CS695 Assignment2

Arohan Hazarika

February 2025

1 Flowchart (Part 1)

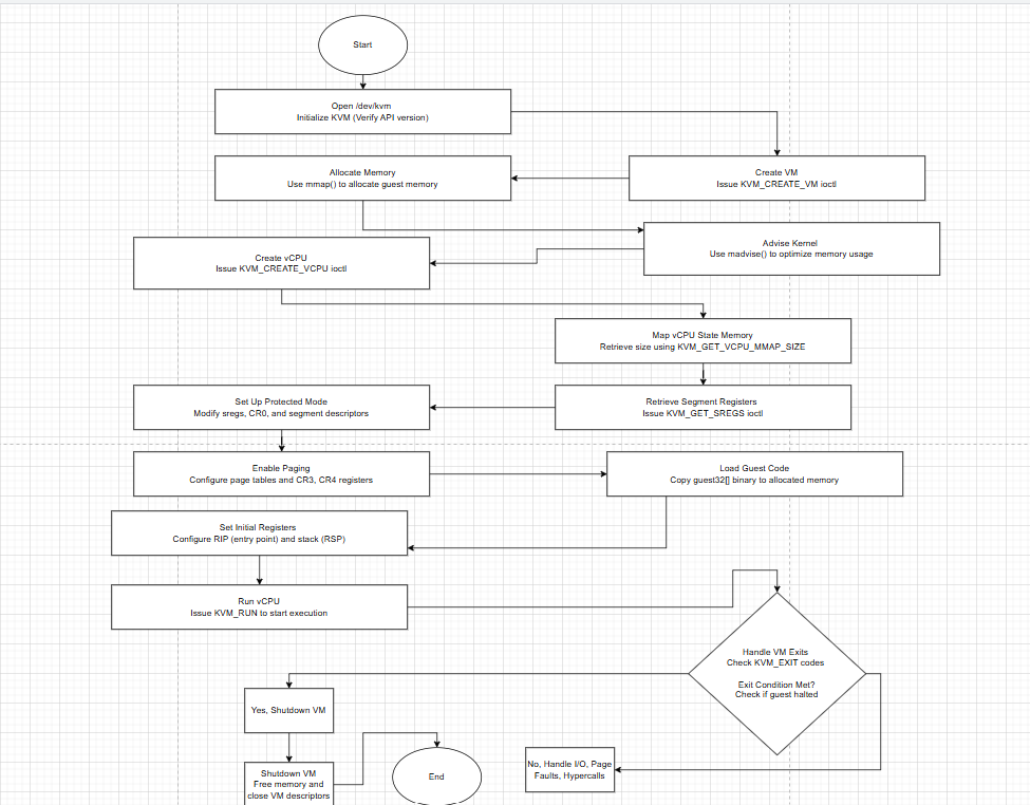

Figure 1: Flowchart explaining execution of VM in protected mode

2 Explanation of Code Snippets

2.1 (a)

```
extern const unsigned char guest64[], guest64_end[];
```

These external variables mark the start and end of the guest code binary, stored in memory. The `guest64` binary is loaded into VM memory (`vm->mem`) to allow execution inside the guest environment.

2.2 (b)

```
pml4[0] = PDE64_PRESENT | PDE64_RW | PDE64_USER | pdpt_addr;
pdpt[0] = PDE64_PRESENT | PDE64_RW | PDE64_USER | pd_addr;
pd[0] = PDE64_PRESENT | PDE64_RW | PDE64_USER | PDE64_PS;

sregs->cr3 = pml4_addr;
sregs->cr4 = CR4_PAE;
sregs->cr0 = CR0_PE | CR0_MP | CR0_ET | CR0_NE | CR0_WP | CR0_AM | CR0_PG;
sregs->efer = EFER_LME | EFER_LMA;
```

This code sets up a 4-level page table for memory management in long mode, configuring the **PML4**, **PDPT**, and **Page Directory** to establish a virtual address space. The `CR3` register is assigned the address of the top-level page table (PML4), allowing address translation. `CR0` and `CR4` enable paging and protected mode, while `EFER` enables long mode execution.

2.3 (c)

```
vm->mem = mmap(NULL, mem_size, PROT_READ | PROT_WRITE,
               MAP_PRIVATE | MAP_ANONYMOUS | MAP_NORESERVE, -1, 0);
madvise(vm->mem, mem_size, MADV_MERGEABLE);
```

The `mmap()` function is used to allocate memory for the virtual machine, mapping an anonymous memory region that serves as the VM's physical memory. The `MADV_MERGEABLE` flag in `madvise()` allows the kernel to optimize memory by merging identical pages, thus reducing redundancy and improving efficiency.

2.4 (d)

```
case KVM_EXIT_IO:
if (vcpu->kvm_run->io.direction == KVM_EXIT_IO_OUT &&
    vcpu->kvm_run->io.port == 0xE9) {
    char *p = (char *)vcpu->kvm_run;
    fwrite(p + vcpu->kvm_run->io.data_offset,
           vcpu->kvm_run->io.size, 1, stdout);
    fflush(stdout);
    continue;
}
```

When a guest tries to write to the I/O port `0xE9`, it triggers a VM exit (`KVM_EXIT_IO`). The hypervisor detects the exit, retrieves the data from the guest memory, and prints it to stdout. This mechanism allows communication between the guest and the host, simulating hypercalls or debugging output.

2.5 (e)

```
memcpy(&memval, &vm->mem[0x400], sz);
```

This snippet accesses memory inside the guest at address `0x400` and copies `sz` bytes into `memval`.