How to write your own KVM client from scratch

Murilo Opsfelder Araújo
Software Engineer
Linux Technology Center, IBM

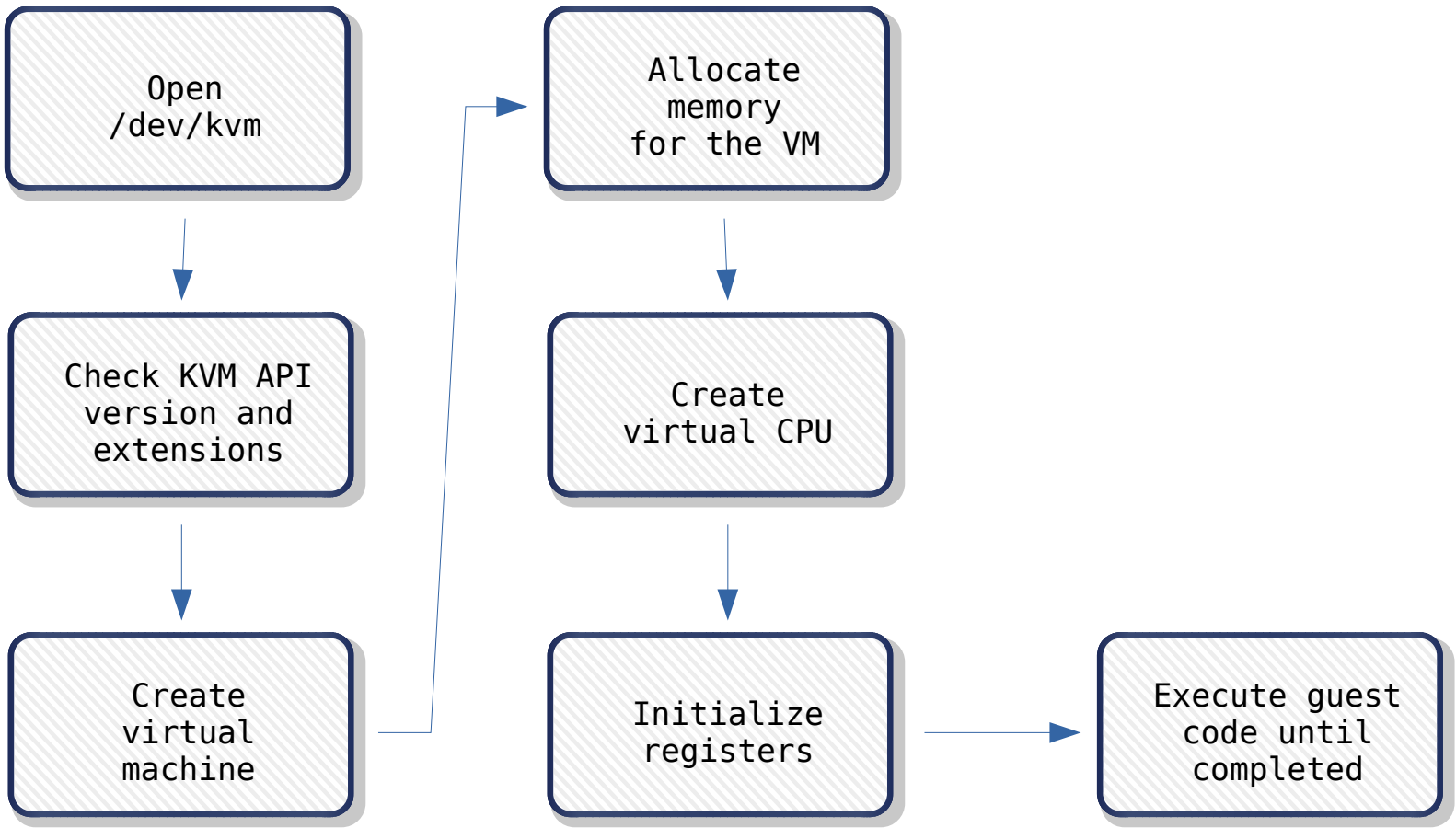
Agenda

- Workflow
- KVM API
- Your own KVM client
- QEMU examples

Workflow

- Open **`/dev/kvm`**
- Obtain KVM API version
- Check extension, if needed
- Create virtual machine
- Allocate memory for the virtual machine
- Create virtual CPU
- Run guest code

Workflow


Open /dev/kvm
Check KVM API version and extensions
Create virtual machine
Allocate memory for the VM
Create virtual CPU
Initialize registers
Execute guest code until completed

KVM API

- More than 11 years of stable API!
  - **`KVM_API_VERSION`** changed to **`12`** with 2.6.22 (released July 2007)
  - Locked as stable interface since 2.6.24 (release January 2008)
- Programs need to match the API version
- **`virt/kvm/kvm_main.c`**: entry point, main code, common to all architectures

KVM API

- **`Documentation/virtual/kvm/api.txt`**
  - "The Definitive KVM API Documentation"
  - All API is documented here (or should be)
  - Around 120 **`ioctl()`**'s available
    - **System**: affects the whole KVM subsystem, e.g.: create virtual machines
    - **VM**: affects virtual machine attributes, e.g.: memory layout, create vCPU
    - **vCPU**: affects a single vCPU, e.g.: set registers, run guest code
    - **Device**: attributes that control the operation of a single device

Your own KVM client

- No need to run a complete operating system
- No need to emulate a full suite of hardware
- Just run code inside a sandbox

Other KVM clients

- **novm**: lightweight hypervisor written in Go that uses KVM API
  - https://github.com/google/novm
- **kvmtool**: lightweight tool for hosting KVM guests
  - https://git.kernel.org/pub/scm/linux/kernel/git/will/kvmtool.git/tree/README
- **Kata Containers**: creates a KVM virtual machine for each container or pod
  - https://github.com/kata-containers/

x86 example

- “Using the KVM API”
  - https://lwn.net/Articles/658511/
- **`kvmtest.c`**
  - https://lwn.net/Articles/658512/

kvmtest.c

```
/* Sample code for /dev/kvm API
 *
 * Copyright (c) 2015 Intel Corporation
 * Author: Josh Triplett <josh@joshtriplett.org>
 *
 * Permission is hereby granted, free of charge, to any person obtaining a copy
 * of this software and associated documentation files (the "Software"), to
 * deal in the Software without restriction, including without limitation the
 * rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
 * sell copies of the Software, and to permit persons to whom the Software is
 * furnished to do so, subject to the following conditions:
 *
 * The above copyright notice and this permission notice shall be included in
 * all copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
 * IN THE SOFTWARE.
 */
...
```

kvmtest.c

```
const uint8_t code[] = {
    0xba, 0xf8, 0x03, /* mov $0x3f8, %dx */
    0x00, 0xd8,       /* add %bl, %al */
    0x04, '0',        /* add $'0', %al */
    0xee,             /* out %al, (%dx) */
    0xb0, '\n',       /* mov $'\n', %al */
    0xee,             /* out %al, (%dx) */
    0xf4,             /* hlt */
};
```

kvmtest.c

```
kvm = open("/dev/kvm", O_RDWR | O_CLOEXEC);

if (kvm == -1)
    err(1, "/dev/kvm");
```

kvmtest.c

```
/* Make sure we have the stable version of the API */
ret = ioctl(kvm, KVM_GET_API_VERSION, NULL);

if (ret == -1)
    err(1, "KVM_GET_API_VERSION");

if (ret != 12)
    errx(1, "KVM_GET_API_VERSION %d, expected 12", ret);
```

kvmtest.c

```
vmfd = ioctl(kvm, KVM_CREATE_VM, (unsigned long)0);

if (vmfd == -1)
    err(1, "KVM_CREATE_VM");
```

kvmtest.c

```
/* Allocate one aligned page of guest memory to hold the code. */
mem = mmap(NULL, 0x1000, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS, -1, 0);

if (!mem)
    err(1, "allocating guest memory");

memcpy(mem, code, sizeof(code));
```

kvmtest.c

```
struct kvm_userspace_memory_region region = {
    .slot = 0,
    .guest_phys_addr = 0x1000,
    .memory_size = 0x1000,
    .userspace_addr = (uint64_t)mem,
};

ret = ioctl(vmfd, KVM_SET_USER_MEMORY_REGION, &region);

if (ret == -1)
    err(1, "KVM_SET_USER_MEMORY_REGION");
```

kvmtest.c

```
vcpufd = ioctl(vmfd, KVM_CREATE_VCPU, (unsigned long)0);

if (vcpufd == -1)
    err(1, "KVM_CREATE_VCPU");
```

kvmtest.c

```
/* Map the shared kvm_run structure and following data. */
ret = ioctl(kvm, KVM_GET_VCPU_MMAP_SIZE, NULL);
if (ret == -1)
    err(1, "KVM_GET_VCPU_MMAP_SIZE");

mmap_size = ret;
if (mmap_size < sizeof(*run))
    errx(1, "KVM_GET_VCPU_MMAP_SIZE unexpectedly small");

run = mmap(NULL, mmap_size, PROT_READ | PROT_WRITE, MAP_SHARED, vcpufd, 0);
if (!run)
    err(1, "mmap vcpu");
```

kvmtest.c

```
/* Initialize CS to point at 0, via a read-modify-write of sregs. */
ret = ioctl(vcpufd, KVM_GET_SREGS, &sregs);
if (ret == -1)
    err(1, "KVM_GET_SREGS");

sregs.cs.base = 0;
sregs.cs.selector = 0;

ret = ioctl(vcpufd, KVM_SET_SREGS, &sregs);
if (ret == -1)
    err(1, "KVM_SET_SREGS");
```

kvmtest.c

```
/* Initialize registers: instruction pointer for our code, addends, and
 * initial flags required by x86 architecture. */
struct kvm_regs regs = {
    .rip = 0x1000,
    .rax = 2,
    .rbx = 2,
    .rflags = 0x2,
};

ret = ioctl(vcpufd, KVM_SET_REGS, &regs);
if (ret == -1)
    err(1, "KVM_SET_REGS");
```

kvmtest.c

```
/* Repeatedly run code and handle VM exits. */
while (1) {
    ret = ioctl(vcpufd, KVM_RUN, NULL);
    if (ret == -1)
        err(1, "KVM_RUN");

    switch (run->exit_reason) {

    case KVM_EXIT_HLT:
        puts("KVM_EXIT_HLT");
        return 0;

    case KVM_EXIT_IO:
        if (run->io.direction == KVM_EXIT_IO_OUT
            && run->io.size == 1
            && run->io.port == 0x3f8
            && run->io.count == 1)
            putchar(*(((char *)run) + run->io.data_offset));
        else
            errx(1, "unhandled KVM_EXIT_IO");
        break;
```

```
case KVM_EXIT_FAIL_ENTRY:
    errx(1, "KVM_EXIT_FAIL_ENTRY:”
            “ hardware_entry_failure_reason = 0x%llx",
         (unsigned long long)run->fail_entry.hardware_entry_failure_reason);

case KVM_EXIT_INTERNAL_ERROR:
    errx(1, "KVM_EXIT_INTERNAL_ERROR: suberror = 0x%x",
         run->internal.suberror);

default:
    errx(1, "exit_reason = 0x%x", run->exit_reason);
    }
}
```

kvmtest.c

```
$ gcc -o kvmtest kvmtest.c


$ ./kvmtest
4
KVM_EXIT_HLT


$ strace ./kvmtest 2>&1 | grep ioctl
ioctl(3, KVM_GET_API_VERSION, 0)        = 12
ioctl(3, KVM_CREATE_VM, 0)              = 4
ioctl(4, KVM_SET_USER_MEMORY_REGION, {slot=0, flags=0, guest_phys_addr=0x1000, memory_size=4096, userspace_addr=0x7f364e640000}) = 0
ioctl(4, KVM_CREATE_VCPU, 0)            = 5
ioctl(3, KVM_GET_VCPU_MMAP_SIZE, 0)     = 12288
ioctl(5, KVM_GET_SREGS, {cs={base=0xffff0000, limit=65535, selector=61440, type=11, present=1, dpl=0, db=0, s=1, l=0, g=0, avl=0}, ...}) = 0
ioctl(5, KVM_SET_SREGS, {cs={base=0, limit=65535, selector=0, type=11, present=1, dpl=0, db=0, s=1, l=0, g=0, avl=0}, ...}) = 0
ioctl(5, KVM_SET_REGS, {rax=0x2, ..., rsp=0, rbp=0, ..., rip=0x1000, rflags=0x2}) = 0
ioctl(5, KVM_RUN, 0)                    = 0
ioctl(5, KVM_RUN, 0)                    = 0
ioctl(5, KVM_RUN, 0)                    = 0
```

QEMU examples

```
int kvm_ioctl(KVMState *s, int type, ...)
{
    int ret;
    void *arg;
    va_list ap;

    va_start(ap, type);
    arg = va_arg(ap, void *);
    va_end(ap);

    trace_kvm_ioctl(type, arg);
    ret = ioctl(s->fd, type, arg);
    if (ret == -1) {
        ret = -errno;
    }
    return ret;
}
```

Source: **accel/kvm/kvm-all.c** from QEMU source code

```
int kvm_vm_ioctl(KVMState *s, int type, ...)
{
    int ret;
    void *arg;
    va_list ap;

    va_start(ap, type);
    arg = va_arg(ap, void *);
    va_end(ap);

    trace_kvm_vm_ioctl(type, arg);
    ret = ioctl(s->vmfd, type, arg);
    if (ret == -1) {
        ret = -errno;
    }
    return ret;
}
```

Source: **accel/kvm/kvm-all.c** from QEMU source code

```
int kvm_vcpu_ioctl(CPUState *cpu, int type, ...)
{
    int ret;
    void *arg;
    va_list ap;

    va_start(ap, type);
    arg = va_arg(ap, void *);
    va_end(ap);

    trace_kvm_vcpu_ioctl(cpu->cpu_index, type, arg);
    ret = ioctl(cpu->kvm_fd, type, arg);
    if (ret == -1) {
        ret = -errno;
    }
    return ret;
}
```

Source: **accel/kvm/kvm-all.c** from QEMU source code

```
int kvm_device_ioctl(int fd, int type, ...)
{
    int ret;
    void *arg;
    va_list ap;

    va_start(ap, type);
    arg = va_arg(ap, void *);
    va_end(ap);

    trace_kvm_device_ioctl(fd, type, arg);
    ret = ioctl(fd, type, arg);
    if (ret == -1) {
        ret = -errno;
    }
    return ret;
}
```

Source: **accel/kvm/kvm-all.c** from QEMU source code

```
static int kvm_init(MachineState *ms)
...
    ret = kvm_ioctl(s, KVM_GET_API_VERSION, 0);
    if (ret < KVM_API_VERSION) {
        if (ret >= 0) {
            ret = -EINVAL;
        }
        fprintf(stderr, "kvm version too old\n");
        goto err;
    }

    if (ret > KVM_API_VERSION) {
        ret = -EINVAL;
        fprintf(stderr, "kvm version not supported\n");
        goto err;
    }
```

Source: **accel/kvm/kvm-all.c** from QEMU source code

```
static int kvm_init(MachineState *ms)
...
    do {
        ret = kvm_ioctl(s, KVM_CREATE_VM, type);
    } while (ret == -EINTR);

    if (ret < 0) {
        fprintf(stderr, "ioctl(KVM_CREATE_VM) failed: %d %s\n", -ret,
                strerror(-ret));
...
```

Source: **accel/kvm/kvm-all.c** from QEMU source code

```
bool kvm_device_supported(int vmfd, uint64_t type)
{
    struct kvm_create_device create_dev = {
        .type = type,
        .fd = -1,
        .flags = KVM_CREATE_DEVICE_TEST,
    };

    if (ioctl(vmfd, KVM_CHECK_EXTENSION, KVM_CAP_DEVICE_CTRL) <= 0) {
        return false;
    }

    return (ioctl(vmfd, KVM_CREATE_DEVICE, &create_dev) >= 0);
}
```

Source: **accel/kvm/kvm-all.c** from QEMU source code

```
bool kvm_arm_create_scratch_host_vcpu(const uint32_t *cpus_to_try, int *fdarray,
                                      struct kvm_vcpu_init *init)
{
    int ret, kvmfd = -1, vmfd = -1, cpufd = -1;

    kvmfd = qemu_open("/dev/kvm", O_RDWR);
    if (kvmfd < 0) {
        goto err;
    }
    vmfd = ioctl(kvmfd, KVM_CREATE_VM, 0);
    if (vmfd < 0) {
        goto err;
    }
    cpufd = ioctl(vmfd, KVM_CREATE_VCPU, 0);
    if (cpufd < 0) {
        goto err;
    }
...
```

Source: **target/arm/kvm.c** from QEMU source code

```
int kvm_destroy_vcpu(CPUState *cpu)
...
    mmap_size = kvm_ioctl(s, KVM_GET_VCPU_MMAP_SIZE, 0);
    if (mmap_size < 0) {
        ret = mmap_size;
        DPRINTF("KVM_GET_VCPU_MMAP_SIZE failed\n");
        goto err;
    }

    ret = munmap(cpu->kvm_run, mmap_size);
    if (ret < 0) {
        goto err;
    }
...
```

Source: **accel/kvm/kvm-all.c** from QEMU source code

```
int kvm_cpu_exec(CPUState *cpu)
...
    run_ret = kvm_vcpu_ioctl(cpu, KVM_RUN, 0);
...
    if (run_ret < 0) {
        if (run_ret == -EINTR || run_ret == -EAGAIN) {
            DPRINTF("io window exit\n");
            kvm_eat_signals(cpu);
            ret = EXCP_INTERRUPT;
            break;
        }
        fprintf(stderr, "error: kvm run failed %s\n",
                strerror(-run_ret));
...
```

Source: **accel/kvm/kvm-all.c** from QEMU source code

```
static void *qemu_kvm_cpu_thread_fn(void *arg)
{
...
    do {
        if (cpu_can_run(cpu)) {
            r = kvm_cpu_exec(cpu);
            if (r == EXCP_DEBUG) {
                cpu_handle_guest_debug(cpu);
            }
        }
        qemu_wait_io_event(cpu);
    } while (!cpu->unplug || cpu_can_run(cpu));
...
```

Source: **cpus.c** from QEMU source code

```
int kvm_arch_get_registers(CPUState *cs)
...
    struct kvm_regs regs;
...
    ret = kvm_vcpu_ioctl(cs, KVM_GET_REGS, &regs);
    if (ret < 0) {
        return ret;
    }

    cr = regs.cr;
...
    env->ctr = regs.ctr;
    env->lr = regs.lr;
    cpu_write_xer(env, regs.xer);
    env->msr = regs.msr;
    env->nip = regs.pc;
...
```

Source: **target/ppc/kvm.c** from QEMU source code

```
int kvm_arch_put_registers(CPUState *cs, int level)
...
    struct kvm_regs regs;
...
    /* Set the registers based on QEMU's view of things */
    for (i = 0; i < 32; i++) {
        regs.gpr[i] = (int64_t)(target_long)env->active_tc.gpr[i];
    }

    regs.hi = (int64_t)(target_long)env->active_tc.HI[0];
    regs.lo = (int64_t)(target_long)env->active_tc.LO[0];
    regs.pc = (int64_t)(target_long)env->active_tc.PC;

    ret = kvm_vcpu_ioctl(cs, KVM_SET_REGS, &regs);

    if (ret < 0) {
        return ret;
    }
...
```

Source: **`target/mips/kvm.c`** from QEMU source code

```
static void kvm_get_one_spr(CPUState *cs, uint64_t id, int spr)
...
    union {
        uint32_t u32;
        uint64_t u64;
    } val;
    struct kvm_one_reg reg = {
        .id = id,
        .addr = (uintptr_t) &val,
    };
    int ret;

    ret = kvm_vcpu_ioctl(cs, KVM_GET_ONE_REG, &reg);
    if (ret != 0) {
        trace_kvm_failed_spr_get(spr, strerror(errno));
...
```

Source: **target/ppc/kvm.c** from QEMU source code

```
static void kvm_put_one_spr(CPUState *cs, uint64_t id, int spr)
...
    union {
        uint32_t u32;
        uint64_t u64;
    } val;
    struct kvm_one_reg reg = {
        .id = id,
        .addr = (uintptr_t) &val,
    };
    int ret;
...
    ret = kvm_vcpu_ioctl(cs, KVM_SET_ONE_REG, &reg);
    if (ret != 0) {
        trace_kvm_failed_spr_set(spr, strerror(errno));
    }
...
```

Source: **target/ppc/kvm.c** from QEMU source code

References

- "Using the KVM API", Josh Triplett, September 2015
  - https://lwn.net/Articles/658511/
- "kvmtest.c", Josh Triplett, September 2015
  - https://lwn.net/Articles/658512/
- QEMU source code
  - https://git.qemu.org/?p=qemu.git
- Linux kernel KVM API documentation
  - https://www.kernel.org/doc/Documentation/virtual/kvm/api.txt

Thanks!

- Family and friends at work
- linuxdev-br orga team, specially Gabriel Gomes
- https://t.me/linuxdevbr

About me

- Murilo Opsfelder Araújo
  - Software Engineer
  - Linux Technology Center, IBM
  - muriloo@br.ibm.com