Please do not read the questions until your class starts, and do not share this document.

There will be no marks for answering these questions correctly. On the other hand, if we find that you knew the questions and/or the answers before your class you will receive zero participation marks for that tutorial.

In all questions below, you need to choose **one answer**.

Q1. (i) What is the meaning of the term busy waiting? (ii) How can the busy waiting be avoided?

A. (i) A task is kept in a waiting queue until an event occurs. (ii) Keep the task constantly checking whether the event has occurred.  
B. (i) Processes wait because the CPU is busy executing other tasks. (ii) Temporarily put the waiting process to sleep and awaken it when the CPU is available.  
C. (i) A process is waiting for a condition to be satisfied in a loop but without occupying the CPU. (ii) Put the task in a waiting queue and run it whenever the condition is satisfied.  
D. (i) A process is waiting for the user to press a key on the keyboard. b) Restart the process when the condition is satisfied to avoid busy waiting.  
E. (i) A process is waiting for a condition to be satisfied in a loop without relinquishing the CPU. b) Temporarily put the waiting process to sleep and awaken it when the appropriate program state is reached.

Q2. If the wait() and signal() semaphore operations are not executed atomically, under which circumstances the mutual exclusion may be violated? Consider processes P1 and P2, and assume that the critical section is between wait() and signal() operation for each process.

A. The initial semaphore value == 0, P1 performs wait() operation and then P2 performs signal() operation.  
B. The initial semaphore value == 1, P1 performs signal() operation and then P2 performs wait() operation.  
C. Two signal() operations are executed on a semaphore when its value == 1.  
D. Two wait() operations are executed on a semaphore when its value == 1.  
E. None of the above choices is correct.

Q3. What is the difference between signaled and non-signaled states with Windows dispatcher objects?

A. An object that is in the signaled state is available, and a thread will not block when it tries to acquire it. When the lock is acquired, it is in the non-signaled state.  
B. Signaled means the semaphore value is zero. Non-signaled means the semaphore value is greater than zero.  
C. An object that is in the signaled state is NOT available. An object is in non-signaled state when it is available.  
D. Signaled means the process acquiring the object is in a running state. Non-signaled means the process is in waiting state.  
E. An object that is in the signaled state is it's a locked mutex. An object is in non-signaled state when it is an unlocked mutex.

Q4.  Which of the following is NOT a deadlocks example?

A. A thread is waiting to enter a critical section that is currently occupied by another thread. The critical section is protected by a mutex lock.  
B. Two cars crossing a single-lane bridge from opposite directions.  
C. A person going down a ladder while another person is climbing up the ladder.  
D. Two trains travelling toward each other on the same track.  
E. Two processes are waiting indefinitely for an event that can be caused only by the other process.

Q5. In which situation data consistency is NOT a concern?

A. Two processes access shared variables a and b. Process P1 reads values from a and b, and process P2 writes values to b only.  
B. Two processes access shared variables a and b. Process P1 reads values from a and b, and process P2 reads values from b only.  
C. Two processes try to call an add() function to add 100 to a shared variable x, if and only if the variable x == 1.  
D. A husband and a wife accessing the same bank account from different devices.  
E. Students write their answers to a shared document

Q6. We have the following algorithmic solution for the critical section problem. Which one of the following statements is correct?

P\_i

while (true) {  
 while (lock) { }; (1)  
 lock = true; (2)  
 critical section (3)  
 lock = false; (4)  
}

A. Using this solution, two processes will always end up in the critical section at the same time.  
B. Using this solution, two processes will never end up in the critical section at the same time.  
C. Using this solution, if two processes arrive in the order of Pa then Pb, Pa would always access the critical section earlier than Pb.  
D. Using the solution, the CPU is utilized in the most efficient way  
E. There is no race condition by using this solution if the lock is an atomic variable.