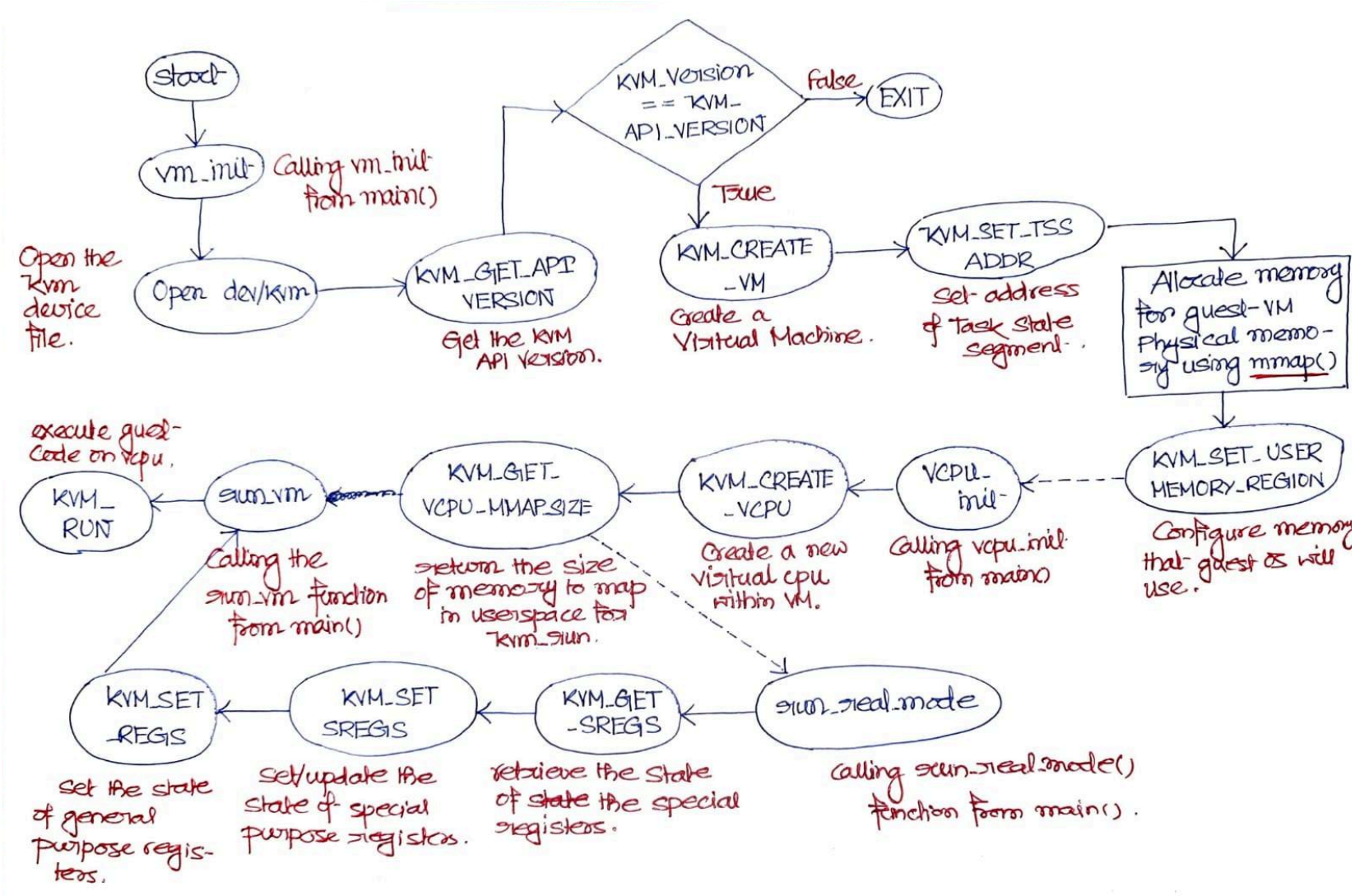
1.a.


start
vm_init
Calling vm_init from main()
Open the kvm device file.
Open dev/kvm
KVM_GET_API VERSION
Get the KVM API version.
KVM_Version == KVM_API_VERSION
False
EXIT
True
KVM_CREATE_VM
Create a Virtual Machine.
KVM_SET_TSS ADDR
Set address of Task State segment.
Allocate memory for guest-VM Physical memory using mmap()
KVM_SET_USER MEMORY_REGION
Configure memory that guest OS will use.
VCPU_init
Calling vcpu_init from main()
KVM_CREATE_VCPU
Create a new virtual cpu within VM.
KVM_GET_VCPU_MMAP_SIZE
Return the size of memory to map in userspace for kvm_run.
kvm_run
Calling the kvm_run function from main()
execute guest-code on vcpu.
KVM_RUN
kvm_real_mode
Calling kvm_real_mode() function from main().
KVM_GET_SREGS
Retrieve the State of state the special registers.
KVM_SET SREGS
Set/update the state of special purpose registers.
KVM_SET_REGS
Set the state of general purpose registers.

1.b.
KVM_APIs:

1. To get the version of KVM API, the KVM_GET_API_VERSION ioctl call is used.

   KVM_GET_API_VERSION:

   Brief description of inputs to the ioctl call:
   No specific input parameters required.

   Functionality that the ioctl call provides:
   Retrieves the version of the KVM API supported by the kernel.

2. To create a virtual machine (VM), which represents everything associated with one emulated system, including memory and one or more CPUs, the KVM_CREATE_VM ioctl call is used.

   KVM_CREATE_VM:

   Brief description of inputs to the ioctl call:
   Machine type identifier (KVM_VM_*) is given as an input parameter. Providing 0 as input implies that the new virtual machine has zero virtual CPUs and no allocated memory.

   Functionality that the ioctl call provides:
   Creates a new KVM virtual machine with the specified machine type and returns its file descriptor.

3. To set the address of the Task State Segment (TSS) for a virtual machine, the KVM_SET_TSS_ADDR ioctl call is used.

   KVM_SET_TSS_ADDR:

   Brief description of inputs to the ioctl call:
   unsigned long tss_address, The address to set as the base address of the TSS.

   Functionality that the ioctl call provides:
   Configuring the virtual machine represented by the given file descriptor to use the specified address i.e., unsigned long tss_address for its

Task State Statement. The Task State Segment (TSS) is a data structure in the x86 architecture that is used to store information about a task during a task switch.

4. To configure the memory region that the guest operating system will use within the virtual machine, The KVM_SET_USER_MEMORY_REGION ioctl call is used.

   KVM_SET_USER_MEMORY_REGION:

   Brief description of inputs to the ioctl call:
   struct kvm_userspace_memory_region, contains information about the memory region, such as its start address, size, and flags.

   Functionality that the ioctl call provides:
   This ioctl allows the user to create, modify or delete a guest physical memory slot.It helps in defining the mapping between guest physical memory and host physical memory.

5. To create a new virtual CPU (vcpu) within a virtual machine, the KVM_CREATE_VCPU ioctl call is used.

   KVM_CREATE_VCPU:

   Brief description of inputs to the ioctl call:
   vcpu id, The vcpu id is an integer in the range [0, max_vcpu_id).

   Functionality that the ioctl call provides:
   This ioctl call is used to spawn new virtual CPUs within an existing virtual machine. The newly created vCPU becomes part of the virtualized environment and can execute guest code.

6. To generate how much memory to map in the user space for kvm_run, the KVM_GET_VCPU_MMAP_SIZE ioctl call is used.

   KVM_GET_VCPU_MMAP_SIZE:

   Brief description of inputs to the ioctl call:
   No specific input parameters required.

   Functionality that the ioctl call provides:

The KVM_RUN ioctl communicates with userspace via a shared memory region. This ioctl returns the size of that region.

7. For running a virtual CPU (vCPU) in the KVM (Kernel-based Virtual Machine) subsystem, the KVM_RUN ioctl call is used.

   KVM_RUN:

   Brief description of inputs to the ioctl call:
   No specific input parameters required.

   Functionality that the ioctl call provides:
   Its primary purpose is to execute the guest code on the virtual CPU.

8. To read the general purpose registers from the vcpu, the KVM_GET_REGS ioctl call is used.

   KVM_GET_REGS:

   Brief description of inputs to the ioctl call:
   struct kvm_regs, Pointer to a structure (e.g., struct kvm_regs) where the register state will be stored.

   Functionality that the ioctl call provides:
   This ioctl call retrieves the register state of the specified vCPU and populates the provided data structure (regs) with the values of various registers.

9. To read the special registers from the vcpu, the KVM_GET_SREGS ioctl call is used.

   KVM_GET_SREGS:

   Brief description of inputs to the ioctl call:
   struct kvm_sregs, Pointer to a structure (e.g., struct kvm_regs) where the register state will be stored.

   Functionality that the ioctl call provides:
   The ioctl call retrieves the segment register state of the specified vCPU and populates the provided data structure (sregs) with the values of various segment registers. These segment registers include the code segment (CS), data segment (DS), and other segment registers that define the memory segmentation in the virtual CPU.

10. To write special registers into the vcpu, the KVM_SET_SREGS ioctl call is used.

    KVM_SET_SREGS:

    Brief description of inputs to the ioctl call:
    struct kvm_sregs, Pointer to a structure (e.g., struct kvm_regs) where the register state will be stored.

    Functionality that the ioctl call provides:
    This ioctl call updates the segment register state of the specified vCPU with the values provided in the data structure (sregs).