ハイパーバイザの作り方〜ちゃんと理解する仮想化技術〜 第８回 Intel VT-x を用いたハイパーバイザの実装その３「vmm.ko による VMEntry」

はじめに

前回は、/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しました。今回は vmm.ko が VM_RUN ioctl を受け取ってから VMEntry するまでの処理を解説します。

解説対象のソースコードについて

本連載では、FreeBSD-CURRENT に実装されている BHyVe のソースコードを解説しています。このソースコードは、FreeBSD の Subversion リポジトリから取得できます。リビジョンは r245673 を用いています。

お手持ちの PC に Subversion をインストールし、次のようなコマンドでソースコードを取得してください。

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

/usr/sbin/bhyve による仮想 CPU の実行処理の復習

/usr/sbin/bhyve は仮想 CPU の数だけスレッドを起動し、それぞれのスレッドが/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 VMX non root mode へ切り替えゲスト OS を実行します (VMEntry)。

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。今回の記事では、vmm.ko に VM_RUN ioctl が届いてから VMX non root mode へ VMEntry するまでを見ていきます。

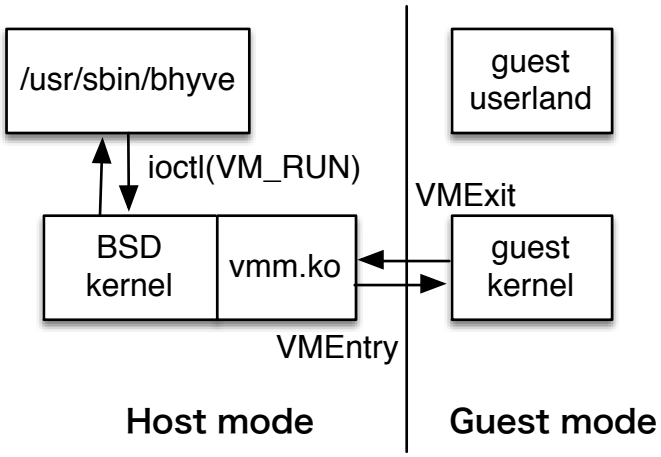

図 1　VM_RUN ioctl による仮想 CPU の実行イメージ

vmm.ko が VM_RUN ioctl を受け取ってから VMEntry するまで

vmm.ko が VM_RUN ioctl を受け取ってから VMEntry するまでの処理について、順を追って見ていきます。リスト 1、リスト 2、リスト 3、リスト 4 にソースコードを示します。白丸の数字と黒丸の数字がありますが、ここでは白丸の数字を追って見ていきます[*1]。

VMExit 時の再開アドレス

(19) で CPU は VMX non-root mode へ切り替わりゲスト OS が実行されますが、ここから VMExit した時に CPU はどこからホスト OS の実行を再開するのでしょうか。直感的には vmlaunch の次の命令ではないかと思いますが、そうではありません。VT-x では、VMEntry 時に VMCS の GUEST_RIP から VMX non-root mode の実行を開始し、VMExit 時に VMCS の HOST_RIP から VMX root mode の実行を開始することになっています。GUEST_RIP は VMExit 時に保存されますが、HOST_RIP は VMEntry 時に保存されません。

このため、VMCS の初期化時に指定された HOST_RIP が常に VMExit 時の再開アドレスとなります。では、VMCS の HOST_RIP がどのように初期化されているか、順を追って見ていきます。リスト 1、リスト 2、リスト 3、リスト 4 にソースコードを示します。今度は黒丸の数字を追って見ていきます。

リスト 1 の解説

vmm_dev.c は、sysctl による/dev/vmm/${name}の作成・削除と/dev/vmm/${name}に対する open(), close(), read(), write(), mmap(), ioctl() のハンドラを定義しています。ここでは/dev/vmm/${name}の作

[*1] データ化における注釈:　機種依存文字を避けるため黒丸囲み数字を []、白丸囲み数字を () に変更しています。

成と VM_RUN ioctl についてのみ見ていきます。

■リスト 1

```
......(省略)......
137: static int
138: vmmdev_ioctl(struct cdev *cdev, u_long cmd, caddr_t data, int fflag,
139:          struct thread *td)                                    (1)
140: {
......(省略)......
234:    switch(cmd) {
235:    case VM_RUN:                                               (2)
236:        vmrun = (struct vm_run *)data;
237:        error = vm_run(sc->vm, vmrun);                         (3)
238:        break;
......(省略)......
371:    }
......(省略)......
382: }
......(省略)......
471: static int
472: sysctl_vmm_create(SYSCTL_HANDLER_ARGS)                        [1]
473: {
......(省略)......
490:    vm = vm_create(buf);                                       [2]
517: }
```

- (1) vmmdev_ioctl は/dev/vmm/${name}の ioctl ハンドラ。
- (2) /usr/sbin/bhyve から送られてきた VM_RUN ioctl を受け取る。
- (3) vm_run() を呼ぶ。
- [1] hw.vmm.create sysctl により/dev/vmm/${name}を作成し VM インスタンスを初期化する時に呼び出される。
- [2] vm_create を呼び出す。

リスト 2 **の解説**

vmm.c は、IntelVT-x と AMD-V の 2 つの異なるハードウェア仮想化支援機能のラッパー関数を提供しています (このリビジョンではラッパーのみが実装されており、AMD-V の実装は行われていません)。Intel/AMD 両アーキテクチャ向けの各関数は vmm_ops 構造体で抽象化され、207〜210 行目のコード CPU を判定して

どちらのアーキテクチャの関数群を使用するかを決定しています。

■リスト 2

```
......(省略)......
 119:  static struct vmm_ops *ops;
......(省略)......
 123:  #define  VMINIT(vm)  (ops != NULL ? (*ops->vminit)(vm): NULL)    [4]
 124:  #define  VMRUN(vmi, vcpu, rip) \
 125:   (ops != NULL ? (*ops->vmrun)(vmi, vcpu, rip) : ENXIO)          (5)
......(省略)......
 195:  static int
 196:  vmm_init(void)
 197:  {
......(省略)......
 207:   if (vmm_is_intel())
 208:       ops = &vmm_ops_intel;
 209:   else if (vmm_is_amd())
 210:       ops = &vmm_ops_amd;
......(省略)......
 216:   return (VMM_INIT());
 217:  }
......(省略)......
 261:  struct vm *
 262:  vm_create(const char *name)
 263:  {
......(省略)......
 275:   vm->cookie = VMINIT(vm);                                        [3]
......(省略)......
 287:  }
......(省略)......
 672:  int
 673:  vm_run(struct vm *vm, struct vm_run *vmrun)
 674:  {
......(省略)......
 681:   vcpuid = vmrun->cpuid;
......(省略)......
 688:   rip = vmrun->rip;
......(省略)......
 701:   error = VMRUN(vm->cookie, vcpuid, rip);                        (4)
```

```
......(省略).....
 757:  }
```

- [4] VMINIT() マクロは vmm_ops 構造体上の関数ポインタ ops->vminit を呼び出す。
- (5) VMRUN() マクロは vmm_ops 構造体上の関数ポインタ ops->vmrun を呼び出す。
- [3] vm_create() は VMINIT() マクロを呼び出す。
- (4) vm_run() は VMRUN() マクロを呼び出す。

リスト 3 の解説

intel/ディレクトリには Intel VT-x に依存したコード群が置かれています。vmx.c はその中心的なコードで、vmm.c で登場した vmm_ops_intel もここで定義されています。

■リスト 3

```
......(省略).....
 666:  static void *
 667:  vmx_vminit(struct vm *vm)                                    [6]
 668:  {
......(省略).....
 725:   for (i = 0; i < VM_MAXCPU; i++) {                           [7]
......(省略).....
 735:       error = vmcs_set_defaults(&vmx->vmcs[i],               [8]
 736:                       (u_long)vmx_longjmp,
 737:                       (u_long)&vmx->ctx[i],
 738:                       vtophys(vmx->pml4ept),
 739:                       pinbased_ctls,
 740:                       procbased_ctls,
 741:                       procbased_ctls2,
 742:                       exit_ctls, entry_ctls,
 743:                       vtophys(vmx->msr_bitmap),
 744:                       vpid);
......(省略).....
 770:   }
 771:
 772:   return (vmx);
 773:  }
......(省略).....
1354:  static int
```

```
1355:  vmx_run(void *arg, int vcpu, register_t rip)                (7)
1356:  {
......(省略)......
1366:   vmxctx = &vmx->ctx[vcpu];
......(省略)......
1375:   VMPTRLD(vmcs);
......(省略)......
1385:   if ((error = vmwrite(VMCS_HOST_CR3, rcr3())) != 0)
1386:       panic("vmx_run: error %d writing to VMCS_HOST_CR3", error);
1387:
1388:   if ((error = vmwrite(VMCS_GUEST_RIP, rip)) != 0)           (8)
1389:       panic("vmx_run: error %d writing to VMCS_GUEST_RIP", error);
1390:
1391:   if ((error = vmx_set_pcpu_defaults(vmx, vcpu)) != 0)
1392:       panic("vmx_run: error %d setting up pcpu defaults", error);
1393:
1394:   do {                                                        (9)
1395:       lapic_timer_tick(vmx->vm, vcpu);
1396:       vmx_inject_interrupts(vmx, vcpu);
1397:       vmx_run_trace(vmx, vcpu);
1398:       rc = vmx_setjmp(vmxctx);                               (10)
......(省略)......
1402:       switch (rc) {
1403:       case VMX_RETURN_DIRECT:                                (12)
1404:           if (vmxctx->launched == 0) {                       (13)
1405:               vmxctx->launched = 1;
1406:               vmx_launch(vmxctx);                            (14)
1407:           } else
1408:               vmx_resume(vmxctx);
1409:           panic("vmx_launch/resume should not return");
1410:           break;
......(省略)......
1458:   } while (handled);
......(省略)......
1480:   VMCLEAR(vmcs);
1481:   return (0);
......(省略)......
1490:  }
......(省略)......
1830:  struct vmm_ops vmm_ops_intel = {
```

```
1831:   vmx_init,
1832:   vmx_cleanup,
1833:   vmx_vminit,                                                    [5]
1834:   vmx_run,                                                       (6)
1835:   vmx_vmcleanup,
1836:   ept_vmmmap_set,
1837:   ept_vmmmap_get,
1838:   vmx_getreg,
1839:   vmx_setreg,
1840:   vmx_getdesc,
1841:   vmx_setdesc,
1842:   vmx_inject,
1843:   vmx_getcap,
1844:   vmx_setcap
1845:  };
```

- [6] 結果として、 vm_create は vmx_vminit が呼び出される。
- [7] vmx_vminit では仮想 CPU ごとの VMCS の初期化処理が行われる。
  その過程で幾つかの関数が呼び出されるが、ここでは vmcs_set_defaults にのみ着目する。
- [8] vmcs_set_defaults の第二引数に vmx_longjmp を指定して呼んでいる。
- (7) 結果として、VM_RUN ioctl は vmx_run 内で実際に処理される。
- (8) VMCS にゲストの RIP レジスタをセットする。
- (9) ユーザランドでハンドルしなければならないイベントが来るまでカーネル内でループし、VMEntry を繰り返す。
- (10) アセンブリコードの vmx_setjmp を呼び出し、vmxctx へホストレジスタを退避、VMX_RETURN_DIRECT がリターンされる。
- (12) rc は VMX_RETURN_DIREC
- (13) 1 度目のループでは launched == 0 で vmx_launch が呼ばれる。二度目以降は vmx_resume が呼ばれる。
- (14) アセンブリコードの vmx_launch を呼び出して VMX non-root mode へ CPU を切り替える。
- [5] ops->vminit は vmx_vminit を指している。
- (6) ops->vmrun は vmx_run を指している。

リスト 4 の解説

vmcs.c は VMCS の設定に関するコードを提供しています。ここでは HOST_RIP の書き込みに注目しています。なお、vmwrite を行う前に vmptrld 命令を発行していますが、これは CPU へ VMCS ポインタをセットして VMCS へアクセス可能にするためです。同様に、vmwrite を行った後に vmclear 命令を発行していますが、これは変更された VMCS をメモリへライトバックさせるためです。

■リスト 4

```
......(省略)......
 309:  int
 310:  vmcs_set_defaults(struct vmcs *vmcs,
 311:         u_long host_rip, u_long host_rsp, u_long ept_pml4,
 312:         uint32_t pinbased_ctls, uint32_t procbased_ctls,
 313:         uint32_t procbased_ctls2, uint32_t exit_ctls,
 314:         uint32_t entry_ctls, u_long msr_bitmap, uint16_t vpid)
 315:  {
......(省略)......
 325:   /*
 326:    * Make sure we have a "current" VMCS to work with.
 327:    */
 328:   VMPTRLD(vmcs);
......(省略)......
 416:   /* instruction pointer */
 417:   if ((error = vmwrite(VMCS_HOST_RIP, host_rip)) != 0)        [9]
 418:       goto done;
......(省略)......
 446:   VMCLEAR(vmcs);
 447:   return (error);
 448:  }
```

- [9] VMCS の HOST_RIP に host_rip を指定している。
  vmx_vminit では vmx_longjmp が指定されているため、結果として VMExit すると常に vmx_longjmp から再開される事となる。

リスト 5 **の解説**

vmx_support.S は C 言語で記述できない、コンテキストの退避/復帰や VT-x 拡張命令の発行などのコードを提供しています。ここでは、ホストレジスタの退避 (vmx_setjmp) と VMEntry(vmx_launch) の処理について見ています。

■リスト 5

```
......(省略)......
  69:  #define  VMX_GUEST_RESTORE                        \           (17)
```

```
 70:   movq    %rdi,%rsp;                       \
 71:   movq    VMXCTX_GUEST_CR2(%rdi),%rsi;                \
 72:   movq    %rsi,%cr2;                      \
 73:   movq    VMXCTX_GUEST_RSI(%rdi),%rsi;                \
 74:   movq    VMXCTX_GUEST_RDX(%rdi),%rdx;                \
 75:   movq    VMXCTX_GUEST_RCX(%rdi),%rcx;                \
 76:   movq    VMXCTX_GUEST_R8(%rdi),%r8;              \
 77:   movq    VMXCTX_GUEST_R9(%rdi),%r9;              \
 78:   movq    VMXCTX_GUEST_RAX(%rdi),%rax;                \
 79:   movq    VMXCTX_GUEST_RBX(%rdi),%rbx;                \
 80:   movq    VMXCTX_GUEST_RBP(%rdi),%rbp;                \
 81:   movq    VMXCTX_GUEST_R10(%rdi),%r10;                \
 82:   movq    VMXCTX_GUEST_R11(%rdi),%r11;                \
 83:   movq    VMXCTX_GUEST_R12(%rdi),%r12;                \
 84:   movq    VMXCTX_GUEST_R13(%rdi),%r13;                \
 85:   movq    VMXCTX_GUEST_R14(%rdi),%r14;                \
 86:   movq    VMXCTX_GUEST_R15(%rdi),%r15;                \
 87:   movq    VMXCTX_GUEST_RDI(%rdi),%rdi; /* restore rdi the last */
 88:
 89:  #define  VM_INSTRUCTION_ERROR(reg)                   \
 90:   jnc     1f;                         \
 91:   movl    $VM_FAIL_INVALID,reg;       /* CF is set */     \
 92:   jmp     3f;                         \
 93:  1:   jnz     2f;                         \
 94:   movl    $VM_FAIL_VALID,reg;     /* ZF is set */     \
 95:   jmp     3f;                         \
 96:  2:   movl    $VM_SUCCESS,reg;                    \
 97:  3:   movl    reg,VMXCTX_LAUNCH_ERROR(%rsp)
.....(省略).....
107:  ENTRY(vmx_setjmp)                                           (11)
108:   movq    (%rsp),%rax         /* return address */
109:   movq    %r15,VMXCTX_HOST_R15(%rdi)
110:   movq    %r14,VMXCTX_HOST_R14(%rdi)
111:   movq    %r13,VMXCTX_HOST_R13(%rdi)
112:   movq    %r12,VMXCTX_HOST_R12(%rdi)
113:   movq    %rbp,VMXCTX_HOST_RBP(%rdi)
114:   movq    %rsp,VMXCTX_HOST_RSP(%rdi)
115:   movq    %rbx,VMXCTX_HOST_RBX(%rdi)
116:   movq    %rax,VMXCTX_HOST_RIP(%rdi)
117:
```

```
118:   /*
119:    * XXX save host debug registers
120:    */
121:   movl    $VMX_RETURN_DIRECT,%eax
122:   ret
123:  END(vmx_setjmp)
......(省略)......
223:  ENTRY(vmx_launch)                                          (15)
224:   VMX_DISABLE_INTERRUPTS
225:
226:   VMX_CHECK_AST
227:
228:   /*
229:    * Restore guest state that is not automatically loaded from the vmcs.
230:    */
231:   VMX_GUEST_RESTORE                                         (16)
232:
233:   vmlaunch                                                  (18)
234:
235:   /*
236:    * Capture the reason why vmlaunch failed.
237:    */
238:   VM_INSTRUCTION_ERROR(%eax)                                (19)
239:
240:   /* Return via vmx_setjmp with return value of VMX_RETURN_VMLAUNCH */
241:   movq    %rsp,%rdi
242:   movq    $VMX_RETURN_VMLAUNCH,%rsi
243:
244:   addq    $VMXCTX_TMPSTKTOP,%rsp
245:   callq   vmx_return
246:  END(vmx_launch)
```

- (17) vmxctx 上のゲストレジスタ値を実レジスタに展開。
  これらのレジスタは VMEntry/VMExit 時に VMCS からセーブ・リストアされないのでハイパーバイザがケアする必要がある。
- (11) vmctx（VMCS ではない）にホストレジスタを退避して VMX_RETURN_DIRECT をリターンする。
- (15) ゲストレジスタをリストアして VMEntry する。
- (16) ゲストレジスタリストアを行うマクロを展開。

- (18) vmlaunch 命令で VMX non-root mode へ VMEntry する。
- (19) ここには VMEntry が失敗したときのみ到達する。
  正常に VMEntry したのち VMExit した場合は、前述の通り vmx_longjmp へジャンプする。

まとめ

vmm.ko が VM_RUN ioctl を受け取ってから VMEntry するまでにどのような処理が行われているかについて、ソースコードを解説しました。次回はこれに対応する VMExit の処理について見ていきます。

ライセンス