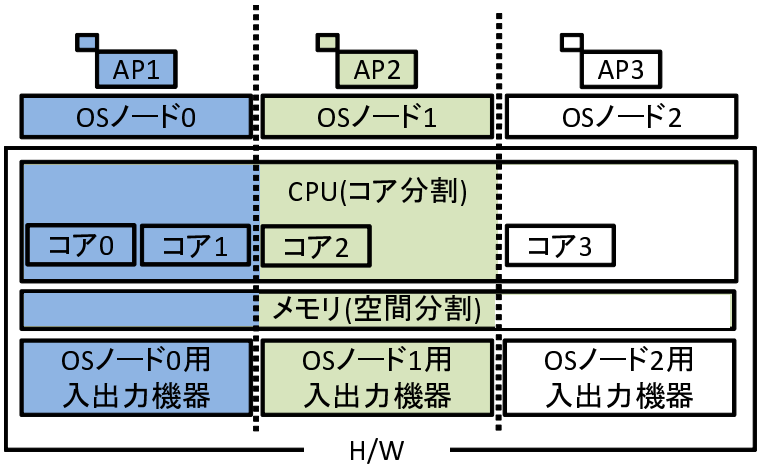

図　2.1　Mint の構成例

Mint では，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．

2.3　実現方式

Mint は Linux カーネルに以下の 8 つの変更を加えることで実現されている．

(1) コアの分割と占有
各 OS ノードの占有するコアに割り込みを正しく通知するため，論理 APIC ID の算出規則を変更している．論理 APIC ID とは，割り込みの通知先として用いられる ID である．

(2) 実メモリの分割と占有
実メモリ分配のため，メモリマップを書き換え，各 OS に重複しない実メモリ領域を認識させている．

(3) デバイスの分割と占有
デバイスの占有のため，占有しないデバイスの初期化を行わないようにしている．

(4) 割り込みの制御
割り込みを各 OS ノードに正しく通知するため，他の OS ノードへの割り込みの設定を操作しないようにしている．また，占有しないデバイスの割り込みを無効化している．

(5) 終了処理と再起動処理
終了処理については，他の OS ノードに影響を与えないように変更している．再起動処理については，他の OS ノードに影響を与えずに終了処理を行い，単独で起動する機能を追加している．

(6) OS ノード 1 以降の起動
Linux の高速な再起動方式である Kexec[3] を改変し，OS ノード 1 以降を起動する機能を実現している．

(7) コア管理機能
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．

(8) カーネルイメージの単一化
単一のカーネルイメージを用いて複数の OS を起動するため，ブートパラメータを追加し，起動する OS を指定可能にしている．

第 3 章

LinuxとAndroidの混載

3.1 Android

Android とは携帯情報端末を対象とする，オープンソースのソフトウェアの集合である．Android の構成図を図 3.1 に示し，以下で説明する．Android のカーネルは Linux カーネルに改変を加えたものであり，このカーネル上に，ライブラリ，ランタイム，アプリケーションフレームワーク，およびアプリケーションがある．本研究では，x86 アーキテクチャ上で動作する Android-x86[4] のカーネル部分を対象とする．

3.2 混載の目的

Mint では，32/64bit Linux の混載を実現している．Mint において用途の異なる OS を混載することで，OS 固有の特性が同時に使用可能になる．これに対するさらなる試みとして，本研究では，Mint において Linux と Android を混載する方式を提案する．Linux と Android を混載する際の利用例として，家電製品への適応が考えられる．この具体例として，制御用の OS として Linux，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．1 つの CPU 上でそれぞれの OS を動作させるため，仮想計算機方式を用いた場合，仮想化によるオーバヘッドが発生する．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生かし，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 とユーザインタフェース用 OS を動作できる．

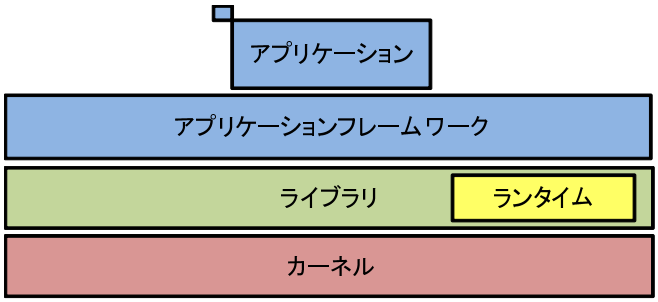

図　3.1　Android の構成

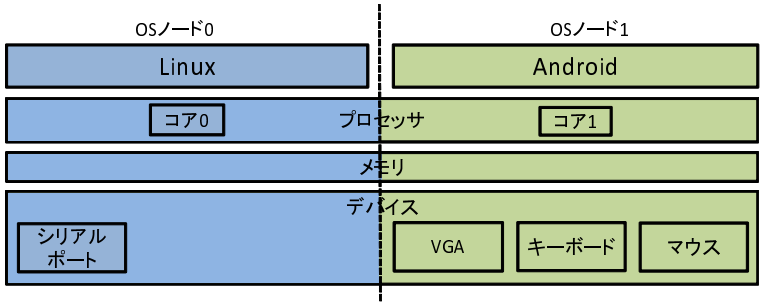

図　3.2　混載時の Linux と Android の構成

3.3　構成

Mint では，まず最初に OS ノード 0 が起動し，OS ノード 0 が OS ノード 1 以降を起動する．OS ノード 1 以降の起動には Kexec を利用する．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 Kexec をそのまま使用することはできない．このため，Android を OS ノード 0 としてしまうと，OS ノード 1 以降を起動できない．そこで，Mint で Linux と Android を混載する際は，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．また，Linux は CUI 環境，Android は GUI 環境で走行させる．このため，デバイスは Linux にシリアルポートを占有させ，Android に VGA，キーボード，およびマウスを占有させる．

3.4　混載方法

Mint において Linux と Android を混載するため，2.2 節で述べた変更を Android のカーネルに加える．これにより Android を OS ノード 1 として動作させる．しかし，3.3 節で述べた

構成で OS を走行させると，Mint では OS ノード 1 にキーボード割り込みが入らなくなるという問題が発生する．この問題の原因は，OS ノード 1 を起動する際の割り込みの設定にある．この問題について，4 章で割り込み処理の流れを述べ，割り込みベクタ調停方式を提案する．

第 4 章

割り込みベクタ調停方式

4.1　割り込み処理

4.1.1　APIC

割り込みコントローラである Advanced Programmable Interrupt Controller(以下，APIC) を利用する割り込み処理について説明する．割り込み処理の流れを以下の 2 つの段階に分ける．

(1) コアに割り込みが通知されるまでの処理

(2) 割り込みがコアに通知された後の割り込み処理を呼び出すまでの処理

以降の項でそれぞれについて説明する．

4.1.2　割り込みがコアに通知されるまでの処理の流れ

device A からの割り込みがコア 0 に通知されるまでの処理の流れを図 4.1 に示し，以下で説明する．

(1) device A から I/O APIC のピン番号 m に割り込みが通知される．

(2) I/O APIC はリダイレクションテーブルを利用し，通知先のコアを求める．通知先は destination に設定されたすべてのコアである．また，割り込みの通知されたピンのピン番号 m をベクタ番号 n に変換する．ベクタ番号とは，割り込みの要因を示す番号である．

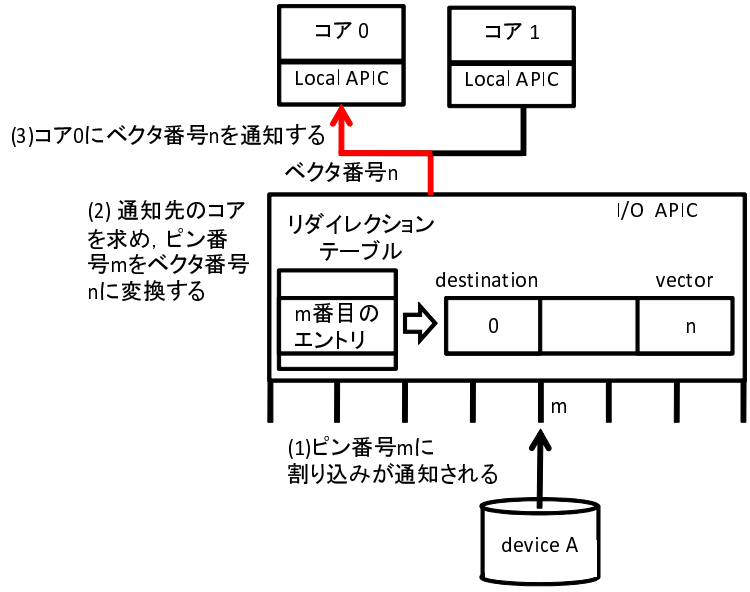

図　4.1　割り込みがコアに通知されるまでの処理の流れ

(3) I/O APIC はコア 0 にベクタ番号 n を通知する．

これらの処理により，ピン番号 m はベクタ番号 n に変換され，コア 0 に通知される．I/O APIC では IRQ 番号とピン番号は 1 対 1 で対応しており，ピン番号とベクタ番号は 1 対 1 で対応している．このため，1 つの IRQ 番号からは，1 つのベクタ番号しか生成できない．ここで，IRQ 番号とは，割り込み要求につけられた Linux で個別の割り込みを区別するための番号である．

4.1.3　割り込み処理を呼び出すまでの処理の流れ

4.1.2 項ではベクタ番号 n がコア 0 に通知されるまでの流れについて述べた．ここでは，割り込みを通知されたコアがベクタ番号を受け取った後，割り込み処理を呼び出すまでの処理の流れを図 4.2 に示し，以下で説明する．

(1) コア 0 はベクタ番号 n を受け取った後，Interrupt Descriptor Table(以下，IDT) の n 番目のエントリに登録された割り込みゲートを呼び出す．

(2) 割り込みゲートはベクタ番号 n を引数に，デバイスからの割り込みを処理する関数 do_IRQ() を呼び出す．

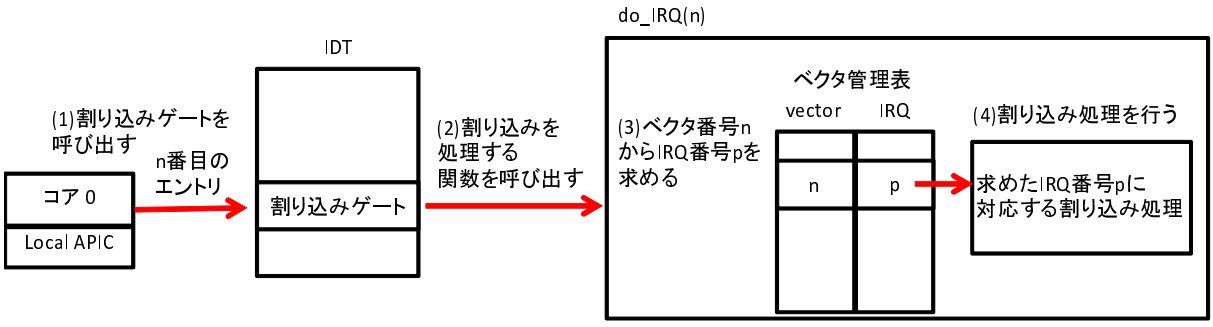

図　4.2　割り込み処理が呼び出されるまでの処理の流れ

(3) do_IRQ() はコア 0 に対応するベクタ管理表を用い，ベクタ番号 n から IRQ 番号 p を求める．

(4) OS は求めた IRQ 番号 p に対応する割り込み処理を行う．

上記の処理で示したように，割り込み処理を呼び出す際，do_IRQ() はベクタ番号から IRQ 番号を求める．これは，割り込み処理は IRQ 番号に対応しており，IRQ 番号にどのベクタ番号を割り当てるかは，OS の起動時に動的に決定されるためである．

4.2　Linux におけるベクタ番号割り当て方法

OS はベクタ番号を IRQ 番号に割り当て，ベクタ番号と IRQ 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．Linux におけるベクタ番号の割り当て例を図 4.3 に示し，以下で Linux におけるベクタ番号の割り当て方法について説明する．

(1) IRQ0〜IRQ15 番にベクタ番号を割り当てる．これは，IRQ0〜IRQ15 番はレガシーデバイスの割り込みであり，必ずベクタ番号を割り当てるためである．IRQ0〜IRQ15 番に割り当てられるベクタ番号は 48〜63 番と決められている．

(2) IRQ16 番以降は，ベクタ管理表を参照し，未使用の割り込みベクタ番号を探索する．探索したベクタ番号にすでに IRQ 番号が割り当てられている場合，未使用のベクタ番号が見つかるまで，ベクタ番号に 8 を足し，探索を続ける．この際，ベクタ番号 48〜63 番はすでに使用されているため，IRQ16 番以降に割り当てられない．

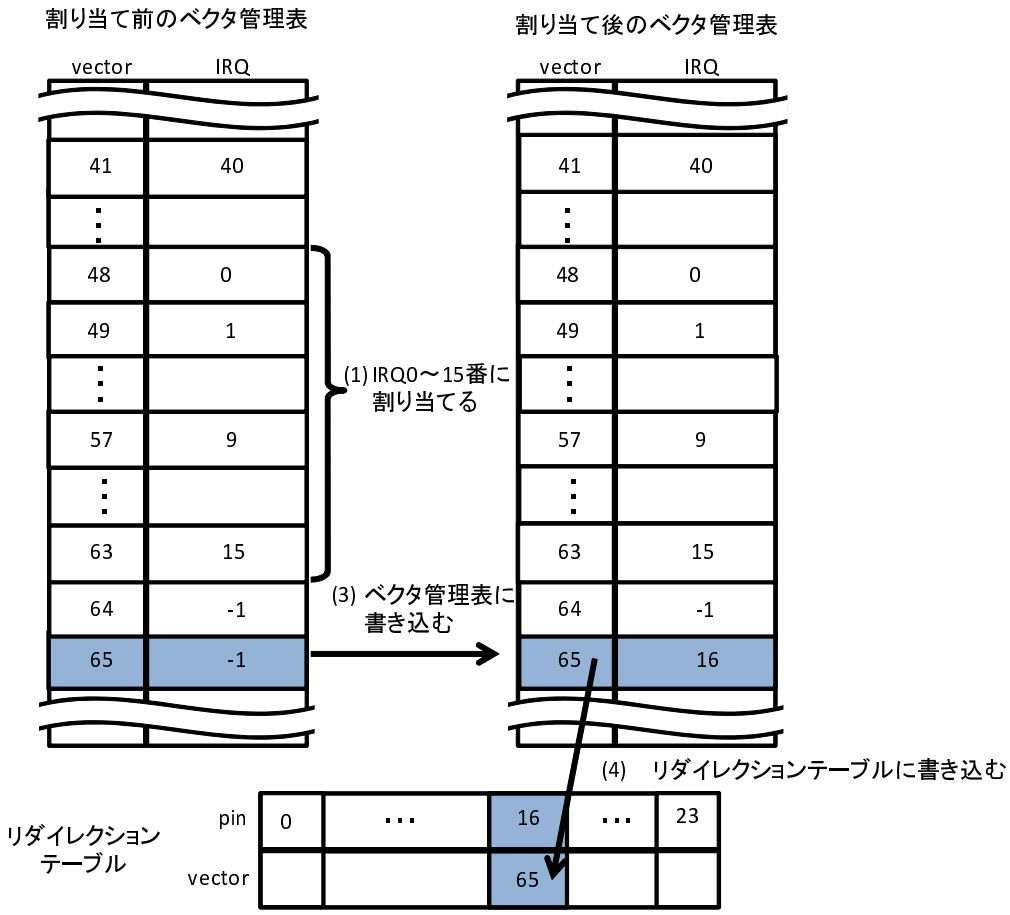

図　4.3　Linux におけるベクタ番号の割り当て例

(3) 未使用のベクタ番号が見つかった場合，そのベクタ番号に IRQ 番号を割り当て，ベクタ管理表に書き込む．

(4) ベクタ番号をリダイレクションテーブルのエントリに書き込む．書き込むエントリは，当該 IRQ を受け取る I/O APIC のピンが持つエントリである．

図 4.3 では，IRQ16 番にベクタ番号を割り当てる様子を示している．まず OS は IRQ0～IRQ15 番にベクタ番号 48 番～63 番を割り当てる．次に OS は IRQ16 番にベクタ番号を割り当てるため，ベクタ管理表を探索する．ベクタ番号 41 番から探索をはじめ，未使用のベクタ番号としてベクタ番号 65 番を見つける．その後，ベクタ番号 65 番を IRQ16 番に割り当て，ベクタ番号 65 番と IRQ 番号 16 番の対応をベクタ管理表と I/O APIC のリダイレクションテーブルに書き込む．

4.3　Mint におけるベクタ番号の割り当て方法

Mint におけるベクタ番号の割り当て方法は，OS ノード 0 と OS ノード 1 以降で異なる．また，Mint ではそれぞれの OS ノードで利用しない割り込みの場合，レガシーデバイスの割り込みであってもベクタ番号を割り当てない．Mint における OS ノード 0 と OS ノード 1 以降のベクタ番号の割り当て方法について以下で説明する．

(1) OS ノード 0 のベクタ番号の割り当て方法
OS ノード 0 では，4.2 節で述べた Linux での割り当て方法と同様にベクタ番号が割り当てられる．ただし，OS ノード 0 で利用しない割り込みの場合，レガシーデバイスの割り込みであってもベクタ番号を割り当てない．

(2) OS ノード 1 以降のベクタ番号の割り当て方法
Mint では，OS ノード間で割り込みを共有するデバイスとしないデバイスがある．例えば，タイマは割り込みを共有するデバイスであり，キーボードは割り込みを共有しないデバイスである．OS ノード 1 以降では，OS ノード間で割り込みを共有するか否かでベクタ番号の割り当て方が異なる．ただし，IRQ0〜IRQ15 番は OS ノード 0 と共有するか否かに関係なく，決まったベクタ番号に割り当てられる．以下で OS ノード 1 以降の割り当て方法について説明する．

(A) OS ノード間で割り込みを共有しない場合
4.2 節で述べた Linux での割り当て方と同様に割り当てる．

(B) OS ノード間で割り込みを共有する場合
割り込みは割り込みを共有す各 OS ノードのコアに通知される．この際，4.1.2 項で述べたように，I/O APIC では 1 つの IRQ 番号から 1 つのベクタ番号しか生成できない．このため，それぞれのコアには同じベクタ番号が通知される．したがって，OS ノード間で共有する割り込みの IRQ 番号 (以下，共有する IRQ 番号) には OS ノード間で同じベクタ番号を割り当てる必要がある．共有する IRQ 番号のベクタ番号の割り当て例を図 4.4 に示し，以下で説明する．

(a) OS ノード 1 はリダイレクションテーブルのエントリから IRQ 番号に対応するベクタ番号を読み込む．このエントリは IRQ 番号に対応するピンが持つエントリである．

(b) 読み込んだベクタ番号を IRQ 番号に割り当て，ベクタ管理表に書き込む．

図 4.4 では，IRQ16 番にベクタ番号を割り当てる例を示す．IRQ16 番は共有する IRQ 番号とする．OS ノード 1 は IRQ16 番のベクタ番号の割り当ての際，リダイ

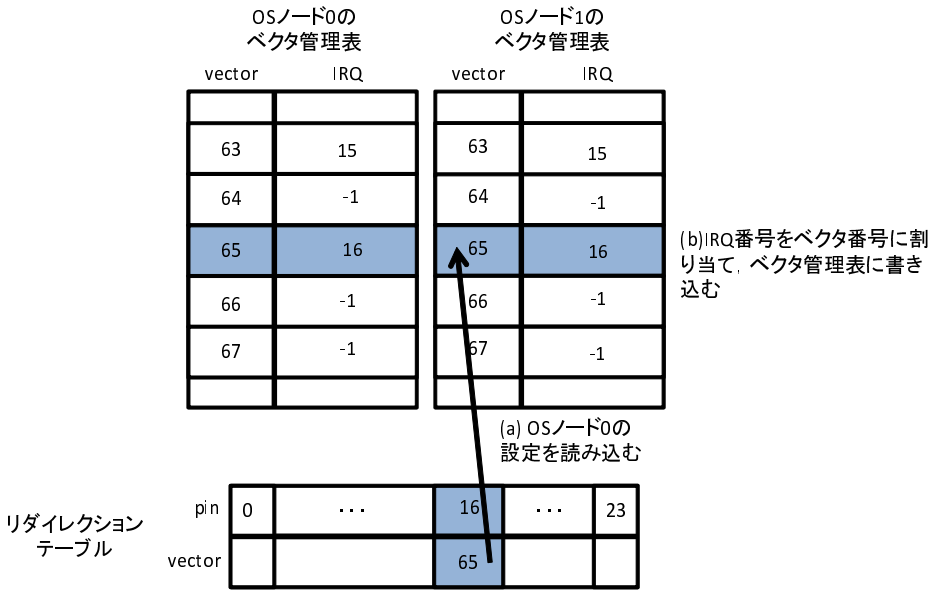

図　4.4　共有する IRQ 番号のベクタ番号割り当て例

レクションテーブルのエントリから IRQ16 番に対応するベクタ番号 65 番を読み込む．IRQ16 番にベクタ番号 65 番を割り当て，OS ノード 1 のベクタ管理表に書きこむ．

4.4　問題点

Mint において，共有する IRQ 番号には，割り込みを共有する OS ノード間で同じベクタ番号を割り当てる必要があるという制約がある．共有する IRQ 番号にベクタ番号を割り当てる際，制約を守るために，ベクタ管理表の設定を上書きしてしまうことがある．この問題の具体例として，IRQ16 番にベクタ番号を割り当てる様子を図 4.5 に示し，以下で説明する．図 4.5 では，IRQ1 番は OS ノード間で共有しない割り込みとする．また，IRQ16 番は OS ノード間で共有する割り込みとする．

(1) 4.3 節で述べた方法を用いて，OS ノード 0 は利用する割り込みの IRQ 番号にベクタ番号を割り当てる．また，IRQ 番号とベクタ番号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．4.3(1) で述べたように，通常ベクタ番号 49〜63 番をレガシーデバイスに割り当てるが，Mint では利用しないことがあるため，IRQ16 番以降に割り

当てられることがある．図 4.5 では，IRQ1 番は OS ノード 0 で利用しない割り込みである．このため，OS ノード 0 において，IRQ1 番にはベクタ番号が割り当てられず，ベクタ番号 49 番は未使用のままとなる．IRQ16 番の割り当てを行う際の探索時に，ベクタ番号 49 番が未使用のベクタ番号として見つかり，IRQ16 番にベクタ番号 49 番を割り当てる．

(2) 4.3 節で述べた方法を用いて，OS ノード 1 は利用する割り込みの IRQ 番号にベクタ番号を割り当てる．また，IRQ 番号とベクタ番号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．図 4.5 では，IRQ1 番は OS ノード 1 で利用する割り込みである．このため，IRQ1 番にはベクタ番号 49 番を割り当てる．

(3) 図 4.5 では，IRQ16 番は OS ノード間で共有する割り込みである．制約を守るため，OS ノード 1 は IRQ16 番に OS ノード 0 と同じベクタ番号 49 番を割り当て，ベクタ管理表に書き込む．この際，OS ノード 1 では，ベクタ番号 49 番には IRQ1 番が割り当てられているため，IRQ16 番の割り当てにより，OS ノード 1 のベクタ管理表を上書きしてしまう．

このようにベクタ管理表が上書きされてしまう場合，OS ノード 1 において上書きされた IRQ 番号の割り込みが利用できなくなる．

4.5 対処

4.5.1 対処の方針

4.4 節で述べた問題に対処する方針について述べる．Mint において，共有する IRQ 番号には，それぞれの OS ノードで同じベクタ番号を割り当てる必要がある．しかし，この制約を守るため，4.4 節で述べた問題が発生することがある．この問題を避けるため，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．

つまり，共有する IRQ 番号に割り当てたベクタ番号を異なる IRQ 番号に割り当てないようにする必要がある．この条件を満たすために，ベクタ番号の割り当て方法を変更する．Mint

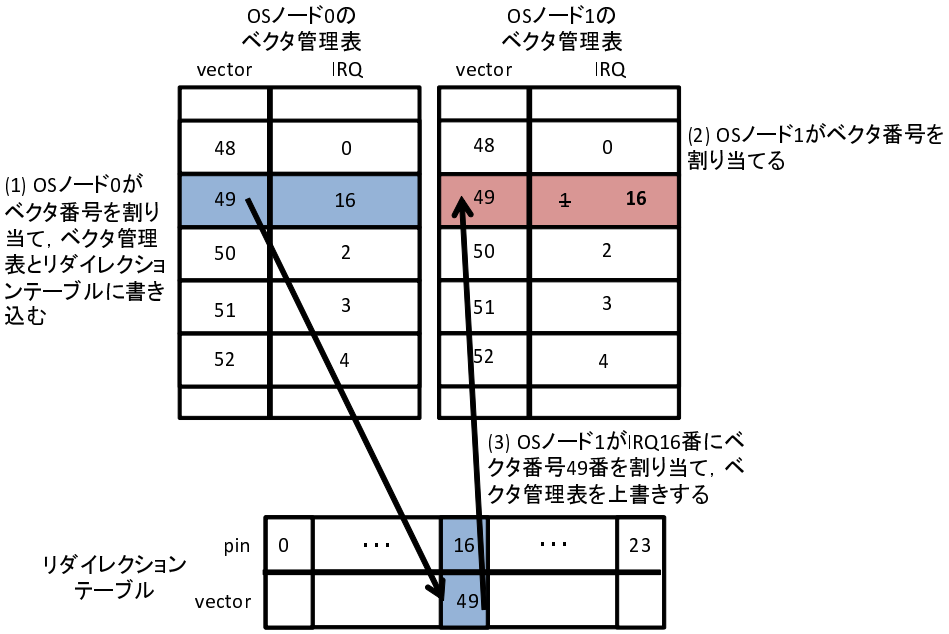

図　4.5　共有する IRQ 番号のベクタ番号割り当ての際にベクタ管理表が上書きされる例

においてベクタ番号の割り当て方法は OS ノード 0 とその他の OS ノードで異なるためそれぞれに対処する．

4.5.2　OS ノード 0 のベクタ番号の割り当て方法の変更

従来の Mint におけるベクタ番号の割り当て方法では，IRQ0～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 48～63 番に未使用のベクタ番号が発生する．この際，ベクタ番号 48～63 番に IRQ16 番以降が割り当てられることがあり，4.4 節で述べた問題が発生する可能性がある．このため，IRQ0～IRQ15 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する必要がある．そこで，IRQ0～IRQ15 番に割り当てるベクタ番号を保護し，IRQ16 番以降には，IRQ0～IRQ15 番に割り当てるベクタ番号を除いたベクタ番号を割り当てる．

変更後のベクタ番号の割り当て例を図 4.6 に示し，以下で説明する．図 4.6 では OS ノード 0 において IRQ16 番にベクタ番号を割り当てる様子を示す．

(1) IRQ0～IRQ15 番にベクタ番号 48～63 番を割り当てる．ただし，利用しない割り込みに関してはベクタ番号を割り当てない

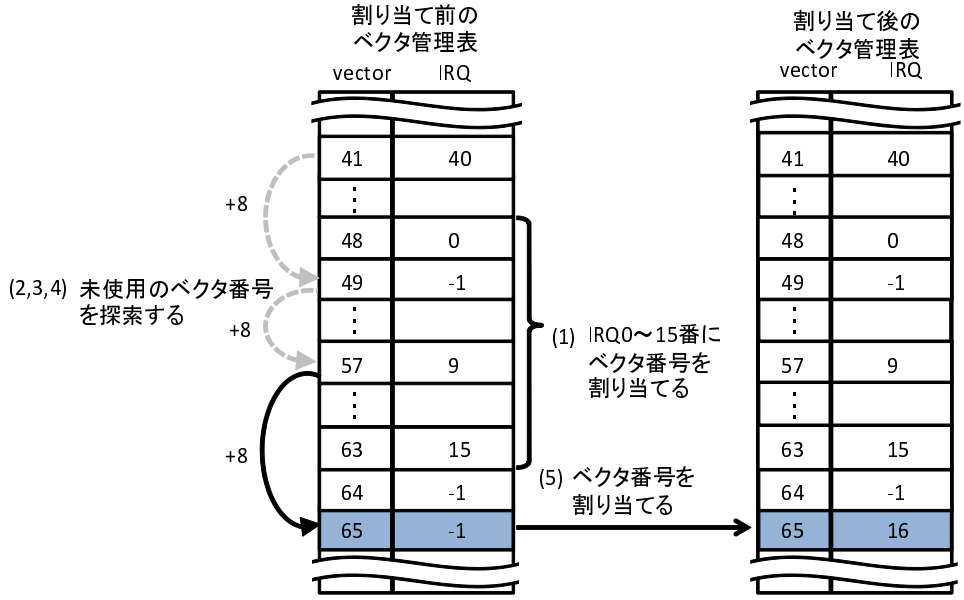

図　4.6　対処後の OS ノード 0 のベクタ番号割り当て例

(2) IRQ16 番を割り当てるために，未使用のベクタ番号を探索する．未使用のベクタ番号としてベクタ番号 49 が見つかる．

(3) ベクタ番号 49 番は，IRQ0～IRQ15 番が割り当てられるベクタ番号である．このため，ベクタ番号を割り当てずに探索を続ける．

(4) 未使用のベクタ番号として，ベクタ番号 65 番が見つかる．

(5) ベクタ番号 65 番は，IRQ0～IRQ15 番が割り当てられるベクタ番号ではないため，IRQ16 番にベクタ番号 65 番を割り当てる．

このように，IRQ0～IRQ15 番に割り当てるベクタ番号には IRQ16 番以降を割り当てないようにする．

4.5.3　OS ノード 1 以降のベクタ番号割り当て方法の変更

OS ノード 1 以降のベクタ番号割り当てでは，共有する IRQ 番号に OS ノード間で同じベクタ番号を割り当てなくてはならない．このため，OS ノード間で共有しない割り込みの IRQ 番号 (以下，共有しない IRQ 番号) に，共有する IRQ 番号に割り当てるベクタ番号を割り当

ててはならない．変更後の割り当て方法では，共有しない IRQ 番号にベクタ番号を割り当てる際，割り当てようとしているベクタ番号が共有する IRQ 番号に割り当てられているか否か確認する．共有する IRQ 番号に割り当てるベクタ番号か否かは，リダイレクションテーブルを用いて確認する．共有する IRQ 番号を割り当てるベクタ番号の場合，このベクタ番号が書かれたエントリがリダイレクションテーブルにある．確認した結果，共有する IRQ 番号を割り当てるベクタ番号ではない場合のみ割り当てる．

変更後の OS ノード 1 の割り込みのベクタ番号の割り当て例を図 4.7 に示し，以下で説明する．図 4.7 では IRQ17 番にベクタ番号を割り当てる様子を示す．ここで，IRQ16，IRQ18 番は OS ノード間で共有する割り込みであり，IRQ17 番は OS ノード間で共有しない割り込みとする．

(1) IRQ17 番を割り当てるために，未使用のベクタ番号を探索する．未使用のベクタ番号として 73 番が見つかる．

(2) リダイレクションテーブルを探索し，ベクタ番号 73 番が共有する割り込みを割り当てるベクタ番号か否か確認する．

(3) リダイレクションテーブルでベクタ番号 73 番が書かれたエントリが見つかる．このため，ベクタ番号 73 番は，OS ノード 0 と共有する割り込みが割り当てられるベクタ番号である．

(4) 未使用のベクタ番号の探索を続け，未使用のベクタ番号としてベクタ番号 81 番を見つける．

(5) リダイレクションテーブルにベクタ番号 81 番が書かれたエントリが見つからないため，ベクタ番号 81 番は OS ノード 0 で使用していないベクタ番号である．

(6) IRQ17 番にベクタ番号 81 番を割り当てる．

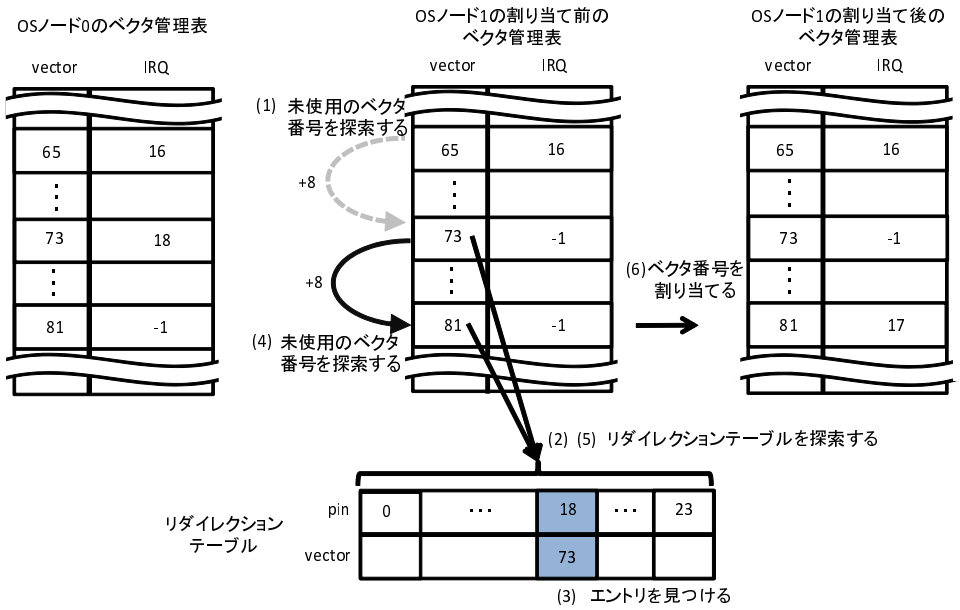

図　4.7　対処後の OS ノード 1 のベクタ番号割り当て例

第 5 章

評価

5.1 評価項目

Mint において混載した Android を以下の 2 つの項目で評価する．

(評価 1) ベンチマークの測定

混載した Android の性能評価として，ベンチマークを測定し，未改変の Android と比較する．

(評価 2) コード変更量

割り込みベクタ調停方式に関するコードの変更量を調査する．Mint では，Linux カーネルのバージョンアップへの追従を容易にするため，カーネルの変更量を抑えたいという要求がある．そこで，コードの変更量を調査し，割り込みベクタ調停方式を少ない工数で実現できることを示す．

5.2 ベンチマークの測定

評価は表 5.1 に示す環境で行った．ベンチマークを測定し，測定結果を未改変の Android と Mint に混載した Android で比較する．ベンチマークは，Android で動作するベンチマークのアプリケーションでよく用いられる 3 つを使用した．具体的には AnTuTu，Geekbench 2，Quadrant の 3 つである．これらは，CPU，メモリ，I/O の性能を測定し，測定結果として得点を算出する．混載した Android は Linux と同時走行している状態で測定する．測定結果を表 5.2 に示す．なお，測定は 5 回ずつ行い平均値を算出した．また，コアの数を 2 個に増やして測定を行った際のベンチマークの測定結果を表 5.3 に示す．

表 5.1 評価環境

OS	Android 4.0
CPU	Intel Core i7-870 @ 2.93GHz
使用するコアの数	1〜2
メモリ	608MB

表 5.2 1 コア時の測定結果

	未改変の Android	混載した Android
AnTuTu	19396.2	19392
Geekbench 2	3494.6	3494.4
Quadrant	6591.4	6589.8

表 5.3 2 コア時の測定結果

	未改変の Android	混載した Android
AnTuTu	36252.4	38242.4
Geekbench 2	4488.6	4484.4
Quadrant	7468.2	7467.8

未改変の Android の測定結果と，混載した Android の測定結果に大きな差はない．また，使用するコアの数を増やして動作させても測定結果に大きな差はない．これらの結果から，Mint において Linux と混載しても混載による影響はなく，Android の性能は低下しないことがわかる．

5.3 コード変更量

Mint 全体でのコード変更量と割り込みベクタ調停方式に関するコード変更量を比較する．これらのコード変更量を表 5.4 に示す．Mint 全体のコード変更量は 664 行であり，割り込みベクタ調停方式に関するコード変更量は 9 行である．割り込みベクタ調停方式に関するコード変更量は Mint 全体のコード変更量の 0.013%である．これらの結果から，割り込みベクタ調停方式は少ない工数で実現できていることがわかる．

表 5.4 カーネルの変更量

評価対象	行数	ファイル数
Mint 全体	673 行	43
割り込みベクタ調停方式	9 行	1

第 6 章

おわりに

本論文では，Mint における Linux と Android の混載について述べた．まず，Android を混載する目的と方法について述べた．次に，Linux と Android の混載時に割り込みが利用できなくなる問題について示し，対処として割り込みベクタ調停方式を提案した．最後に混載した Android について評価した．

具体的には，Mint の変更点 8 点を示し，Android の混載も同様の変更で行えることを示した．Linux と Android の混載時に割り込みが利用できなくなる問題について対処として割り込みベクタ調停方式を提案した．

提案した割り込みベクタ調停方式は，共有する IRQ 番号に割り当てたベクタ番号を異なる IRQ 番号に割り当てないようにするという方針に従い，ベクタ番号の割り当て方法を変更するものである．

評価として，ベンチマークを測定した．測定結果より，混載した Android は，コードを改変していない未改変の Android と比較し，性能が低下しないことを示した．また，割り込みベクタ調停方式に関するコード変更量は，Mint 全体のコード変更量の 1%未満と小さいことを示した．

残された課題として，Android から Linux を起動できないという問題への対処がある．

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] 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）.

[2] 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）.

[3] 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).

[4] Android-x86: Android-x86 Project - Run Android on Your PC, Android-x86 - Porting Android to x86(online), available from ⟨http://www.android-x86.org/⟩ (accessed 2013-02-07).