Chapter 7

Multiple Choice Questions

1. In the structure of the producer process shown in Figure 7.1, what would be a possible outcome if wait(empty) is replaced with signal(empty) and signal(full) is replaced with wait(full)?
A) Producer will remain blocked after adding an item in the buffer.
B) Consumer will remain blocked after taking out an item from the buffer.
C) Producer and consumer may access the buffer at the same time.
D) All of the above.

Ans: A
Feedback: 7.1.1
Difficulty: Difficult

2. Which of the following is true about the two versions of readers-writers problem?
A) In the first readers-writers problem, a writer may starve if a continuous series of readers arrive before the earlier readers exit their critical section.
B) In the second readers-writers problem, a reader may starve if a continuous series of readers arrive before the earlier readers exit their critical section.
C) In the first readers-writers problem, a writer may starve if a continuous series of writers arrive before the earlier writers exit their critical section.
D) In the second readers-writers problem, a writer may starve if a continuous series of readers arrive before the earlier readers exit their critical section.

Ans: A
Feedback: 7.1.2
Difficulty: Medium

3. A reader-writer lock is useful when
A) there are a significantly large number of processes attempting to enter a critical section.
B) there are a significantly large number of consumer processes attempting to read data from a bounded buffer.
C) there are a significantly small number of reader processes attempting to read in the critical section.
D) there are a significantly large number of reader processes attempting to read in the critical section.

Ans: D
Feedback: 7.1.2
Difficulty: Medium

4. In the solution provided for readers-writers problem in Section 7.1.2, if a writer is in the critical section, and multiple readers and writers are waiting,

A) all waiting readers will be allowed to enter the critical section when the writer in the critical section exits.
B) all waiting writers will be allowed to enter the critical section when the writer in the critical section exits.
C) exactly one of the waiting writers will be allowed to enter the critical section when the writer in the critical section exits.
D) either all waiting readers or exactly one writer will be allowed to enter the critical section.

Ans: D
Feedback: 7.1.2
Difficulty: Medium

5. In an asymmetric solution for the dining philosophers problem, deadlock is avoided, because
A) there is no contention for acquiring chopsticks.
B) neighboring philosophers will never get hungry at the same time.
C) any neighboring philosophers would have already acquired one of their chopsticks before attempting to acquire the shared chopstick.
D) a philosopher will release a chopstick in case she is unable to acquire the other chopstick.

Ans: C
Feedback: 7.1.3
Difficulty: Medium

6. In the monitor solution for dining-philosophers problem (Figure 7.7), a philosopher may start eating

A) at the end of the pickup( ) function before exiting the function.
B) in the beginning of the putdown( ) function
C) after exiting the pickup( ) function and before entering the putdown( ) function.
D) All of the above.

Ans: C
Feedback: 7.1.3
Difficulty: Easy

7. When the state for a dispatcher object moves to signaled, the Windows kernel
A) moves all threads waiting on that object to ready state if the dispatcher object is a mutex object.
B) moves a fixed number of threads (possibly greater than one) waiting on that object to ready state if the dispatcher object is a mutex object.
C) moves all threads waiting on that object to ready state if the dispatcher object is an event object.
D) moves a fixed number of threads (possibly greater than one) waiting on that object to ready state if the dispatcher object is an event object.

Ans: C
Feedback: 7.2.1
Difficulty: Easy

8. A critical-section object in Windows
A) is particularly efficient when there is lots of contention for the object.
B) completely avoids kernel intervention.
C) uses spinlocks on single processor systems.
D) None of the above.

Ans: D
Feedback: 7.2.1
Difficulty: Medium

9. In a single processor system running Windows, when the kernel accesses a global resource, it

A) uses spinlocks.
B) masks all interrupts.
C) uses a dispatcher object.
D) atomic integers.

Ans: B
Feedback: 7.2.1
Difficulty: Easy

10. Atomic integers in Linux are useful when

A) several variables are involved in a race condition.
B) a single process access several variable involved in a race condition.
C) an integer variable needs to be updated.
D) All of the above.

Ans: C
Feedback: 7.2.2
Difficulty: Easy

11. Which of the following statements is not true about spinlocks in Linux?
A) Spinlocks cannot be used on single processor machines.
B) A thread may disable kernel preemption on Symmetric Multi Processing machines instead of acquiring spinlocks.
C) A thread that acquires a spinlock cannot acquire the same lock a second time without first releasing the lock.
D) The Linux kernel is designed so that the spinlock is held only for only short durations.

Ans: B
Feedback: 7.2.2
Difficulty: Medium

12. POSIX named semaphores
A) can easily be used by multiple unrelated processes.
B) can be initialized during creation time.
C) uses sem_wait( ) and sem_post( ) to acquire and release a semaphore respectively.
D) All of the above.

Ans: D
Feedback: 7.3.2
Difficulty: Medium

13. A thread using POSIX condition variables, a thread

A) must lock the associated mutex lock before calling pthread_cond_wait( ) and unlock it after calling pthread_cond_signal( ).
B) must lock the associated mutex lock before calling pthread_cond_wait( ), but doesn't need to unlock it after calling pthread_cond_signal( ).
C) doesn't need to lock the associated mutex lock before calling pthread_cond_wait( ), but must unlock it after calling pthread_cond_signal( ).
D) doesn't need to lock the associated mutex lock before calling pthread_cond_wait( ) and doesn't need to unlock it after calling pthread_cond_signal( ).

Ans: A
Feedback: 7.3.3
Difficulty: Medium

14. In JAVA, when a thread calls wait( ) inside a synchronized method,
A) the thread releases the object lock and continues its execution.
B) the thread releases the object lock, blocks and is put in the entry set.
C) the thread releases the object lock, blocks and is put in the wait set.
D) the thread continues its execution without releasing the object lock.

Ans: C
Feedback: 7.4.1
Difficulty: Medium

15. A notify( ) operation in Java monitors
A) removes a process from the wait set and puts it in the entry set.
B) removes a process from the entry set and puts it in the wait set.
C) removes a process from the entry set and provides the lock for the object to that process.
D) removes a process from the wait set and provides the lock for the object to that process.

Ans: A
Feedback: 7.4.1
Difficulty: Medium

16. In the solution for bounded buffer problem using JAVA monitors (Figure 7.9), functions insert( ) and remove( ) are synchronized to ensure that

A) a thread may insert an item and different thread may remove an item from a different location in the buffer simultaneously.
B) at most one thread may enter or remove an item at any time.
C) at most one thread may enter an item at any time, but multiple threads may remove items from different locations at the same time.
D) multiple thread may enter items at different locations at the same time, but at most one thread may remove an item at the same time.

Ans: B
Feedback: 7.4.1
Difficulty: Easy

17. A key difference between Reentrant locks and JAVA monitor's synchronized statements is that
A) there is a possibility of deadlock when using a monitor while deadlock cannot occur when using reentrant locks.
B) a reentrant lock favors granting the lock to the longest-waiting thread while there is no specification for the order in which threads in the wait set for an object lock.
C) multiple processes may own a reentrant lock at the same time while at most one process may execute inside a synchronized method at any time.
D) at most one process may own a reentrant lock, while multiple processes may execute inside a synchronized method at any time.

Ans: B
Feedback: 7.4.2
Difficulty: Medium

18. The signal( ) operation in the example using JAVA condition variables ensures that the thread with the thread number turn

A) that is blocked on await( ) is unblocked.
B) that may be blocked on await( ) is unblocked.
C) does not block on await( ) even if hasn't yet called this function.
D) blocks on await( ) if it hasn't yet called this function.

Ans: B
Feedback: 7.4.4
Difficulty: Difficult

19. Emergence of multicore systems has put greater emphasis on developing novel techniques for concurrency problems, because
A) some fundamentally new concurrency problems have arisen that cannot be solved using traditional techniques such as mutex locks, semaphores, and monitors.
B) race conditions are much more difficult to solve in multicore systems.
C) deadlocks are much more difficult to prevent or avoid in multicore systems.
C) with increased number of processing cores, there is an increased risk of race conditions and deadlocks.

Ans: D
Feedback: 7.5
Difficulty: Easy

20. Alternate approaches such as transactional memory or OpenMP are useful, because

A) some race condition problems that cannot be solved using synchronization mechanisms such as mutex locks and semaphores can be solved using these alternate approaches.
B) these approaches scale much better than synchronization mechanisms such as mutex locks and semaphores as the number of threads increases.
C) developers do not need to identify race conditions when using these approaches.
D) All of the above.

Ans: B
Feedback: 7.5
Difficulty: Medium

21. A(n) _____ is a sequence of read-write operations that are atomic.
A) atomic integer
B) semaphore
C) memory transaction
D) mutex lock

Ans: C
Feedback: 7.5.1
Difficulty: Easy

22. An advantage of using transactional memory is that
A) the atomicity is guaranteed by the transactional memory system.
B) there is no possibility of deadlocks.
C) the transactional memory system can identify which statements in atomic blocks can be executed concurrently.
D. All of the above.

Ans: D
Feedback: 7.5.1
Difficulty: Easy

23. A difference between software transactional memory (STM) and hardware transactional memory (HTM) is that

A) HTM uses cache hierarchy and cache coherency protocols while STM uses software implementation in addition to the cache hierarchy and cache coherency protocols.
B) STM requires no special code instrumentation and thus has less overhead than HTM.
C) In HTM, code is inserted by a compiler, while in STM, user enters the appropriate code.
D) HTM requires existing cache hierarchies and cache coherency protocols be modified, while STM does not.

Ans: D
Feedback: 7.5.1
Difficulty: Medium

24. Critical-section compiler directive in OpenMP is generally considered easier to use than standard mutex locks, because
A) management of entry and exit code is managed by OpenMP library reducing the possibility of programming errors.
B) programmers don't need to worry about race conditions.
C) a thread cannot block inside a critical section.
D) deadlock cannot occur.

Ans: A
Feedback: 7.5.2
Difficulty: Medium

25. In functional programming languages,
A) race conditions cannot occur.
B) there is no need to maintain program state.
C) deadlocks cannot occur.
D) All of the above.

Ans: D
Feedback: 7.5.3
Difficulty: Easy

Short Answer Questions

1. Explain the difference between the first readers–writers problem and the second readers–-writers problem.

Ans: The first readers–writers problem requires that no reader will be kept waiting unless a writer has already obtained permission to use the shared database; whereas the second readers–writers problem requires that once a writer is ready, that writer performs its write as soon as possible.
Feedback: 7.1.2
Difficulty: Easy

2. Describe a scenario when using a reader–writer lock is more appropriate than another synchronization tool such as a semaphore.

Ans: A tool such as a semaphore only allows one process to access shared data at a time. Reader–writer locks are useful when it is easy to distinguish if a process is only reading or reading/writing shared data. If a process is only reading shared data, it can access the shared data concurrently with other readers. In the case when there are several readers, a reader–writer lock may be much more efficient.
Feedback: 7.1.2
Difficulty: Medium

3. Describe the dining-philosophers problem and how it relates to operating systems.

Ans: The scenario involves five philosophers sitting at a round table with a bowl of food and five chopsticks. Each chopstick sits between two adjacent philosophers. The philosophers are allowed to think and eat. Since two chopsticks are required for each philosopher to eat, and only five chopsticks exist at the table, no two adjacent philosophers may be eating at the same time. A scheduling problem arises as to who gets to eat at what time. This problem is similar to the problem of scheduling processes that require a limited number of resources.
Feedback: 7.1.3
Difficulty: Medium

4. In the solution for dining philosophers problem using monitors, provide a scenario in which a philosopher may starve to death.

Ans: Consider a scenario where philosopher 2 is eating and philosopher 3 wants to eat (starving). Philosopher 3 has to wait since one of her neighbor is eating. All other philosophers are thinking. Before philosopher 2 finishes eating philosopher 4 gets hungry and starts eating since both of her neighbors are not eating. Now philosopher 2 finishes eating, but philosopher 3 still cannot eat because one of her neighbor is eating. Now before philosopher 4 finishes eating, philosopher 2 gets hungry and starts eating. This process of philosophers 2 and 4 alternating their eating states can continue forever, and as a result, philosopher 3 will never get to eat.
Feedback: 7.1.3
Difficulty: Difficult

5. Describe the mechanisms used for synchronization in Windows kernel in single and multiprocessor systems. Explain why Windows uses different mechanisms for the two systems.

Ans: On single processor systems, Windows temporarily masks interrupts for all interrupt handlers when accessing a global resource. This mechanism does not work in multiprocessor systems, because masking interrupts on processor still allows threads in other processes to run concurrently. Instead, Windows provides spinlocks to access global resources on multiprocessor systems. Spinlocks can be inefficient, particularly for longer code segments, so the kernel uses spinlocks only to protect short code segments. Furthermore, for reasons of efficiency, the kernel ensures that a thread will never be preempted while holding a spinlock.
Feedback: 7.2.1
Difficulty: Medium

6. Explain the relationship between the state of a dispatcher object and the state of a thread in Windows.

Ans: When a thread blocks on a dispatcher object in a nonsignaled state its state changes from ready to waiting. When the state for a dispatcher object moves to signaled, the kernel checks whether any

threads are waiting on the object. If so, the kernel moves one thread—or possibly more—from the waiting state to the ready state, where they can resume executing.
Feedback: 7.2.1
Difficulty: Easy

7. Explain the role of the variable preempt_count in the Linux kernel.

Ans: Each task maintains a value preempt_count which is the number of locks held by each task. When a lock is acquired, preempt_count is incremented. When a lock is released, preempt_count is decremented. If the task currently running in the kernel has a value of preempt_count > 0, the kernel cannot be preempted as the task currently holds a lock. If the count is zero, the kernel can be preempted.
Feedback: 7.2.2
Difficulty: Difficult

8. Explain how Linux manages race conditions on single-processor systems such as embedded devices.

Ans: On multiprocessor machines, Linux uses spinlocks to manage race conditions. However, as spinlocks are not appropriate on single processor machines, Linux instead disables kernel preemption which acquiring a spinlock, and enables it after releasing the spinlock.
Feedback: 7.2.2
Difficulty: Medium

9. Explain how can a POSIX unnamed semaphore be shared between multiple processes.

Ans: The unnamed semaphore must be created using flag 1 (second parameter of sem_init( )). This semaphore can be shared by placing it in a region of shared memory, or by any child processes created by the parent.
Feedback: 7.3.2
Difficulty: Difficult

10. Explain what will happen if a process A locks the associated mutex before calling pthread cond wait( ), but another process B does not lock the associated mutex before call pthread_cond_signal( ).

Ans: In this case, there is a race condition between wait and signal statements. Process A may miss the signal and will remain blocked on the wait condition.
Feedback: 7.3.3
Difficulty: Difficult

11. In JAVA monitors, when a thread is placed in the entry set, is that thread guaranteed to own the object lock some time in future.

Ans: The JAVA specification does not require that threads in the entry set be organized in any particular order, so it is possible that a thread in the entry set may never be selected to own the lock assuming that there are always multiple threads in the set. However, in practice, most virtual machines order threads in the entry set according to a FIFO policy, in which case, every thread in the entry set is guaranteed to own the object sometime in future as long as every thread that gains the lock does release the lock (e.g., by exiting the method or calling wait( )).
Feedback: 7.4.1
Difficulty: Difficult

12. Explain the key limitation of wait( ) and notify( ) operations of Java monitors that Java condition variables remedy.

Ans: Each Java monitor is associated with just one unnamed condition variable and the wait() and notify( ) operations apply only to this single condition variable. Condition variables remedy this by allowing programmers to create (possibly multiple) named condition variables.
Feedback: 7.4.4
Difficulty: Easy

13. What is the difference between software transactional memory and hardware transactional memory?

Ans: Software transactional memory (STM) implements transactional memory entirely in software, no special hardware is required. STM works by inserting instrumentation code inside of transaction blocks and typically requires the support of a compiler. Hardware transactional memory (HTM) implements transactional memory entirely in hardware using cache hierarchies and cache coherency protocols to resolve conflicts when shared data resides in separate caches.
Feedback: 7.5.1
Difficulty: Difficult

14. Does nested critical sections in OpenMP provide any utility? Explain why.

Ans: Since at most one thread may be active at any time inside an OpenMP critical section, at most one thread can reach the start of the nested critical section at any time. Thus, nesting a critical section with in another does not provide any utility.
Feedback: 7.5.2
Difficulty: Difficult

15. Describe how a critical-section object functions.

Ans: A critical-section object can be acquired and released without kernel intervention. On a multiprocessor system, a critical-section object first uses a spinlock while waiting for the other thread to release the object. If it spins too long, the acquiring thread allocates a kernel mutex and yields its CPU.

Feedback: 7.5.2
Difficulty: Difficult

True/False Questions

1. The solution for bounded buffer problem provided in Section 7.1.1 does not work correctly if there are more than one producer or consumer.

Ans: True
Feedback: 7.1.1
Difficulty: Easy

2. A reader-writer lock gives preference to writer processes in the readers–writers problem.

Ans: False
Feedback: 7.1.2
Difficulty: Medium

3. A solution to the readers-writers problem that avoids starvation and allows some concurrency among readers is not possible.

Ans: False
Feedback: 7.1.2
Difficulty: Difficult

4. Allowing at most four philosophers to sit simultaneously prevents deadlock.

Ans: True
Feedback: 7.1.3
Difficulty: Easy

5. Dining philosophers problem is important because it represents a class of problems where multiple processes need to share multiple resources.

Ans: True
Feedback: 7.1.3
Difficulty: Easy

6. In Windows, a thread may get preempted while holding a spinlock.
Ans: False
Feedback: 7.2.1
Difficulty: Easy

7. Dispatcher objects in Windows are used for synchronization outside the kernel.

Ans: True
Feedback: 7.2.1
Difficulty: Easy

8. A critical section object in the user mode needs kernel intervention to ensure mutual exclusion.

Ans: False
Feedback: 7.2.1
Difficulty: Easy

9. To lock the kernel on a single processor machine in Linux, kernel preemption is disabled.

Ans: True
Feedback: 7.2.2
Difficulty: Easy

10. POSIX unnamed semaphores can be shared either only by threads with in a single process, or between processes.

Ans: True
Feedback: 7.3.2
Difficulty: Medium

11. A call to pthread_cond_signal() (used by POSIX threads, called Pthread) releases the associated mutex lock.

Ans: False
Feedback: 7.3.3
Difficulty: Medium

12. In JAVA, calling a synchronized method always blocks the calling thread.

Ans: False
Feedback: 7.4.1
Difficulty: Medium

13. If a thread has the ownership of a Reentrant lock, it will block forever if it calls lock( ) before calling unlock( ).

Ans: False
Feedback: 7.4.2
Difficulty: Medium

14. With reentrant locks, programming construct try and finally is used to ensure that unlock( ) is called even when an exception occurs in the critical section.

Ans: True
Feedback: 7.4.2
Difficulty: Medium

15. Semaphores in JAVA can be initialized to a negative value.

Ans: True
Feedback: 7.4.3
Difficulty: Easy

16. JAVA provides support for both named and unnamed condition variables.

Ans: False
Feedback: 7.4.4
Difficulty: Medium

17. Transactional memory may particularly be useful for multicore systems.

Ans: True
Feedback: 7.5.1
Difficulty: Easy

18. Each critical section must be assigned a different name in OpenMP.

Ans: False
Feedback: 7.5.2
Difficulty: Medium