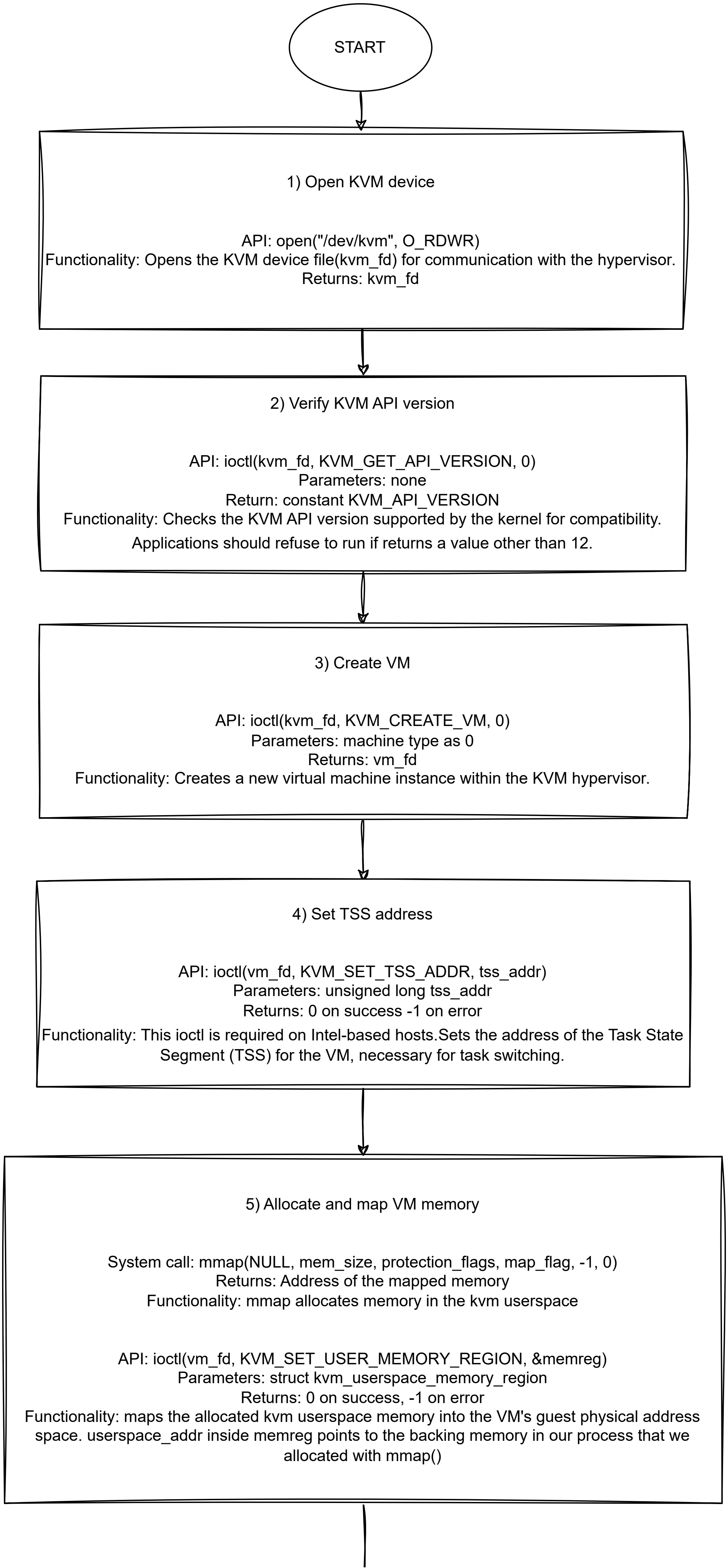


START

1) Open KVM device

API: open("/dev/kvm", O_RDWR)
Functionality: Opens the KVM device file(kvm_fd) for communication with the hypervisor.
Returns: kvm_fd

2) Verify KVM API version

API: ioctl(kvm_fd, KVM_GET_API_VERSION, 0)
Parameters: none
Return: constant KVM_API_VERSION
Functionality: Checks the KVM API version supported by the kernel for compatibility.
Applications should refuse to run if returns a value other than 12.

3) Create VM

API: ioctl(kvm_fd, KVM_CREATE_VM, 0)
Parameters: machine type as 0
Returns: vm_fd
Functionality: Creates a new virtual machine instance within the KVM hypervisor.

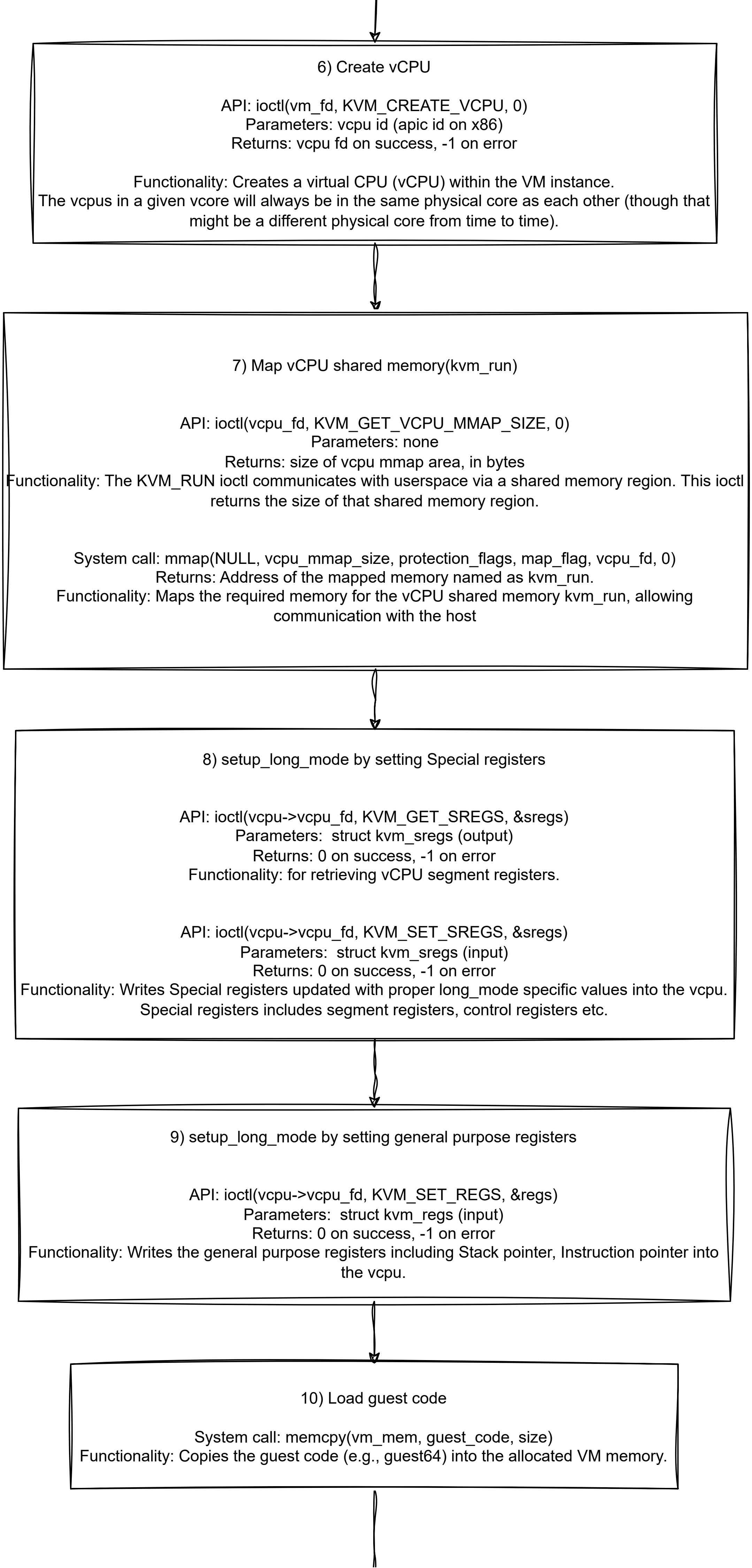
4) Set TSS address

API: ioctl(vm_fd, KVM_SET_TSS_ADDR, tss_addr)
Parameters: unsigned long tss_addr
Returns: 0 on success -1 on error
Functionality: This ioctl is required on Intel-based hosts.Sets the address of the Task State Segment (TSS) for the VM, necessary for task switching.

5) Allocate and map VM memory

System call: mmap(NULL, mem_size, protection_flags, map_flag, -1, 0)
Returns: Address of the mapped memory
Functionality: mmap allocates memory in the kvm userspace

API: ioctl(vm_fd, KVM_SET_USER_MEMORY_REGION, &memreg)
Parameters: struct kvm_userspace_memory_region
Returns: 0 on success, -1 on error
Functionality: maps the allocated kvm userspace memory into the VM's guest physical address space. userspace_addr inside memreg points to the backing memory in our process that we allocated with mmap()



6) Create vCPU

API: ioctl(vm_fd, KVM_CREATE_VCPU, 0)
Parameters: vcpu id (apic id on x86)
Returns: vcpu fd on success, -1 on error

Functionality: Creates a virtual CPU (vCPU) within the VM instance.
The vcpus in a given vcore will always be in the same physical core as each other (though that might be a different physical core from time to time).

7) Map vCPU shared memory(kvm_run)

API: ioctl(vcpu_fd, KVM_GET_VCPU_MMAP_SIZE, 0)
Parameters: none
Returns: size of vcpu mmap area, in bytes
Functionality: The KVM_RUN ioctl communicates with userspace via a shared memory region. This ioctl returns the size of that shared memory region.

System call: mmap(NULL, vcpu_mmap_size, protection_flags, map_flag, vcpu_fd, 0)
Returns: Address of the mapped memory named as kvm_run.
Functionality: Maps the required memory for the vCPU shared memory kvm_run, allowing communication with the host

8) setup_long_mode by setting Special registers

API: ioctl(vcpu->vcpu_fd, KVM_GET_SREGS, &sregs)
Parameters: struct kvm_sregs (output)
Returns: 0 on success, -1 on error
Functionality: for retrieving vCPU segment registers.

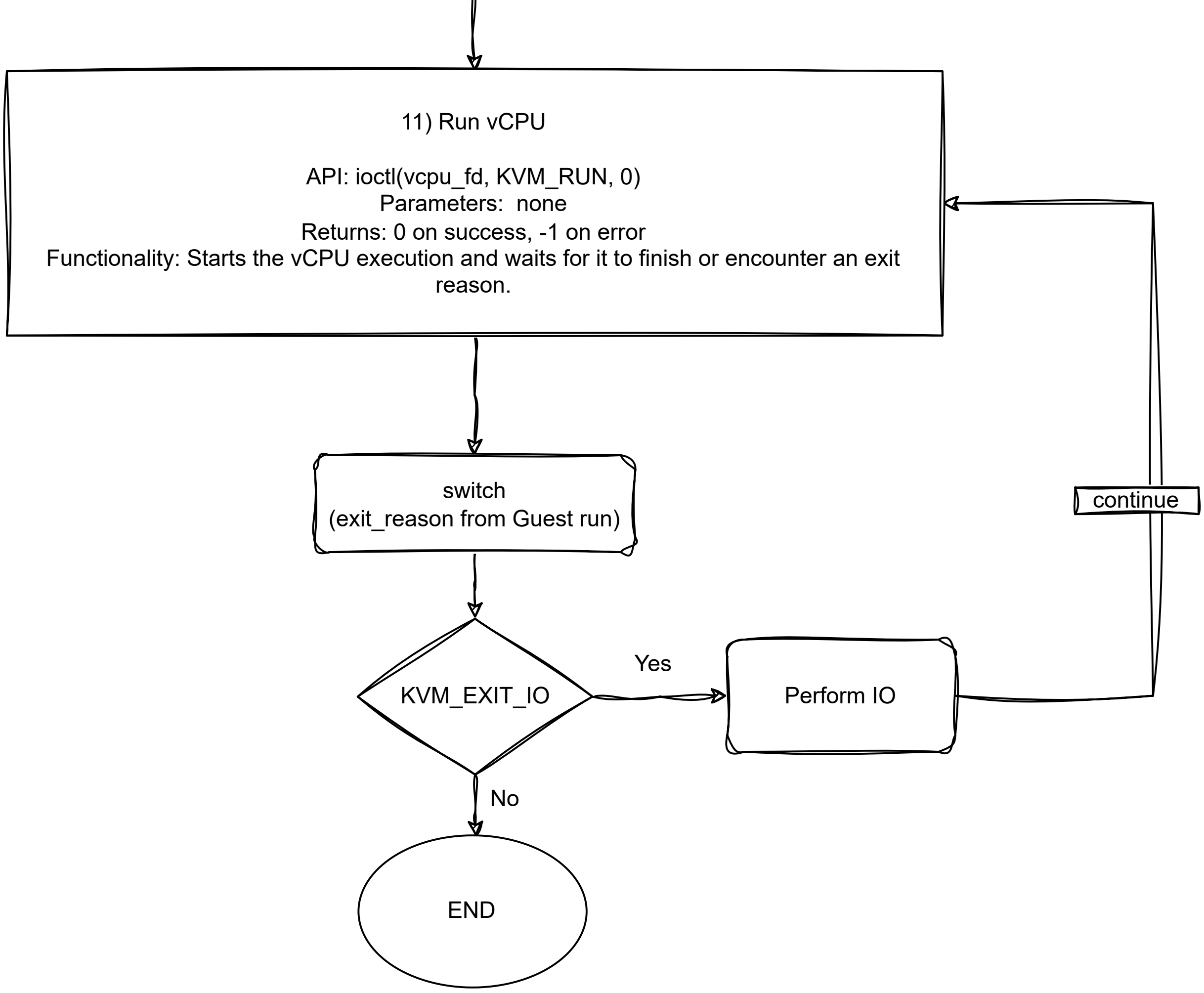
API: ioctl(vcpu->vcpu_fd, KVM_SET_SREGS, &sregs)
Parameters: struct kvm_sregs (input)
Returns: 0 on success, -1 on error
Functionality: Writes Special registers updated with proper long_mode specific values into the vcpu.
Special registers includes segment registers, control registers etc.

9) setup_long_mode by setting general purpose registers

API: ioctl(vcpu->vcpu_fd, KVM_SET_REGS, &regs)
Parameters: struct kvm_regs (input)
Returns: 0 on success, -1 on error
Functionality: Writes the general purpose registers including Stack pointer, Instruction pointer into the vcpu.

10) Load guest code

System call: memcpy(vm_mem, guest_code, size)
Functionality: Copies the guest code (e.g., guest64) into the allocated VM memory.


11) Run vCPU
API: ioctl(vcpu_fd, KVM_RUN, 0)
Parameters: none
Returns: 0 on success, -1 on error
Functionality: Starts the vCPU execution and waits for it to finish or encounter an exit reason.
switch
(exit_reason from Guest run)
KVM_EXIT_IO
Yes
Perform IO
continue
No
END