COS 226 S2 2022 Final Examination 19/11

Review Test Submission: COS226 November 2022 Examination

Review Test Submission: COS226 November 2022 Examination

User	
Course	COS 226 S2 2022
Test	COS226 November 2022 Examination
Started	11/19/22 8:01 AM
Submitted	11/19/22 11:16 AM
Due Date	11/19/22 11:45 AM
Status	Completed
Attempt Score	52 out of 80 points
Time Elapsed	3 hours, 14 minutes out of 3 hours and 15 minutes
Results Displayed	All Answers, Submitted Answers, Correct Answers, Feedback, Incorrectly Answered Questions

Question 1

1 out of 1 points

If a thread is interrupted during a call to a Condition's await() method.

Selected Answer: ✓ The call throws InterruptedException.

Answers:

The call continues waiting.

✓ The call throws InterruptedException.

The call goes into a blocking state.

None of the above.

Question 2

0 out of 1 points

Which of the following is a disadvantage of the Backoff lock ?

Selected Answers: ✗ Fairness.

✓ Cache-coherence traffic.

Answers:

Fairness.

✓ Cache-coherence traffic.

Critical section over-utilization.

Reduced cache-coherence traffic.

Question 3

0 out of 1 p ← OK

What is the output of the following code ?

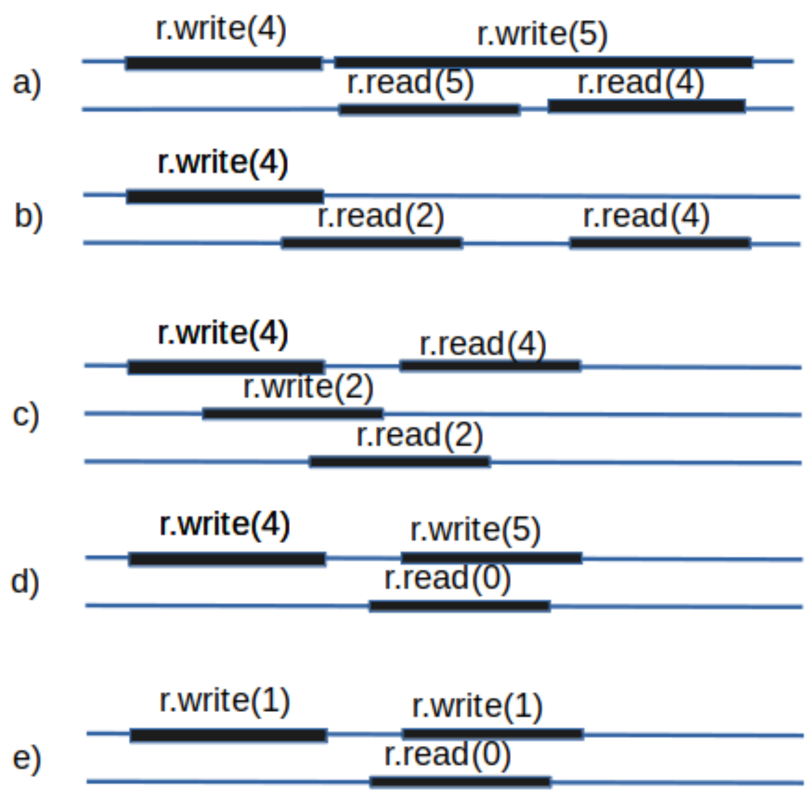
```
public class ManyThreads {
private static class MyThread extends Thread {
public MyThread(String name) {
    super(name);
  }
@Override
public void run() {
    System.out.println(Thread.currentThread().getName());
 }
}
public static void main(String[] args) {
  MyThread myThread = new MyThread("myThread");
  myThread.run();
   }
}
```

Selected Answer:	✗ An exception is thrown.
Answers:	An exception is thrown.
	myThread.
	✓ main.
	None of the above.

Question 4

10 out of 12 points

This question contains 5 sub-parts a), b), c), d), e) and f)



With respect to the concurrent histories depicted above, answer the following questions. You must justify each answer.

a) Is the history of (a) atomic ? (2 mark)

b) Is the history of (a) regular? (2 mark)

c) Is the history of (b) atomic? (2 marks)

d) Is the history of (b) safe? (2 marks)

e) Is the history of (c) atomic? (2 marks)

f) Is the history of (d) regular? (2 marks)

Selected Answer:	a) No. Since the first read returns a 5 (which is the latest value), then the second read should have returned 5 as well b) Yes. The first read returns the new value (5), but the old read returns the old value (4) c) Yes. The first read could be returning an old value, which could have been 2, but the new read is returning the latest value (4) d) Yes. Since the first read is overlapping a write, it is possible that the value being read is in the range of the old value and the value currently being written by the write, which is 4. The second read is returning the latest value (4) as it is not overlapping a write e) No. The first read is returned the new value (2), therefore the second read should've also returned this new value f) No. The read is returning neither value from the writes
Correct Answer:	[None]
Response Feedback:	e-0

Question 5

1 out of 3 points

Explain how the Filter lock may allow some threads to overtake others an arbitrary number of times. (3 marks)

Selected Answer:	At first, threads start at the first level, but as threads progress up through the levels, they can be easily overtaken by one another. This can happen when you have a thread who is a victim in their level and there is another thread in the level above. A new thread can come in can see that the level above is now open, so even though it is the victim of the level, the first condition of spinning in the filter lock no longer holds. So this thread will immediately jump to the next level even though it was not the first thread to enter the level. The other thread may jump to the next level shortly afterwards. This scenario can happen multiple times.
Correct Answer:	[None]
Response Feedback:	[None Given]

Question 6

4 out of 5 points

This question has two parts a) and b).

Given the following snippet of code.

```
public class TestLock{
 .
 .
 boolean available = true;

 static void acquire(){
       while(!available){}
       available = true;
 }
 static void release(){
       available = false;
 }

 do{
      acquire();
      // enter the critical section
      release();
      // remainder of the section.
}while(true);

}
```

a) Does the code guarantee starvation freedom. Justify. (2 marks)

b) Does the code guarantee mutex behaviour. Justify. If not what changes need to be made to guarantee mutex behaviour (3 marks)

Selected Answer:	a) No. The first thread to acquire the lock will set the available to true and proceed into the critical section. But when it releases the lock, it will set available to false. Since acquiring a lock spins on available, specifically the `while (!available),` since the available is now false, and the only way to set it back to true is after the while loop, the other threads that try to acquire the lock will forever spin on the `!available`. Therefore, no thread other than the first thread that came in will make any progress. b) No. There is a possiblity that two threads can see that the lock is available at the same time and then both can set the available to true and proceed into the critical section. This can be fixed by making the available volatile, so that threads will always have the latest value of the variable.
Correct Answer:	[None]
Response Feedback:	a-2 b-2

Question 7

4 out of 7 points

The following question consists of three parts a) , b) and c)

Consider the following class code of a proposed lock named TestLock. Answer the questions that follow.

```
class TestLock implements Lock {
    private int myChance;
    private boolean busy = false;
    public void lock() {
        int myTurn = ThreadID.get();
        do {
            do {
                myChance = myTurn;
            } while (busy);
            busy = true;
        } while (myChance != myTurn);
    }
    public void unlock() {
        busy = false;
    }
}
```

a) Does the proposed lock guarantee Mutual Exclusion. Justify (3 marks)

b) Does the proposed lock guarantee Deadlock Freedom. Justify (2 marks)

c) Does the proposed lock guarantee Starvation Freedom.Justify (2 marks)

Selected Answer:	a) Yes. The lock guarentees deadlock freedom so it therefore guarantees mutual exclusion. If two threads try to acquire the lock at the same time, the myChance variable will alternate between the two threads' ids. But when busy is set to false, both threads will exit the first do while and set the busy back to true. But only one of them would be equal to the value in myChance, this thread will then proceed into the critical section. The other thread will return back to the second do while loop. b) Yes. At least one thread will always make it out the second do while loop. At maximum only one thread will always make it out because there is only one thread that can be currently set to the myChance. No thread depends on another thread to make progress. c) No. Any new thread can come in and set the myChance to itself and at the first instance of busy being true, it can overtake the other threads and proceed into the critical section because the myChance is currently set to that thread. This forces the other threads to spin again, regardless of how long they were waiting to acquire the lock
Correct Answer:	[None]
Response Feedback:	a-3 b-0 c-1

Question 8

1 out of 1 points

In which of the following real-world application areas is the use of locks relevant ?

Selected Answer:	✓ All of the above.
Answers:	Operating systems.
	Web applications.
	Distributed architecture.
	✓ All of the above.

Question 9

4 out of 6 points

This question has sub-parts a) and b)

a) Some non-Queue based locks are vulnerable to cache-coherence. In your own words explain the concept of cache-coherence including cache coherence traffic. (3 marks)

b) In your own words, describe the limitations of the TASLock and explain how the TTASLock overcomes these limitations. (3 marks)

Selected Answer:

a) Cache coherence is the process of ensursing that the cache that is related to certain processes all are all consistent and can detetct synchronization conflicts. Cache coherence traffic is when multiple processes spin on the same location, which causes bus traffic on every access or update
b) One of the limitions of the TASLock is that the lock's getAndSet() invalidates the values in cache for all threads. Even for the ones that aren't trying to even aquire the lock. The TTAS is an improvement because it has a much better management of cache coherence, by repeatedly spinning on a value in cache and only updating the cache when the lock is free

Correct Answer: [None]

Response Feedback:

a-2

b-2

Question 10

0 out of 1 points

Consider the following execution history H of a FIFO queue objects **p** and **q**

```
H =  B p.enq(3)
     A q.enq(5)
     B p:void
     B q.deq()
     A q:void
     B q:5
```

Selected Answer:	✗ The execution history is not well formed.
Answers:	The execution history is not well formed.
	✓ The execution history is well formed.
	An exception will be thrown.
	None of the above.

Question 11

3 out of 5 points

This question has two parts a) and b)

Fairness is one of the properties of the filter lock.

a) Given concurrent threads, thread A and thread B, provide a description of how the filter lock enforces fairness. (3 marks)

b) Why is the fairness provided by the filter lock considered as weak. (2 marks)

Selected Answer:

a) The Filter lock does not enforce fairness. If threads A and B both start at the first level then they will both progress to the next level at the same time. Thread A is the first thread to enter the level, and will stay in that level if there is another thread in the level above it. After the level above clears and thread B enters the level, there is a possibility it may overtake thread A. Alternatively, if the level above thread A clears out before thread B arrives, then thread A will progress to the next level without an issue.
b) Because threads can be easily overtaken by each other. There is no guarantee that the first thread in will be the first thread out.

Correct Answer: [None]

Response Feedback:

a-1 (By the late arriving thread being a victim it fairly defers to threads that arrived earlier. It also enforces fairness by splitting the lock() method into 2 sections of code. A doorway section and and a waiting section. If thread A arrives at the doorway ahead of thread B it will enter the critical section first.)

b-2

Question 12

1 out of 1 points

What happens when a thread holding a lock calls the wait() method ?

Selected Answer:

✓ The thread will immediately release the lock of that object and enter into the waiting state.

Answers:

✓ The thread will immediately release the lock of that object and enter into the waiting state.

The thread will immediately release all locks it is holding and then enter into the blocking state.

The thread will immediately enter into the waiting state without releasing any locks.

None of the above.

Question 13

0 out of 2 points

Which of the following are true about the unbounded LockFreeQueue

Selected Answers:	✓ Threads may encounter half finished enqueue methods and may need to complete the tasks.
Answers:	The enqueue method is partial.
	✓ Threads may encounter half finished enqueue methods and may need to complete the tasks.
	✓ The linearization point of the enqueue method is when it calls the compareAndSet
	The deq() method makes sure that the queue is not empty in the same way by checking that the next field of the tail node is not null.

Question 14

3 out of 6 points

This question consists of two parts a) and b)

```
int num = 5; // line 1
num++; // line 2
```

a) Given the code snippet above. Is **line 2** an atomic operation. Justify (3 marks)

b) A vaccination center operates a protocol that involves people being put in front of the queue as they arrive. One can only be vaccinated if they are at the front of the queue. Discuss how atomic registers may (or may not) be used to simulate this protocol. (3 marks)

Selected Answer:	a) No. Multiple concurrent threads can still modify x, there is no form of mutual exclusion to ensure that the incrementation only happens one at a time b) Atomic registers can be used to create a FIFO queue. This queue can be used to always retrieve the person at the front of the queue
Correct Answer:	[None]

Response Feedback:	a-3 b-0 (No atomic registers have a consensus of 1 and cannot be used to construct a LIFO (stack))

Question 15

1 out of 1 points

If getAndSet() is a broadcast, which of the following is true?

Selected Answer:	✓ All of the above
Answers:	Unlocking is affected.
	All threads are affected.
	All processors are affected.
	✓ All of the above

Question 16

2 out of 2 points

Discuss how the random BackOff algorithm detects high contention compared to the LockFreeStack algorithm. (2 marks)

Selected Answer:	The BackOff algorithm detects high contention when it sees that a lock is free but when it tries to acquire the lock, another thread came in and acquired it. LockFreeStack detects high contention when it tries to push or a pop a value from the stack and the compareAndSet calls in the methods return a false, meaning another thread is trying to pop/push.
Correct Answer:	[None]
Response Feedback:	[None Given]

Question 17

1 out of 1 points

The **synchronize** keyword can be applied to

Selected Answer:	✓ methods and blocks.
Answers:	✓ methods and blocks.
	variables and classes.
	methods and classes.
	variables and methods.

Question 18

1 out of 1 points

Thread Local variables such as **ThreadLocal<QNode> myNode** field in the **MCS** Lock generate coherence traffic and require synchronization.

Selected Answer:	✓ False
Answers:	True
	✓ False

Question 19

0 out of 1 points

What brings a thread out of its waiting state?

Selected Answer:	✗ If the waiting time expires.
Answers:	If the waiting thread gets a notification.
	If the waiting time expires.
	If the waiting thread gets interrupted.
	✓ All of the above.

Question 20

2 out of 2 points

Consider the following lines of code.

```
public static synchronized void printerOne(int number){
    System.out.println(number--);
    System.out.println(++number);
}
```

If the method was to re-written as follows:

```
public static synchronized void printOne(AtomicInteger number){
    System.out.println(number.x());
    System.out.println(number.y());
}
```

Which of the following can replace methods x() and y() in the printOne() method to produce the same results as printerOne()?

Selected Answer:	✓ getAndDecrement() and incrementAndGet().
Answers:	decrementAndGet() and getAndIncrement().
	decrementAndGet() and incrementAndGet().
	getAndDecrement() and getAndIncrement().
	✓ getAndDecrement() and incrementAndGet().

1 out of 1 points

Question 21

How many concurrent threads can simultaneously lock a single object at any given time ?

Selected Answer:	✓ one.
Answers:	hashcode dependent.
	two.
	multiple.
	✓ one.

Question 22

0 out of 0 points

The University of Pretoria commits itself to produce academic work of integrity. I affirm that I am aware of and have read the Rules and Policies of the University, more specifically the Disciplinary Procedure and the Tests and Examinations Rules, which prohibit any unethical, dishonest or improper conduct during tests, assignments, examinations and/or any other forms of assessment. I am aware that no student or any other person may assist or attempt to assist another student, or obtain help, or attempt to obtain help from another student or any other person during tests, assessments, assignments, examinations and/or any other forms of assessment.

Selected Answer:	✓ True
Answers:	✓ True
	False

Question 23

1 out of 1 points

What is not true about Readers and Writers.

Selected Answer:	✓ Synchronization among readers is necessary.
Answers:	✓ Synchronization among readers is necessary.
	Synchronization among writers is necessary.
	Readers, return the object's state without modifying the object.
	Writers modify the object's state.

Question 24

1 out of 1 points

Calling start() on a thread that is already running will throw an IllegalThreadSt ateException.

Selected Answer:	✓ True
Answers:	✓ True
	False

Question 25

3 out of 4 points

This question consist of two parts a) and b).

Bottleneck and contention are inherent problems associated with a LockFreeStack.

a) Explain how each of these may arise. (2 marks)

b) Can a semaphore be used to solve the bottleneck problem ? Justify (2 marks)

Selected Answer:	a) Contention arises when multiple threads are trying to push or pop something from the stack. A bottleneck arises because method calls can only proceed one after the other as ordered by the compareAndSet() b) No. Because the compareAndSet() would still create a bottleneck. And the stack will no longer be lock free
Correct Answer:	[None]
Response Feedback:	a-2 b-1

Question 26

1 out of 1 points

Synchronized variables prevent concurrent acess to variables.

Selected Answer:	✓ False
Answers:	True
	✓ False

Question 27

1 out of 1 points

Which of the following is not a characteristic of a CLH lock ?

Selected Answer:	✓ Requires knowledge of thread count.
Answers:	✓ Requires knowledge of thread count.
	Maintains fairness.
	Reduces cache-coherence traffic.
	None of the above.

Question 28

0 out of 2 points

A webserver with an old processor spends 70% of its time searching. This is deemed inefficient and the IT manager decides to integrate a new CPU which is 20 times faster than the old processor. Calculate the speed-up gained by the new architecture.

Selected Answer:	1.02
Correct Answer:	[None]
Response Feedback:	[None Given]

Question 29

4 out of 6 points

This question is in two parts a) and b)

```
private boolean validate(Node pred, Node curr) {
Node node = head;
while (node.key <= pred.key) {
if (node == pred)
return pred.next == curr;
node = node.next;
 }
 return false;
}
```

a) The given snippet of code outlines the *validate* method associated with the OptimisticList class. The class also contains the contains() method.Justify the necessity of the *validate* method. (3 marks)

b) In the Fine-grained synchronization approach, methods acquire locks and apply them in an ordered manner down the list of the data structure.

Despite this, deadlocks still occur.

Present a case supporting or contradicting the highlighted statement. (3 marks)

Selected Answer:	a) The validate method is necessary so that a thread can make sure that it did not traverse parts of a list that were missing. If the validation succeeds, the method can continue as normal, but if it fails it has to restart b) The fine-grained synchronization approach is starvation free, it is therefore deadlock free. If a thread attempts to lock the head, it will eventually succeed and eventually all the locks held in front of the head will also be released. Therefore the thread will be able to acquire all the locks it needs
Correct Answer:	[None]
Response Feedback:	a-2 (where is the explanation regarding contains()) b-2 (because of hand-over transition down the list)

Question 30

1 out of 3 points

```
public class LockFreeStack<T> { // lock implementation
 .
 .
 protected boolean tryPush(Node node){
 Node node = new top.get();
 node.next = oldTop;
 return(top.getAndSet(true));
 }
 public void push(T value){
 Node node = new Node(value);
 while(true){
  if(tryPush(node)){
     return;
     }else{
     backoff.backoff();
     }
  }
 }
```

Given lockFree Stack A is initialized with three elements **[a,b,c]** where element c is at the top of the Stack. Describe the final state of A that would result from executing concurrent threads in the following order: push(d) (using the given push method) followed by pop()(which just returns the top element). (3 marks)

Selected Answer:	The push(d) will push the d onto the stack. But since we are not using compareAndSet in the tryPush, pop is the only method that has the compareAndSet. What will happen is that d will be pushed onto the stack, but the pop will not be successful if these two methods are concurrent. So the final state of A would be [a,b,c,d], where d is the first element
Correct Answer:	[None]
Response Feedback:	1- for compareAndSet() 0-Code wont compile. The tryPush method is wrongly implemented.

Wednesday, November 30, 2022 10:22:21 AM SAST