TinyVisorの開発と設計

2013年12月6日

渡辺 祐一

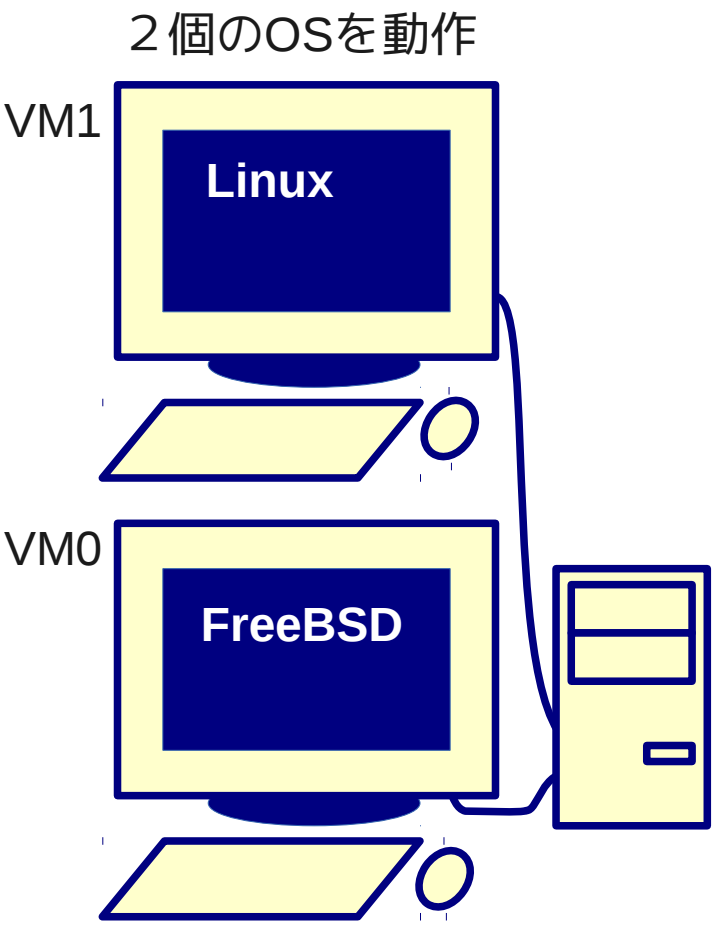
はじめに

- TinyVisorの紹介、および、開発の経緯と今後の目標について説明

- TinyVisorの設計について説明

TinyVisorとは

- 一台のパソコン(PC)でOSを二個動作させるためのハイパーバイザ(VMM)
  - CPU、メモリ、I/Oを各OSに割り当て、OSが直接制御
  - ホストOSなしで各OSが独立動作
- オープンソースソフトウェアをベースに開発し成果物を公開
  - VMM: BitVisor(BSDライセンス)を改造
  - ゲストBIOS: SeaBIOS(GPL)をカスタマイズ



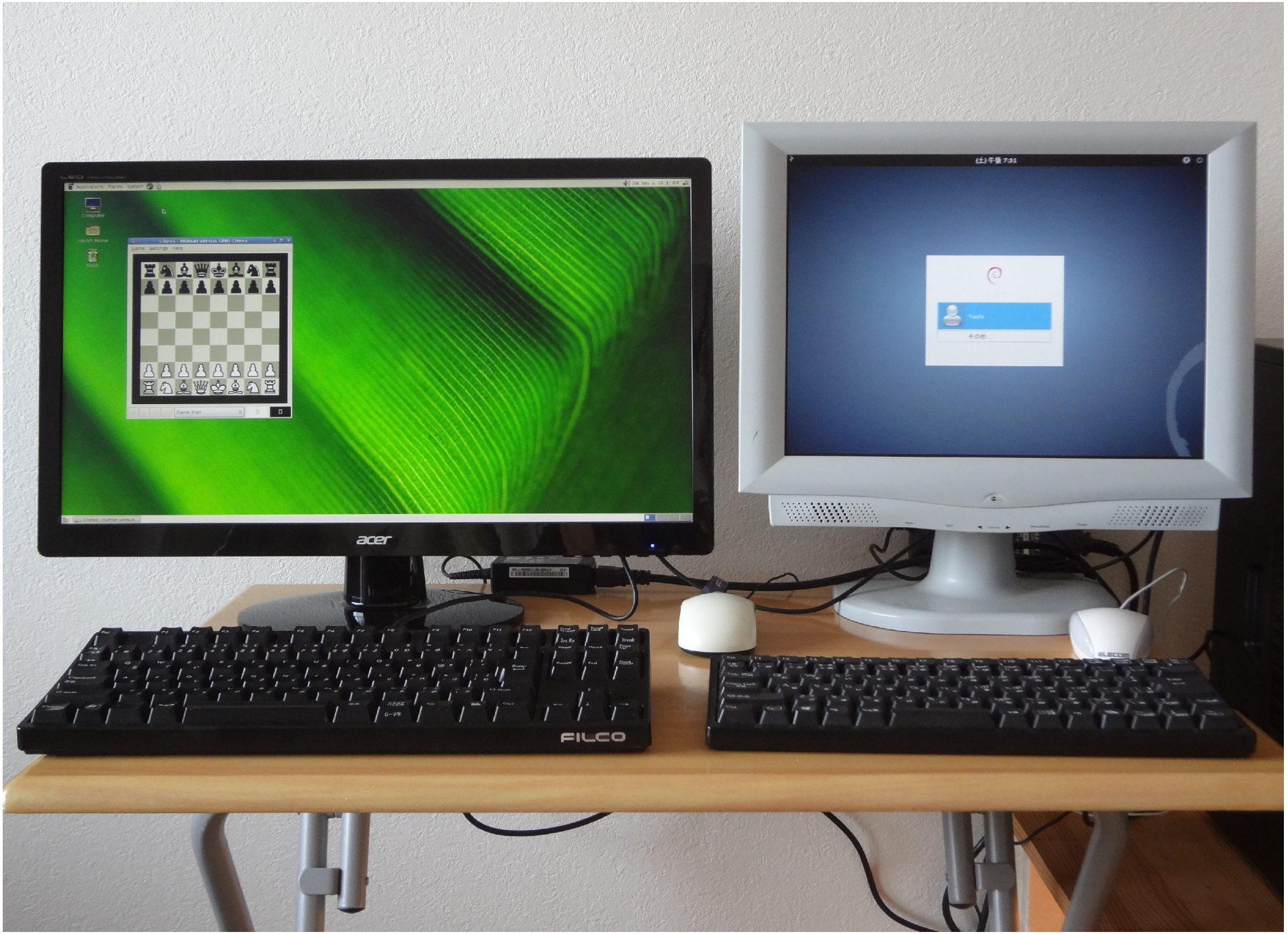
使い方

1. ディスプレイ、キーボード、マウス、ビデオカード、HDD、 HBA カード(例えばAHCI)、NICカードなどのデバイスを２組用意
2. ２つのHDDに、それぞれOSをインストール
3. USBメモリにVMMとゲストBIOSをインストール

```
./install_to_usb.sh -c 'vm0.boot_int18 vm=vm0,vm1 vm1.cpu=4,6
vm1.mem=80000000-21f5fffff vm1.pci=00:01.0,00:14.0,00:1c.5,00:1c.7' /dev/sdd
```

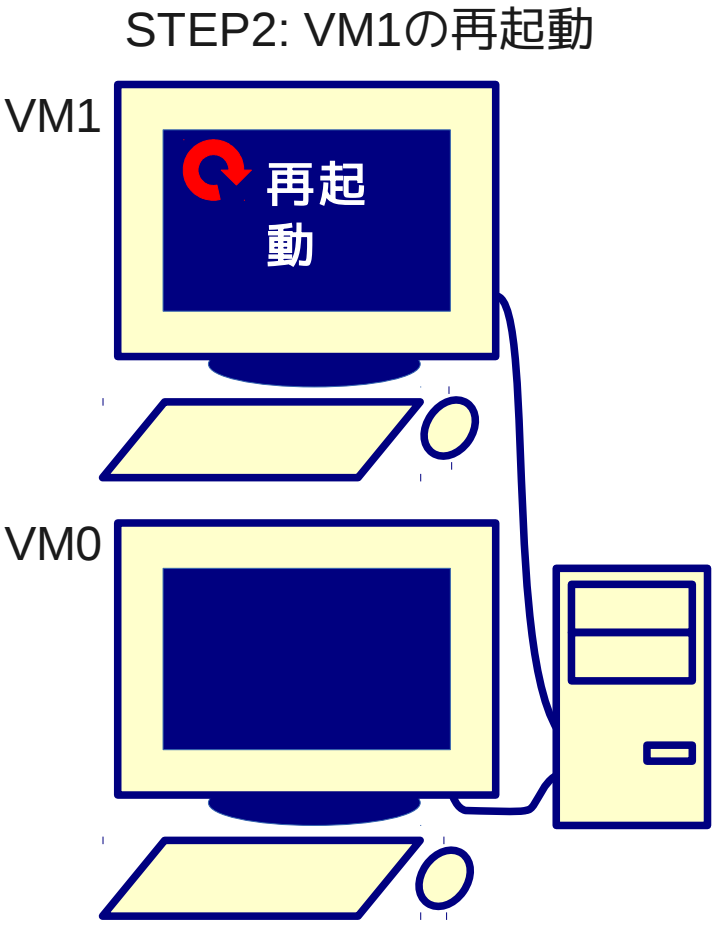
4. USBメモリと、２組用意したデバイスを全てPCに接続
5. PCの電源を入れると、OSが2個同時に起動

AHCI: Advanced Host Controller Interface (SATAのコントローラのインタフェース)


acer
FILCO

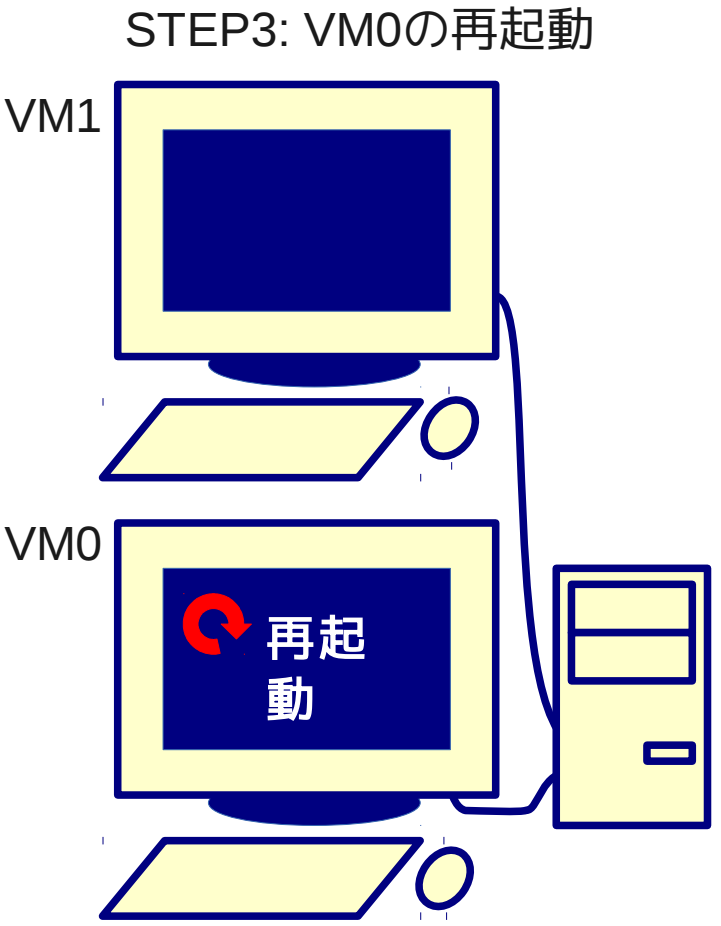
開発の経緯

- STEP1 (2012年3月)
  - 2個のOSを動作
- STEP2 (2013年5月)
  - VM1の再起動



今後の目標

- STEP3 (2014年?)
  - VM0の再起動
- STEP4 (2015年?)
  - VMごとにVMMの独立性向上
- STEP5 (2016年?)
  - VMMの部分アップデート



応用の可能性

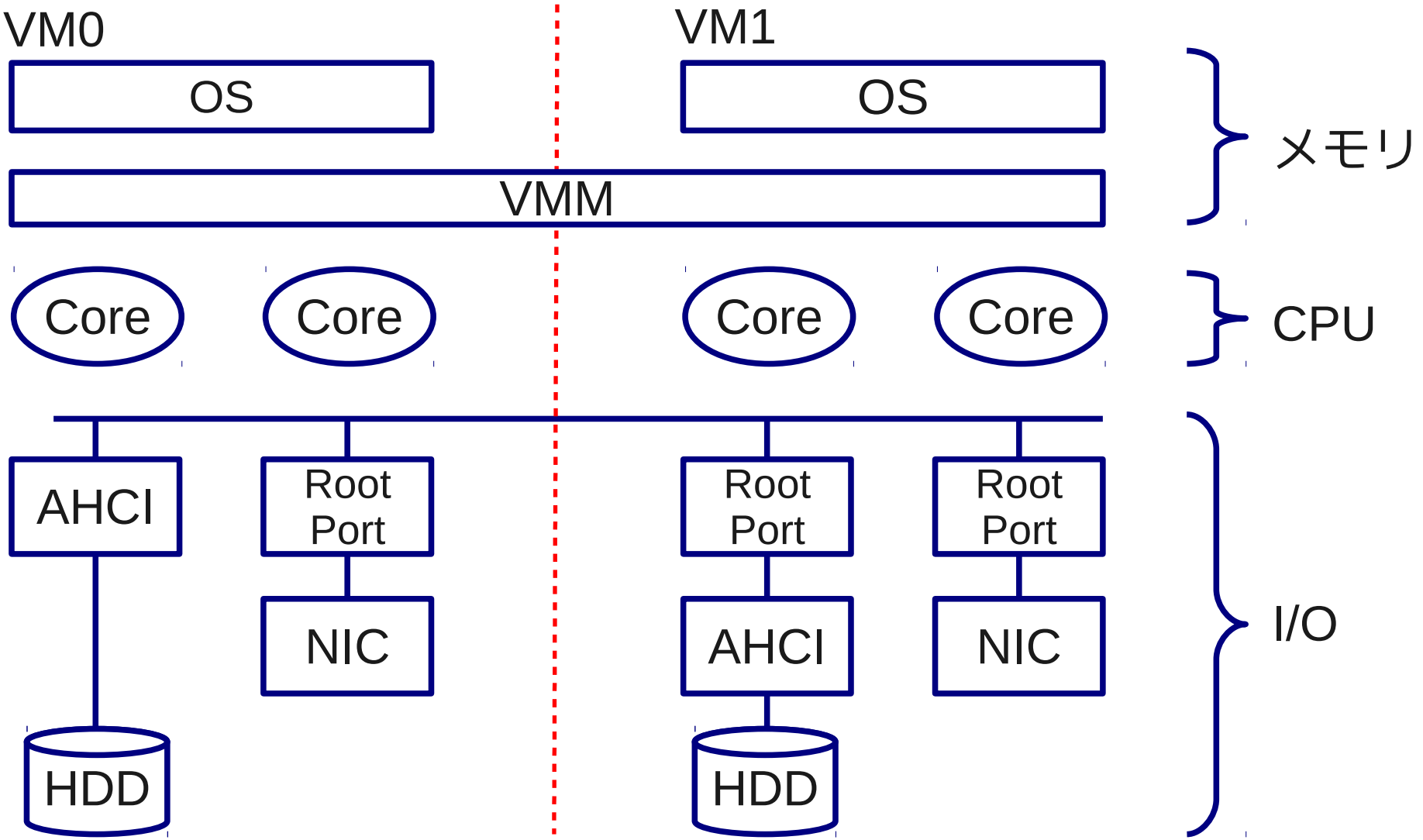
- 夫婦でパソコンを共用？
  - 同じ部屋で隣り合って使えば会話が増える？
- ネット用パソコンと、自宅サーバを、統合？
  - 自宅のパソコンを削減して、夏の暑さを軽減？
- 製品に組み込む？
  - OSが二つ動けば面白いことができるかも？

重複するCPU、メモリ、I/O

- CPU
  - マルチコア
    ハイパースレッディングが有効なら論理プロセッサはコアの２倍
- メモリ
  - 数ギガバイトから数十ギガバイト
- I/O
  - チップセット内臓AHCI(SATA) + マザーボード上のAHCIチップ
  - xHCI(USB3.0) + EHCI(USB2.0)
  - PIT + HPET

PIT: Programmable Interval Timer
HPET: High Precision Event Timer

CPU、メモリ、I/Oの割り当て



CPUの割り当て

- 論理プロセッサ単位
  - ハイパースレッディングが無効な場合は論理プロセッサ=コア
  - ハイパースレッディングが有効な場合はスレッド単位

- APIC IDで指定
  - ローカルAPICに付与されたID
  - 論理プロセッサを特定可能

```
./install_to_usb.sh -c 'vm0.boot_int18 vm=vm0,vm1
vm1.cpu=4,6 vm1.mem=80000000-21f5fffff
vm1.pci=00:01.0,00:14.0,00:1c.5,00:1c.7' /dev/sdd
```

割り当てた論理プロセッサをOSに認識させる

- OSは、ACPIのMADTのProcessor Local APIC Structureを参照して論理プロセッサを認識
- VM0
  ホストBIOSが生成したStructureのうち、VM1に割り当てた論理プロセッサのStructureをVMMが無効化
- VM1
  ゲストBIOSが、VM1に割り当てられた論理プロセッサを検出し、Structureを生成

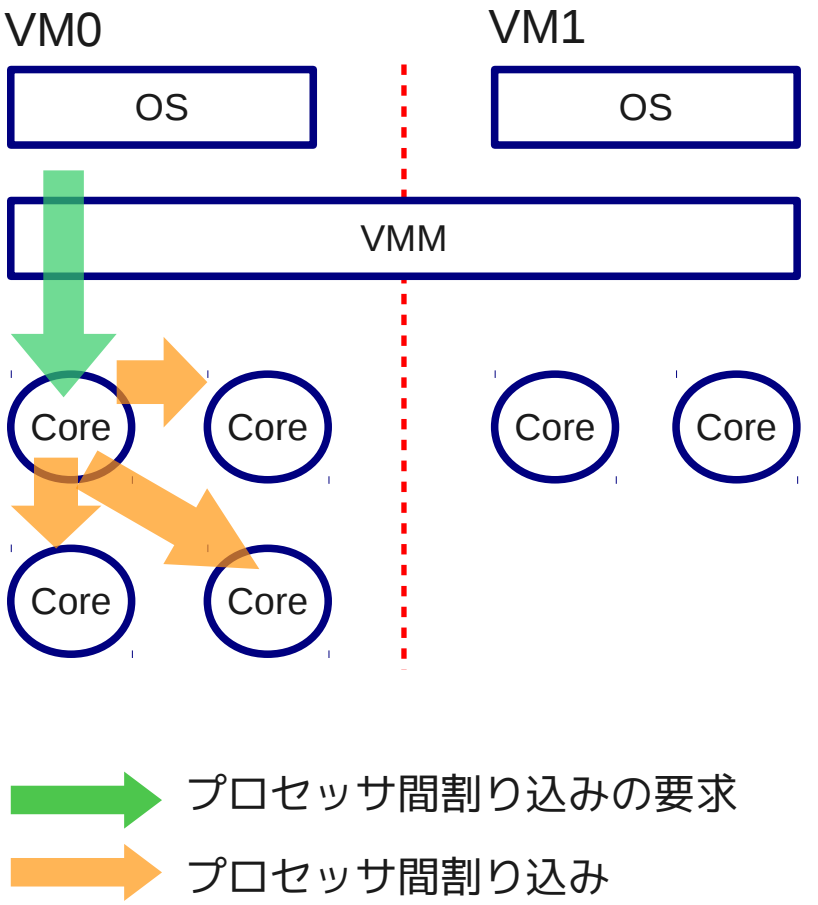
ACPIのMADTの
Processor Local APIC Structure [1]

オフセット	フィールド	説明
0	Type	常に0
1	Length	常に8
2	ACPI Processor ID	この論理プロセッサに対応するACPI Processor Objectに付けられたID
3	APIC ID	論理プロセッサにHW的に割り当てられているID
4-7	Flags	**ビット0: 有効ビット**

MADT: Multiple APIC Description Table
[1]「Advanced Configuration and Power Interface Specification, Revision 4.0a」を要約
http://www.acpi.info

プロセッサ間割り込みの送信

- OSがプロセッサ間割り込みを要求
- OSが宛先を指定していれば、その割り込み要求をVMMがパススルー
- OSが宛先として「全プロセッサ」や「他のプロセッサ全て」を指定していれば、割り当てられた論理プロセッサのみを宛先にした割り込みにVMMが変換



割り込み
ディスティネーションモード

- プロセッサ間割り込みや、I/Oデバイスの割り込みの宛先の指定方法
- VM0
  - 論理ディスティネーションモード
  - 物理ディスティネーションモード
- VM1
  - 物理ディスティネーションモードのみ

    ゲストBIOSのACPI FADTのフラグで物理ディスティネーションモードをOSに要求

メモリの割り当て

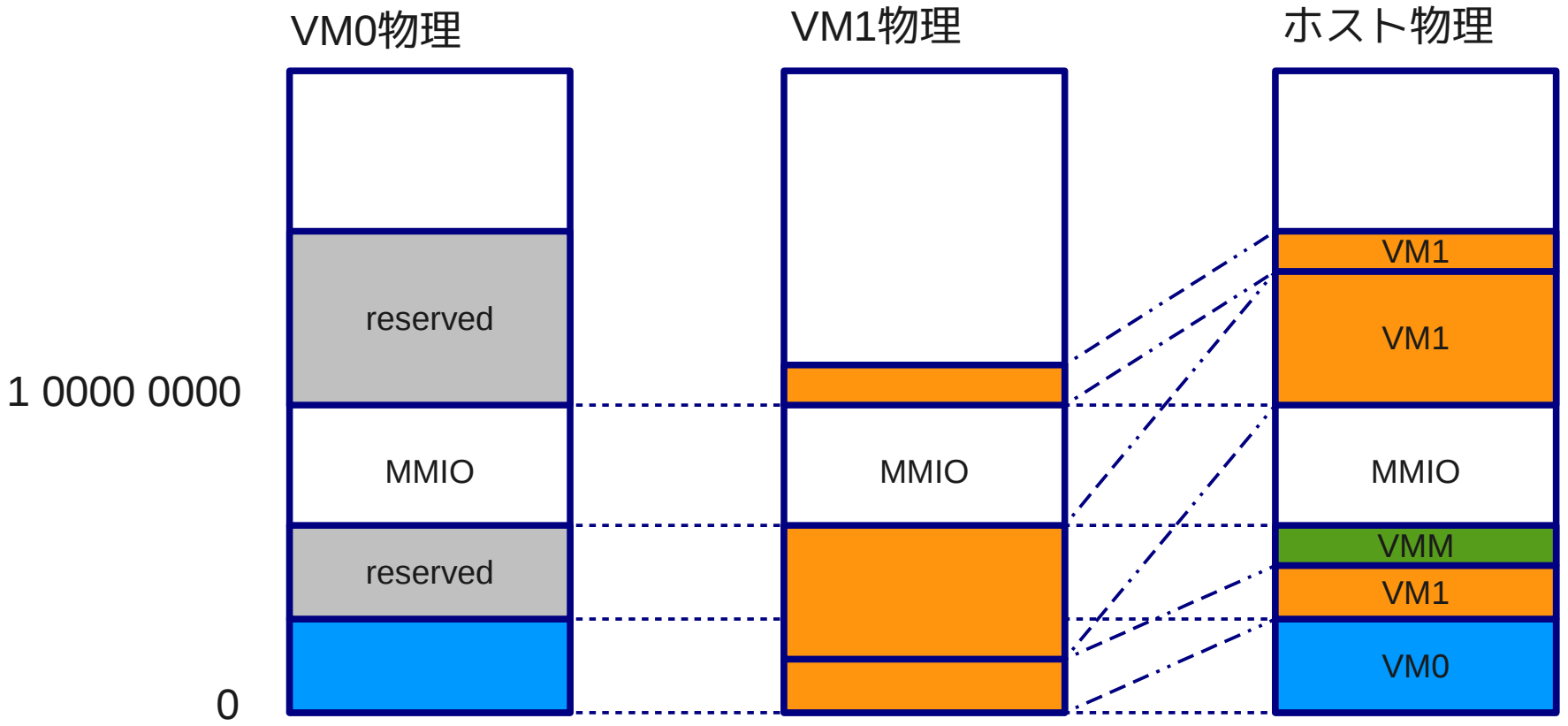
- ページ単位
  - 4KB単位
  - Shadow PagingやEPT(Extended Page Table)でアドレス変換するため

- 物理アドレスの範囲で指定

```
./install_to_usb.sh -c 'vm0.boot_int18 vm=vm0,vm1
vm1.cpu=4,6 vm1.mem=80000000-21f5fffff
vm1.pci=00:01.0,00:14.0,00:1c.5,00:1c.7' /dev/sdd
```

各VMの物理アドレスと ホスト物理アドレスの関係

- VMMが使用するメモリと、VM1に割り当てたメモリを、VM0のOSにreserved領域として見せる
- VM1割り当てたメモリを、アドレス変換して、VM1のOSに見せる



割り当てたメモリを OSに認識させる

- OSは、INT 15H, E820H BIOSコールでメモリマップを取得
  メモリマップは、右のAddress Range Descriptor Structureの配列で示す
- VM0
  INT 15H, E820H BIOSコールをフックして、VMMがメモリマップを返却
- VM1
  ゲストBIOSが、予めVMMからメモリの範囲を取得しておき、INT 15H, E820H BIOSコールにメモリマップを返却する

INT 15H, E820H BIOSコールで取得できるAddress Range Descriptor Structure [2]

オフセット	フィールド
0-3	BaseAddrLow
4-7	BaseAddrHigh
8-11	LengthLow
12-15	LengthHigh
16-19	Type
20-23	Extended Attributes

Type
1 AddressRangeMemory
2 AddressRangeReserved
3 AddressRangeACPI
4 AddressRangeNVS
5 AddressRangeUnusuable
6 AddressRangeDisabled

[2]「Advanced Configuration and Power Interface Specification, Revision 4.0a」を要約
http://www.acpi.info

I/Oの割り当て

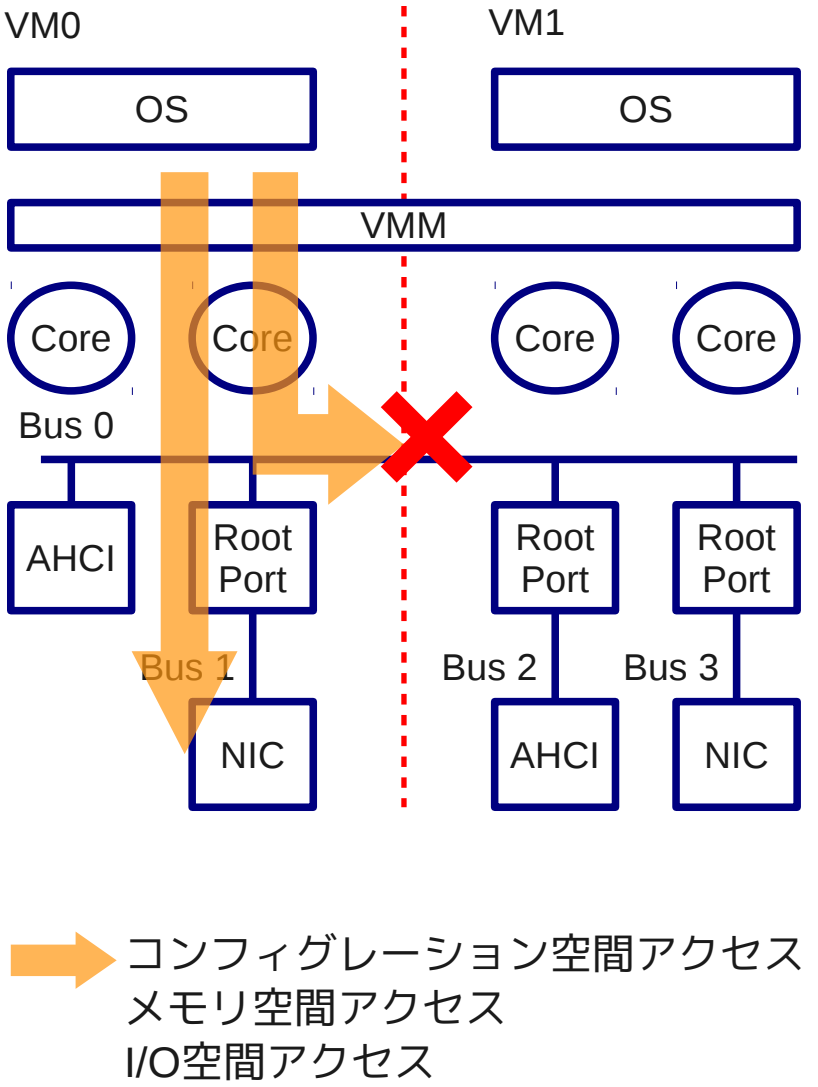
- Root Port単位
  - Root Portを割り当てると、その背後のデバイスも一緒に割り当てられる

- Root Portのバス番号、デバイス番号、ファンクション番号で指定

```
./install_to_usb.sh -c 'vm0.boot_int18 vm=vm0,vm1
vm1.cpu=4,6 vm1.mem=80000000-21f5fffff
vm1.pci=00:01.0,00:14.0,00:1c.5,00:1c.7' /dev/sdd
```

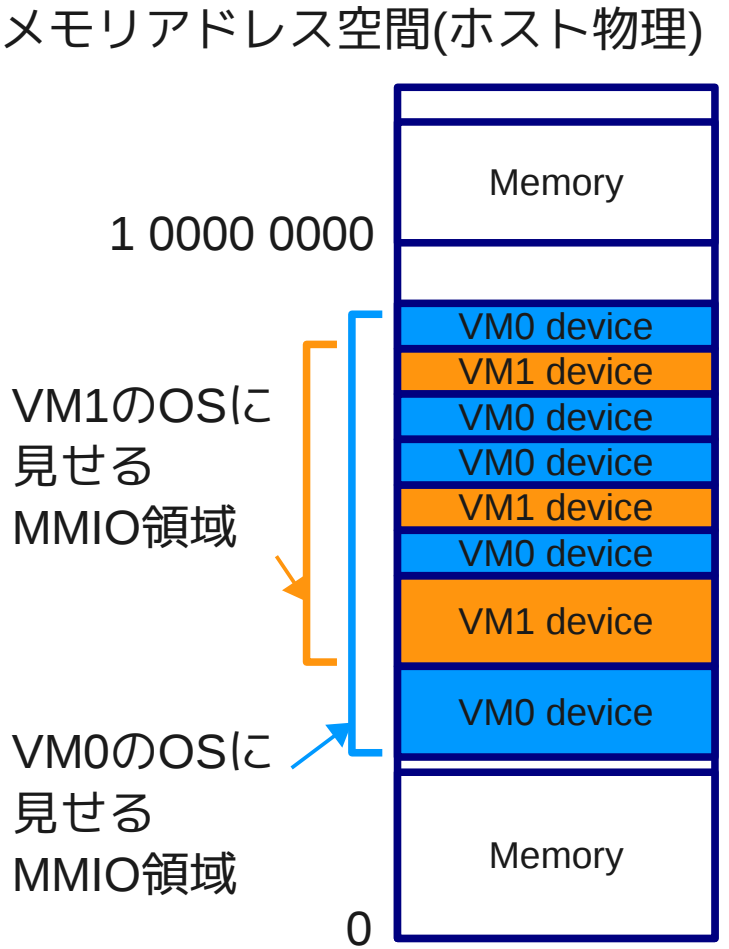
OSからI/Oデバイスへのアクセス

- 自分のVMに割り当てられたデバイスへのアクセスは許可
- 別のVMに割り当てられたデバイスに対するアクセスは禁止
  - OSがリードアクセスすると、VMMがOSにAll Fを返す
  - OSがライトアクセスすると、VMMが無効化
- VMMはアドレスを変換しない
- OSはBus 0を探索するため、割り当てたRoot PortやI/Oデバイスを見つけることができる



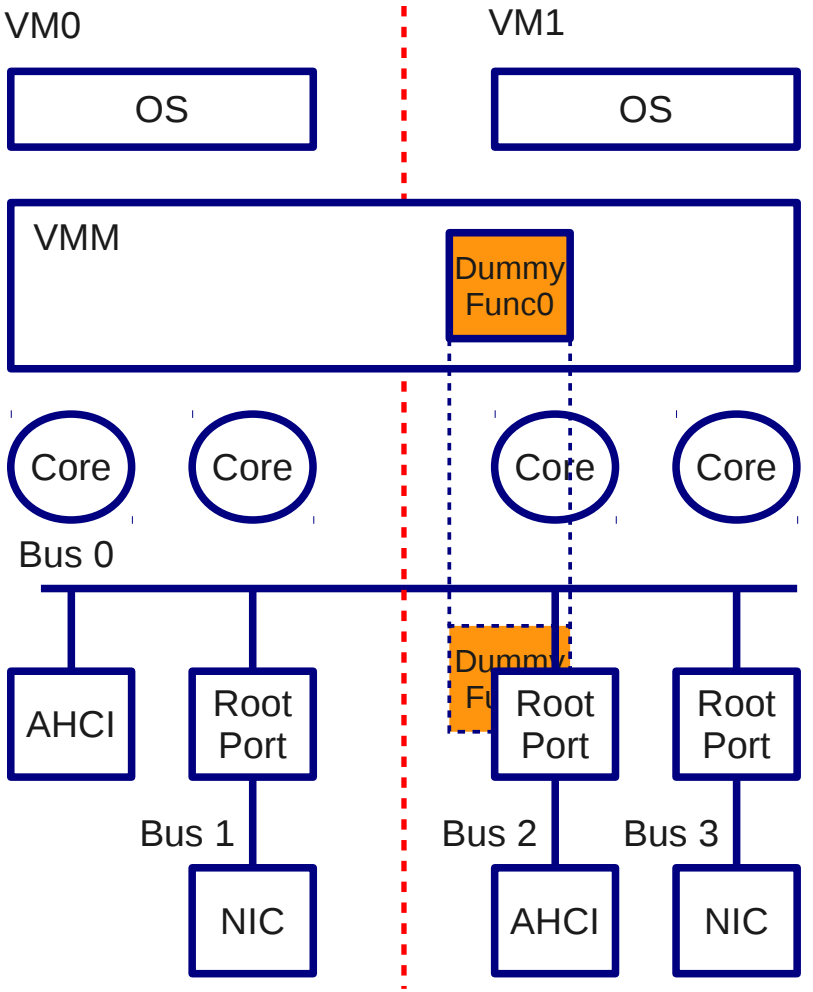
MMIO領域をOSに認識させる

- メモリアドレス空間の一部がMMIO領域
- OSは、BIOSが提供するACPI PCI Root Bridge オブジェクトの_CRSメソッドを実行して MMIO領域の範囲を取得する
- VM0
  ホストBIOSは、MMIO領域全体をOSに返却
- VM1
  ゲストBIOSは、VM1に割り当てられたデバイスがマップされている領域の上端から下端までをMMIO領域として返却



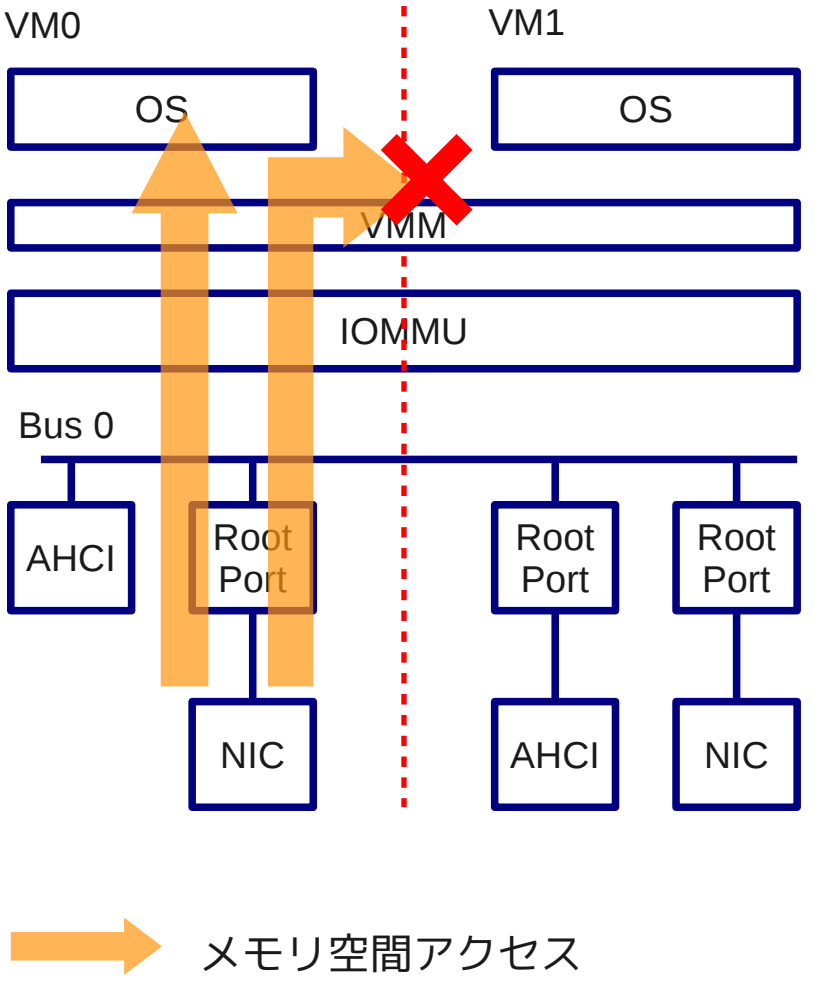
マルチファンクションデバイス

- Root Portがマルチファンクションデバイスの場合がある
  - Root Portを割り当ててもファンクション0が存在しないとOSが検出不可
- VMMがダミーのファンクション0をエミュレートし、割り当てたRoot PortをOSが見つけられるようにする



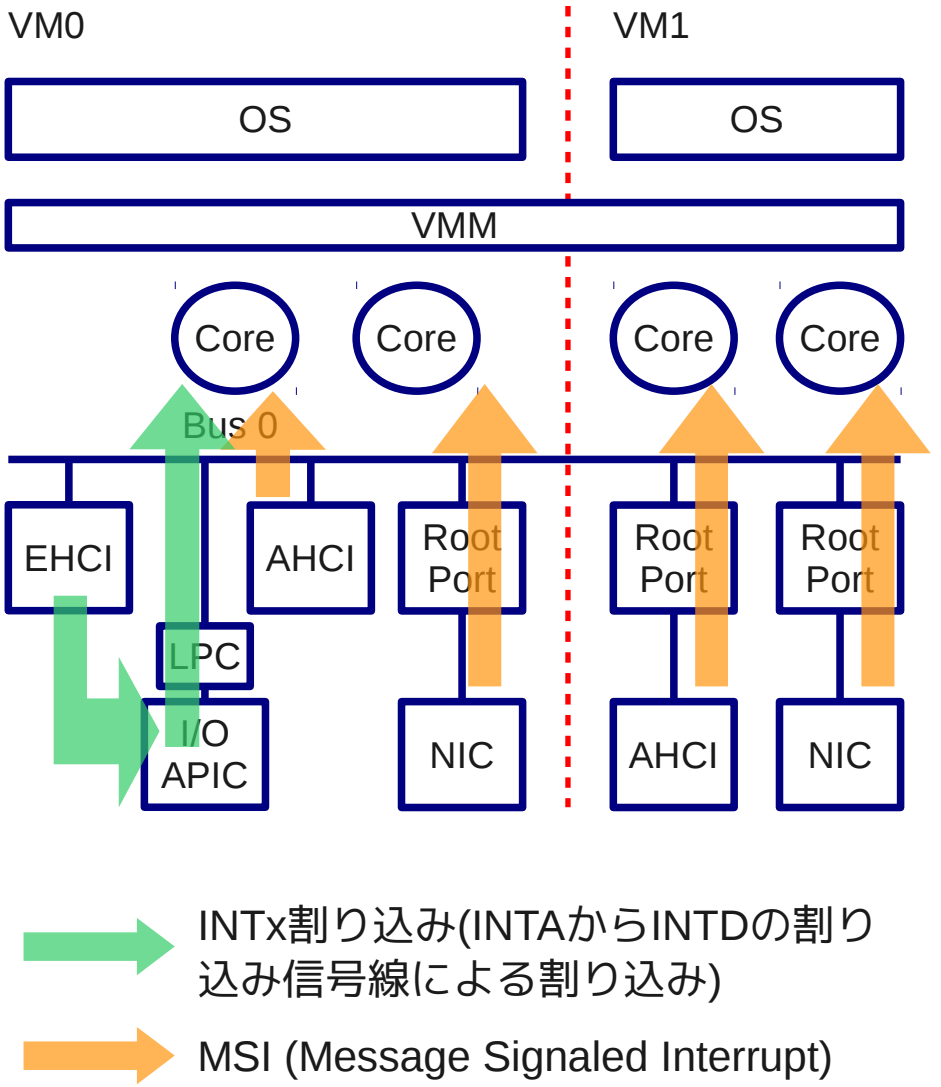
I/Oデバイスからメモリへのアクセス

- VT-dのIOMMUによってアドレス変換
- VMに割り当てられたメモリ以外へのアクセスは禁止
  - I/Oデバイスから見ると、存在しないメモリにアクセスした時と同じ振る舞い



I/Oデバイスからの割り込み

- VM0
  - INTx割り込みで I/OAPIC経由で コアへ
  - MSIで直接コアへ
- VM1
  - MSIで直接コアへ

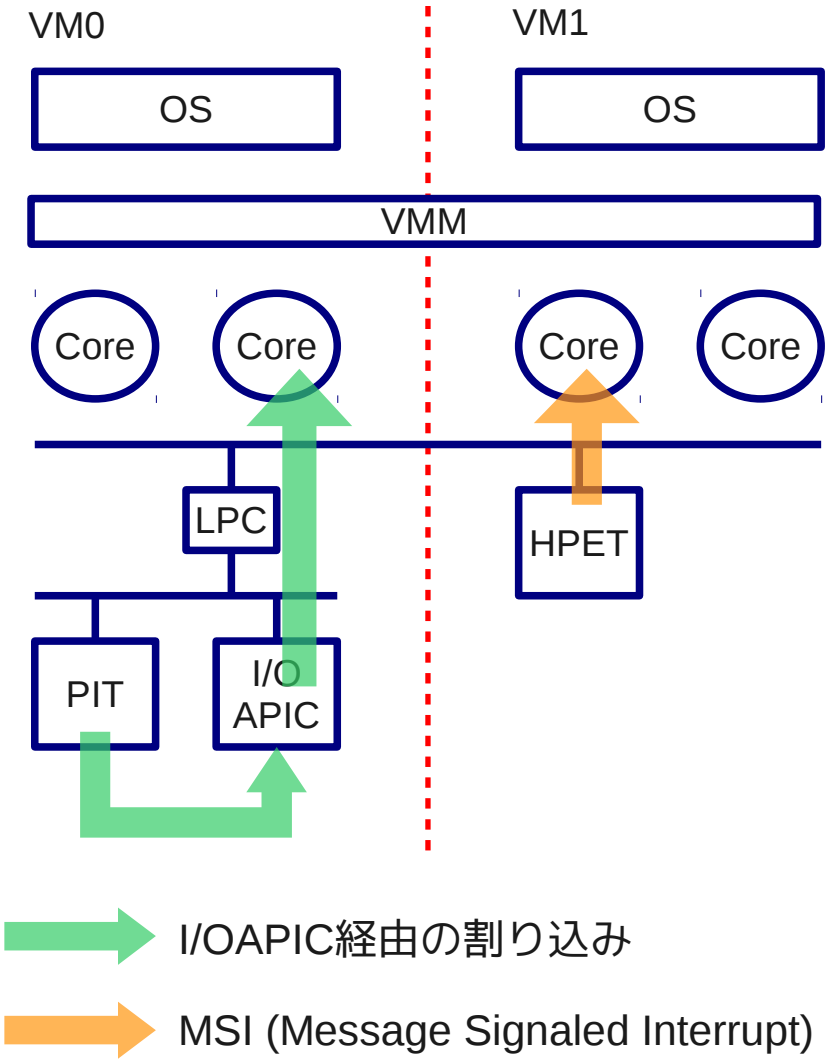


VM1に割り当てたことのあるI/Oデバイス

	HWのMSI対応	Linux	FreeBSD
AHCI (SATA)	対応	動作	動作
Realtek RTL8111E (NIC)	対応	動作	動作
xHCI (USB 3.0)	対応	動作	動作
GeForce 8400 GS (VGA)	対応	動作	動作しない
EHCI (USB 2.0)	非対応	動作しない	動作しない

PITとHPET

- PIT
  - VM0に割り当て
- HPET
  - VM1に割り当て
  - 常にMSIを有効にした状態で使用
  - OSがレガシー割り込みを有効にした場合、VMMがI/OAPIC経由の割り込みをエミュレート



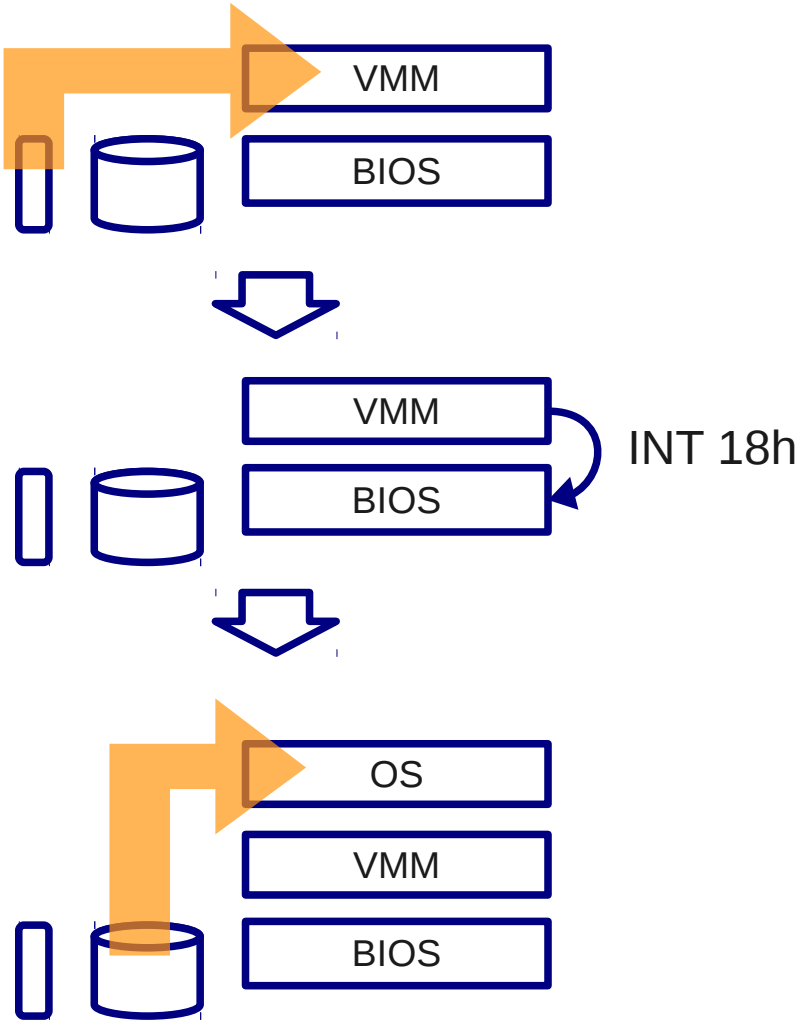
PIT: Programmable Interval Timer
HPET: High Precision Event Timer

VM1向けにPITのエミュレート

- ゲストBIOSが、経過時間の測定などにPITを使用
- TSCの値をもとに、PITのカウンタの値をVMMがエミュレート
- ISA IRQ0がPIC(8259A)によってマスクされるまでの間、PITによる周期割り込みをエミュレート

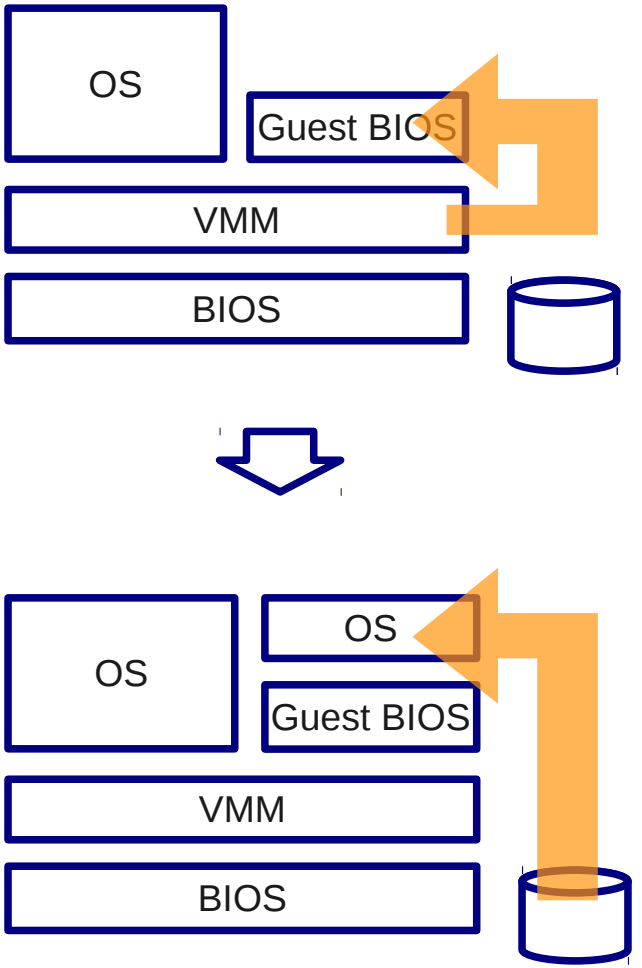
VM0のOSの起動

- BIOSはUSBメモリに格納されているVMMをロード
- VMMはINT 18Hを実行
  - ブートに失敗した時に呼び出すBIOSコール
- BIOSはHDDからOSをロード



VM1のOSの起動

- VMMのロード時にゲストBIOSもロードし、VMMのメモリ中に保存しておく
- VM1の起動時、VMMがゲストBIOSをVM1のメモリにロード
- ゲストBIOSがVM1に割り当てられたHDDからOSをロード



ゲストBIOS

- SeaBIOSを利用
  - 16ビット X86 BIOSのオープンソース実装[3]
  - AHCIドライバが含まれている
- VMMの設計に合わせてカスタマイズ
  - APIC IDが不連続でも、割り当てられた論理プロセッサに対応するProcessor Local APIC StructureをACPI MADTに登録
  - I/Oデバイスにリソースを割り当てない
  - OSに物理ディスティネーションモードを要求

[3] Wikiページ「SeaBIOS」を和訳 http://www.seabios.org/SeaBIOS

Linuxの起動時に発生した問題

- タイマ割り込み(HPET/PIT)がISA IRQ0で発生しないと、タイマ割り込みのチェック処理でパニック
  - HPETのMSI割り込みをGSI 2のI/O APICの経由の割り込みに見せる
  - ゲストBIOSのACPI MADTにて、ISA IRQ 0とGSI 2を関連付け

- シリアルコントローラ(UART)の割り込みが発生しないと、sttyプロセスがスリープし続け、起動しない
  - シリアルコントローラのTBR Empty割り込みをエミュレート

TSC: Time Stamp Counter
PIC: Programmable Interrupt Controller

Free BSDの起動時に発生した問題

- ブートローダがINT 15h / AH=86h BIOSコールを呼び出した時に、ゲストBIOSがRTCの割り込み待ちでストール
  - RTC割り込みのエミュレートは実装していない
  - RTCの代わりに、TSCを参照しながらビジーループするようゲストBIOSを改造

- ACPI MADTに記述されているプロセッサを検出できないと、FreeBSDがページフォールト
  - ホストBIOSのMADTをVMMが書き換え、VM1に割り当てた論理プロセッサをVM0のOSが認識しないようにした

VM1の再起動

1.OSによる、PCのリセット要求を検出

2.論理プロセッサをリアルモードに戻し、レジスタをデフォルト値に設定

3.メモリのクリア

4.ゲストBIOSをメモリへ再ロード

5.割り当てられたデバイスのリセット

6.ゲストBIOSの実行を開始

VM1のFree BSDの再起動

- 再起動中にAHCIタイムアウトが発生し、ルートファイルシステムのマウント失敗

```
ahcich1: Timeout on slot 0 port 0
ahcich1: is 00000002 cs 00000000 ss 00000000 rs 00000001 tfd 50 serr 00000000 cmd
0004c017
(aprobe0:ahcich1:0:0:0): ATA_IDENTIFY. ACB: ec 00 00 00 00 40 00 00 00 00 00 00
(aprobe0:ahcich1:0:0:0): CAM status: Command timeout
(aprobe0:ahcich1:0:0:0): Error 5, Retry was blocked
run_interrupt_driven_hooks: still waiting after 60 seconds for xpt_config
ahcich1: Timeout on slot 0 port 0
ahcich1: is 00000002 cs 00000000 ss 00000000 rs 00000001 tfd 50 serr 00000000 cmd
0004c017
(aprobe0:ahcich1:0:0:0): ATA_IDENTIFY. ACB: ec 00 00 00 00 40 00 00 00 00 00 00
(aprobe0:ahcich1:0:0:0): CAM status: Command timeout
(aprobe0:ahcich1:0:0:0): Error 5, Retry was blocked
SMP: AP CPU #1 Launched!
Trying to mount root from ufs:/dev/ada0p2 [rw]...
mountroot: waiting for device /dev/ada0p2 ...
Mounting from ufs:/dev/ada0p2 failed with error 19.
```

まとめ

- 一台のパソコンでOSを二個動作
  - CPU、メモリ、I/Oを各OSに割り当て、OSが直接制御
  - ホストOSなしで各OSが独立動作

- 設計のポイント
  - 割り当てられたリソースのみをOSに認識させる
  - 割り当てられていないリソースへのOSやI/Oデバイスによるアクセスを制限する
  - ホストBIOSを使用してVM0のOSをロード、ゲストBIOSを使用してVM1のOSをロード

おわりに

- 趣味としてコツコツ開発してきた
- だいぶ動くようになってきているので、何かに応用できるかもしれない
- 本日の発表がきっかけとなって、新しい取り組みができたら嬉しい

TinyVisor

http://sourceforge.jp/projects/tinyvisor/

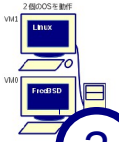
はじめに

- TinyVisorの紹介、および、開発の経緯と今後の目標について説明
- TinyVisorの設計について説明

2

TinyVisorとは

- 一台のパソコン(PC)でOSを二個動作させるためのハイパーバイザ(VMM)
  - CPU、メモリ、I/Oを各OSに割り当て、OSが直接制御
  - ホストOSなしでOSが独立動作
- オープンソースソフトウェアをベースに開発し成果物を公開
  - VMM: BitVisor(BSDライセンス)を改造
  - ゲストBIOS: SeaBIOS(LGPL)をカスタマイズ



3

使い方

1. ディスプレイ、キーボード、マウス、ビデオカード、HDD、 HBAカード(例えばAHCI)、NICカードなどのデバイスを２組用意
2. ２つのHDDに、それぞれのOSをインストール
3. USBメモリにVMMとゲストBIOSをインストール

```
./install_to_usb.sh -c 'vm0.boot_int18 vmm=vm0,vm1 vm0.cpu=4,6
vm1.mem=8000C000-21ffffff vm1.pci=00:01.0,00:14.3,00:1c.5,00:1c.7' /dev/sdd
```

4. USBメモリと、２組用意したデバイスを全てPCに接続
5. PCの電源を入れると、OSが2個同時に起動

AHCI: Advanced Host Controller Interface (SATAのコントローラのインタフェース

4

5

開発の経緯

- STEP1 (2012年3月)
  - 2個のOSを動作
- STEP2 (2013年5月)
  - VM1の再起動

6

今後の目標

- STEP3 (2014年?)
  - VM0の再起動
- STEP4 (2015年?)
  - VMごとにVMMの独立性向上
- STEP5 (2016年?)
  - VMMの部分アップデート

7

応用の可能性

- 夫婦でパソコンを共用？
  - 同じ部屋で隣り合って使えば会話が増える？
- ネット用パソコンと、自宅サーバを、統合？
  - 自宅のパソコンを常駐して、夏の暑さを軽減？
- 製品に組み込む？
  - OSが二つ動けば面白いことができるかも？

8

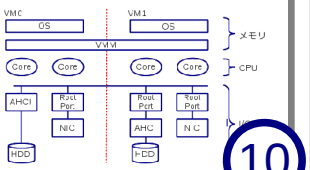
重複するCPU、メモリ、I/O

- CPU
  - マルチコア
    ハイパースレッディングが有効なら論理プロセッサは二つの２倍
- メモリ
  - 数ギガバイトから数十ギガバイト
- I/O
  - チップセット内蔵AHCI(SATA)＋マザーボード上のAHCIチップ
  - xHCI(USB3.0)＋EHCI(USB2.0)
  - PIT＋HPET

PIT: Programmable Interval Timer
HPET: High Precision Event Timer

9

CPU、メモリ、I/Oの割り当て



10

CPUの割り当て

- 論理プロセッサ単位
  - ハイパースレッディングが無効な場合は論理プロセッサ=コア
  - ハイパースレッディングが有効な場合はスレッド単位
- APIC IDで指定
  - ローカルAPICに付与されたID
  - 論理プロセッサを特定可能

```
./install_to_usb.sh -c 'vm0.boot_int18 vmm=vm0,vm1
vm1.cpu=4,6 vm1.mem=8000C000-21ffffff
vm1.pci=00:01.0,00:14.0,00:1c.5,00:1c.7' /dev/sdd
```

11

割り当てた論理プロセッサをOSに認識させる

- OSは、ACPIのMADTのProcessor Local APIC Structureを参照して論理プロセッサを認識
- VM0
  ホストBIOSが生成したStructureのうち、VM1に割り当てた論理プロセッサのStructureをVMMが無効化
- VM1
  ゲストBIOSが、VM1に割り当てられた論理プロセッサを検出し、Structureを生成

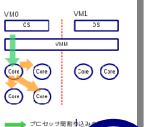
ACPI(Multiple APIC Description Table
[1] Advanced Configuration and Power Interface Specification, Revision 4.0a を参照)
http://www.acpi.info

オフセット	フィールド名	説明
0	Type	常に0
1	Length	常に8
2	ACPI Processor ID	この論理プロセッサに対応するACPI Processor Objectに対応付けられた値
3	APIC ID	論理プロセッサに対応付けられているAPIC ID
4-7	Flags	ビット0

12

プロセッサ間割り込みの送信

- OSがプロセッサ間割り込みを要求
- OSが宛先を指定していれば、その割り込み要求をVMMがパススルー
- OSが宛先として「全プロセッサ」や「他のプロセッサ全て」を指定していれば、割り当てられた論理プロセッサのみを宛先にした割り込みにVMMが変換



13

割り込み ディスティネーションモード

- プロセッサ間割り込みや、I/Oデバイスの割り込みの宛先の指定方法
- VM0
  - 論理ディスティネーションモード
  - 物理ディスティネーションモード
- VM1
  - 物理ディスティネーションモードのみ
    ゲストBIOSのACPI TABLEのフラグで物理ディスティネーションモードをOSに要求

14

メモリの割り当て

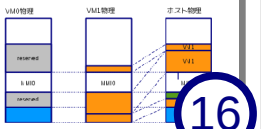
- ページ単位
  - 4KB単位
  - Shadow PagingやEPT(Extended Page Table)でアドレス変換するため
- 物理アドレスの範囲で指定

```
./install_to_usb.sh -c 'vm0.boot_int18 vmm=vm0,vm1
vm1.cpu=4,6 vm1.mem=8000C000-21ffffff
vm1.pci=00:01.0,00:14.0,00:1c.5,00:1c.7' /dev/sdd
```

15

各VMの物理アドレスとホスト物理アドレスの関係

- VMMが使用するメモリと、VM1に割り当てたメモリを、VM0のOSにreserved領域として見せる
- VM1割り当てたメモリを、アドレス変換して、VM1のOSに見せる



16

割り当てたメモリをOSに認識させる

- OSは、INT 15H, E820H BIOSコールでメモリマップを取得
  メモリマップは、右のAddrase Range Descriptor Structureの配列です
- VM0
  INT 15H, E820H BIOSコールをフックして、VMMがメモリマップを返却
- VM1
  ゲストBIOSが、予めVMMからメモリの範囲を取得しておき、INT 15H, E820H BIOSコールにメモリマップを返却する

INT 15H, E820H BIOSコールで取得できるAddress Range Descriptor Structure [2]

オフセット	フィールド
0-3	BaseAddrLow
4-7	BaseAddrHigh
8-11	LengthLow
12-15	LengthHigh
16-19	Type
20-23	Extended Attributes

[2] Advanced Configuration and Power Interface Specification, Revision 4.0a を参照)
http://www.acpi.info

17

I/Oの割り当て

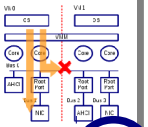
- Root Port単位
  - Root Portを割り当てると、その背後のデバイスも一緒に割り当てられる
- Root Portのバス番号、デバイス番号、ファンクション番号で指定

```
./install_to_usb.sh -c 'vm0.boot_int18 vmm=vm0,vm1
vm1.cpu=4,6 vm1.mem=8000C000-21ffffff
vm1.pci=00:01.0,00:14.0,00:1c.5,00:1c.7' /dev/sdd
```

18

OSからI/Oデバイスへのアクセス

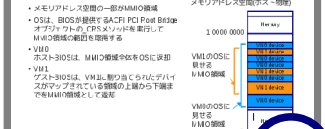
- 自分のVMに割り当てられたデバイスへのアクセスは許可
- 別のVMに割り当てられたデバイスに対するアクセスは禁止
  - OSがリードアクセスすると、VMMがOS にAll 1Fを返す
  - OSがライトアクセスすると、VMMが無視化
- VMMはアドレスを変換しない
- OSはBus 0を探索するため、割り当てたRoot Portやそのデバイスを見つけることができる



19

MMIO領域をOSに認識させる

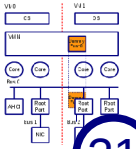
- メモリアドレス空間の一部がMMIO領域
- OSは、BIOSが提供するACPI PCI Root Bridge オブジェクトの_CRSメソッドを実行してMMIO領域の範囲を取得する
- VM0
  ホストBIOSは、MMIO領域全体をOSに返却
- VM1
  ゲストBIOSは、VM1に割り当てられたデバイスがアップされている領域の上限から下限までをMMIO領域として返却



20

マルチファンクションデバイス

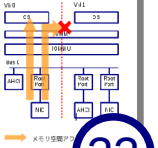
- Root Portがマルチファンクションデバイスの場合かつ Root Portを割り当ててもファンクションの少なくとも1つはOSが検出可
- VMMがダミーのファンクション0をエミュレートし、割り当てたRoot Portを OSが見つけられるようにする



21

I/Oデバイスからメモリへのアクセス

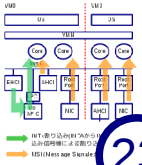
- VT-dのIOMMUによってアドレス変換
- VMに割り当てられたメモリ以外へのアクセスは禁止
  - I/Oデバイスから見ると、存在しないメモリにアクセスした時と同じ振る舞い



22

I/Oデバイスからの割り込み

- VM0
  - INTx割り込みでI/OAPIC経由でコアへ
  - MSIで直接コアへ
- VM1
  - MSIで直接コアへ



23

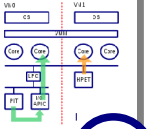
VM1に割り当てたことのあるI/Oデバイス

	HWのMSI対応	Linux	FreeBSD
AHCI (SATA)	対応	動作	動作
Realtek RTL8111E (NIC)	対応	動作	動作
xHCI (USB 3.0)	対応	動作	動作
GeForce 8400 GS (VGA)	対応	動作	動作しない
EHCI (USB 2.0)	非対応	動作しない	動作しない

24

PITとHPET

- PIT
  VM0に割り当て
- HPET
  - VM1に割り当て
  - 常にMSIを有効にした状態で使用
  - OSがレガシー割り込みを有効にした場合、VMMがI/OAPIC経由の割り込みをエミュレート



PIT: Programmable Interval Timer
HPET: High Precision Event Timer

25

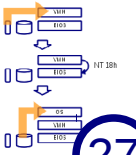
VM1向けにPITのエミュレート

- ゲストBIOSが、経過時間の測定などにPITを使用
- TSCの値をもとに、PITのカウンタの値をVMMがエミュレート
- ISA IRQ0がPIC(8259A)によってマスクされるまでの間、PITによる周期割り込みをエミュレート

26

VM0のOSの起動

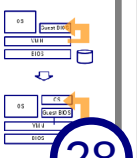
- BIOSはUSBメモリに格納されているVMMをロード
- VMMはINT 18Hを実行
  - ブートに失敗した時に呼び出すBIOSコール
- BIOSはHDDからOSをロード



27

VM1のOSの起動

- VMMのロード時にゲストBIOSもロードし、VMMのメモリ中に保存しておく
- VM1の起動時、VMMがゲストBIOSをVM1のメモリにロード
- ゲストBIOSがVM1に割り当てられたHDDからOSをロード



28

ゲストBIOS

- SeaBIOSを利用
  - 16ビット x86 BIOSのオープンソース実装[3]
  - AHCIドライバが含まれている
- VMMの設計に合わせてカスタマイズ
  - APIC IDが不連続でも、割り当てられた論理プロセッサに対応するProcessor Local APIC StructureをACPI MADTに登録
  - I/Oデバイスにリソースを割り当てない
  - OSし物理ディスティネーションモードを要求

[3] VMwareページ SeaBIOSを参照 http://www.seabios.org/SeaBIOS

29

Linuxの起動時に発生した問題

- タイマ割り込み(HPET/PIT)がISA IRQ0で発生しないと、タイマ割り込みのチェック処理でパニック
  - HPETのMSI割り込みをOSおよびI/O APICの経由の割り込みに変更
  - ゲストBIOSのACPI MADTで、ISA IRQ 0にOSと同様の設定
- シリアルコントローラ(UART)の割り込みが発生しないと、sttyプロセスがスリープし続け、起動しない
  - シリアルコントローラのIER Empty割り込みをエミュレート

TSC: Time Stamp Counter
PIC: Programmable Interrupt Controller

30

Free BSDの起動時に発生した問題

- ブートローダがINT 15h / AH=86h BIOSコールを呼び出した時に、ゲストBIOSがRTCの割り込み待ちでストール
  - RTC割り込みのエミュレートは実装していない
  - RTCの代わりに、TSCを参照しながらビジーループするようゲストBIOSを改造
- ACPI MADTに記述されているプロセッサを検出できないと、FreeBSDがページフォールト
  - ホストOSのMADTをVMMが書き換え、VM1に割り当てたプロセッサをVM0のOSが認識しないようにした

31

VM1の再起動

1. OSによる、PCのリセット要求を検出
2. 論理プロセッサをリアルモードに戻し、レジスタをデフォルト値に設定
3. メモリのクリア
4. ゲストBIOSをメモリへ再ロード
5. 割り当てられたデバイスのリセット
6. ゲストBIOSの実行を開始

32

VM1のFree BSDの再起動

- 再起動中にAHCIタイムアウトが発生し、ルートファイルシステムのマウント失敗

```
ahcich1: Timeout on slot 0 port 0
ahcich1: is 00000002 cs 00000000 ss 00000000 rs 00000001 tfd 50 serr 00000000 cmd
0004c017
(aprobe0:ahcich1:0:0:0): ATA_IDENTIFY. ACB: ec 00 00 00 40 00 00 00 00 00 00 00
(aprobe0:ahcich1:0:0:0): CAM status: Command timeout
(aprobe0:ahcich1:0:0:0): Error 5, Retry was blocked
(aprobe0:ahcich1:0:0:0): Unfreezing devq for sim_config
ahcich1: Timeout on slot 0 port 0
ahcich1: is 00000002 cs 00000000 ss 00000000 rs 00000001 tfd 50 serr 00000000 cmd
0004c017
(aprobe0:ahcich1:0:0:0): ATA_IDENTIFY. ACB: ec 00 00 00 40 00 00 00 00 00 00 00
(aprobe0:ahcich1:0:0:0): CAM status: Command timeout
(aprobe0:ahcich1:0:0:0): Error 5, Retry was blocked
(xpt_AP_CPU:0:1):Launching!
Trying to mount root from ufs:/dev/ada0p2 [rw]...
mountroot: waiting for device /dev/ada0p2 ...
Mounting from ufs:/dev/ada0p2 failed with error 19.
```

33

まとめ

- 一台のパソコンでOSを二個動作
  - CPU、メモリ、I/Oを各OSに割り当て、OSが直接制御
  - ホストOSなしで各OSが独立動作
- 設計のポイント
  - 割り当てられたリソースのみをOSに認識させる
  - 割り当てられていないリソースへのOSやI/Oデバイスによるアクセスを制限する
  - ホストOSを使用してVM0のOSをロードし、ゲストOSをVM1のOSをロード

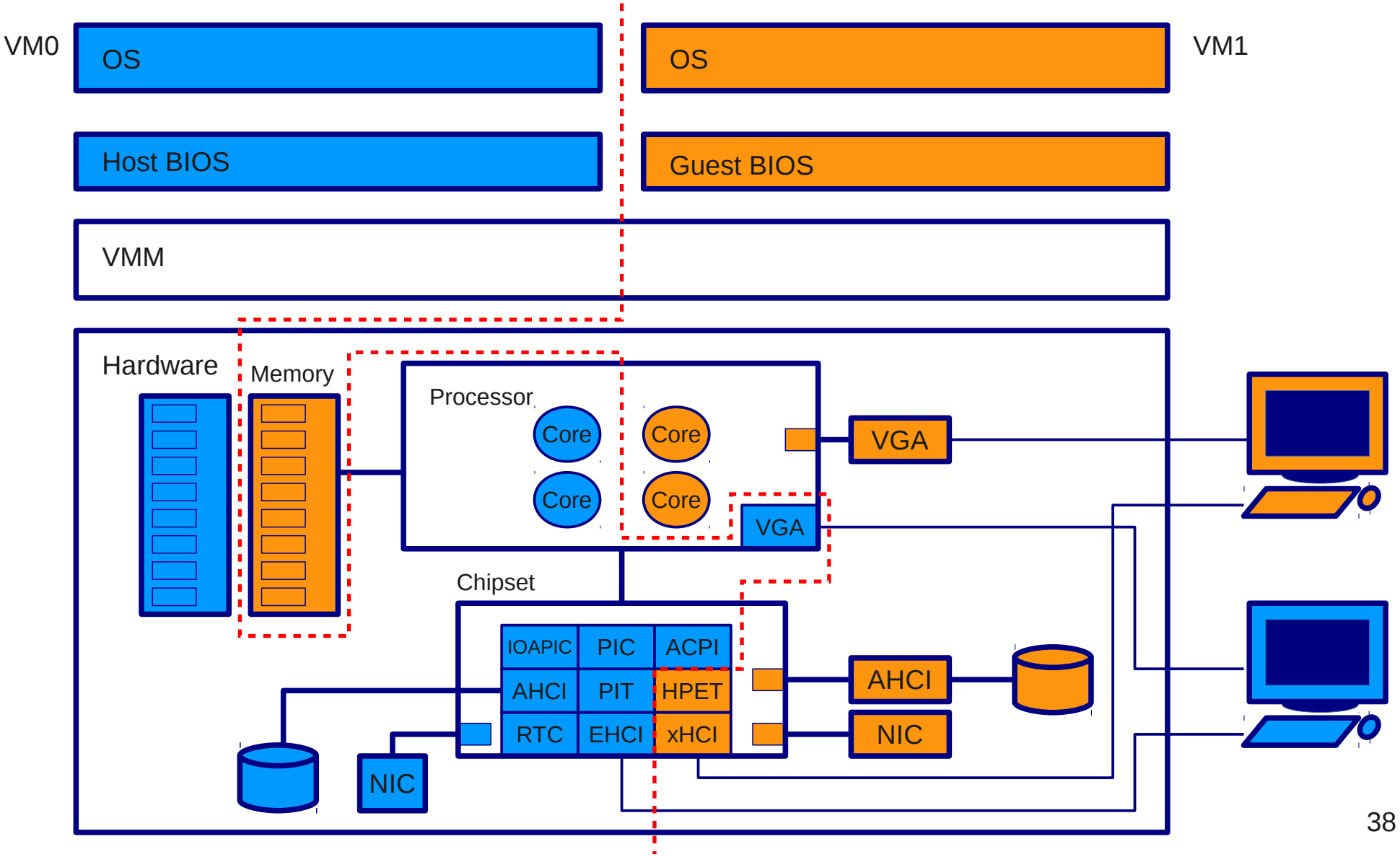
34

おわりに

- 趣味としてコツコツ開発してきた
- だいぶ動くようになってきているので、何かに応用できるかもしれない
- 本日の発表がきっかけとなって、新しい取り組みができたら嬉しい

35

Tiny Visor概念図



TinyVisor

http://sourceforge.jp/projects/tinyvisor/