ハイパーバイザーの作り方第１７回　仮想マシンの初期化と BHyVe のゲスト OS ローダ

はじめに

前回は、VT-d の詳細について解説しました。
今回は、仮想マシンの初期化と BHyVe のゲスト OS ローダについて解説していきます。

物理マシンの初期化と仮想マシンの初期化の違い

まず、物理マシンの初期化手順について考えていきましょう。
物理マシンの電源投入時には、次のような手順で初期化処理が実行されます。

1. 物理的な初期化

コンピュータの電源が投入されると、CPU は決められたアドレスから命令の実行を開始します。
このアドレスには ROM がメモリマップされており、電源投入時に CPU は ROM 上のファームウェア[*1]の初期化ルーチンから実行を開始します。
電源を上げたときのレジスタ値やメモリ内容、周辺デバイスの状態は初期化されていないため不定ですが、ファームウェアが最低限の初期化を行います。

2. ファームウェアのロード

初期化が終わると、ファームウェアはブートローダまたは OS をロードします。

3. OS のロード

ブートローダを実行した場合、ブートローダが OS をロードします。
この初期化手順は PC に限ったものではなく、スマートフォンのようなデバイスやマイコンのようなより単純なデバイスでもおおむね同じです。

[*1] PC では通常 BIOS または UEFI がファームウェアとして用いられています。
同じ x86 CPU が搭載されていてもファームウェアの仕様が異なれば別々のブートローダを用意しなければならなくなるため、BIOS も UEFI もベンダ間で差異が生じないよう仕様が定められています。

一般的に、CPU のレジスタやメモリ上のデータは電源を遮断すると状態を維持できないため、電源投入ごとに上述のような初期化処理を行う必要があります。

しかしながら、仮想マシンはそのような物理デバイスの特性に起因する実装上の制約がなく、ホスト側でどのようにでも初期状態を設定してから実行することができます。

ゲスト OS のダイレクトブート

いくつかのハイパーバイザでは、上述のような仮想マシンの特性を用いてブートローダを実行することなく、直接 OS をブートするモードを実装しています。

たとえば QEMU（KVM）では、以下のようなコマンドでホスト OS 上に置かれた Linux カーネルをロード・ブートできます。

```
$ qemu -kernel vmlinuz -append "ro root=LABEL=/" -initrd initrd.img
```

BHyVe では、今のところ BIOS が実装されていません。

このため、ブートセクタから起動する方式をサポートできず、ホスト OS 上で動作するプログラムを用いてゲスト OS をロードしています。

FreeBSD ゲストのロードでは、FreeBSD ブートローダをホスト OS のユーザランドへ移植しレジスタアクセスやメモリアクセスをゲストマシンに対するアクセスで置き換えることによって、通常のブートローダと同じインターフェースを持つ OS ローダを実装しています。

Linux ゲストや OpenBSD ゲストのロードでは、同様の手法で GRUB2 をホスト OS へ移植することにより[*2]、通常の GRUB2 と同じインターフェースを持つ OS ローダを実装しています。

このとき、ゲスト OS ローダはゲスト OS をメモリ上にロードし、レジスタの初期値を設定し、CPU のモードをプロテクトモードに切り替えてから仮想マシンを始動します。

これは従来ブートローダが行っていたことであり、この方法では BIOS へ依存するブートローダをホスト OS 側で動作するプログラムで置き換えています。

ただし、ゲスト OS が実行中に BIOS を呼ぼうとすると BIOS コールハンドラが存在しないため、エラーが発生してしまって OS が正常に動作しません。

これに対処するには部分的にせよ BIOS サポートを導入するしかなく、そのため BHyVe では動作しないゲスト OS が存在します[*3]。

[*2] https://github.com/grehan-freebsd/grub2-bhyve
[*3] FreeBSD/i386 など。

具体的な実装方法

BHyVe では、VM インスタンスの作成は sysctl を用いて/dev/vmm/を作成することにより、VM インスタンスの操作は/dev/vmm/に対して ioctl を発行することにより、VM インスタンスのメモリ空間へのアクセスは/dev/vmm/に対して mmap することにより行えます。

ただし、これらの処理を書きやすくするために vmmapi というライブラリが用意されているので、通常こちらを利用します。

以降に行いたい操作の種類によってどのような実装を行えばよいかを示します。

メモリへの書き込み

ページテーブルの作成など、ゲストメモリ空間への書き込みは vmmapi を用いてゲストメモリ空間を mmap することによって行います（リスト 1)。

まず、vm_setup_memory() でゲストマシンのメモリサイズを指定し、/dev/vmm/を mmap してプロセスのメモリ空間へゲストメモリ空間をマップします。

次に、vm_map_gpa() へオフセットを渡すことによりポインタを取得できます。

■リスト１，メモリへの書き込み

```
vm_create(VM_NAME);
ctx = vm_open(VM_NAME);
vm_setup_memory(ctx, VM_MEM_SIZE, VM_MMAP_ALL);
ptr = vm_map_gpa(ctx, addr, len);
```

レジスタへの書き込み

ゲストマシンのレジスタへの書き込みは vmmapi を通じて ioctl を発行することによって行います（リスト 2)。
セグメントレジスタとそれ以外のレジスタでは VMCS のフォーマットが異なるため、API が異なります。
セグメントレジスタでは vm_set_desc() で base、limit、access を設定し、vm_set_register() で selector を設定します。
その他のレジスタでは、vm_set_register() で値を設定します。

■リスト２，レジスタへの書き込み

```
tx = vm_open(vm_name)
vm_set_register(ctx, cpuno, VM_REG_GUEST_RFLAGS, val)

vm_set_desc(ctx, cpuno, VM_REG_GUEST_CS, base, limit, access)
vm_set_register(ctx, cpuno, VM_REG_GUEST_CS, selector)
```

コンソールへの文字列表示

コンソールへの文字列表示に関しては、printf() や puts() を用いればよいため、vmmapi は使用しません。

ディスクの読み込み

ディスクイメージは通常のファイルであるため、通常のファイル API を使用できます。

このため、vmmapi は使用しません。

簡易ローダの実装例

BHyVe における OS ローダの実装方法を例示するため、シリアルコンソールにアスタリスクを表示し続けるだけの簡単なプログラムをゲストマシンへロードする簡易ローダを実装しました[*4]。

このローダを使うと、ゲストマシンが起動した最初の瞬間から、64bit モード・ページングが有効な状態で実行されます。
リスト 3 にソースコードを示します。

■リスト３，簡易ローダのソースコード

```
#include <sys/cdefs.h>
__FBSDID("$FreeBSD$");
#include <sys/param.h>
#include <sys/stat.h>
#include <machine/specialreg.h>
#include <machine/vmm.h>
#include <x86/segments.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

[*4] https://github.com/syuu1228/bhyve-embedded-guest

```
#include <unistd.h>
#include <vmmapi.h>

#define VM_NAME         "test1"
#define VM_MEM_SIZE         128 * 1024 * 1024UL

#define MSR_EFER            0xc0000080
#define CR4_PAE             0x00000020
#define CR4_PSE             0x00000010
#define CR0_PG              0x80000000
#define CR0_PE          0x00000001
#define CR0_NE          0x00000020

#define PG_V            0x001
#define PG_RW           0x002
#define PG_U            0x004
#define PG_PS           0x080

#define ADDR_PT4        0x2000
#define ADDR_PT3        0x3000
#define ADDR_PT2        0x4000
#define ADDR_GDT        0x5000
#define ADDR_STACK      0x6000
#define ADDR_ENTRY      0x10000

#define DESC_UNUSABLE       0x00010000

#define GUEST_NULL_SEL      0
#define GUEST_CODE_SEL      1
#define GUEST_DATA_SEL      2
#define GUEST_GDTR_LIMIT    (3 * 8 - 1)

int
main(int argc, char** argv)
{
    struct vmctx *ctx;
    uint64_t *gdt, *pt4, *pt3, *pt2;
    int i;
    unsigned char *entry;
    unsigned char program[] = {・・・・(1)
```

```
    0xb0, 0x2a,             /* mov    $0x2a,%al */
    0xba, 0xf8, 0x03, 0x00, 0x00,   /* mov    $0x3f8,%edx */
    0xee,               /* out    %al,(%dx) */
    0xeb, 0xfd          /* jmp    7 <loop> */
};

vm_create(VM_NAME);・・・・(2)
ctx = vm_open(VM_NAME);
vm_setup_memory(ctx, VM_MEM_SIZE, VM_MMAP_ALL);・・・・(3)

pt4 = vm_map_gpa(ctx, ADDR_PT4, sizeof(uint64_t) * 512);・・・・(4)
pt3 = vm_map_gpa(ctx, ADDR_PT3, sizeof(uint64_t) * 512);
pt2 = vm_map_gpa(ctx, ADDR_PT2, sizeof(uint64_t) * 512);
gdt = vm_map_gpa(ctx, ADDR_GDT, sizeof(uint64_t) * 3);
entry = vm_map_gpa(ctx, ADDR_ENTRY, sizeof(program));
bzero(pt4, PAGE_SIZE);
bzero(pt3, PAGE_SIZE);
bzero(pt2, PAGE_SIZE);
for (i = 0; i < 512; i++) {・・・・(5)
    pt4[i] = (uint64_t)ADDR_PT3;
    pt4[i] |= PG_V | PG_RW | PG_U;
    pt3[i] = (uint64_t)ADDR_PT2;
    pt3[i] |= PG_V | PG_RW | PG_U;
    pt2[i] = i * (2 * 1024 * 1024);
    pt2[i] |= PG_V | PG_RW | PG_PS | PG_U;
}

gdt[GUEST_NULL_SEL] = 0;・・・・(6)
gdt[GUEST_CODE_SEL] = 0x0020980000000000;
gdt[GUEST_DATA_SEL] = 0x0000900000000000;

memcpy(entry, program, sizeof(program));・・・・(7)

vm_set_desc(ctx, 0, VM_REG_GUEST_CS, 0, 0, 0x0000209B);・・・・(8)
vm_set_desc(ctx, 0, VM_REG_GUEST_DS, 0, 0, 0x00000093);
vm_set_desc(ctx, 0, VM_REG_GUEST_ES, 0, 0, 0x00000093);
vm_set_desc(ctx, 0, VM_REG_GUEST_FS, 0, 0, 0x00000093);
vm_set_desc(ctx, 0, VM_REG_GUEST_GS, 0, 0, 0x00000093);
vm_set_desc(ctx, 0, VM_REG_GUEST_SS, 0, 0, 0x00000093);
vm_set_desc(ctx, 0, VM_REG_GUEST_TR, 0, 0, 0x0000008b);
```

```
    vm_set_desc(ctx, 0, VM_REG_GUEST_LDTR, 0, 0, DESC_UNUSABLE);
    vm_set_desc(ctx, 0, VM_REG_GUEST_GDTR, ADDR_GDT, GUEST_GDTR_LIMIT, 0);・・・・(9)

    vm_set_register(ctx, 0, VM_REG_GUEST_CS, GSEL(GUEST_CODE_SEL, SEL_KPL));
    vm_set_register(ctx, 0, VM_REG_GUEST_DS, GSEL(GUEST_DATA_SEL, SEL_KPL));
    vm_set_register(ctx, 0, VM_REG_GUEST_ES, GSEL(GUEST_DATA_SEL, SEL_KPL));
    vm_set_register(ctx, 0, VM_REG_GUEST_FS, GSEL(GUEST_DATA_SEL, SEL_KPL));
    vm_set_register(ctx, 0, VM_REG_GUEST_GS, GSEL(GUEST_DATA_SEL, SEL_KPL));
    vm_set_register(ctx, 0, VM_REG_GUEST_SS, GSEL(GUEST_DATA_SEL, SEL_KPL));
    vm_set_register(ctx, 0, VM_REG_GUEST_TR, 0);
    vm_set_register(ctx, 0, VM_REG_GUEST_LDTR, 0);

    vm_set_register(ctx, 0, VM_REG_GUEST_CR0, CR0_PG | CR0_PE | CR0_NE);・・・・(10)
    vm_set_register(ctx, 0, VM_REG_GUEST_CR3, ADDR_PT4);・・・・(11)
    vm_set_register(ctx, 0, VM_REG_GUEST_CR4, CR4_PAE | CR4_VMXE);・・・・(12)
    vm_set_register(ctx, 0, VM_REG_GUEST_EFER, EFER_LMA | EFER_LME);・・・・(13)
    vm_set_register(ctx, 0, VM_REG_GUEST_RFLAGS, 0x2);
    vm_set_register(ctx, 0, VM_REG_GUEST_RSP, ADDR_STACK);・・・・(14)
    vm_set_register(ctx, 0, VM_REG_GUEST_RIP, ADDR_ENTRY);・・・・(15)
    return (0);
}
```

(1) ゲストマシンで実行するプログラムです。

(2) AL レジスタに*（ASCII コードで 0x2a）をロード、DX レジスタにシリアルポートのデータレジスタのアドレスをロード、outb で AL レジスタの内容を DX レジスタで指定したポートへ書き込み、を繰り返し行っています。

(3) ここではプログラムのロード処理を省略するため、配列上にプログラムの HEX ダンプを持ってきています。

(4) vm_create() で vmm.ko へ sysctl を発行し、VM インスタンスを作成します。作成したインスタンスは/dev/vmm/で表され、ioctl で制御できます。

(5) vm_setup_memory() でゲストマシンのメモリサイズを指定し、/dev/vmm/を mmap してプロセスのメモリ空間へゲストメモリ空間をマップしています。

(6) vm_map_gpa() でゲストメモリ空間へのポインタを取得しています。ゲストメモリ空間上のアドレスは引数で指定しています（ここでは ADDR_PT4 = 2000h)。 vm_map_gpa() では渡されたアドレスをオフセットとしてポインタを計算します。

(7) ここでは、ページテーブル（pt4,pt3,pt2)、GDT、プログラム領域（entry）のアドレスを指定してそれぞれのポインタを取得しています。

(8) ゲストメモリ空間上のページテーブルを初期化しています。

(9) ゲストメモリ空間上の初期化しています。

(10) ゲストメモリ空間へ (1) で記述したプログラムをコピーしています。
(11) 各セグメントレジスタを初期化してます。
(12) GDTR に作成した GDT のアドレスをセットしてます。
(13) CR0 レジスタにプロテクテッドモード有効、ページング有効などのビットを設定しています。
(14) CR3 レジスタに作成したページテーブルのアドレスを設定しています。
(15) CR4 レジスタに PAE 有効化などのビットを設定しています。
(16) EFER レジスタに 64bit 有効化などのビットを設定しています。
(17) RSP レジスタにスタックアドレスの初期値を設定しています。
(18) RIP レジスタにロードしたプログラムのアドレスを設定しています。

実行イメージ

ビルドしたローダは以下のようなコマンドで実行出来ます（リスト４）。

■リスト４，画面出力

```
sudo ./load
sudo bhyve -c 1 -m 128 -s 0:0,hostbridge -S 31,uart,stdio test1
***********************************************************************************
***********************************************************************************
***********************************************************************************
***********************************************************************************
***********************************************************************************
***********************************************************************************
***********************************************************************************
***********************************************************************************
***********************************************************************************
*****************************************************************
```

まとめ

仮想マシンには物理マシンのような制約がないため、ユーザが自由に仮想マシンの状態をプログラムしてから実行できること、その特性を利用して BIOS を使わないブート方式を実装出来ることを示しました。次回は、仮想マシンのネットワークデバイスについて解説します。

ライセンス