Chapter 7 Deadlock

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)?

```
while (true) {
   . . .
  /* produce an item in next_produced */
   . . .
  wait(empty);
  wait(mutex);
   . . .
  /* add next_produced to the buffer */
   . . .
  signal(mutex);
  signal(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.

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.

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.

4. In the Writers-Priority solution provided for readers-writers problem, 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.

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.

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.

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.

10. Atomic integers in Linux are useful when
A) several variables are involved in a race condition.
B) a single process accesses several variables involved in a race condition.
C) an integer variable needs to be updated.
D) All of the above.

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.

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( ).

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.

16. In the solution for bounded buffer problem using JAVA monitors, 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.

```
public class BoundedBuffer<E>
{
   private static final int BUFFER_SIZE = 5;

   private int count, in, out;
   private E[] buffer;

   public BoundedBuffer() {
      count = 0;
      in = 0;
      out = 0;
      buffer = (E[]) new Object[BUFFER_SIZE];
   }

   /* Producers call this method */
   public synchronized void insert(E item) {
      /* See Figure 7.11 */
   }

   /* Consumers call this method */
   public synchronized E remove() {
      /* See Figure 7.11 */
   }
}
```

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.

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.

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.

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.

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

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.

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.

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.

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.

Short Answer Questions

1. Explain the difference between the Reader-Priority Readers–Writers problem and the Writer-Priority Readers–-Writers problem.

2. Describe how is that writer lock may be much more efficient and thus more appropriate than another synchronization tool such as a semaphore.

3. Describe the dining-philosophers problem.

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

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

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

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( ).

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.

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

True/False Questions

T 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.

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

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

T 4. Allowing at most four philosophers to sit simultaneously prevents deadlock (assuming the table still has five chopsticks).

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

6. In Windows, a thread may get preempted while holding a spinlock.

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

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

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

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

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