# Section 3: Process Description and Control

## Question 1

1. What is the purpose of the Process Control Block?
2. Describe the three categories of information present in the process control block as shown in Figure 3.13 in the text book.
3. What is being represented in Figure 3.14?

### Answer

1. The process control block contains the information needed by the operating system to maintain and schedule the process.
2. A) Process Identification Information contains information that identified the process. For example the PID, PPID, UID, and GID.   
   B) **Processor** State Information contains information (PC, SP, registers, etc.) that describes the processor state when the processor was last interrupted and is needed to restore the process back into a running state.   
   C) Process Control Information contains the information used by the OS to schedule and manage the execution of individual processes. For example, process state, priority, owned resources (file & socket descriptors), accounting information (CPU & wall clock times).
3. Figure 3.14 describes how the operating system maintains a number of queues and collections that reference lists of PCB. In the figure, an operating system data structure maintains references to processes that are in the Running, Ready, or Blocked state.

## Question 2

1. Explain the meaning of a process’s instruction trace.
2. Explain the instruction trace presented in Figure 3.4 in the book. What is meant by Time-Out and by I/O Request?
3. Explain the role of the dispatcher service in all of this. How is the dispatcher represented in Figure 3.4?

### Answer

1. The instruction trace is a list (or trace) of the instruction address that a process executes over some given time range.
2. Figure 3.4 illustrates the execution of three processes interleaved with the dispatcher’s execution. The execution of a process divided into slices and every eligible process is to be given a turn executing. The process’s execution can be interrupted for two reasons in this example. Either the process executes a Blocking I/O Request (Blocking SYSCALL) or it executes for the full duration of its time-slice and is preempted by time-out interrupt.
3. The dispatcher is the OS mechanism that selects, installs, and allows to execute a “ready to execute” process from the Read Process Queue. The dispatcher’s execution is represented by the blue shaded traces which execute the same instructions each time.

## Question 3

1. What is the Process Image?
2. What two places does a process image reside? Hint: Suspended Process.
3. What are the five generic steps involved with building a new process image?

### Answer

1. The process image is the region of memory (disk or main memory) containing the instructions / data / stack of the program being executed. Note that in some models the image may contain also contain the Process Control Block.
2. The process image resides in both the system drive and main memory. The image is initially created on the drive and copied into main memory for execution. A suspended process is one which has been removed from main memory but still resides on the disk so that it can later be restored to main memory for further execution.
3. The five generic steps are:
   1. Allocate space for the new image on the system (swap) drive.
   2. Assign a unique process ID (PID) to the new process.
   3. Create the Process Control Block and other data structures needed by the OS to maintain the process’s execution.
   4. Copy the image from the system drive into main memory.
   5. Mark the new process Ready and place in the Ready Queue.

## Question 4

Using Figure 3.6, describe the process states and conditions that cause a process to transition from one state to another i.e. 1) The transition from New to Ready. 2) The transition from Ready to Running. 3), 4), etc.

### Answer

New to Ready: When a new process has been fully installed by the OS i.e. its Process Control Block has been created, its process image has been created, and it is ready to be installed in the ready queue for its turn at execution.

Ready to Running: The dispatcher selects a ready process from the Ready Process FIFO. The process’s Processor Context is installed on the processor and the process begins executing its current slice.

Running to Ready: The process has completed it slice (time-out). Is processor context is saved and it is moved to the back of the FIFO waiting for its turn to be re-scheduled.

Running to Exit: The process has been released by the OS for one of several reasons given in the book and slides. The process control block is release as is all system resources allocated to the process.

Running to Blocked: The process has executed a blocking system call (e.g. I/O request) and its execution is blocked while it waits for the associated event (interrupt) signaling that the request has completed and the process can resume execution.

Blocked to Ready: The blocked process’s event has occurred and it is once again eligible for execution.

## Question 5

Describe the difference between the monolithic kernel architecture and the process-based kernel architecture.

Include a description about how user processes access kernel services in each model.

### Answer

The monolithic kernel architecture implements the kernel as a single (monolithic) block of executable instructions that is loaded and executed at system boot. The kernel instructions provide the facilities needed to create and maintain processes and the syscall instructions. This includes syscall mechanism used to transfer control from a process to code residing in the kernel program.

The process-based kernel architecture (also known as a Microkernel architecture) implements its services as ‘kernel processes’ (see Slide 42). Both user programs and kernel services are implemented as individual processes. Processes (user and kernel) are interconnected by a relatively simple service layer that provides process management (creation and switching) and a fast Inter-Process Communication. The later allows user processes to make service requests (syscalls) to kernel processes.

## Question 6

Describe the differences between the user control stack and a kernel control stack.

Describe the reasons for having both a user control stack and a kernel control stack.

### Answer

The user stack is the control stack in user space (memory) that is used by the user process to pass function (subroutine) arguments and results made by the process when running in user mode.

The kernel stack is a separate control stack that is maintained in kernel space. It serves the same purpose as the user stack except that 1) the kernel stack in used to facilitate subroutine call made during the kernel’s execution 2) because the kernel stack exists in kernel space, the kernel stack cannot be manipulated modified by a user process.

An operating system needs two stacks to enforce security between the user and kernel spaces i.e. so that the user space process cannot access kernel data or corrupt the execution of the kernel by manipulating arguments and return addresses pushed on the kernel stack by kernel instructions.

## Question 7

1. Describe what it means to “Suspend a Process”.
2. What is the benefit of suspending a process?

### Answer

1. A suspended process is one in which the contents of its image in physical memory (text, data, & stack) is copied onto the system drive and removed from memory. Because the image is maintained on the system drive, it can later be restored to main memory and its execution continued.
2. The benefit of suspending a process is that the main memory occupied by the suspended process is freed for other READY processes to be installed and scheduled for execution.

## Question 8

1. What are the two system calls used to create and launch a new executable process in the Linux (Unix) operating system.
2. Describe how a parent process uses these two calls to create a new child process from a given executable file.
3. What is meant by copy on write?

You can use the book “Linux Kernel Development”, Section 3 found in the Supplemental Materials eLearning folder.

### Answer

The two system calls are fork() and exec().

1. The fork() system call logically duplicates the process image that makes the call. The new process is a logical clone of the original with the exception of the process ID and parent process ID.
2. The exec(‘exec\_file’) system call replaces the process image of the process making the call with a new image obtained from an executable file that was passed to the syscall.

So creating a new executable process involves 1) having the parent process call fork() to create a new distinct child process and 2) having the child process call exec() to replace the parent’s image with the image of the desired executable.

Copy on Write is the technique the Linux kernel uses to avoid physically copying the entire parent process image into the new child process. This copy would be especially wasteful considering that under most circumstances the child process is going to be immediately replaced with the new image of the desired executable.

## Question 9

Does every mode switch (i.e. system call) result in a blocked process? Explain why or why not.

### Answer

No. The process may switch to kernel mode in response to a system call that involves no I/O. For example a call to malloc() or free() to allocated or free memory can be executed by the process causing a mode switch. However, these calls only update internal OS tables and so control can be returned to the calling process.

## Question 10

Describe the four mechanisms that are used to preempt a process’s execution. Two of the four mechanisms are derived from interrupts (two types of hardware interrupts) and the two others as presented in the slides.

### Answer

The four mechanisms are:

1. Controller Interrupt: A hardware signal from a system controller that signals that the controller will soon need servicing.
2. Timer Interrupt: The interrupt generated by the hardware timer that interrupts the currently executing process so that the dispatcher can gain control and perform a context switch. Although the timer interrupt basically a hardware interrupt, its purpose is fundamentally different from a the interrupts generated by system controllers.
3. Signals: Also known as Traps. These are events generated by the OS that ‘signal’ the occurrence of something of interest. For example, division by zero errors or memory bounds violations are examples of signals that would cause the termination of the executing process.
4. Blocking System Calls (SYSCALL): A request by the process to the OS for some service. Implemented by a software interrupt instruction (INT).

## Also know for exam:

* The meaning and implementation of “Context Switch” in the context of the OS switching from the execution of one process to another.