第 7 回 Intel VT-x を用いたハイパーバイザの実装その 2「/usr/sbin/bhyve による仮想 CPU の実行処理」

はじめに

前回は、BHyVe の概要や使い方について紹介してきました。いよいよソースコードの解説に入っていきます。今回は、とくに/usr/sbin/bhyve の初期化と VM インスタンスの実行機能の実装について解説をしていきます。

解説対象のバージョン

BHyVe は、現在開発の初期段階です。日々開発が進められており、さまざまな機能が追加されていますが、リリースバージョンが存在していません。

そこで、本連載では執筆時点での最新リビジョンである r245673 を用いて解説を行います。r245673 のインストールディスクは、以下のアドレスからダウンロードできます。

ftp://ftp.freebsd.org/pub/FreeBSD/snapshots/amd64/amd64/ISO-IMAGES/10.0/FreeBSD-10.0-CURRENT-amd64-20130119-r245673-release.iso

r245673 のソースコードは次のコマンドで取得できます。

svn co -r245673 svn://svn.freebsd.org/base/head src

/usr/sbin/bhyve と/usr/sbin/bhyveload の役割分担

まずはじめに、前回簡単に紹介した/usr/sbin/bhyve と/usr/sbin/bhyveload の役割分担について解説します。ゲスト OS を起動するには、/usr/sbin/bhyve を実行する前に/usr/sbin/bhyveload を実行してゲストカーネルのロードを行います。/usr/sbin/bhyveload を実行すると、/dev/vmm/へ VM インスタンスのデバイスファイルが作成されます。このデバイスファイルを通じてゲストメモリ領域にゲストカーネルがロードされ、ゲストマシンのレジスタ初期値や GDT・IDT などのデスクリプタの設定が行われます。

これに対して/usr/sbin/bhyve の仕事は、初期化済みの VM インスタンスのデバイスファイルをオープンし、VM インスタンスの実行を開始するすることになります。また、ゲスト OS が使用する各種デバイスをエミュレーションするのも/usr/sbin/bhyve の役割となります。

なお、CPU のモードを VMX non root mode に切り替えるなどのハードウェアに近い処理は、特権モードで実行する必要があるためカーネルモジュール (vmm.ko) の仕事になります。

/usr/sbin/bhyve が起動されたら、まずはじめにゲストマシンが使用する各種デバイス (HDD/NIC/コンソール) のエミュレータを使用可能な状態に初期化する必要があります。

初期化が終わると、/usr/sbin/bhyve は仮想 CPU の数だけスレッドを起動し、/dev/vmm/${name}に対して VM_RUN ioctl を発行します (図 1)。 vmm.ko は ioctl を受けて CPU を VT-x non root mode へ切り替えゲスト OS を実行します (VMEntry)。

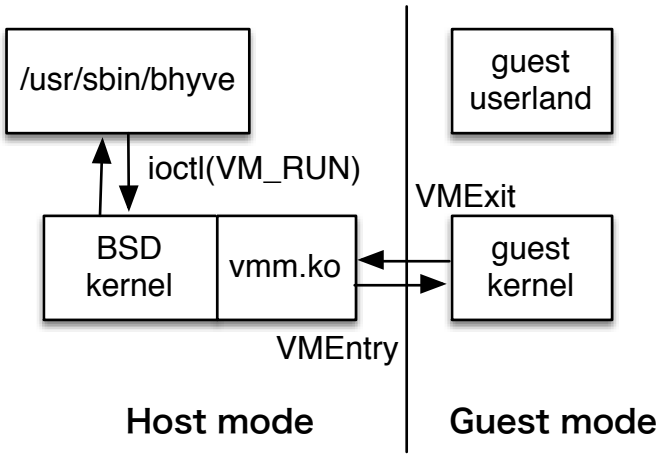

図 1　VM_RUN ioctl による仮想 CPU の実行イメージ

VMX non root mode でハイパーバイザの介入が必要な何らかのイベントが発生すると制御が vmm.ko へ戻され、イベントがトラップされます (VMExit)。イベントの種類が/usr/sbin/bhyve でハンドルされる必要のあるものだった場合、ioctl はリターンされ、制御が/usr/sbin/bhyve へ移ります。イベントの種類が/usr/sbin/bhyve でハンドルされる必要のないものだった場合、ioctl はリターンされないままゲスト CPU の実行が再開されます。

それでは、実際に BHyVe のソースコードを読んでいきましょう。リスト 1 とリスト 2 にソースコードを示します。

リスト 1 usr.sbin/bhyve/bhyverun.c

```
static void *
fbsdrun_start_thread(void *param)
{
    char tname[MAXCOMLEN + 1];
    struct mt_vmm_info *mtp;
    int vcpu;

    mtp = param;
    vcpu = mtp->mt_vcpu;

    snprintf(tname, sizeof(tname), "%s vcpu %d", vmname, vcpu);
    pthread_set_name_np(mtp->mt_thr, tname);
```

```
    vm_loop(mtp->mt_ctx, vcpu, vmexit[vcpu].rip);                    (17)

    /* not reached */
    exit(1);
    return (NULL);
}

void
fbsdrun_addcpu(struct vmctx *ctx, int vcpu, uint64_t rip)
{
    int error;

    if (cpumask & (1 << vcpu)) {
        fprintf(stderr, "addcpu: attempting to add existing cpu %d\n",
            vcpu);
        exit(1);
    }

    cpumask |= 1 << vcpu;
    foundcpus++;

    /*
     * Set up the vmexit struct to allow execution to start
     * at the given RIP
     */
    vmexit[vcpu].rip = rip;
    vmexit[vcpu].inst_length = 0;

    if (vcpu == BSP || !guest_vcpu_mux){
        mt_vmm_info[vcpu].mt_ctx = ctx;
        mt_vmm_info[vcpu].mt_vcpu = vcpu;

        error = pthread_create(&mt_vmm_info[vcpu].mt_thr, NULL,
                fbsdrun_start_thread, &mt_vmm_info[vcpu]);           (16)
        assert(error == 0);
    }
}
......(省略)......
static vmexit_handler_t handler[VM_EXITCODE_MAX] = {                 (22)
```

```
    [VM_EXITCODE_INOUT]  = vmexit_inout,
    [VM_EXITCODE_VMX]    = vmexit_vmx,
    [VM_EXITCODE_BOGUS]  = vmexit_bogus,
    [VM_EXITCODE_RDMSR]  = vmexit_rdmsr,
    [VM_EXITCODE_WRMSR]  = vmexit_wrmsr,
    [VM_EXITCODE_MTRAP]  = vmexit_mtrap,
    [VM_EXITCODE_PAGING] = vmexit_paging,
    [VM_EXITCODE_SPINUP_AP] = vmexit_spinup_ap,
};

static void
vm_loop(struct vmctx *ctx, int vcpu, uint64_t rip)
{
    int error, rc, prevcpu;

    if (guest_vcpu_mux)
        setup_timeslice();

    if (pincpu >= 0) {
        error = vm_set_pinning(ctx, vcpu, pincpu + vcpu);
        assert(error == 0);
    }

    while (1) {
        error = vm_run(ctx, vcpu, rip, &vmexit[vcpu]);              (18)
        if (error != 0) {
            /*
             * It is possible that 'vmmctl' or some other process
             * has transitioned the vcpu to CANNOT_RUN state right
             * before we tried to transition it to RUNNING.
             *
             * This is expected to be temporary so just retry.
             */
            if (errno == EBUSY)
                continue;
            else
                break;
        }

        prevcpu = vcpu;
```

```
                rc = (*handler[vmexit[vcpu].exitcode])(ctx, &vmexit[vcpu],
                                                       &vcpu);      (21)
        switch (rc) {                                               (23)
                case VMEXIT_SWITCH:
            assert(guest_vcpu_mux);
            if (vcpu == -1) {
                stats.cpu_switch_rotate++;
                vcpu = fbsdrun_get_next_cpu(prevcpu);
            } else {
                stats.cpu_switch_direct++;
            }
            /* fall through */
        case VMEXIT_CONTINUE:
                        rip = vmexit[vcpu].rip + vmexit[vcpu].inst_length;
            break;
        case VMEXIT_RESTART:
                        rip = vmexit[vcpu].rip;
            break;
        case VMEXIT_RESET:
            exit(0);
        default:
            exit(1);
        }
    }
    fprintf(stderr, "vm_run error %d, errno %d\n", error, errno);
}
......(省略)......
int
main(int argc, char *argv[])
{
......(省略)......

    while ((c = getopt(argc, argv, "abehABHIPxp:g:c:z:s:S:n:m:M:")) != -1) {
......(省略)......
    vmname = argv[0];

    ctx = vm_open(vmname);                                          (1)
    if (ctx == NULL) {
        perror("vm_open");
        exit(1);
```

```
    }
.....(省略).....
    if (lomem_sz != 0) {
        lomem_addr = vm_map_memory(ctx, 0, lomem_sz);              (5)
        if (lomem_addr == (char *) MAP_FAILED) {
            lomem_sz = 0;
        } else if (himem_sz != 0) {
            himem_addr = vm_map_memory(ctx, 4*GB, himem_sz);       (7)
            if (himem_addr == (char *) MAP_FAILED) {
                lomem_sz = 0;
                himem_sz = 0;
            }
        }
    }

    init_inout();                                                  (8)
    init_pci(ctx);                                                 (9)
    if (ioapic)
        ioapic_init(0);                                            (10)
.....(省略).....
    error = vm_get_register(ctx, BSP, VM_REG_GUEST_RIP, &rip);     (11)
    assert(error == 0);
.....(省略).....
    /*
     * build the guest tables, MP etc.
     */
    mptable_build(ctx, guest_ncpus, ioapic);                       (13)

    if (acpi) {
        error = acpi_build(ctx, guest_ncpus, ioapic);              (14)
        assert(error == 0);
    }

    /*
     * Add CPU 0
     */
    fbsdrun_addcpu(ctx, BSP, rip);                                 (15)
```

(1) getopt で処理されなかった一番端の引数が VM 名となります。

libvmmapi を用いて VM 名に対応するデバイスファイルをオープンします。

(5) 4GB 未満のゲストメモリ空間を lomem_addr へマッピングします。

(7) 4GB 以上のゲストメモリ空間を himem_addr へマッピングします。

(8) IO ポートエミュレーションの初期化を行います。

(9) PCI デバイスエミュレーションの初期化を行います。

(10)IO APIC エミュレーションの初期化を行います。

(11)/usr/sbin/bhyveload で設定された RIP レジスタの値を取得します。

(13)MP テーブルを生成します。2 つ以上の CPU 数で起動する時に必要です。

(14)ACPI テーブルを生成します。無変更な FreeBSD カーネルを起動するのに必要です。

(15)cpu0(BSP) のスレッドを起動します。実行開始アドレスとして RIP を渡します。

(16)pthread_create(fbsdrun_start_thread) します。
ここまでがハイパーバイザの初期化の処理になります。
ここからはゲストマシンの実行処理に移っていきます。

(17)vm_loop() で仮想 CPU を実行します。

(18)while ループの中で vm_run() を呼びます。

(21)ioctl リターンの理由 (vmexit.exitcode) に対応したイベントハンドラ (handler[]) を呼び出します。

(22)/usr/sbin/bhyve に定義されている handler[] です。
IO ポートへのアクセス、MSR レジスタの読み書き、メモリマップド IO、
セカンダリ CPU の起動シグナルなどがハンドラとして定義されています。

(23)handler[] からの返り値により、CPU を現在の rip から再開するか・次の命令から再開するか・
ハイパーバイザの実行を中止するか、などの処理を行なっています。
実行が再開される場合は、while ループにより再び vm_loop() の実行に戻ります。

リスト 2 lib/libvmmapi/vmmapi.c

```
.....(省略).....
static int
vm_device_open(const char *name)
{
        int fd, len;
        char *vmfile;

    len = strlen("/dev/vmm/") + strlen(name) + 1;
    vmfile = malloc(len);
    assert(vmfile != NULL);
    snprintf(vmfile, len, "/dev/vmm/%s", name);

        /* Open the device file */
        fd = open(vmfile, O_RDWR, 0);                                    (4)

    free(vmfile);
        return (fd);
}
.....(省略).....
vm_open(const char *name)
{
    struct vmctx *vm;

    vm = malloc(sizeof(struct vmctx) + strlen(name) + 1);                (2)
    assert(vm != NULL);

    vm->fd = -1;
    vm->name = (char *)(vm + 1);
    strcpy(vm->name, name);

    if ((vm->fd = vm_device_open(vm->name)) < 0)                         (3)
        goto err;

    return (vm);
err:
    vm_destroy(vm);
    return (NULL);
```

```
}
.....(省略).....
char *
vm_map_memory(struct vmctx *ctx, vm_paddr_t gpa, size_t len)
{

    /* Map 'len' bytes of memory at guest physical address 'gpa' */
    return ((char *)mmap(NULL, len, PROT_READ | PROT_WRITE, MAP_SHARED,
             ctx->fd, gpa));                                          (6)
}
.....(省略).....
int
vm_get_register(struct vmctx *ctx, int vcpu, int reg, uint64_t *ret_val)
{
    int error;
    struct vm_register vmreg;

    bzero(&vmreg, sizeof(vmreg));
    vmreg.cpuid = vcpu;
    vmreg.regnum = reg;

    error = ioctl(ctx->fd, VM_GET_REGISTER, &vmreg);                (12)
    *ret_val = vmreg.regval;
    return (error);
}
.....(省略).....
int
vm_run(struct vmctx *ctx, int vcpu, uint64_t rip, struct vm_exit *vmexit)
{
    int error;
    struct vm_run vmrun;

    bzero(&vmrun, sizeof(vmrun));
    vmrun.cpuid = vcpu;
    vmrun.rip = rip;

    error = ioctl(ctx->fd, VM_RUN, &vmrun);                         (19)
    bcopy(&vmrun.vm_exit, vmexit, sizeof(struct vm_exit));          (20)
    return (error);
}
```

(2) vm_open() は struct vmctx をアロケートし、

(3) vmctx->fd にファイルディスクリプタを保存します。

(4) vm_device_open() は/dev/vmm/${name}を読み書き用でオープンします。

(6) /dev/vmm/${name}を mmap() します。
vmm.ko からゲストメモリ空間のアドレスが渡されます。

(12)/dev/vmm/${name}へ VM_GET_REGISTER ioctl を行います。
vmm.ko からレジスタの値が渡されます。

(19)/dev/vmm/${name}へ VM_RUN ioctl を行います。
vmm.ko はこのスレッドが実行されている CPU を VT-x non root mode へ移行し、
vmrun.rip で指定されたアドレスから実行を開始します。

(20)VMX non root mode でハイパーバイザの介入が必要な何らかのイベントが発生すると
vmm.ko の中でトラップされ、/usr/sbin/bhyve でイベントを処理する必要がある場合は
ioctl がリターンされます。
リターンされた理由を vmm.ko から struct vmexit で受け取ります。

まとめ

/usr/sbin/bhyve の初期化と VM インスタンスの実行機能の実装について、ソースコードを解説しました。極めてシンプルなループにより VM インスタンスの実行処理が実現されていることを確認して頂けたかと思います。

今回はユーザランド側の実装のみ解説しましたが、次回はこれに対応するカーネル側の実装を見て行きたいと思います。

ライセンス