QUESTION 1

1. In Algorithm 1 presented in the lecture, progress is violated because

- (●) Process P0 executes the critical section once, enters a long remainder section, context switch occurs to process P1, process P1 is waiting to get access to the critical section indefinitely.
- ( ) All of the other options are correct.
- ( ) Process P0 executes the critical section once, enters a short remainder section, context switch occurs to process P1, process P1 is waiting to get access to the critical section indefinitely.
- ( ) Process P1 executes the critical section once, enters a long remainder section, context switch occurs to process P0, process P0 executes the critical section once and then waits indefinitely to get access to the critical section again.

0.15 points

QUESTION 2

1. What is meant by the Progress property?

- (●) It is liveness; when no process is in the critical section, then a process cannot wait indefinitely to get access to the critical section.
- ( ) It is fairness; when a process wants access to the critical section, another process cannot repeatedly get access to the critical section indefinitely.
- ( ) It is liveness; when one process is in the critical section, another process cannot wait indefinitely to get access to the critical section.

0.15 points

QUESTION 3

1. In the Producer-Consumer Bounder-Buffer implementation, why is the Wait() on empty or full semaphore done before the Wait() on mutex?

- (●) Otherwise, it can lead to a deadlock when the buffer is empty/full, because to add/remove items in the buffer the mutex semaphore must be acquired.
- ( ) Otherwise, it can lead to a deadlock when the buffer is empty/full, because to add/remove items in the buffer the empty/full semaphore must be acquired.
- ( ) Otherwise, it can lead to a deadlock when the buffer is empty/full, because to add/remove items in the buffer the full/empty semaphore must be acquired.

0.15 points

QUESTION 4

1. In Algorithm 2 in the lecture, Progress property is violated because

- (●) Process P0 updates flag to true, context switch to Process P1, Process P1 updates flag to true, both the processes are now stuck in the entry while loop.
- ( ) Process P0 updates flag to false, context switch to Process P1, Process P1 updates flag to false, both the processes are now stuck in the entry while loop.
- ( ) Process P0 updates flag to true and enters critical section, context switch to Process P1, Process P1 updates flag to true and is now stuck indefinitely.

0.15 points

QUESTION 5

1. Algorithm 3 in the lecture satisfies Progress property because

- (●) Suppose Process P0 wants access to the critical section; then flag for Process P0 is true; if Process P1 is in the entry section then either P0 or P1 will get access to the critical section; if Process P1 is in the critical section then Progress is vacuously satisfied; if Process P1 is in the remainder section then Process P0 can get access to the critical section.
- ( ) Progress property is satisfied because Bounded Waiting is satisfied.
- ( ) Process P0 sets its flag to true, turn to 1, and waits at the entry while loop; context switch to Process P1; Process P1 sets its flag to true, turn to 0, and waits at the entry while loop; context switch to Process P0; Process P0 enters the critical section.

0.15 points

QUESTION 6

1. Explain why Bounded Waiting property is not satisfied in the TestAndSet(lock) implementation of the Entry section.

- (●) TestAndSet(lock) does not store information about failed attempts to acquire the lock. Hence, a process that attempts to acquire an available lock will be given the lock, irrespective of the number of failed attempts by other processes.
- ( ) It is only true when Round Robin scheduling is used as shown in the lecture slide.
- ( ) Both the other options are correct.

0.15 points

QUESTION 7

1. In Algorithm 2 in the lecture, Bounded Waiting is satisfied because

- (●) A process sets its flag to true at the beginning of entry section; once the flag is set to true another process cannot subsequently enter its critical section.
- ( ) Progress is not satisfied and this implies that Bounded Waiting will be satisfied.
- ( ) Process P0 starts and enters critical section, context switch to Process P1, Process P1 sets flag to true and gets stuck in the entry while loop, context switch to Process P0, Process P0 completes critical and remainder sections and tries to enter the critical section again, Process P0 gets stuck.

0.15 points

QUESTION 8

1. In TestAndSet(lock) based implementation of the Entry section, explain why Mutual Exclusion is successful.

- (●) A process enters its critical section only when the lock value is changed from false to true by the TestAndSet() call. The lock remains true while the process is in the Critical section. Since the call to TestAndSet() is atomic, any other process would see the lock as true when executing TestAndSet().
- ( ) A process enters its critical section only when the lock value is changed from true to false by the TestAndSet() call. The lock remains true while the process is in the Critical

section. Since the call to TestAndSet() is atomic, any other process would see the lock as false when executing TestAndSet().

- A process enters its critical section only when the lock value is changed from false to true or the value remains unchanged by the TestAndSet() call. The lock remains true while the process is in the Critical section. Since the call to TestAndSet() is atomic, any other process would see the lock as true when executing TestAndSet().

0.15 points

QUESTION 9

1. In the blocking implementation of Wait(), why can we not decrement the semaphore value after the if() condition check?

- This would imply a context-switch between the if() condition check and the decrement, which can lead to mutual exclusion violation.
- This would imply a context-switch between the if() condition check and the decrement, which can lead to a race condition for the semaphore value.
- It is possible, but the implementation is less efficient than when the decrement is before the if() condition check.

0.15 points

QUESTION 10

1. What is meant by the mutual exclusion property?

- No two processes can be in the critical section at the same time.
- At most two processes can be in the critical section at the same time.
- No process can context switch in the critical section.
- The OS cannot context switch from one process in its critical section to another process in its critical section.

QUESTION 1

1. In the Reader-Writer implementation, explain what it means by "the reader is given preference".

- (●) While a reader is reading, other readers will be allowed access to the database, even if writers are currently waiting to get access.
- ( ) If a reader and writer both want access to the database at the same time, then the reader is given preference over the writer irrespective of who is currently accessing the database.
- ( ) Readers can always access the database, even if writers are currently accessing it.

QUESTION 3

1. Algorithm 3 in the lecture satisfies the Bounded Waiting property because

- (●) If Process P0 is waiting in the entry while loop, Process P1's flag is true and turn is 1; when Process P1 tries to enter the critical section subsequently it will set turn to 0; Process P0 is no longer blocked. Identical argument applies to Process P1 because the solution is symmetric.
- ( ) Process P0 sets its flag to true, turn to 1 and waits in the entry while loop; context switch to Process P1; Process P1 sets its flag to true, turn to 0 and waits in the entry while loop; context switch to Process P0; Process P0 enters its critical section. Identical argument applies to Process P1 because the solution is symmetric.
- ( ) Bounded Waiting is satisfied because Progress property is satisfied.

QUESTION 7

1. In the producer-consumer bounded buffer implementation, explain the role of each semaphore.

- ( ) mutex is a binary semaphore for mutually exclusive access to the shared buffer; full is a counting semaphore tracking the number of empty slots in the buffer; empty is a counting semaphore tracking the number of items in the buffer.
- ( ) mutex is a binary semaphore for mutually exclusive access to the shared buffer; full is used by the Producer to check if the buffer is full; empty is used by the Consumer to check if the buffer is empty.
- ( ) mutex is a binary semaphore for mutually exclusive access to the shared buffer; full is used by the Consumer to check if the buffer is full; empty is used by the Producer to check if the buffer is empty.
- (●) mutex is a binary semaphore for mutually exclusive access to the shared buffer; empty is used by the Producer to check if the buffer is full; full is used by the Consumer to check if the buffer is empty.

QUESTION 9

1. In the Dining-Philosopher problem, explain how the "asymmetric" solution solves deadlock.

- ( ) Odd philosopher first picks left chopstick and then right, whereas even philosopher first picks right chopstick and then left. This ensures that at least one philosopher will

not be able to pick any chopstick when all of them try to acquire it simultaneously and hence deadlock is avoided.

- Odd philosopher first picks right chopstick and then left, whereas even philosopher first picks left chopstick and then right. This ensures that at least one philosopher will not be able to pick any chopstick when all of them try to acquire it simultaneously and hence deadlock is avoided.
- Both the other answers are correct.

QUESTION 10

1. Present one advantage and one dis-advantage of busy waiting implementation of semaphores.
   - reduces context-switch overheads; infeasible on single-core CPUs.
   - increases context-switch overheads; infeasible on single-core CPUs.
   - reduces context-switch overheads; infeasible on multi-core CPUs.
   - increases context-switch overheads; infeasible on multi-core CPUs.

QUESTION 1

1. The busy waiting implemention of semaphores is infeasible only on single-core CPUs.

- (•) True; because to release the semaphore the busy waiting must be interrupted on a single-core CPU which then leads to the Wait() system call being non-atomic.
- ( ) False; it is infeasible even on multi-core CPUs.
- ( ) False; it is feasible on both single as well as multi-core CPUs.
- ( ) True; because to lock the semaphore the busy waiting must be interrupted on a single-core CPU.

QUESTION 2

1. What are the sequence of atomic operations in the TestAndSet(lock) function call?

- (•) fetch the current value of lock in a local variable; store the value true in the lock; return the fetched value to the caller.
- ( ) fetch the current value of lock in a local variable; store the value false in the lock; return the fetched value to the caller.
- ( ) store the value true in the lock; fetch the current value of lock in a local variable; return the fetched value to the caller.
- ( ) store the value false in the lock; fetch the current value of lock in a local variable; return the fetched value to the caller.

QUESTION 3

1. Race conditions can be prevented by disabling interrupts in critical sections.

- (•) True; disabling interrupts will prevent context switches which is necessary and sufficient to prevent a race condition.
- ( ) True; disabling interrupts will prevent the execution of any other process or even the OS. This will ensure mutual exclusion.
- ( ) False; interrupts are unrelated to the race condition problem.

QUESTION 6

1. Race condition is impossible with a single writer, multiple reader process system.

- (•) False; race condition can occur even without any writers in the system.
- ( ) True; in this case the writes and reads can be causally ordered without any issues.
- ( ) False; can still cause a race condition between the writer and some of the readers.

QUESTION 3

1. Semaphore implementations do not require any special hardware support.

   - (●) False; Wait() and Signal() system calls must be executed atomically, which requires hardware support (either disabling of interrupts or atomic instructions like TestAndSet()).
   - ( ) True; Wait() and Signal() system calls must be executed atomically, but they do not require any hardware support.
   - ( ) False; Wait() system call must be executed atomically, which requires hardware support (either disabling of interrupts or atomic instructions like TestAndSet()). However, Signal() system call does not require hardware support.

QUESTION 8

1. Why must Wait() and Signal() system calls with blocking implementation be atomic? Give all reasons.

   - (●) To prevent race condition in the updates to the value of the semaphore; To ensure mutual exclusion by guaranteeing that the update to the semaphore value and its checking in the if() condition happen atomically in each system call.
   - ( ) To ensure mutual exclusion by guaranteeing that the update to the semaphore value and its checking in the if() condition happen atomically in each system call.
   - ( ) To prevent race condition in the updates to the value of the semaphore.
   - ( ) They don't have to be atomic. Only busy waiting implementations require atomic Wait() and Signal() system calls.

QUESTION 9

1. In the Reader-Writer implementation, explain all the functions for each semaphore.

   - ( ) mutex is a binary semaphore to ensure mutually exclusive updates to the readcount variable; wrt is a binary semaphore to ensure mutually exclusive access among writers.
   - (●) mutex is a binary semaphore to ensure mutually exclusive updates to the readcount variable; wrt is a binary semaphore to ensure mutually exclusive access among writers as well as between readers and writers.
   - ( ) mutex is a binary semaphore to ensure mutually exclusive updates to the readcount variable; wrt is a counting semaphore to count the number of readers.