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デバイス接続状況の把握法

指導教員

報 告 者

左海 裕庸

岡山大学工学部 情報工学科

平成23年2月14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を走行させる方式が活発に研究されている．この方式の1つとして，ハードウェアの仮想化により1台の計算機上に複数のOSを走行させる仮想計算機方式がある．例えば，VMwareやXenがある．仮想計算機方式では，入出力要求をハードウェアへ直接送らず，仮想化したハードウェアを通すため，オーバーヘッドが存在する．また，マルチコアプロセッサ上で複数のOSが独立に走行する方式として，Mint(Multiple Independent operating systems with New Technology)が研究開発されている．Mintでは，占有したデバイスを直接制御するため，仮想計算機方式と比べ，入出力性能が高い．

Mintの各OSは，起動時に指定されたデバイスしか利用できない．Mintの各OS間で占有デバイスを移譲したいという要求がある．この要求に対処するためには，各OSの割り込みルーティングを把握する必要がある．つまり，各OSのデバイス接続状況の把握が必要である．

本論文では，Linuxに実装されている仮想化手法であるKVM(Kernel-based Virtual Machine)の割り込み制御を解析し，KVMの割り込み制御を参考に割り込み管理層を実現する．ここで，実現の際にKVMとMintの違いを考慮する必要がある．具体的には，KVMがQEMUと呼ばれるアプリケーションと連携して割り込みを管理していることに対して，MintはLinux単体とほぼ同様の機構によって割り込み処理を実現している点である．また，KVMはCPUの仮想化支援機能を利用している．これに対し，Mintは仮想化支援機能を使用していない．これらの違いによって生じる課題を検討し，Mintオペレーティングシステムにおけるデバイス接続状況の把握法を設計した．

目次

図 目 次

表 目 次

第 1 章

はじめに

計算機の高性能化により，1 台の計算機上に複数の OS を走行させる方式が活発に研究されている．この方式の 1 つとして，VMware[1] や Xen[2] といったハードウェアの仮想化により 1 台の計算機上に複数の OS を走行させる仮想計算機方式がある．仮想計算機方式では，入出力要求をハードウェアへ直接送らず，ソフトウェアで再現された仮想的なハードウェアを通すため，オーバーヘッドが存在する．また，マルチコアプロセッサ上で複数の OS が独立に走行する方式として，Mint(Multiple Independent operating systems with New Technology)[3] 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，占有したデバイスを直接制御するため，仮想計算機方式と比べ，入出力性能が高い．

Mint の各 OS は，起動時に指定されたデバイスしか利用できない．このため，起動後に Mint の各 OS 間で占有デバイスを移譲したいという要求がある．この要求に対処するためには，各 OS の割り込みルーティングを把握する必要がある．つまり，各 OS のデバイス接続状況の把握が必要である．

本論文では，デバイス接続状況の把握を実現するために，ハードウェアと OS の間に割り込みを集約し，管理する層を設計する．この設計は，仮想化に似た概念である．Linux に実装されている仮想化手法である KVM(Kernel-based Virtual Machine)[4] の割り込み制御を解析し，KVM の割り込み制御を参考に割り込み管理層を実現する．ここで，実現の際に KVM と Mint の違いを考慮する必要がある．具体的には，KVM が QEMU と呼ばれるアプリケーションと連携して割り込みを管理していることに対して，Mint は Linux 単体とほぼ同様の機構によって割り込み処理を実現している点である．また，KVM は CPU の仮想化支援機能を利用している．これに対し，Mint は仮想化支援機能を使用していない．これらの違いによって生じる課題を検討する．

第 2 章

背景

2.1 Mint オペレーティングシステム

2.1.1 概要

Mint は，マルチコアプロセッサ上で複数の Linux カーネルが独立に走行する方式である．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．

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分割し，占有する必要がある．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．CPU については，マルチコアプロセッサを対象とし，コア毎に分割して 1 つ以上のコアを各 OS が占有する．メモリについては，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．入出力機器については，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占有する．

2.1.2 Mint における割り込み制御

図 2.2 に Mint における割り込み制御を示し，以下で説明する．割り込みの種類によっては，各カーネル間で割り込みを共有する必要がある．図 2.2 の入出力機器の分割例では，SATA に OS1 の HDD と OS2 の HDD が接続されているため，これらは割り込みを共有しなければならない．

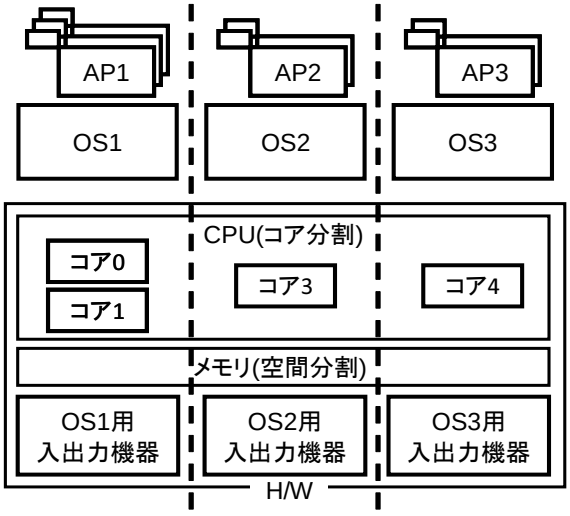

図 2.1 Mint の構成例

また，割り込みの通知先について，Linux カーネルは起動時に I/O APIC の設定を一旦クリアし，自身の占有するデバイスに対して，自身のコアに割り込みを通知するよう設定する．このとき，先に起動したカーネルが I/O APIC に設定した内容は，後に起動したカーネルに上書きされてしまう．このため，各カーネルについて，他カーネルのコアに関する設定内容を操作しないようにする．具体的には，既に I/O APIC に設定されている割り込みの割り込み先と設定しようとする割り込み先の両方に通知するように変更する．

さらに，I/O APIC の各割り込みには，対応するベクタ番号がある．ベクタ番号は，各割り込みに対して 1 つのみ設定できるため，割り込みを共有するカーネル間で同じ値を用いる必要がある．このため，Mint の各カーネルは，既にベクタ番号が設定されている割り込みに対しては，I/O APIC からベクタ番号の設定を読み出し，同じベクタ番号を利用するように変更する．

Mint の割り込み制御では，各 OS ごとに I/O APIC を設定する．このとき，他の OS の割り込み設定を意識しなければならない．このため，割り込み制御が複雑である．

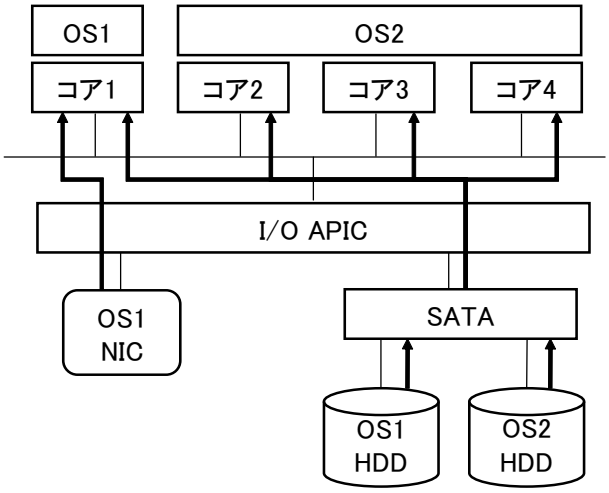

図 2.2 Mint における割り込み制御

2.2 目的

Mint の各 OS は，起動時に指定されたデバイスしか利用できない．例えば，図 2.2 では，OS1 は NIC を使用できるが，OS2 では NIC を使用できない．そこで，Mint の各 OS 間で占有デバイスの移譲を行いたいという要求がある．この要求を満たすためには，割り込みルーティングの把握が必要である．つまり，各 OS のデバイスの接続状況の把握が必要である．Mint では，割り込み設定時に他の OS の割り込み設定を意識する必要があるため，割り込み制御が複雑である．また，デバイスの移譲には他の OS の割り込み設定を知る必要がある．各 OS は，他の OS を意識せず走行できることが望ましい．そこで，デバイスの接続状況の把握を実現するために，ハードウェアと OS の間に割り込みを集約し，どの OS へ割り込みを通知するか管理する層を設計する．本研究の目的は，割り込みを集約して管理する層をハードウェアと Mint の間に構成することである．この割り込み管理層は，割り込みを直接 OS に通知せず，割り込み管理テーブルによって，割り込みを OS へ通知する点が仮想化による割り込み管理に似た構造である．したがって，本研究では，仮想化手法の 1 つである KVM の割り込み制御を解析し，応用する．この際，KVM と Mint の構成の違いを考慮に入れた設計変更についての検討が課題となる．

第 3 章

KVM

3.1 概要

Linuxで仮想計算機の機能を実現するソフトウェアとしてKVMとQEMUがある．KVMとQEMUによる仮想化の様子を 図3.1 に示す．KVMはLinuxカーネル内に存在し，以下の5つの機能を持つ．

(1) 仮想マシンの作成

(2) 仮想マシンへのメモリ割り当て

(3) 仮想CPUのレジスタの読み書き

(4) 仮想CPUへの割り込みの挿入

(5) 仮想CPUの実行

これに対し，QEMUはLinuxカーネル外に存在し，仮想デバイスのエミュレーション機能を持つ．

QEMUは，KVMの機能を利用するためにioctlを発行する．ioctlによって提供されるKVMの機能のうち，割り込みに関係するものは5種類ある．それぞれの概要と対応するリクエストコードを表3.1に示す．

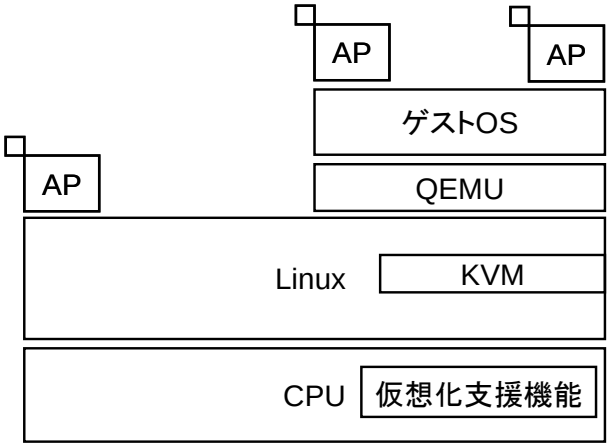

図 3.1 KVM と QEMU による仮想化の様子

表 3.1 KVM の機能と対応するリクエストコード

リクエストコード	概要
KVM_CREATE_IRQCHIP	割り込みコントローラの作成
KVM_SET_GSI_ROUTING	割り込みコントローラの構成の設定
KVM_CREATE_VCPU	仮想 CPU の作成
KVM_IRQ_LINE_STATUS	割り込みコントローラに割り込みをセット
KVM_RUN	ゲスト OS の実行

3.2 KVM における割り込み制御

3.2.1 KVM が仮想化する割り込みとハードウェアの構成

図 3.2 に KVM が仮想化する割り込みコントローラの構成を示し，以下で説明する．割り込みコントローラには，APIC と PIC が存在する．APIC は，I/O APIC と Local APIC によって構成される．I/O APIC には 24 本の IRQ ラインが存在し，ICC(Interrupt Controller Communication) バスを通して各 CPU が持つ Local APIC へ割り込みを送信する．また，I/O APIC には，各 IRQ ラインごとにリダイレクションテーブルが存在する．リダイレクションテーブルには，割り込みに対応するベクタ番号，割り込み通知先として APIC ID が保存される．この APIC ID により割り込み通知先の Local APIC を

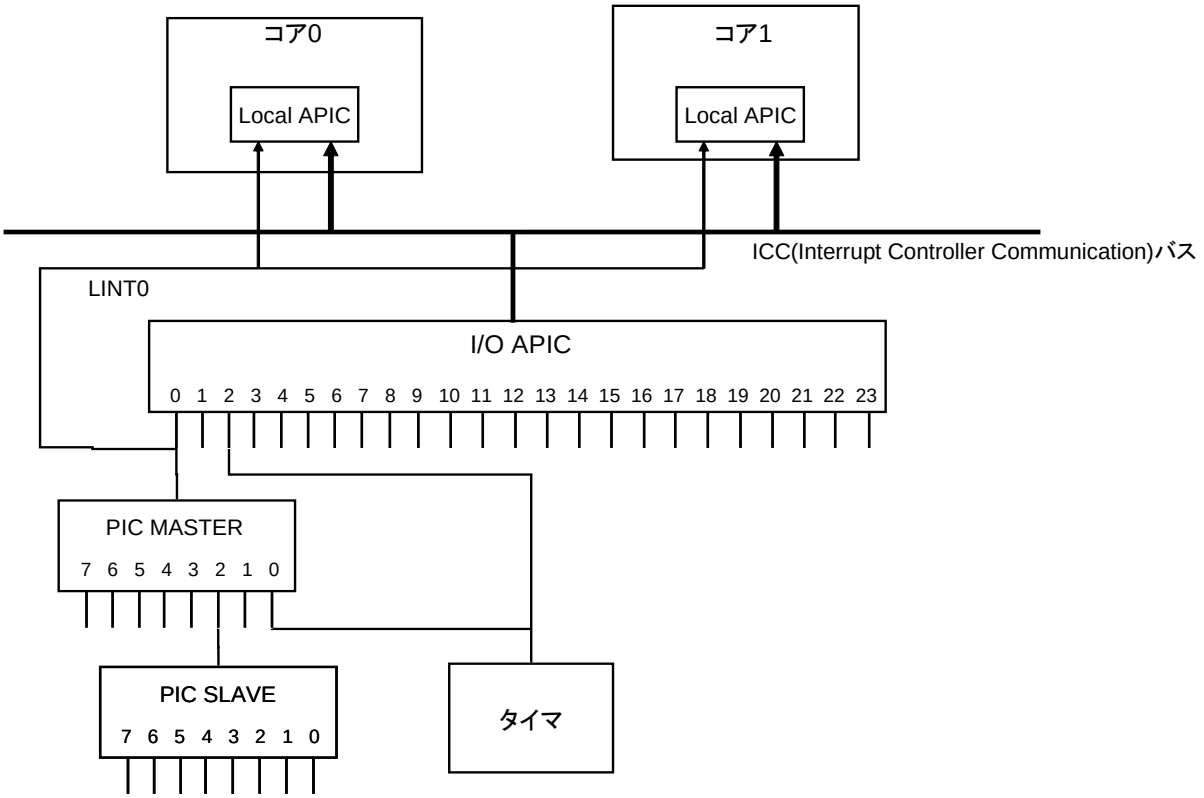

図 3.2 割り込みコントローラの構成

決定する．また，I/O APIC の 0 番ピンは PIC が接続され，2 番ピンはタイマが接続されている．PIC は，MASTER と SLAVE の 2 種類によって構成され，15 本の IRQ ラインを持つ．PIC の MASTER の 0 番ピンにはタイマが接続され，2 番ピンは PIC の SLAVE がカスケード接続される．PIC は，I/O APIC の 0 番ピンや Local APIC の LINT0 ピンに接続され，I/O APIC や Local APIC を経由して CPU のコアに割り込みを送信できる．

3.2.2 Linux の割り込み処理の流れ

割り込みの流れを 図3.3 に示し，以下で説明する．

(1) I/O APIC が IRQ ラインを監視し，信号の発生を調べる．

(2) I/O APIC では，IRQ ラインに信号が発生した場合には，割り込みが発生したピンのピン番号 m に対応するリダイレクションテーブルのエントリを参照し，ベクタ番号 n への変換と割り込み通知先の決定を行う．

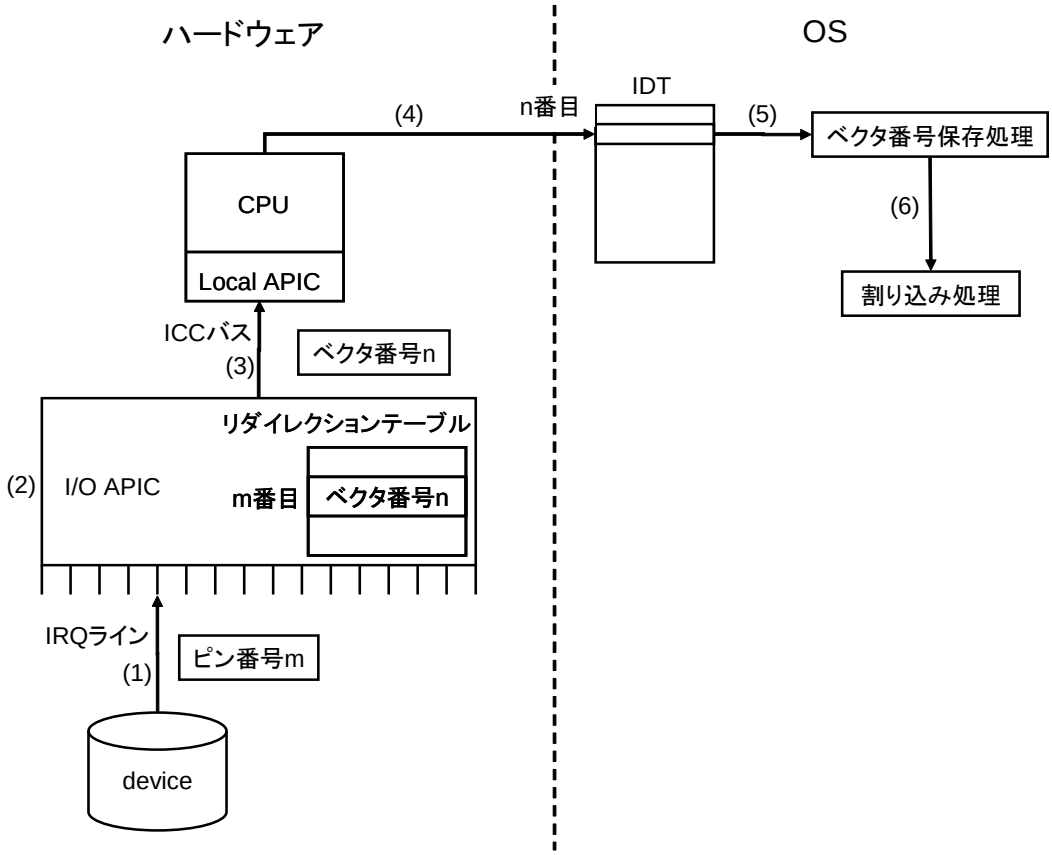

図 3.3 割り込みの流れ

(3) 発生した信号を ICC バスを通じて CPU の Local APIC に通知する.

(4) CPU が割り込みを受け取ると，ベクタ番号 n に対応する IDT(Interrupt Discriptor Table) のエントリへジャンプする.

(5) IDT のエントリからベクタ保存処理を呼び出す.

(6) ベクタを保存後，割り込み処理を呼び出す.

3.2.3 KVM の割り込みに関連するデータ構造

KVM の割り込み制御の解析に必要であるため，KVM の割り込みに関連するデータ構造を以下に示し，説明する.

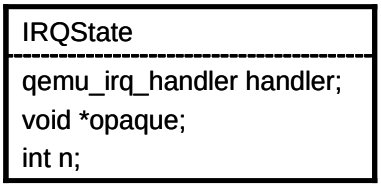

図 3.4 IRQState 構造体

(1) IRQState 構造体

図 3.4 に IRQState 構造体を示し，以下で説明する．この構造体は，QEMU から KVM へ割り込みを通知する際に使用する． 図 3.2 中の各割り込みコントローラの IRQ ラインとデバイスを繋ぐ線を表す．この構造体の重要なメンバと意味を以下に示す．

(A) qemu_irq_handler handler

仮想デバイスから KVM が作成した割り込みコントローラへ割り込みを通知する関数を登録する．

(B) int n

KVM が作成した割り込みコントローラの割り込み番号に対応する．

(2) kvm_irq_routing_entry 構造体

図 3.5 に kvm_irq_routing_entry 構造体を示し，以下で説明する．この構造体は，IRQ ラインの構成の作成に使用される．この構造体の重要なメンバと意味を以下に示す．

(A) __u32 gsi

ピン毎に対応する割り込み番号を示す．

(B) struct kvm_irq_routing_irqchip

割り込みコントローラの種類とピン番号を格納している．割り込みコントローラの種類は，I/O APIC，PIC MASTER，PIC SLAVE のいずれかである．

(3) kvm_kernel_irq_routing_entry 構造体

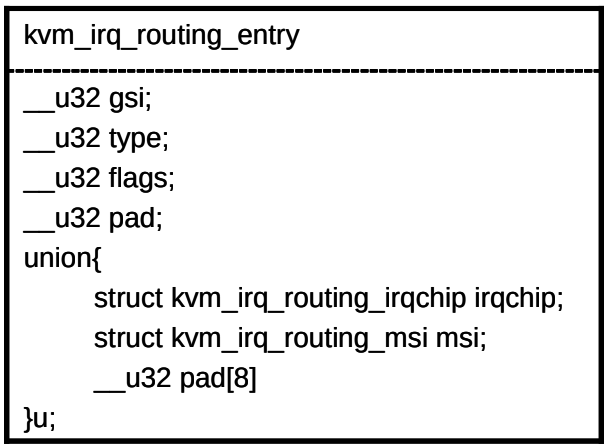

図 3.5 kvm_irq_routing_entry 構造体

図 3.6 に kvm_kernel_irq_routing_entry 構造体を示し，以下で説明する．この構造体は，図 3.2 中の各割り込みコントローラの IRQ ラインを構成する．この構造体の重要なメンバと意味を以下に示す.

(A) u32 gsi

ピン毎に対応する割り込み番号を示す.

(B) int (*set)(struct kvm_kernel_irq_routing_entry *e, struct kvm *kvm, int level)

割り込みコントローラに割り込みをセットする関数を示す.

(C) unsigned irqchip

割り込みコントローラの種類を示す. このメンバが示す割り込みコントローラの種類は，I/O APIC，PIC MASTER，PIC SLAVE のいずれかである．

(4) kvm_ioapic 構造体

図 3.7 に kvm_ioapic 構造体を示し，以下で説明する．この構造体は，I/O APIC のレジスタを構成する．この構造体の重要なメンバと意味を以下に示す.

(A) u32 irr

I/O APIC の IRR を表す.

```
kvm_kernel_irq_routing_entry
----------
u32 gsi;
u32 type;
int (*set)(struct kvm_kernel_irq_routing_entry *e, struct kvm *kvm, int level);
union{
        struct{
                unsigned irqchip;
                unsigned pin;
        }irqchip;
        struct msi_msg msi;
};
struct list_head link;
```

図 3.6 kvm_kernel_irq_routing_entry 構造体

(B) union kvm_ioapic_redirect_entry redirtbl[IOAPIC_NUM_PINS]

I/O APIC のリダイレクションテーブルレジスタを表す.

(5) kvm_lapic 構造体

図 3.8 に kvm_lapic 構造体を示し，以下で説明する．この構造体は，Local APIC を表す．この構造体の重要なメンバと意味を以下に示す.

(A) void *regs

Local APIC のレジスタを表す．オフセットを指定することで Local APIC の各レジスタにアクセスできる.

3.2.4 KVM の割り込み処理の流れ

KVM 使用時の割り込みの流れは，割り込みコントローラのエミュレーションとゲスト OS への割り込みの挿入の 2 つに分けられる．図 3.9 に割り込みコントローラのエミュレーションの流れを示し，以下で説明する.

(1) QEMU が仮想デバイスをエミュレーションし，KVM が作成した割り込みコントローラへ通知する割り込みを生成する.

```
kvm_ioapic
---------------------------------------------------------------------------
u64 base_address;
u32 ioregsel;
u32 id;
u32 irr;
u32 pad;
union kvm_ioapic_redirect_entry redirtbl[IOAPIC_NUM_PINS];
struct kvm_io_device dev;
struct kvm *kvm;
void (*ack_notifier)(void *opaque, int irq);
```

図 3.7 kvm_ioapic 構造体

(2) 割り込み番号 n に対応した QEMU から KVM へ割り込みを通知するハンドラを IRQState 構造体 (図 3.4) から決定する.

(3) ハンドラを呼び出す.

(4) KVM_IRQ_LINE_STATUS リクエストコードの送信により, KVM へ割り込みが通知される.

(5) KVM_IRQ_LINE_STATUS リクエストコードに対応した KVM の処理を呼び出す.

(6) gsi と割り込み番号 n が一致する kvm_kernel_irq_routing_entry 構造体 (図 3.6) を探す.

(7) kvm_ker
nel_irq_routing_entry 構造体 (図 3.6) の set 関数を呼び出す.

(8) kvm_ioapic 構造体 (図 3.7) の irr に割り込み番号に対応したビットを立てる.

(9) kvm_ioapic 構造体 (図 3.7) の redirtbl と割り込み番号から割り込み通知先の仮想 CPU を決定する.

(10) 割り込み通知先の仮想 CPU が持つ kvm_lapic 構造体 (図 3.8) の regs に IRR(Interrupt Request Register) へのオフセットを足した場所へベクタ番号をセットする.

図 3.10 にゲスト OS への割り込み挿入の流れを示し, 以下で説明する.

```
kvm_lapic
-----------------------------------------------------------------------------
unsigned long base_address;
struct kvm_io_device dev;
struct kvm_timer lapic_timer;
u32 divide_count;
struct kvm_vcpu *vcpu;
struct page *regs_page;
void *regs;
gpa_t vapic_addr;
struct page *vapic_page;
 void (*ack_notifier)(void *opaque, int irq);
```

図 3.8 kvm_lapic 構造体

(1) QEMU が KVM_RUN リクエストコードを KVM へ送信する.

(2) kvm_lapic 構造体 (図 3.8) の regs の IRR にベクタ番号がセットされているかチェックする.

(3) kvm_lapic 構造体 (図 3.8) の regs の IRR にベクタ番号がセットされていた場合, KVM はゲスト OS へ割り込みを挿入する. VMCS(Virtual Machine Control Structure) の VM-entry interruption information field にベクタ番号を書き込む. ここで VMCS とは, Intel-VT の仮想化支援機能を制御するための構造体である. また, VM-entry interruption information field は, ゲスト OS 走行開始時の割り込みの挿入に使用される領域である.

(4) KVM はゲスト OS を走行させ, ゲスト OS は割り込みを処理する.

3.2.5 割り込みコントローラの作成と設定

割り込みコントローラの作成と設定について, 以下で説明する.

(1) 割り込みコントローラの作成

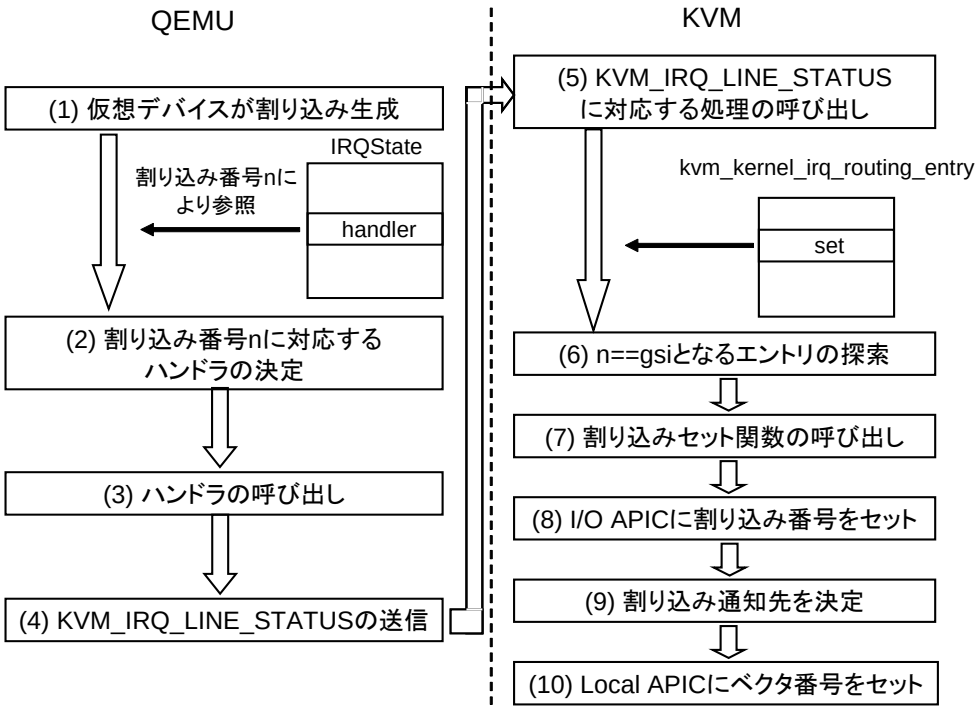

図 3.9 割り込みコントローラのエミュレーションの流れ

QEMU が KVM_CREATE_IRQCHIP リクエストコードを KVM へ送信することにより，行われる．KVM が kvm_ioapic 構造体 (図 3.7) の領域確保と初期化を行う．また，kvm_kernel_irq_routing_entry 構造体 (図 3.6) を PIC と I/O APIC の IRQ ライン数だけ作成し，初期化する．

(2) x86 アーキテクチャ用の割り込みコントローラの構成を作成

QEMU が x86 アーキテクチャ用の割り込みコントローラの構成を作成する．割り込みコントローラの IRQ ライン毎に割り込み番号を設定し，kvm_irq_routing_entry 構造体 (図 3.5) を作成する．

(3) 割り込みコントローラの構成を設定

QEMU が KVM_SET_GSI_ROUTING リクエストコードを KVM へ送信することにより，行われる．(2) で作成した割り込みコントローラの構成を KVM に設定する．割り込みコントローラの IRQ ライン毎に (2) で作成した kvm_irq_routing_entry 構造体 (図 3.5) の割り込み番号を kvm_kernel_irq_routing_entry 構造体 (図 3.6) へ設

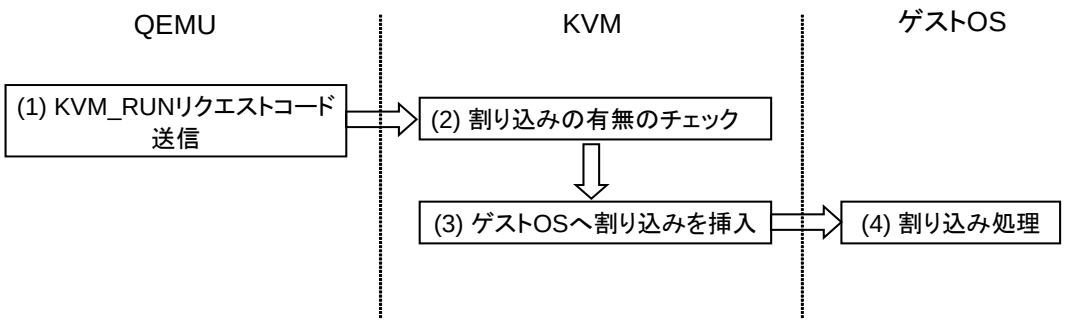

図 3.10 ゲスト OS への割り込み挿入の流れ

定する．また，割り込みコントローラの IRQ ライン毎に割り込みコントローラに割り込みをセットする関数を設定する．作成した kvm_kernel_irq_routing_entry 構造体 (図 3.6) をリスト化し，仮想マシンに保存する．

3.2.6 仮想デバイスの割り込み設定

IRQState 構造体の handler に仮想デバイスから KVM へ割り込みを通知するハンドラを割り込みコントローラの割り込み番号毎に設定する．その後，仮想デバイスを作成する．このとき，各仮想デバイスに割り込みコントローラ用の割り込み番号を割り当て，割り込み番号に対応したハンドラを設定する．

第 4 章

Mint におけるデバイス接続状況の把握法

4.1 設計

3.2 節に述べた KVM の割り込み制御方式を Mint に導入することを考える．図 4.1 に割り込み管理層を加えた Mint の構成例を示し，以下で説明する．ハードウェアと OS の間に割り込み管理層を挿入する．この割り込み管理層で割り込みを集約し，どの OS へ割り込みを通知するか管理する．この割り込み管理層を用いた場合の割り込みの流れを図 4.2 に示し，以下で説明する．

(1) ハードウェアから割り込みが発生する．

(2) I/O APIC を通して CPU へ割り込みが通知される．

(3) CPU は IDT を用いてベクタ番号保存処理を呼び出す．その後，割り込み管理層へジャンプする．

(4) 割り込み管理層は割り込み管理テーブル (IRQ_manage_table) をベクタ番号で参照する．

(5) 割り込み通知先 OS の割り込み処理を決定する．

(6) 割り込み通知先 OS の割り込み処理を呼び出す．

(7) 割り込み通知先 OS にて割り込みを処理する．

また，割り込み管理層への割り込み処理登録は次の 2 つの契機が考えられる．

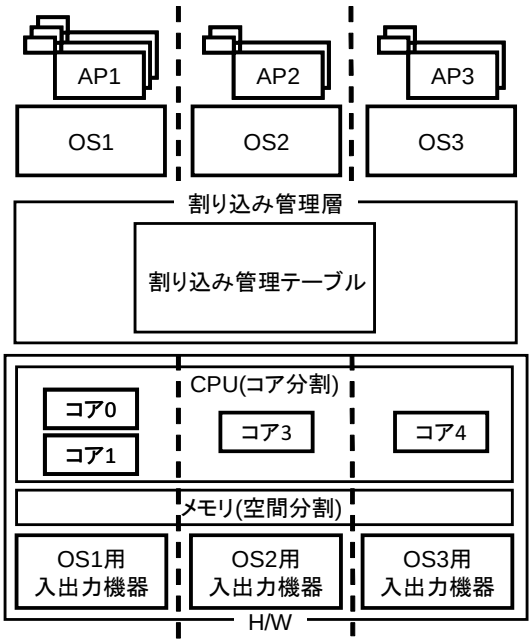

図 4.1 割り込み管理層を加えた Mint の構成例

(1) デバイス初期化処理

(2) デバイス解放処理

この 2 つの割り込み処理登録に関して，契機以外の割り込み処理登録の流れは同様であると考えられる．このため，図 4.3 に割り込み処理を登録する関数を呼び出してからの処理の流れを示し，以下で説明する．

(1) 割り込み管理層に割り込み処理を登録する関数を呼び出す.

(2) 割り込み管理層へ割り込み処理のアドレスとベクタ番号を渡す.

(3) 割り込み管理テーブルのベクタ番号に対応する位置に割り込み処理のアドレスを登録する．また，割り込み通知先 OS を登録する．

(4) 対応する I/O APIC のリダイレクションテーブルに割り込み通知先を設定する．

これらの処理により，割り込み管理テーブルにどの OS の割り込み処理が登録されているかによって，デバイスの接続状況が分かる．

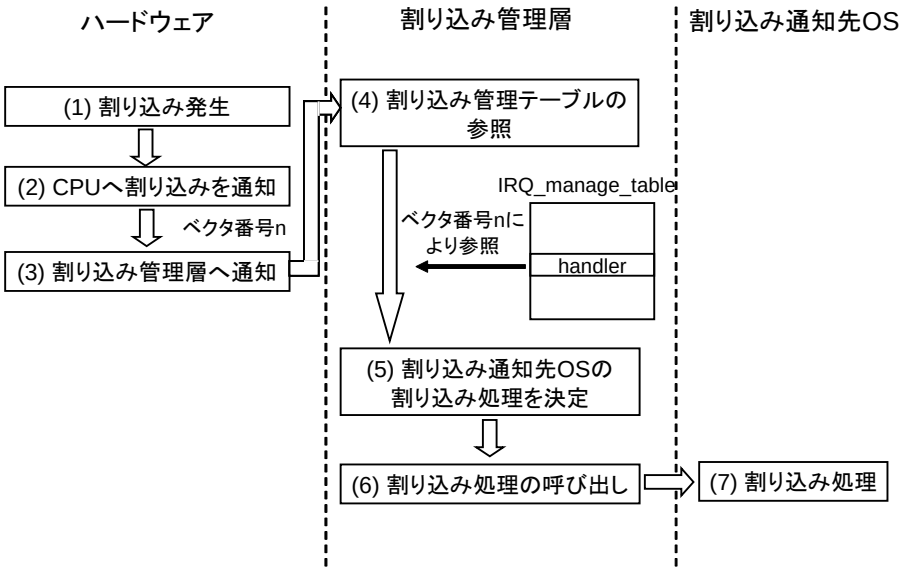

図 4.2 割り込み管理層を用いた割り込みの流れ

4.2 課題

4.1 節で示した割り込みの流れを実現するためには，以下の 4 つの課題がある．

(1) CPU から割り込み管理層を呼び出す方法

本来の割り込みの流れは，3.2.2 節に示すように，CPU が IDT を用いてベクタ保存処理を呼び出し，その後，OS の割り込み処理が呼び出されるという流れである．割り込み管理層を用いる場合，割り込み処理を呼び出す前に割り込み管理層に割り込みを集約する必要がある．

(2) 割り込み管理層に配置する割り込み管理テーブルの作成

割り込み管理層にて，受け付けた割り込みを処理する OS へ通知するためのテーブルが必要である．このテーブルの内容を決定し，作成する．

(3) 割り込み通知先 OS の割り込み処理の登録

割り込み管理層から割り込み通知先の OS へ割り込みを渡すため，割り込み通知先 OS の割り込み処理を登録する．

(4) デバイス接続状況の通知法

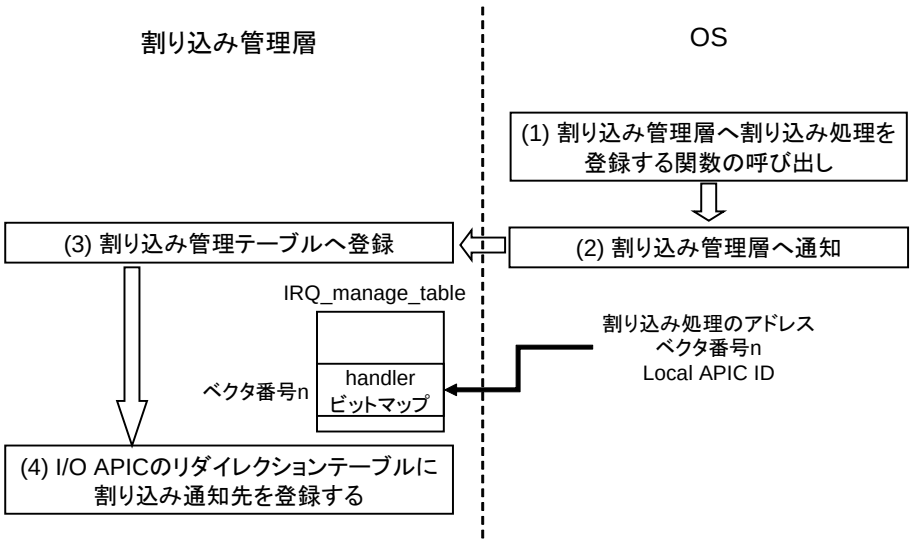

図 4.3 割り込み処理登録の流れ

割り込み管理層から OS へデバイス接続状況を通知する方法が必要である.

4.3 対処

4.2 節で示した 4 つの課題に対する対処を以下に示す.

(1) ベクタ番号保存後，割り込み管理層へジャンプするように改変する.

(2) 割り込み管理テーブルには，以下の要素を持たせる.

(A) 割り込み処理の呼び出し

KVM の kvm_kernel_irq_routing_entry 構造体 (図 3.6) を参考にし，kvm_kernel_irq_routing_entry 構造体のメンバのうち，set 相当のものを用意する．これには，割り込み通知先 OS の割り込み処理のアドレスを登録し，登録した割り込み処理を呼び出すことで OS に割り込みを通知する.

(B) 割り込み通知先の明示

割り込み通知先を明示するビットマップを用意する.

割り込み管理テーブルは，ベクタ番号で参照し，ベクタ番号で参照されたエントリに登録している割り込み処理を呼び出す．このとき，ベクタ番号が同一で

表 4.1 割り込み管理層に渡すパラメータ

パラメータ	概要
ベクタ番号	割り込み管理テーブルのエントリ参照に使用
割り込み処理アドレス	割り込み処理の登録に使用
Local APIC ID	通知先 OS の登録に使用

複数の OS へ通知する割り込みが存在する場合がある．この場合，割り込み処理を複数登録できなければならない．また，Mint では，起動できる OS の数は CPU のコア数に依存し，現在使用している CPU のコア数は 4 つである．このため，1 エントリの割り込み処理の最大登録数は 4 つとする．

(3) 割り込み管理テーブルには，割り込み通知先 OS の割り込み処理を登録する．OS が割り込み管理層に渡すパラメータを表 4.1 に示し，以下で説明する．この処理は，割り込みを登録したい OS がベクタ番号，OS の割り込み処理のアドレスと Local APIC ID を割り込み管理層に渡し，割り込み管理テーブルのベクタ番号に対応するエントリに割り込み処理のアドレスを登録する．また，通知先 OS を判別するためのビットマップを Local APIC ID をもとに変更する．その後，登録された割り込みに対応する I/O APIC のリダイレクションテーブルに割り込み通知先の Local APIC ID を設定する．

(4) 割り込み管理テーブルに割り込み通知先 OS を明示するためのビットマップを用意している．OS から Local APIC ID を受け取り，これをもとに各エントリのビットマップを参照する．割り込み通知先と Local APIC ID が一致したエントリのベクタ番号を OS へ通知する．

以上の対処により，Mint においてデバイス接続状況を把握する．

第 5 章

おわりに

本論文では，Mint におけるデバイス接続状況の把握法を設計した．

まず，Mint の概要と割り込み制御について述べ，デバイス接続状況の把握の目的を述べた．

次に，KVM の割り込み制御について述べた．KVM が仮想化するハードウェア構成を述べ，このハードウェア構成を仮想化するためのデータ構造を述べた．また，KVM の割り込みの流れを述べた．QEMU が仮想デバイスの割り込みを生成し，KVM へ通知する．KVM は，割り込みコントローラをエミュレーションし，ゲスト OS へ割り込みを挿入する準備を行う．その後，ゲスト OS の実行時に割り込みをゲスト OS へ挿入する．

最後に，KVM の割り込み制御を参考にし，Mint におけるデバイス接続状況の把握法の設計を述べた．割り込みを集約し，どの OS へ通知するか管理する割り込み管理層をハードウェアと OS の間に挿入した．この割り込み管理層で割り込みを管理することにより，各デバイスの割り込みがどの OS へ通知されるか管理できるようにした．

残された課題として，Mint におけるデバイス接続状況の把握法の実装がある．

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，山内利宏准教授，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] Jeremy Sugerman，Ganesh Venkitachalam，and Beng-Hong Lim,“ Virtualizing I/O Devices on VMware Workstation’s Hosted Virtual Machine Monitor，”Proc. of the General Track: 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 symposium on 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). 電子情報通信学会 コンピュータシステム研究会(CPSY)

[4]“ Main Page -KVM-，”http://www.linux-kvm.org/page/Main_Page