How can multiple processes run on a single system (Basis of virtualization)?

- Time-sharing: Allow every process to run for a brief period of time on a CPU and then execute the next process (This is called as round-robin approach)

Main goal while running an application: High performance

How is performance achieved?

- Direct execution: Allowing the application code to run directly on the CPU. OS creates process and then transfers control to main() of the program.

What is the problem with direct execution and what is the solution?

- You can access all the resources. e.g. You could just issue I/O request to a disk and read the file for which you don’t have access.
- Limited direct execution: Split instruction as privileged instructions and normal instructions. The privilege instructions (e.g. to access device) could be executed only in the kernel (privileged) mode. The OS runs in the kernel mode. Normal instructions could be run by processes in user (regular) mode and privileged instructions cannot be executed while in this mode.

How can processes access devices?

- System calls: its like a function call made by the program except that call enters the kernel mode completes running the privileged instructions (e.g. contacting disk and fetching data) and then comes back to the user mode.
- System calls are implemented by the OS (Applications/Programs cannot modify them)
  - <Learn about the Trap>

What happens when a system call is made by the process?

- Executes special instruction (int 0x80 in intel machines) - This generates an interrupt to the processor
- Processor is changed to privilege mode
- Process’s user-mode stack is switched to kernel stack (done by hardware)
- A trap frame is generated by the hardware that saves the program's state (like stack, instruction pointer, registers) and the trap frame is pushed onto the kernel stack
- Trap handler is invoked which then invokes the right system call function (like open or read or write or getpid)
- Reverse operation happens during the return

What are interrupts?

- Interrupt is special message sent by devices to the processor
- Every interrupt has its associated interrupt handler that is nothing but a code function
- When devices are plugged into the system, OS assigns a unique interrupt number to that particular device
- When device wishes to communicate with the OS, it sends an interrupt with this interrupt number
• OS maintains a table called as interrupt table that is basically an array of interrupt handlers
• On receiving the interrupt, OS fetches the interrupt number and indexes into the interrupt table to get the interrupt handler which is then executed

How can process switching be done?
• cooperative
• non-cooperative

How is non-cooperative control achieved?
• Use timer to generate interrupts at periodic intervals and at each interval switch to a new process from the currently running process. Whether to switch is a decision made by an OS component called **scheduler**.

What is context switch and what happens during a context switch?

Questions
• Assume there are multiple cpus in the system, what would happen if space-sharing is enforced?
• What are the other alternatives to direct execution?
• When you write a normal C program or Java program, you don't invoke the system calls directly. Then how are they getting invoked?
• What are the pros/cons of cooperative and non-cooperative approach in process switching?
• Can you achieve non-cooperative control without using a timer?
• Context switch is a mechanism or policy?
• Why do you need a separate kernel stack and user-mode stack?