Chapter 6

Multiple Choice Questions

1. Assume count is a variable name, which of the following operations is atomic?
A) count++
B) count--
C) both of the above
D) none of the above

Ans: D
Feedback: 6.1
Difficulty: Easy

2. Which of the following is true for race condition?
A) race condition occurs where several processes access and manipulate the same data concurrently
B) when race condition occurs, the outcome of the execution depends on the particular order in which the access takes place
C) both of the above
D) none of the above

Ans: C
Feedback: 6.1
Difficulty: Easy

3. Which of the following actions should be performed among cooperating processes?
A) process synchronization
B) coordination
C) both of the above
D) none of the above

Ans: C
Feedback: 6.1
Difficulty: Easy

4. In _____, the process may be changing common variables, updating a table, writing a file, and so on.
A) entry section
B) critical section
C) exit section
D) remainder section

Ans: B
Feedback: 6.2
Difficulty: Easy

5. In _____, the process requests permission to access and modify variables shared with others.
A) entry section
B) critical section
C) exit section
D) remainder section

Ans: A
Feedback: 6.2
Difficulty: Easy

6. Which of the following critical-section problem's requirements ensures only one process is active in its critical section at a time?
A) mutual exclusion
B) progress
C) bounded waiting
D) none of the above

Ans: A
Feedback: 6.2
Difficulty: Easy

7. Which of the following critical-section problem's requirements ensures programs will cooperatively determine what process will next enter its critical section?
A) mutual exclusion
B) progress
C) bounded waiting
D) none of the above

Ans: B
Feedback: 6.2
Difficulty: Easy

8. Which of the following critical-section problem's requirements limits the amount of time a program will wait before it can enter its critical section?
A) mutual exclusion
B) progress
C) bounded waiting
D) none of the above

Ans: C
Feedback: 6.2
Difficulty: Easy

9. Which of the following is true regarding the requirements for the solutions to critical-section problem?
A) mutual exclusion implies progress
B) progress implies bounded waiting
C) bounded waiting implies progress
D) none of the above

Ans: C
Feedback: 6.2
Difficulty: Medium

10. Which of the following is true for the solutions to critical-section problems?
A) No deadlock implies progress, and progress implies bounded waiting
B) Bounded waiting implies progress, and progress implies no deadlock
C) Progress implies no deadlock, and no deadlock implies bounded waiting
D) Bounded waiting implies no deadlock, and no deadlock implies progress

Ans: B
Feedback: 6.2
Difficulty: Medium

11. Which of the following is NOT true for Peterson's solution?
A) Mutual exclusion is preserved
B) The progress requirement is satisfied
C) The bounded-waiting requirement is met
D) Peterson's solution works for synchronization among more than two processes

Ans: D
Feedback: 6.3
Difficulty: Easy

12. Which of the following variables are shared between the processes in Peterson's solution?
A) int turn
B) boolean flag[2]
C) both of the above
D) none of the above

Ans: C
Feedback: 6.3
Difficulty: Medium

13. Which of the following indicates that P_i can enter the critical section in Peterson's solution?
A) flag[j] == false or turn == i
B) flag[j] == true or turn == i
C) flag[j] == false or turn == j
D) flag[j] == true and turn == j

Ans: A
Feedback: 6.3
Difficulty: Medium

14. Which of the following is a software-based solution to the critical-section problem?
A) Peterson's solution
B) test_and_set
C) compare_and_swap
D) all of the above

Ans: A
Feedback: 6.3
Difficulty: Easy

15. Which of the following solutions needs hardware support for the critical section problem?
A) memory barriers
B) compare_and_swap instruction
C) atomic variables
D) all of the above

Ans: D
Feedback: 6.4
Difficulty: Easy

16. Which of the following is not true about *test_and_set* instruction?
A) It is a hardware instruction
B) It is executed atomically
C) Returns the original value of passed parameter
D) Returns the new value of passed parameter
E) Set the new value of passed parameter to "TRUE"

Ans: D
Feedback: 6.4.2
Difficulty: Easy

17. Which of the following is not true about *compare_and_swap* instruction?
A) It is a hardware instruction
B) It is executed atomically
N) Returns the original value of passed parameter
D) Set the new value of passed parameter to "TRUE"
E) Set the variable "value" the value of the passed parameter "new_value" but only if "value" =="expected"

Ans: D
Feedback: 6.4.2
Difficulty: Easy

18. Assume the binary variable *lock* is initialized to be 0, which of the following can be an implementation of the entry section to solve the critical-section problem?
A) while (compare and swap(&lock, 0, 1) != 0), do nothing;
B) while (test and set(&lock)), do nothing;
C) both A and B
D) none of the above

Ans: C
Feedback: 6.4.2
Difficulty: Easy

19. Which of the following regarding mutex lock is NOT true?
A) mutex lock is a hardware solution to critical-section problem
B) mutex lock is a higher-level software solution to critical-section problem
C) mutex lock suffers from busy waiting
D) the general rule of thumb is to use a mutex lock if the lock will be held for a duration less than two context switches

Ans: A
Feedback: 6.5
Difficulty: Easy

20. When mutex lock is implemented as a binary semaphore, what should its value be initialized to be?
A) 0
B) 1
C) -1
D) none of the above

Ans: B
Feedback: 6.6
Difficulty: Easy

21. The counting semaphore is initialized to _____.
A) 0
B) 1
C) the number of resources available
D) none of the above

Ans: C
Feedback: 6.6
Difficulty: Easy

22. Which of the following is NOT true regarding semaphore implementation?
A) It suffers from the busy waiting problem
B) Semaphore has a waiting queue associated with the semaphore
C) When a process executes the *wait()* operation and finds that the semaphore value is not positive, it will suspend itself
D) A process that is suspended, waiting on the semaphore, should be restarted when some other process executes a signal() operation.

Ans: A
Feedback: 6.6
Difficulty: Medium

23. What is the correct order of operations for protecting a critical section using a binary semaphore?
A) release() followed by acquire()
B) acquire() followed by release()
C) wait() followed by signal()
D) signal() followed by wait()

Ans: C
Feedback: 6.6
Difficulty: Easy

24. Which of the following statements is true?
A) A counting semaphore can never be used as a binary semaphore.
B) A binary semaphore can never be used as a counting semaphore.
C) Spinlocks can be used to prevent busy waiting in the implementation of semaphore.
D) Counting semaphores can be used to control access to a resource with a finite number of instances.

Ans: C
Feedback: 6.6
Difficulty: Medium

25. Which of the following is NOT true?
A) Since semaphore and mutex lock are tools for synchronization, process that have used semaphores or mutex locks should not cause deadlocks
B) Semaphores and mutex locks may be shared resources that difference processes contend for, and hence deadlocks may occur
C) a set of processes is in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set
D) all of the above

Ans: A
Feedback: 6.6
Difficulty: Easy

26. Which of the following is NOT true regarding conditional variable, e.g. *x*?
A) The only operations that can be invoked on a condition variable are *wait()* and *signal()*
B) *x.wait()* means that the process invoking this operation is suspended until another process invokes *x.signal()*
C) The *x.signal()* operation resumes exactly one suspended process
D) If no process is suspended, then the signal() operation still affects the state of the semaphore

Ans: D
Feedback: 6.7.1
Difficulty: Medium

27. Which of the following may cause a liveness failure?
A) an infinite loop
B) a busy waiting loop
C) a deadlock
D) all of the above

Ans: D
Feedback: 6.8
Difficulty: Easy

28. Which of the following is true?
A) No deadlock implies no starvation;
B) No starvation implies no deadlock;
C) Deadlock doesn't imply starvation;
D) Starvation implies deadlock.

Ans: C
Feedback: 6.8.1
Difficulty: Medium

29. Under which of the following contention loads does traditional synchronization become faster than CAS-based synchronization?
A) uncontended
B) moderate contention
C) high contention
D) none of the above

Ans: C
Feedback: 6.9
Difficulty: Easy

30. Which of the following are efforts to towards developing scalable, efficient tools that address the demands of concurrent programming?
A) designing compilers that generate more efficient code
B) developing languages that provide support for concurrent programming
C) improving the performance of existing libraries and APIs
D) all of the above

Ans: D
Feedback: 6.9
Difficulty: Easy

Essay Questions

1. Explain what race condition is.
Ans: A situation where several processes access and manipulate the same data concurrently and the outcome of the execution depends on the particular order in which the access takes place, is called a race condition.
Feedback: 6.1
Difficulty: Easy

2. Explain what critical section problem is.
Ans: A critical section is a section of code where shared data may be manipulated and a possible race condition may occur. The critical section problem is to design a protocol whereby processes can synchronize their activity to cooperatively share data.
Feedback: 6.2
Difficulty: Easy

3. What three conditions must be satisfied in order to solve the critical section problem?

Ans: In a solution to the critical section problem, no thread may be executing in its critical section if a thread is currently executing in its critical section. Furthermore, only those threads that are not executing in their critical sections can participate in the decision on which process will enter its critical section next. Finally, a bound must exist on the number of times that other threads are allowed to enter their critical state after a thread has made a request to enter its critical state.
Feedback: 6.2
Difficulty: Medium

4. Explain two general approaches to handle critical sections in operating systems.

Ans: Critical sections may use preemptive or nonpreemptive kernels. A preemptive kernel allows a process to be preempted while it is running in kernel mode. A nonpreemptive kernel does not allow a process running in kernel mode to be preempted; a kernel-mode process will run until it exits kernel mode, blocks, or voluntarily yields control of the CPU. A nonpreemptive kernel is essentially free from race conditions on kernel data structures, as the contents of this register will be saved and restored by the interrupt handler.
Feedback: 6.2
Difficulty: Medium

5. Assume you had a function named update() that updates shared data. Illustrate how a mutex lock named mutex might be used to prevent a race condition in update().

Ans:

```
void update()
{
        mutex.acquire();

        // update shared data

        mutex.release();
}
```

Feedback: 6.5
Difficulty: Easy

6. Explain what busy waiting is.
Ans: While a process is in its critical section, any other process that tries to enter its critical section must loop continuously in the call to *acquire()*.This continual looping is clearly a problem in a real multiprogramming system, where a single CPU is shared among many processes. Busy waiting wastes CPU cycles that some other process might be able to use productively.
Feedback: 6.5
Difficulty: Easy

7. What is meant by “short duration”?
Ans: Given that waiting on a lock requires two context switches: a context switch to move the thread to the waiting state, and a second context switch to restore the waiting thread once the lock becomes available, the general rule of thumb is to use a spinlock if the lock will be held for a duration less than two context switches.
Feedback: 6.5
Difficulty: Easy

8. What is the difference between semaphore and mutex lock?
Ans: Semaphores are similar to mutex lock in that they can be used to provide mutual exclusion. However whereas a mutex lock has a binary value which indicates if the lock is available or not, a semaphore has an integer value, and can therefore be used to solve a variety of synchronization problems.
Feedback: 6.6
Difficulty: Medium

9. What is the difference between counting semaphore and binary semaphore.
Ans: The value of a counting semaphore can range over an unrestricted domain. The value of a binary semaphore can range only between 0 and 1.
Feedback: 6.6
Difficulty: Easy

10. Explain how semaphore implementation overcomes the busy waiting problem of mutex lock.
Ans: When a process executes the wait() operation and finds that the semaphore value is not positive, it must wait. However, rather than engaging in busy waiting, the process can suspend itself. The suspend operation places a process into a waiting queue associated with the semaphore, and the state of the process is switched to the waiting state. Then control is transferred to the CPU scheduler, which selects another process to execute.
Feedback: 6.6
Difficulty: Medium

11. Write two short methods that implement the simple semaphore wait() and signal() operations on global variable S.

Ans:
```
wait (S) {
  while (S <= 0);
  S--;
}

signal (S) {
  S++;
}
```
Feedback: 6.6
Difficulty: Difficult

12. What is the difference between a semaphore and conditional variable?
Ans: If no process is suspended, then the signal() operation for conditional variable has no effect; that is, the state of the variable is the same as if the operation had never been executed. However,

the operation with the signal() operation associated with semaphores always affects the state of the semaphore.
Feedback: 6.7.1
Difficulty: Medium

13. Explain what a deadlock is.
Ans: We say that a set of processes is in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set.
Feedback: 6.8.1
Difficulty: Easy

True/False Questions

1. Race conditions can result in corrupted values of shared data.
Ans: True
Feedback: 6.1
Difficulty: Easy

2. Bounded waiting implies progress, and progress implies mutual exclusion.
Ans: True
Feedback: 6.2
Difficulty: Easy

3. The preemptive kernel is more suitable for real-time programming than non-preemptive kernel.
Ans: True
Feedback: 6.2
Difficulty: Easy

4. The preemptive kernel may be more responsive than non-preemptive kernel.
Ans: True
Feedback: 6.2
Difficulty: Easy

5. Peterson’s solution works on modern computer architectures.
Ans: False
Feedback: 6.3
Difficulty: Easy

6. A mutex lock is released immediately after entering a critical section.
Ans: False
Feedback: 6.5
Difficulty: Easy

7. Semaphores and mutex locks both provide mutual exclusion.
Ans: True
Feedback: 6.5 and 6.6
Difficulty: Easy

8. Spinlocks are not appropriate for single-processor systems.
Ans: True
Feedback: 6.5
Difficulty: Medium

9. Mutex lock variable is binary.
Ans: True
Feedback: 6.5
Difficulty: Easy

10. Semaphore implementation overcomes the busy waiting problem.
Ans: True
Feedback: 6.6
Difficulty: Easy

11. The value of a counting semaphore can range only between 0 and 1.

Ans: False
Feedback: 6.6
Difficulty: Easy

12. Mutex locks and counting semaphores are essentially the same thing.

Ans: False
Feedback: 6.6

Difficulty: Easy

13. A semaphore has an integer value.
Ans: True
Feedback: 6.6
Difficulty: Easy

14. When the mutex lock is implemented based on a binary semaphore, it should be initialized to be 0.
Ans: False
Feedback: 6.6
Difficulty: Easy

15. Every object in Java has associated with it a single lock.

Ans: True
Feedback: 6.7
Difficulty: Medium

16. A monitor is an abstract data type that is based on semaphore implementation.
Ans: False
Feedback: 6.7
Difficulty: Easy

17. Solutions to the critical section problem may suffer from liveness failures.
Ans: True
Feedback: 6.8
Difficulty: Easy

18. CAS-based synchronization is always faster than traditional synchronization.
Ans: False
Feedback: 6.9
Difficulty: Easy