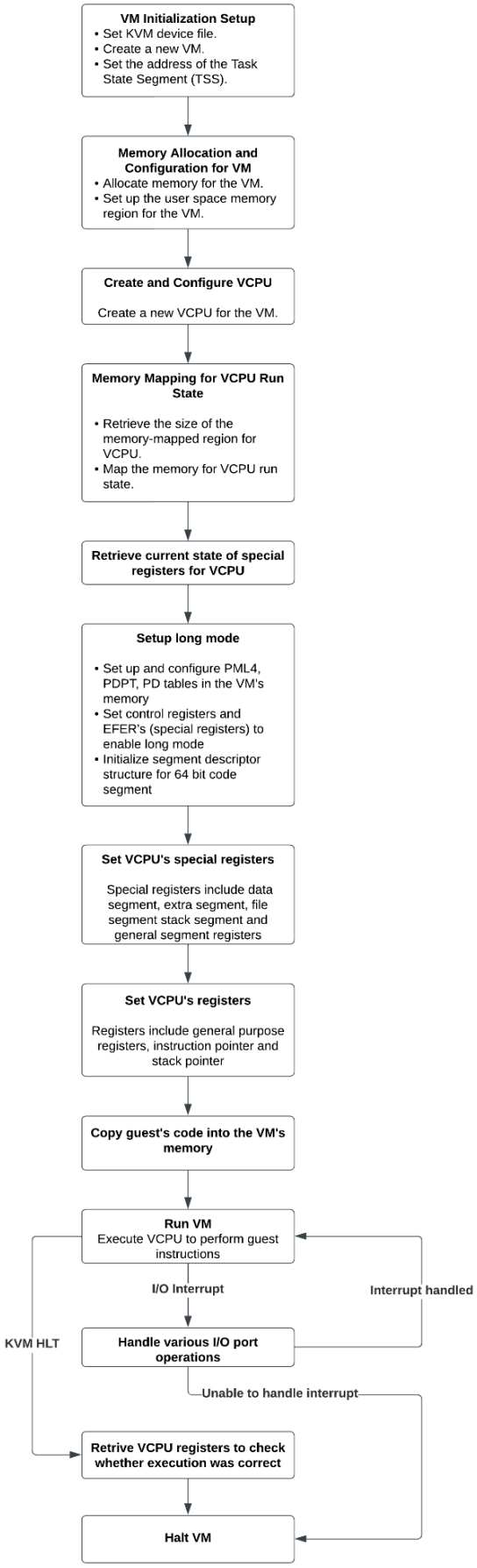
CS-695 Assignment-2
23M0743

Q 1_a)



1) VM initialization setup

Set KVM device file: The program opens the /dev/kvm device file using the open() system call, enabling communication with the Kernel-based Virtual Machine (KVM) subsystem.

Create a new VM: It creates a new virtual machine by issuing the **KVM_CREATE_VM** ioctl call to the KVM device file descriptor obtained in the previous step.

Set the address of the Task State Segment (TSS): Using **the KVM_SET_TSS_ADDR** ioctl call, the program sets the address of the Task State Segment (TSS). This is essential for managing task-related information during virtual machine execution.

2) Memory allocation and configuation for VM

Allocate memory for the VM: Then the program allocates memory for the virtual machine using the **mmap()** system call. This memory will be used to store the guest's memory.

Set up the user space memory region for the VM: It configures the memory region for the virtual machine in user space by defining the guest physical address, memory size, and user space address. This is done using the **KVM_SET_USER_MEMORY_REGION** ioctl call to the VM file descriptor.

3) Create and Configure VCPU

Create a new VCPU for the VM: The program creates a new Virtual CPU (VCPU) for the virtual machine by issuing the **KVM_CREATE_VCPU** ioctl call to the VM file descriptor obtained earlier. This call creates a new VCPU instance associated with the VM.

4) Memory mapping for VCPU run state

Retrieve the size of the memory-mapped region for VCPU: After this, the program retrieves the size of the memory-mapped region for VCPU run state by issuing the **KVM_GET_VCPU_MMAP_SIZE** ioctl call to the KVM device file descriptor. This size determines the amount of memory needed to store the VCPU's run state.

Map the memory for VCPU run state: It maps the memory for the VCPU's run state by calling **mmap()** with the appropriate size and permissions. This memory mapping allows the program to access and manipulate the VCPU's run state data.

5) Retrieve current state of special registers for VCPU

Then it retrieves the current state of special registers for the VCPU by issuing the **KVM_GET_SREGS** ioctl call to the VCPU file descriptor. This call fetches the current values of

special registers such as control registers (CR0, CR3, CR4), segment registers (CS, DS, SS, etc.), and others, providing crucial information about the VCPU's execution context.

6) Setup long module

Configure page tables: Manually set up and configure the necessary page tables (PML4, PDPT, PD) in the VM's memory.

Enable long mode: Set control registers (CR0, CR3, CR4) and EFER to enable long mode operation.

Initialize 64-bit code segment: Initialize the segment descriptor structure for the 64-bit code segment (CS) with appropriate values.

7) Set VCPU’s special registers

Set special registers: Then the program sets the VCPU's special registers, including data segment, extra segment, file segment, stack segment, and general segment registers. This is achieved by updating the appropriate fields in the segment descriptor structure and issuing the **KVM_SET_SREGS** ioctl call to the VCPU file descriptor.

8) Set VCPU’s registers

Set general-purpose registers: Sets the VCPU's general-purpose registers along with the instruction pointer (RIP) and stack pointer (RSP). This is accomplished by updating the appropriate fields in the struct kvm_regs structure and issuing the **KVM_SET_REGS** ioctl call to the VCPU file descriptor.

9) Copy guest’s code into the VM’s memory

Copies the guest's code into the VM's memory. This is achieved by using the **memcpy()** to copy the guest code from its source location to the allocated memory region for the VM.

10) Run VM

Execute VCPU to run guest code: Then the program tells the VCPU to start running the guest's code. This is done by asking the VCPU to begin executing instructions using the ioctl command **KVM_RUN**.

11) Handle various I/O port operations

Handling various input/output (I/O) port operations, such as reading from or writing to specific I/O ports.

12) Retrive VCPU registers to check whether execution was correct

Retrieving the VCPU registers to check whether the execution was correct. This involves using the **KVM_GET_REGS** ioctl call to fetch the current values of the VCPU's registers. By examining these registers, the program can verify the outcome of the execution and determine whether it was successful.

IOCTL calls	**Input arguments**	**Description**
ioctl(vm->dev_fd, KVM_CREATE_VM, 0)	vm->dev_fd: Device file descriptor KVM_CREATE_VM: Command 0: No i/p to KVM_CREATE_VM	Create a new virtual machine (VM)
ioctl(vm->vm_fd, KVM_SET_TSS_ADDR, 0xfffbd000)	vm->vm_fd: VM file descriptor KVM_SET_TSS_ADDR: Command 0xfffbd000: TSS address	Set the address of the TSS for the VM
ioctl(vm->vm_fd, KVM_SET_USER_MEMORY_REGION, &memreg)	vm->vm_fd: VM file descriptor KVM_SET_USER_MEMORY_REGION: Command &memreg: Pointer to memory region descriptor	Set up user space memory region for the VM
ioctl(vm->vm_fd, KVM_CREATE_VCPU, 0)	vm->vm_fd: VM file descriptor KVM_CREATE_VCPU: Command 0: No i/p to KVM_CREATE_CPU	Create a new virtual CPU (VCPU) for the VM
ioctl(vm->dev_fd, KVM_GET_VCPU_MMAP_SIZE, 0)	vm->dev_fd: Device File descriptor KVM_GET_VCPU_MMAP_SIZE: Command 0: No i/p to KVM_GET_VCPU_MMAP_SIZE	Retrieve the size of the VCPU's memory-mapped region
ioctl(vcpu->vcpu_fd, KVM_GET_SREGS, &sregs)	vcpu->vcpu_fd: VCPU file descriptor KVM_GET_SREGS: Command &sregs: Pointer to special registers struct	Retrieve current state of special registers for VCPU
ioctl(vcpu->vcpu_fd, KVM_SET_SREGS, &sregs)	vcpu->vcpu_fd: VCPU file descriptor KVM_SET_SREGS: Command &sregs: Pointer to special registers struct	Set VCPU's special registers

ioctl(vcpu->vcpu_fd, KVM_SET_REGS, &regs)	vcpu->vcpu_fd: VCPU file descriptor KVM_SET_REGS: Command &regs: Pointer to general-purpose registers struct	Set VCPU's general-purpose registers
ioctl(vcpu->vcpu_fd, KVM_RUN, 0)	vcpu->vcpu_fd: VCPU file descriptor KVM_RUN: Command 0: No i/p to KVM_RUN	Execute VCPU to perform guest instructions
ioctl(vcpu->vcpu_fd, KVM_GET_REGS, &regs)	Nonevcpu->vcpu_fd: VCPU file descriptor KVM_GET_REGS: Command &regs: Pointer to general-purpose registers struct	Retrieve VCPU registers to check execution