# Question 1: Process Switching

Pre-emptive multitasking in a modern operating system involves the cooperation of both the CPU hardware and the operating system kernel to efficiently manage multiple processes. Let's address each of your questions step by step:

1. Following a Clock Interrupt:

* CPU Hardware: When a clock interrupt occurs, the CPU hardware performs the following actions:
  + Saves the current state of the running process (Process A) by saving its registers, program counter, and other relevant information into its Process Control Block (PCB).
  + Switches the CPU execution mode to kernel mode, which allows it to execute privileged instructions.
  + Jumps to a predefined interrupt handler routine in the operating system kernel.
* Operating System Kernel: Upon receiving the clock interrupt, the kernel performs several tasks:
  + Determines whether it's time to perform a context switch, i.e., whether Process A has run for its time slice (time quantum). If so, it chooses the next process to run based on a scheduling algorithm and updates the process control block accordingly.
  + If Process A still has time remaining in its time slice, the kernel simply updates its timer and returns control to it.
  + If a context switch is required, the kernel updates the CPU's memory management unit to load the memory space of the selected process.
  + Restores the saved state of the chosen process (Process B or C) from its PCB.
  + Sets the CPU's program counter to the saved value for the chosen process.
  + Resets the CPU execution mode to user mode, allowing the selected process to run.

1. When Process A Needs to Read a Word from the Disk:

* Process A: When Process A initiates a disk read request, it enters a blocked state and yields the CPU.
* CPU Hardware: The CPU hardware, upon recognizing that Process A is blocked and cannot continue executing, generates an interrupt known as a "trap" or "system call" to transfer control to the operating system kernel.
* Operating System Kernel: Upon receiving the disk read request interrupt, the kernel performs the following actions:
  + Suspends Process A and places it in the blocked state, typically in a queue or a list of processes waiting for I/O operations.
  + Initiates the disk read operation, which may involve scheduling the operation, configuring the hardware controller, and issuing the appropriate commands to the disk subsystem.
  + The kernel then allows another process, such as Process B or C, to run while Process A is waiting for the disk operation to complete.

1. When the Read Disk Operation for Process C Completes:

* Disk Hardware: The disk hardware performs the requested read operation and signals its completion to the operating system.
* Operating System Kernel: Upon receiving the completion signal, the kernel performs the following actions:
  + Updates the status of Process C from a blocked state to a ready state, indicating that it is now ready to run.
  + If Process C has the highest priority or is the next process to run according to the scheduling algorithm, the kernel may choose to schedule it immediately.
  + If Process C is not immediately scheduled, the kernel may continue running the currently running process (e.g., Process B) or select another process for execution based on its scheduling policy.
  + The kernel handles the context switch if necessary and updates the CPU hardware to run the selected process.

In summary, pre-emptive multitasking involves a seamless cooperation between the CPU hardware and the operating system kernel. The hardware generates interrupts (e.g., clock interrupts, I/O interrupts) to trigger kernel intervention, and the kernel manages the scheduling of processes, context switches, and I/O operations to ensure efficient and fair execution of multiple processes. This coordination allows for the illusion of concurrent execution of multiple processes on a single CPU core.

Top of Form

# Question 2: The Memory Layout of a Process

# Question 3: Executing Commands in Child Processes