Topics in Virtualization and Cloud Computing (Assignment 3)

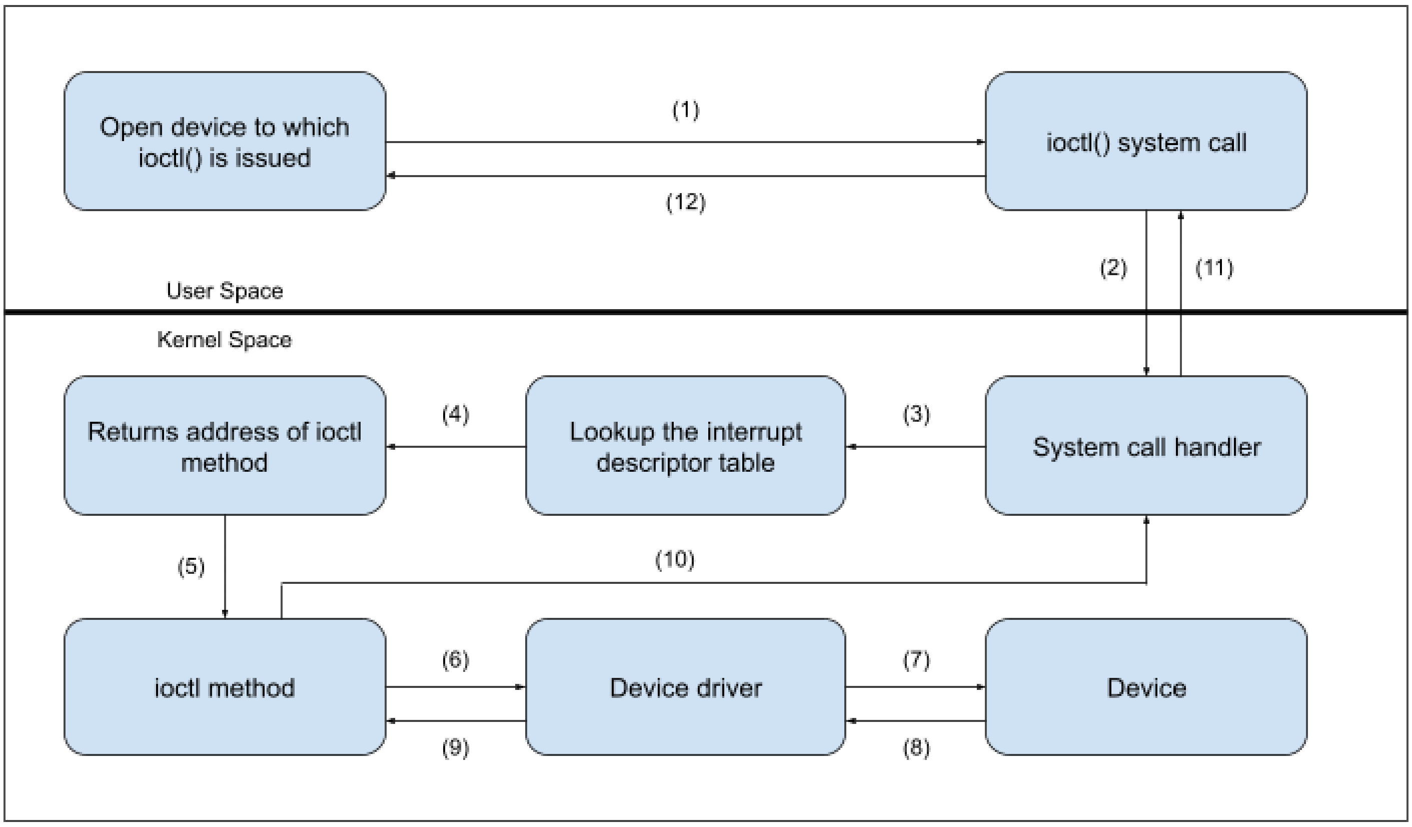
Shivansh Dhiman (22m2120)

PART 1

Question 1

a. There are 3 arguments of the `ioctl` system call in newer Linux kernels.
   i. First is `**fd**`, the file descriptor of the target device. This is used to tell which IO device should the command be executed on.
   ii. Second is `**cmd**`, the command in the form of a number that is defined in the device driver. IOCTL identifies the appropriate command using this number.
   iii. Third is `**arg**`, the argument which is just the memory pointer to the argument structure where all the arguments for the command are stored.

b. The steps are displayed in the following diagram.



Question 2

a. After opening the KVM device the following API calls are made:
   i. **KVM_GET_API_VERSION**: Obtain the KVM API version
   ii. **KVM_CREATE_VM**: Create a Virtual Machine
   iii. **KVM_SET_TSS_ADDR**: Allocate 3-page memory for the Task Set Segment
   iv. **KVM_SET_USER_MEMORY_REGION**: Allocate memory to the VM
   v. **KVM_CREATE_VCPU**: Create Virtual CPUs

   vi. **KVM_GET_VCPU_MMAP_SIZE**: Returns the size of shared memory used to communicate with the VM
   vii. **KVM_RUN**: Start the virtual machine

b. These are all the KVM API commands used:
   i. **ioctl(vm->sys_fd, KVM_GET_API_VERSION, 0):** Used this to check the API version of KVM. If it is not a positive integer, the program will stop.
   ii. **ioctl(vm->sys_fd, KVM_CREATE_VM, 0):** Create an empty VM with no virtual CPUs and no memory.
   iii. **ioctl(vm->fd, KVM_SET_TSS_ADDR, 0xfffbd000):** Used to allocate a memory of 3 pages for storing the Task Set Segment in the guest physical address space.
   iv. **ioctl(vm->fd, KVM_SET_USER_MEMORY_REGION, &memreg):** Create a guest physical memory slot of size 2MB. Size of memory is passed through `memreg` argument.
   v. **ioctl(vm->fd, KVM_CREATE_VCPU, 0):** Create one virtual CPU with id 0.
   vi. **ioctl(vm->sys_fd, KVM_GET_VCPU_MMAP_SIZE, 0):** Returns the size of the shared memory region used to communicate with the guest.
   vii. **ioctl(vcpu->fd, KVM_RUN, 0):** Run a guest virtual CPU.
   viii. **ioctl(vcpu->fd, KVM_GET_REGS, &regs):** Read the general-purpose registers.
   ix. **ioctl(vcpu->fd, KVM_SET_REGS, &regs):** Write to the general-purpose registers.
   x. **ioctl(vcpu->fd, KVM_GET_SREGS, &sregs):** Read the special registers of the virtual CPU.
   xi. **ioctl(vcpu->fd, KVM_SET_SREGS, &sregs):** Write to the special registers of the virtual CPU.

Question 3

a. GVA is converted to HVA in the following way:
   i. **Real mode:** GVA is directly taken as HVA as there is no page table present. This HVA is converted to HPA using the host's page tables.
   ii. **Protected mode:** GVA is directly taken as HVA as there is no page table present. This HVA is converted to HPA using the host's page tables.
   iii. **Paged 32-bit mode:** Here, a single level page directory is set up which addresses one 4MB page. When a GVA has to be translated, it is converted to HVA using the guest’s page table and then converted to HPA by the host. This larger page size is enabled by setting the PSE bit in the CR4 register.
   iv. **Long mode:** In this mode, there are 3 levels of page tables, namely PML4, PDPT, and PD. When a GVA arrives, it is looked up in the first page of PML4 (Page Map Level 4) page. Then, PDPT (Page Directory Pointers Table) is looked up followed by final PD (Page Directory). THe address resulting from this is the HVA. This address is then converted to HPA using the host's page tables.

b. **2MB** memory is allocated to the guest running in Long mode.

c. **`sregs->cr3 = pml4_addr;`** This line sets up the guest page table. Here the address of the top-level page directory is stored in the CR3 register.

d. **CR4_PAE** allows setting up a 3-level page table with table entries of 64-bit instead of 32-bit. This allows for more room for address bits.

Question 4

a. The guest starts its execution at **address 0x0**. This is configured in the RIP register (Instruction pointer for 64-bit processes) of the guest.

b. **`if (ioctl(vcpu->fd, KVM_RUN, 0) < 0) {`** This line starts the guest execution. The KVM API that is in use for running the guest is KVM_RUN.

Question 5

a. `**outb**` function is an inline assembly instruction that takes two arguments. `a` takes a value and stores it in the EAX register. `Nd` takes a port number and stores it in the EDX register. `outb` function essentially sends the value to the desired serial port. The guest uses a function `**inb**` that is used to read values from this serial port in a similar way.
b. The hypervisor checks the reason for VM_EXIT. In the case of transferring data through serial ports, **KVM_EXIT_IO** is the reason. The hypervisor then checks the IO direction and port and reads the value sent by the guest from this serial port.
c. For sanity check, when the **VM_HALT** occurs, the guest stored a value of 42 in the **shared memory buffer**. The hypervisor accesses the memory and reads the value. If this value doesn't match the value in the EAX register, the hypervisor knows something is incorrect.