1. Which of the following about mutex is not correct?

A. We can protect our data and ensure access by only one thread at a time by using the  
pthreads mutual-exclusion interfaces.

B. A mutex is basically a lock that we set (lock )before accessing a shared resource and release (unlock) when we’re done.

C. While it is set, any other thread that tries to set it will block until we release it.

D. If more than one thread is blocked when we unlock the mutex, then all threads blocked on the lock will be made runnable, and the first one to run will be able to set the lock.

E. The others will see that the mutex is not locked and go back to waiting for it to become available again.

Answer: E) The others will see that the mutex is still locked. APUE 11.6.1 page 400

2. To initialize a mutex with the default attributes, we set attr to \_\_\_\_.

A. 1

B. -1

C. NULL

D. 0

E. 2

Answer: C) To initialize a mutex with the default attributes, we set attr to NULL. APUE 11.6.1 page 400

3. If a thread can’t afford to block, it can use \_\_\_\_\_\_\_to lock the  
mutex conditionally.

A. pthread\_mutex\_trylock

B. pthread\_mutex\_lock

C. pthread\_mutex\_unlock

D. pthread\_mutex\_init

E. pthread\_mutex\_destroy

Answer A) If a thread can’t afford to block, it can use pthread\_mutex\_trylock to lock the  
mutex conditionally. APUE 11.6.1 page 400

4. Which of the following about deadlock is not correct?

A. Deadlocks can be avoided by carefully controlling the order in which mutexes are locked.

B. For example, assume that you have two mutexes, A and B, that you need to lock at the same time. If all threads always lock mutex A before mutex B, no deadlock can occur from the use of the two mutexes.

C. but you can still deadlock on other resources

D. Similarly, if all threads are not locked mutex B before mutex A, no deadlock will occur

E. You’ll have the potential for a deadlock only when one thread attempts to lock the mutexes in the opposite order from another thread.

Answer D) if all threads always lock mutex B before mutex A, no deadlock will occur. APUE 11.6.2 page 402

5. Which of the following about deadlock is not correct?

A. The lock-ordering issues surrounding the hash list and the reference count go away when we use the same lock for both purposes.

B. Multithreaded software design involves these types of trade-offs.

C. If your locking granularity is too coarse, you end up with too many threads blocking behind the same locks, with little improvement possible from concurrency.

D. If your locking granularity is too fine, then you get good performance from excess locking overhead, and you end up with complex code.

E. As a programmer, you need to find the correct balance between code complexity and performance, while still satisfying your locking requirements.

Answer D) If your locking granularity is too fine, then you suffer bad performance from excess locking overhead, and you end up with complex code. APUE 11.6.2 page 407

6. What does pthread\_mutex\_timedlock return if OK?

A. 1

B. -1

C. NULL

D. 0

E. 2

Answer D) Returns: 0 if OK, error number on failure. APUE 11.6.3 page 407

7. Which of the following is not correct about reader-writer locks?

A. Reader–writer locks are similar to mutexes, except that they allow for higher degrees of  
parallelism.

B. With a mutex, the state is either locked or unlocked, and only one thread can lock it at a time.

C. Three states are possible with a reader–writer lock: locked in read mode, locked in write mode, and unlocked.

D. Multiple threads at a time can hold a reader–writer lock in write mode.

E. But multiple threads can hold a reader–writer lock in read mode at the same time.

Answer D) Only one thread at a time can hold a reader–writer lock in write mode. APUE 11.6.4 page 409

8. Which of the following is not correct about condition variables?

A. Condition variables are another synchronization mechanism available to threads.

B. These synchronization objects provide a place for threads to rendezvous.

C. When used with mutexes, condition variables allow threads to wait in a race-free way for arbitrary  
conditions to occur.

D. The condition itself is protected by a mutex.

E. A thread can change the condition state anytime.

Answer E) A thread must first lock the mutex to change the condition state. APUE 11.6.6 page 413