*1. Available at the end of "text book"*

*A*

**2. C**

*4. Available at the end of "text book"*

*C*

5. True

7. C

10. A, B, C, D

11. A

14. 2

*15. Available at the end of "text book"*

*1*

16.

a. Preemptive Kernel: All of the code inside the SendPacket function comprises a critical section and must be protected against an interrupt which my trigger a context switch (and thus possible reentry) by another thread. Immediately upon entry to the function we need a semaphore pend, and just before exit we need a semaphore post. Disabling interrupts is not acceptable since the Send1Byte function may be interrupt-driven.

b. Non-Preemptive Kernel: A context switch in a non-preemptive application would require an explicit yield call to be coded by the application programmer. Although none appear within the SendPacket funtion, it is possible that one is coded within the Send1Byte function while it waits for the serial device to finish sending the previous byte. Disabling interrupts won't provide protection since it won't prevent the yield call. Again, the only reliable solution is as before – protecting the critical section with a semaphore.

*1. Available at the end of "text book"*

*B*

3. A

4. B

*5. Available at the end of "text book"*

*A*

6.

a. L+M
b. L
c. L

7.

a. N
b. L+H
c. L+H

10.

| | State of Thread A | State of Thread B | Resource 1 Owned by | Resource 2 Owned by |
|---|---|---|---|---|
| Start: | Running | Ready | Thread B | No Thread |
| Event 1: | Running | Ready | Thread B | Thread A |
| Event 2: | Pending | Running | Thread B | Thread A |
| Event 3: | Pending | Pending | Thread B | Thread A |

11.

a. Threads – 2, Shared Resources – 1
b. Threads – 3, Shared Resources – 1
*c. Available at the end of "text book"*
Threads – 2, Shared Resources – 2

14. & 15.

B

18.

L + M seconds

19.

a. L seconds