igel

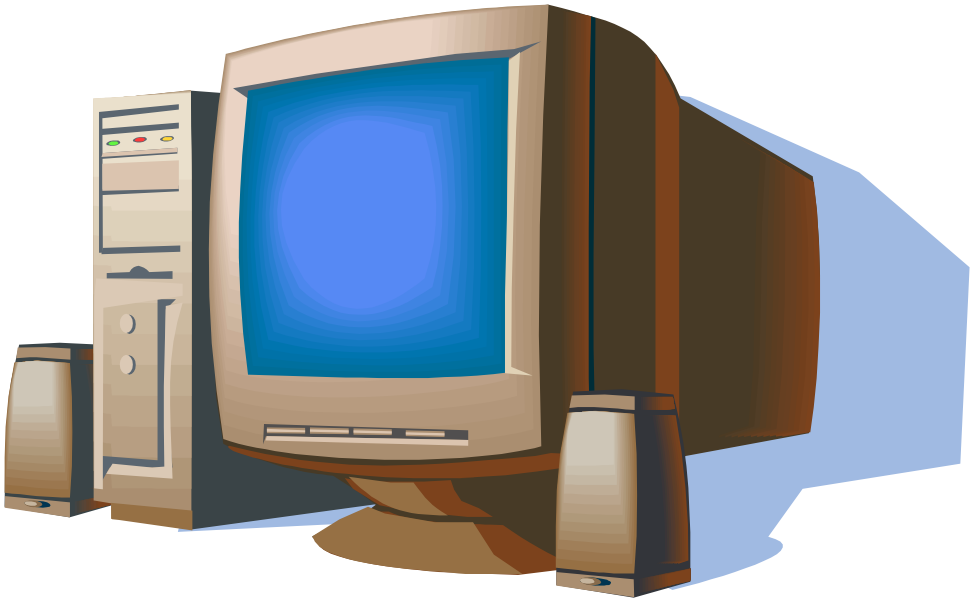
Technology Consulting Company
Research, Development &
Global Standard

準パススルー型VMM開発の難しいところ

榮樂 英樹
株式会社イーゲル

VMとは

- VMMというソフトウェアによって作り出された仮想的なPCハードウェアのこと



VMMが仮想化するPC内部のデバイスの例

igel

- CPU・ROM・RAM
- 割り込みコントローラー
- プログラマブル・インターバル・タイマー
- カレンダー時計
- キーボードコントローラー
- ビデオコントローラー
- スピーカー
- 補助記憶装置 (HDD, FDD, DVDドライブなど)
- etc.

準パススルー型VMMについて

igel

- 暗号化などの処理を施したいデバイスのみを仮想化し、ほとんどのデバイスをVMに直接見せる
  - CPU: 基本的に通常のVMMと同じである
  - RAM: 一部分をVMMが占拠する
  - ACPI (電源制御): ほとんどをVMに直接見せる
  - ファームウェア (BIOS等): VMに直接見せる
  - ビデオコントローラー: VMに直接見せる
  - その他: そのまま見せたくないものだけを仮想化する
- 仮想化するデバイスが少ないので、通常のVMMより簡単になる

準パススルー型VMM開発の何が難しいか

igel

- 新しいCPUへの対応
- パススルーしてはいけない機能への対応

新しいCPUへの対応

igel

- CPUID命令等で得られる未知の機能の情報について隠蔽していない場合、CPUの持つ新しい機能が使用されて問題が発生することがある
  - BitVisorは新しい機能を隠蔽していない
- 実際に問題が発生した例:
  - **XSETBV命令** (Linux on Let'snote CF-W8): 新しい制御レジスターを変更する命令であり、ゲストOSが実行するとVMMに制御が移るため (BitVisor 1.1で対応)
  - **Page1GB** (1GiBのページ) (Linux AMD64): ページテーブル構造の変更があるため (BitVisor 1.3で対応)
- 問題が発生しない例:
  - **AES-NI**: ただの演算命令であり、ゲストOSが使用しても全く問題ない

パススルーしてはいけない機能への対応

- Local Advanced Programmable Interrupt Controller (LAPIC)
- Advanced Configuration and Power Interface (ACPI)
- ファームウェア

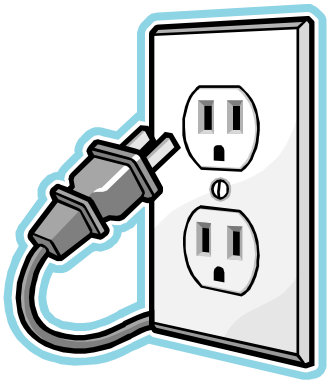
Local Advanced Programmable Interrupt Controller

igel

割り込み制御、特に、マルチプロセッサ/マルチコア環境において、他のプロセッサを初期化・開始したり、プロセッサ間割り込み (IPI) を生成したりするときに用いられる

- Intel VT-x環境ではパススルーしてよい
  - INITおよびStartup IPIによるVM Exitを処理することで、仮想化する
- AMD-V環境では、VM内をINIT状態に置くことができないため、仮想化の必要がある
  - Startup IPI生成をフックする
  - BitVisor 1.3で対応

Advanced Configuration and Power Interface (ACPI)



電源制御の規格

- 本当はすべてゲストOSに任せたい
  - 電源制御を正しく行わないと、ノートPCの電池が持たない等の問題につながる
  - 準パススルー型VMMではすべてゲストOSに任せることで問題を回避できる

- 準パススルー型VMMが一部面倒を見なければならない理由
  - スリープ対応の問題
  - シリアルポートの電源問題

スリープ対応の問題

igel

- ACPIのスリープの種類
  - S1 Sleeping State: CPU・チップセットのコンテキスト保持
  - S2 Sleeping State: CPUとシステムキャッシュコンテキストが失われる
  - S3 Sleeping State: (Sleep, Standby, Suspend-to-RAM) システムメモリ以外のコンテキストが失われる
  - S4 Sleeping State: Hibernation
  - S5 Soft Off State: Power off

- S2とS3が問題
  - CPUのコンテキストが失われるため、VMMが消えてしまう

スリープの制御方法

igel

- PM1a_CNTというレジスターに適切な内容を書き込む
  - I/Oアドレスは**FADT** (Fixed ACPI Description Table) というテーブルを参照する
- 書き込む値に応じてS1, S2, S3, S4, S5のいずれかに遷移できる
  - 書き込む値は**DSDT** (Differentiated System Description Table) というテーブルを参照する

FADTの参照 (SignatureはFACP)

```
RSD PTR @ 0xf97a0
  0000: 52 53 44 20 50 54 52 20 e1 44 45 4c 4c 20 20 02  RSD PTR .DELL  .
  0010: 00 30 ee 7f 24 00 00 00 80 30 ee 7f 00 00 00 00  .0..$....0......
  0020: bf 00 00 00                                      ....
```

```
RSDT @ 0x7fee3000
  0000: 52 53 44 54 40 00 00 00 01 f0 44 45 4c 4c 20 20  RSDT@.....DELL
  0010: 46 58 30 39 20 20 20 00 31 2e 30 42 41 57 52 44  FX09   .1.0BAWRD
  0020: 00 00 00 00 80 31 ee 7f c0 73 ee 7f 00 74 ee 7f  .....1...s...t..
  0030: 40 74 ee 7f c0 75 ee 7f 00 73 ee 7f a0 7c ee 7f  @t...u...s...|..
```

```
FACP @ 0x7fee3180
  0000: 46 41 43 50 74 00 00 00 01 13 44 45 4c 4c 20 20  FACPt.....DELL
  0010: 46 58 30 39 20 20 20 00 31 2e 30 42 41 57 52 44  FX09   .1.0BAWRD
  0020: 00 00 00 00 00 00 e9 7f 00 32 ee 7f 01 00 09 00  .........2......
  0030: b2 00 00 00 a1 a0 00 34 00 04 00 00 00 00 00 00  .......4........
  0040: 04 04 00 00 00 00 00 00 00 00 00 00 08 04 00 00  ................
  0050: 20 04 00 00 00 00 00 00 04 02 00 04 10 00 00 00   ...............
  0060: 65 00 e9 03 00 00 00 00 01 01 0d 00 00 00 00 00  e...............
  0070: a5 04 00 00                                      ....
```

PM1a_CNTのI/Oポートアドレス

DSDTのアドレス

DSDTの参照

```
DSDT @ 0x7fee3200
  0000: 44 53 44 54 fc 3f 00 00 01 50 44 45 4c 4c 20 20  DSDT.?...PDELL
  0010: 41 57 52 44 41 43 50 49 00 10 00 00 4d 53 46 54  AWRDACPI....MSFT
  0020: 00 00 00 03 10 43 05 5c 5f 50 52 5f 5b 83 11 5c  .....C.¥_PR_[..¥
  0030: 2e 5f 50 52 5f 43 50 55 30 00 00 00 00 00 00 5b  ._PR_CPU0......[
  0040: 83 11 5c 2e 5f 50 52 5f 43 50 55 31 01 00 00 00  ..¥._PR_CPU1....
  0050: 00 00 5b 83 11 5c 2e 5f 50 52 5f 43 50 55 32 02  ..[..¥._PR_CPU2.
  0060: 00 00 00 00 00 5b 83 11 5c 2e 5f 50 52 5f 43 50  .....[..¥._PR_CP
  0070: 55 33 03 00 00 00 00 00 08 5c 5f 53 30 5f 12 0a  U3.......¥_S0_..
  0080: 04 0a 00 0a 00 0a 00 0a 00 08 5c 5f 53 33 5f 12  ..........¥_S3_.
  0090: 0a 04 0a 05 0a 00 0a 00 0a 00 08 5c 5f 53 34 5f  ...........¥_S4_
  00a0: 12 0a 04 0a 06 0a 00 0a 00 0a 00 08 5c 5f 53 35  ............¥_S5
  00b0: 5f 12 0a 04 0a 07 0a 00 0a 00 0a 00 08 46 4c 41  _............FLA
```

- 中身はAML (ACPI Machine Language) という仕様に基づいていて、機械語プログラムのようになっている
- これを解釈していくと¥_S1, ¥_S3などの名前で、書き込む値が見つかる

AML (ACPI Machine Language)の解釈方法

igel

- ACPIの仕様書にAML仕様が載っている
- Intelのリファレンス実装がある (ACPICA)
  - Linux kernelなどにも使われている
  - ASL (ACPI Source Language) からコンパイルしたり、逆コンパイルしたりするためのiaslというコマンドもある
- 逆コンパイル例:

```
DefinitionBlock ("DSDT.aml", "DSDT", 1, "DELL  ", "AWRDACPI", 0x00001000)
{
    Scope (¥_PR)
    {
        Processor (¥_PR.CPU0, 0x00, 0x00000000, 0x00) {}
        Processor (¥_PR.CPU1, 0x01, 0x00000000, 0x00) {}
        Processor (¥_PR.CPU2, 0x02, 0x00000000, 0x00) {}
        Processor (¥_PR.CPU3, 0x03, 0x00000000, 0x00) {}
    }
```

BitVisorにおけるAML解釈

igel

- 興味のある部分は一部だけなので、独自実装した
  - 仕様書をもとに解釈ルーチンを機械的に作成した

AML解釈の難しさ

igel

- 引数の終わりが識別できない
- 時間がかかる
- 仕様書に沿っていない実装がある

引数の終わりが識別できない

igel

- iaslコマンドによる逆コンパイル結果の一部:

```
If (¥_SB.PCI0.PEX0.XPM1 (0x00)) {
    Notify (¥_SB.PCI0.PEX0, 0x02)
}
```

- 2行目をちょっと編集してコンパイル:

```
    Notify (¥_SB.PCI0.PEX0.XPM1 (0x01), 0x02)
```

- そして逆コンパイル:

```
    Notify (¥_SB.PCI0.PEX0.XPM1, One)
    0x02
```

Intelのリファレンス実装でも正しく識別できていない!

時間がかかる

igel

- 機械的に作成した解釈ルーチンは、先頭から1バイトずつ順に処理し、解釈結果を絞り込んでいくため、時間がかかる
  - 前述の問題により絞り込みがなかなかできない場合がある

時間がかかる問題の対策

igel

- Method() の中身には興味がないので、Method() の中身を単なるバイト列として無視する
  - “Advanced Configuration and Power Interface Specification” Revision 4.0a

    ```
    DefMethod := MethodOp PkgLength NameString MethodFlags TermList
    ```

  - BitVisor

    ```
    case AML_DefMethod:
            addbuf (d, AML_MethodOp, AML_PkgLength,
                    AML_NameString, AML_MethodFlags, AML_ByteList,
                    AML_PkgEND, OK);
            break;
    ```

- 他のプロセッサで他の初期化処理を行っている間に、並行して解釈を行う (BitVisor 1.3)

仕様書に沿っていない実装がある (1/2)

igel

- Device() の直下には本来存在してはいけない If() が、存在する実装がある
- 仕様書によると:
  - DefDevice := DeviceOp PkgLength NameString ObjectList
  - ObjectList := Nothing | <Object ObjectList>
  - Object := NameSpaceModifierObj | NamedObj
  - NameSpaceModifierObj := DefAlias | DefName | DefScope
  - NamedObj := DefBankField | DefCreateBitField | DefCreateByteField | DefCreateDWordField | DefCreateField | DefCreateQWordField | DefCreateWordField | DefDataRegion | DefDevice | DefEvent | DefField | DefIndexField | DefMethod | DefMutex | DefOpRegion | DefPowerRes | DefProcessor | DefThermalZone

仕様書に沿っていない実装がある (2/2)

igel

- 例:
  - ONKYO TW2B-A31B7PH:

```
Scope (_SB) {
    Device (PCI0) {
        Device (SATA) {
            Name (_ADR, 0x00110000)
            If (LEqual (STCL, 0x0101)) {
```

  - HP ProBook 4320s:

```
Scope (¥_SB) {
    Device (PCI0) {
        Device (GFX0) {
            If (CondRefOf (FPED)) {
```

- 対策: Device() のObjectListだけ特別扱いする (BitVisor 1.1.1)

スリープを禁止する

- PM1a_CNTレジスターのI/Oをフックし、S2・S3への遷移ができないようにする
- DSDTの中の¥_S2, ¥_S3を消す
- BitVisorでは、以下のように¥_S3を¥_S3Dという名前に変更している

```
Name (¥_S3, Package (0x04)
{
    0x05,
    0x05,
    0x00,
    0x00
})
```

通常時

```
Name (¥_S3D, Package (0x04)
{
    0x05,
    0x05,
    0x00,
    0x00
})
```

スリープ禁止時

スリープに対応する

igel

- DSDTの中の値を保存しておく
- PM1a_CNTレジスターのI/Oをフックしておき、スリープが行われようとした時には、再開時に実行されるVMMのプログラムを転送し、そのアドレスから再開されるように設定を変更した後、実際にスリープするようにする

- 再開時にVMMがパスワード認証を行うことができるか
  - 画面出力、キーボード入力などのデバイスの再初期化が必要なため、難しい
  - 特定の環境に限定すれば、できるかも

シリアルポートの電源問題

igel

- Windowsを起動するとBitVisorからシリアルポートが使えなくなってしまう問題があった
- 原因は、DSDTに書かれているシリアルポートの電源制御プログラムがゲストOSによって実行されてしまうことだった
- 同プログラムの部分にNoopOpを埋めることによって回避している

```
Name (_HID, EisaId ("PNP0501"))
#中略
Method (_DIS, 0, NotSerialized)
{
    Noop
    Noop
    Noop
    #以下省略
```

BIOS (ファームウェア) について

igel

- パススルーでないVMMは仮想マシン用の独自のBIOSを持っているが、BitVisorはシステムのBIOSをそのまま利用する

- メリット
  - 起動ストレージの種類を区別せずに動作できる (CD、USB等)
  - 電源管理 (ACPI) もほとんどゲストOSまかせにできる
- デメリット
  - BIOSの実装ごとに異なる内容への対応が必要になる
  - ほとんどのBIOSはブラックボックスでデバッグが難しい

BIOSで問題となった内容

- BIOSでの64ビットモード移行
- SMM (System Management Mode)
- AHCI (Serial ATA)

BIOSでの64ビットモード移行 (1/3)

igel

- ThinkPad X200のTCG BIOS
  - TCG BIOSはリアルアドレスモードで呼び出さなければならない仕様になっていて、内部で64ビットのプロテクトモードへ移行していた

(参考) 従来のDisk BIOS等は、DOS環境において仮想8086モードから実行される場合があるため、内部でのプロテクトモード移行等は行われない

BIOSでの64ビットモード移行 (2/3)

igel

- ゲストOSからではなく、VMMから直接呼び出す時に制御レジスター等を保存しておく必要がある
  1. CR0, CR3, CR4、および、EFERを保存
  2. リアルアドレスモードへ移行
  3. TCG BIOS呼び出し
  4. CR0, CR3, CR4、および、EFERを復元
  - ただし、Intel Atom Z520などEFERを持たないプロセッサ、および、AMDなどEFERのLMAビットを変更しようとすると例外が生成されるプロセッサで、例外が発生しないようにする必要がある

BIOSでの64ビットモード移行 (3/3)

igel

- 64ビットモードではページングが必須だが、ページングの設定がOSでは使われない設定になっていることもある
  - ThinkPad X200では、OSではほとんど使用されていないCR0.WP=0の設定が使われていた
    - この設定では、特権レベル0の時に読み取り専用ページにも書き込みができる
    - この設定では、特権レベル0で、ページフォールトを用いたcopy-on-writeが実現できないため、通常は使用されていない
    - 80386ではこの設定しかできなかった
  - しかも、PTEだけでなくPDEなどに読み取り専用の設定がされており、PTEまでしか考慮していなかったシャドウページテーブルではページフォールトが発生し続けて先に進まない状況に陥った

SMM (System Management Mode) とは

igel

- 電源管理等を目的として386SLから導入された動作モードで、System Management Interrupt (SMI) によって起動される
- SMIハンドラーは通常は見えないところにあるファームウェア (BIOS) に入っていて、独立した動作環境で実行される (用途はマシン依存)

- 以下のI/O命令を実行するとSMIが発行される
  - outb %al, $0xB2 ; I/Oポート0xB2にALを出力

BIOSによるSMIの利用

igel

- BIOSでは例えば以下のようにしてSMIを発行している:

```
cmp %al, %al        ; フラグレジスターを偶数パリティにセット
outb %al, $0xB2     ; I/Oポート0xB2にALを出力
jp .                ; 偶数パリティの間ビジーウェイト
```

- SMI発行後、SMMで処理が実行され、フラグレジスターなどが更新された後、元のプログラムに戻る
- BitVisor開発初期段階において、ポートI/O命令をすべてフックしてVMMで代行していたところ、ThinkPad X60でWindows XPが起動しなかった
  - VMMで代行するときにはレジスターやアドレス空間が全く違うため処理が実行されず、無限ループに陥っていた
  - 問題を回避するため、フックをやめていた

SMIのフックをやめても発生する問題

igel

- TCG BIOSの中のSMI
  - ThinkPad X61において、フックをしていないにも関わらず、止まってしまう問題があった
  - 当初はTCG BIOSを無効化する機能を実装し、デフォルト設定ではゲストOSからTCG BIOSを呼び出すことができないようにして回避していた (BitVisor 1.1)
- Disk BIOSのReset Disk処理中のSMI
  - ThinkPad T410iにおいて、TCG BIOSと同じように止まってしまう問題があった
  - これは無効化するわけにはいかなかった

BIOSで止まってしまった原因(推測)

igel

- VMMのシャドウページテーブルが使われているので、実際にはページングが有効になっている
- アドレス空間をすべてマップしているわけではなく、実際にアクセスされた時のページフォールトでマップする

- **SMMのページフォールトはVMMで検出できない**
  - SMMに移行する時点でアクセスされていなかったページには、SMMのプログラムはアクセスすることができない

SMIで止まってしまう問題の解決方法 (BitVisor 1.1.1)

igel

- SMI生成の前に下位1MiBをマップするようにした
  - 下位1MiB: 起動時のリアルアドレスモードから参照できるアドレス範囲
- 詳細:
  - SMIを生成するポートI/O (通常0xB2, 正確にはACPIのテーブルで示されている) をフックする
  - 同ポートI/OをVMMが代行することはできないので、フックハンドラーでは、1MiBをマップした後、I/Oフックを一時的に無効にして、同じI/O命令から実行を再開する
  - CR3の変更など、TLBフラッシュの時には再びI/Oフックを有効にする

未解決の問題: Disk BIOSのRead/Write処理中のSMI

igel

- ThinkPad X220では、ディスクI/Oの処理がSMMで行われている
  - AHCIへのアクセスをVMMがフックできない!
- もし、独自のBIOSをVMMが提供していたならば気づかなかった問題
  - しかし、このSMIを許していると、悪意のあるプログラムは自由にディスクI/Oを実行できてしまうことになる

BIOSがAHCIのSerial ATAストレージにアクセスする方法

igel

BIOSはAHCIレジスターの32bitアドレスにアクセスできない

- Parallel ATAと同じ
  - ホストコントローラーがParallel ATA互換のアクセスに対応しており、BIOSは最初AHCIを無効にしておき、Parallel ATA互換の方法によってアクセスする
    例: ThinkPad X61, ThinkPad X200, Marvell 88SE91xx
- Index-Data Pair mechanism
  - PCIで設定されるI/Oポートの一部を用いてAHCIの各レジスターにアクセスする
    例: Let‘snote CF-W8, ThinkPad T410i, HP ProBook 4320s
  - PCI configuration spaceの一部を用いてAHCIの各レジスターにアクセスする (BitVisor 1.3で正式に対応)
    例: AMD FX-4100搭載PC (ドスパラPrime A Galleria FRS-X4)

BIOSによるAHCIへのアクセスでの問題

igel

- PxCIへの書き込みの際にはPxCMD.ST=1でなければならないとの記載がAHCIの仕様書にはある:

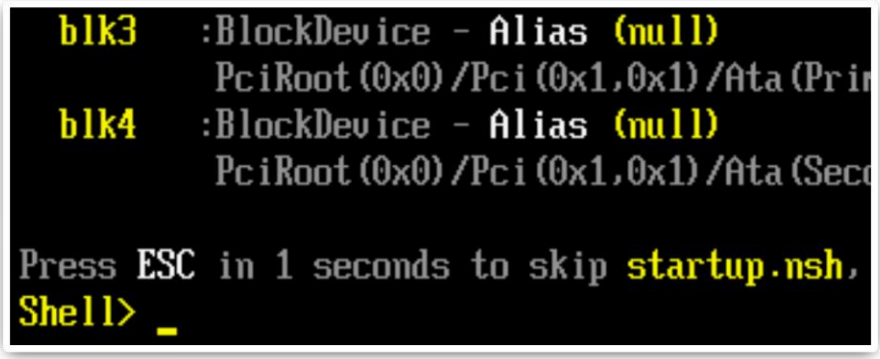
  (Intel “Serial ATA AHCI 1.3 Specification” より)

3.3.14 Offset 38h: PxCI – Port x Command Issue

Bit	Type	Reset	Description
31:0	RW1	0	**Commands Issued (CI):** This field is bit significant. Each bit corresponds to a command slot, where bit 0 corresponds to command slot 0. This field is set by software to indicate to the HBA that a command has been built in system memory for a command slot and may be sent to the device. When the HBA receives a FIS which clears the BSY, DRQ, and ERR bits for the command, it clears the corresponding bit in this register for that command slot. Bits in this field shall only be set to ‘1’ by software when PxCMD.ST is set to ‘1’. This field is also cleared when PxCMD.ST is written from a ‘1’ to a ‘0’ by software.

- ThinkPad T410iのBIOSは、PxCMD.STが0であるにも関わらず、PxCIへの書き込みを行う
  (BitVisor 1.1.1で対応)

UEFI起動のしくみ

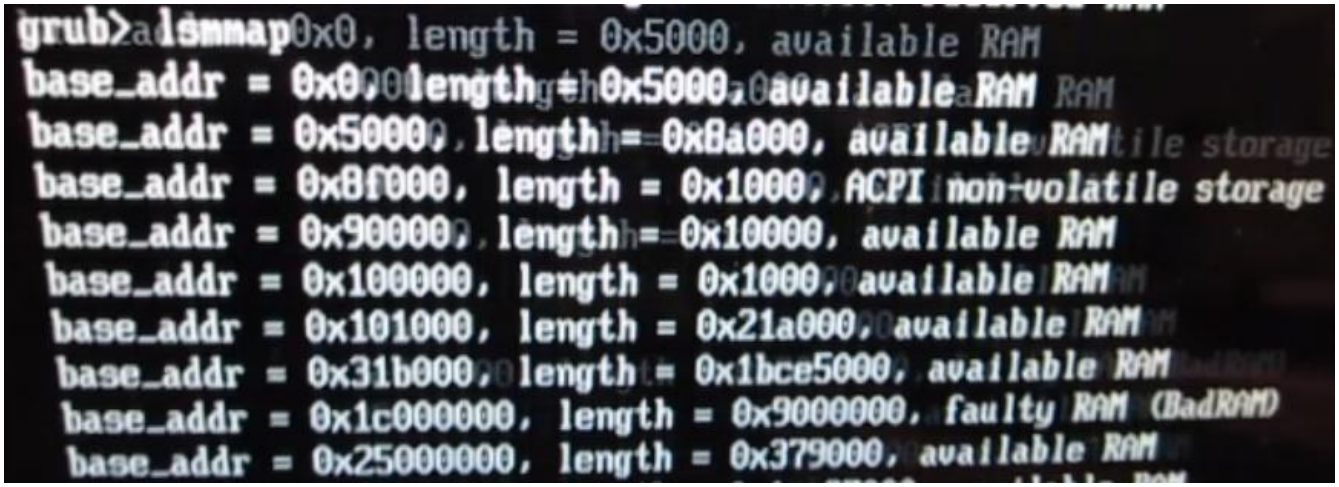


- UEFI用のプログラムというのを作ることができて、シェルからそのプログラムを実行したり、ファームウェアから自動的に特定のプログラムを実行させるようにしたりすることができる
- GNU GRUBのようなブートローダーも、UEFI対応版ではUEFI用のプログラムとなる
- シェルなどもUEFI用のプログラムであって、プログラムから他のプログラムを実行することができる (プロテクトモード用のDOSのような環境)
- BitVisorもUEFI用のプログラムとして読み込ませ、本来のブートローダーを実行するようにすればいいはず

メモリ

igel

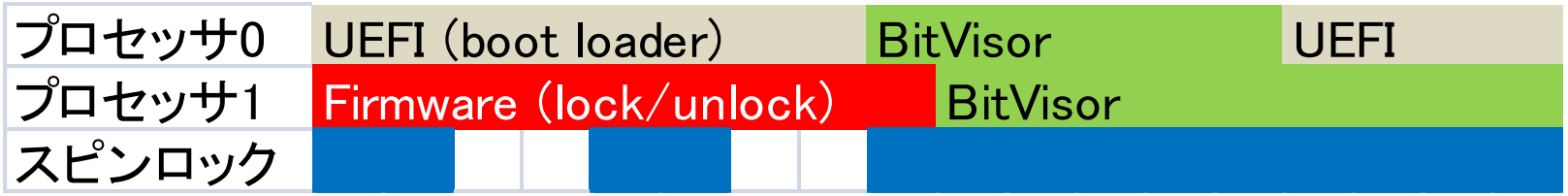
- UEFIの呼び出しでallocate_pagesというのがあって、指定したメモリ領域を使用不可とすることができる
- 従来のBIOSのように、int $0x15をフックしたりしなくても良いので簡単になる
- 例: GNU GRUBのlsmmapで確認 (faulty RAMの部分)



MacBook Air (Mid 2009) 対応 (1/2)

igel

- マルチプロセッサ/マルチコアの対応の問題
  - 2つ目のプロセッサ/コアで、スピンロックを使って何かやっている
  - スピンロックがロック状態のままBitVisorがマルチプロセッサの初期化をして止めてしまうと、後でスピンロックをロックしようとしても解除するプログラムがないのでフリーズしてしまう
  - 本来のOSは、UEFIの起動時に使用する機能を終了させ、以降使わないという状態にした上でマルチプロセッサの初期化を行うので、問題ないようだ。BitVisorはマルチプロセッサの初期化をした後でまたUEFIの機能を使用したいので、問題となる。



MacBook Air (Mid 2009) 対応 (2/2)

igel

- ACPI Enable、システムリセット、電源オフなどのタイミングでシステム全体がフリーズする問題

  キャッシュ制御 (PAT) の値の内容によって発生する。

  – 起動直後の値: 00 07 04 06 00 07 04 06
  – BitVisorが設定する値: 00 00 01 05 00 07 04 06
  – 設定を変更し、正常に動作する値: 00 01 05 06 00 07 04 06

  SMMで実行されるプログラムの問題か?

準パススルー型VMM開発の難しいところ まとめ

igel

- 新しいCPUへの対応
- パススルーしてはいけない機能への対応
  - Local Advanced Programmable Interrupt Controller (LAPIC)
    - マルチプロセッサ/マルチコア初期化処理
  - Advanced Configuration and Power Interface (ACPI)
    - スリープ対応の問題
    - シリアルポートの電源問題
    - AML (ACPI Machine Language) 解釈
  - ファームウェア
    - BIOS
      - SMM (System Management Mode)
      - AHCI
    - UEFI