## Worksheet 3.2: Processes, Multiple-Choice Questions

**In the questions below, circle the right answer. There is only one correct answer.**

**1.** Which of the following events will cause an OS to put a process in the waiting state?
a. I/O completion b. I/O request
c. Expiration of the time quantum d. Execution of an invalid operation
e. Both a and b are correct. f. Both b and c are correct. g. Both b and d are correct.

Explanation: A process moves to the waiting state when it makes an I/O request, as it must wait for the I/O operation to complete. I/O completion moves a process from waiting to ready, not to waiting.

**2.** Which of the following is (are) true about context switching (CS)?
a. In CS, the OS copies from CPU registers to memory.
b. In CS, the OS copies from memory to CPU registers.
c. CS time does not depend on the hardware design.
d. During CS, the kernel is not necessarily running on the CPU.
e. Both a and c are correct. f. Both b and d are correct. g. Both a and b are correct.

Explanation: Context switching involves saving the current state of a process (copying from CPU registers to memory) and restoring the state of the next process to run (copying from memory to CPU registers).

**3.** What's the purpose of storing the values of the CPU registers in the Process Control Block (PCB)?
a. To have a backup copy of the registers in case the process crashes.
b. To improve the performance of the process, because registers are the fastest storage devices.
c. To restore them when the process gets the CPU again so that the process picks up from where it left off.
d. To give the OS easy access to CPU registers.
e. Both b and c are true. f. Both a and d are true. g. Both c and d are true.

Explanation: The primary purpose of storing CPU register values in the PCB is to ensure that a process can be resumed correctly after it has been switched out of the CPU.

**4.** When the kernel performs context switching from $P_1$ to $P_2$, in which order will it perform the following steps?
a. Saving $P_1$ state, restoring $P_2$ state, updating the ready queue, and finally invoking the scheduler.
b. Saving $P_1$ state, updating the ready queue, restoring $P_2$ state, and finally invoking the scheduler.
c. Saving $P_1$ state, updating the ready queue, invoking the scheduler, and finally restoring $P_2$ state.
d. Updating the ready queue, invoking the scheduler, saving $P_1$ state, and finally restoring $P_2$ state.
e. Updating the ready queue, saving $P_1$ state, invoking the scheduler, and finally restoring $P_2$ state.
f. Different operating systems perform these steps in different orders.
g. These steps can be performed in any order.

Explanation: Typically, the OS saves the current process state, updates the ready queue, invokes the scheduler to determine the next process, and then restores the next process's state.

**5.** What is the difference between an I/O-bound process and a CPU-bound process?
a. A CPU-bound process never requests I/O.
b. An I/O-bound process has longer CPU bursts than a CPU-bound process.
c. A CPU-bound process has longer CPU bursts than an I/O-bound process.
d. A CPU-bound process has a larger number of CPU bursts than an I/O-bound process.
e. Both c and d are correct. f. Both b and d are correct. g. Both a and c are correct.

Explanation: A CPU-bound process spends more time doing computations and has longer CPU bursts, while an I/O-bound process spends more time waiting for I/O operations to complete.

**6.** Which of the following is true about process creation using the fork() system call?
a. The child will have the same address space as the parent.
b. The child will run the same code as the parent.
c. The parent will automatically wait for the child.
d. Both the parent and the child will be running on the same CPU at the same time.
e. Both b and c are correct. f. Both a and b are correct. g. a, b and c are correct

Explanation: The fork() system call creates a new process (the child) that is a copy of the parent process. The child has its own address space but starts execution at the same point in the code as the parent. The parent does not automatically wait for the child; this must be done explicitly using a separate system call like wait().