UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Engineering, Built Environment and IT
Department of Computer Science

Concurrent Systems
COS 226

Semester Test 2 (ST2)

12 October 2020 (17:30 to 19:00)

Examiners

Internal: Ms Ntombikayise Banda and Mr Theo Naidu

Instructions

1. Read the question paper carefully and answer all the questions.
2. The assessment opportunity comprises of **7** questions on **8** pages.
3. **1.5** hours have been allocated for you to complete this paper. An additional 30 min has been allocated for you to download the paper and upload your answer document.
4. This paper is **take home** and is subject to the University of Pretoria Integrity statement provided below.
5. You are allowed to consult any literature.
6. You are **not** allowed to discuss the questions with anyone.
7. If you have any queries when writing the paper, post them in good time on the **Semester Test 2 Discord channel**. You are also allowed to direct message Ms Banda or Mr Naidu on Discord should you wish for privacy. NO emails during the test will receive a response. A Collaborate LIVE session will also be available to answer queries.
8. Write your answers in a separate document (eg. using a word processor or handwritten and scanned) and submit the document as a **single PDF** attachment. An upload slot will be open on the ClickUP module page under the **Semester Test 2 - Question paper and upload** menu option for the duration of the test (17:30 to 19:00), and then for an additional 30 minutes to give you enough time to download this paper and upload your PDF. The submission deadline is therefore **19:30**. **No late submissions will be accepted**.
   - Please make sure that your **name and student number are clearly visible** in the document you upload.
   - You need to provide photographic evidence of yourself in the form of a student card, identity document, passport document, or driver's licence. This only needs to be shown on the first page of your answer script.
9. **Marks per question**

Question:	1	2	3	4	5	6	7	Total
Marks:	12	4	11	8	4	5	16	60

Integrity statement:
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.

Multiple choice questions

1. For each of the questions that follow, choose the most appropriate option.

   (*Note:* possible answers are denoted by letters A, B, C, etc.)

   (a) Which correctness condition does a regular register follow? (1)

   A. Sequential consistency
   B. Quiescent consistency
   C. Linearizability
   D. Mutual exclusion

   (b) Given a multi-reader multi-writer (MRMW) atomic register `r` with a capacity of 4 and the following array table state: (1)

a_table =

stamp	value
2	670
0	0
1	510
0	0

   What would be the result of a `r.write(720)` method call by a writer thread with id 2?

   A.

2	670
0	0
1	720
0	0

   B.

2	720
0	720
1	720
0	720

   C.

2	670
0	0
3	720
0	0

   D.

2	670
0	720
1	510
0	0

   (c) Given a multi-reader multi-writer (MRMW) atomic register `r` with a capacity of 4 and the following array table state: (1)

a_table =

stamp	value
3	670
2	110
1	510
3	320

   What would be the result of an `r.read()` method call by a reader thread with id 1?

   A. 670
   B. 110
   C. 510
   D. 320

(d) A regular M-valued MRSW register (with M = 4) represents its values using unary notation. Which of the following options shows the correct representation of the value 3? (1)

A. | 0 | 0 | 1 | 1 |

B. | 0 | 0 | 0 | 1 |

C. | 0 | 0 | 0 | 1 | 0 |

D. | 0 | 0 | 1 | 0 |

(e) Which of the following is NOT an example of synchronization primitives? (1)

A. CLH lock

B. MRSW atomic register

C. Binary semaphore

D. Condition object

(f) Consider an atomic register `r` with an initial value of **0**. What is the output of the following code snippet? (1)

```
r.compareAndSet(1, 2);
r.compareAndSet(0, 1);
System.out.println(r.get());
```

A. 0

B. 1

C. 2

D. Compiler error

(g) Consider an atomic register `r` with an initial value of **1**. What is the output of the following code snippet? (1)

```
r.compareAndSet(1, 0);
r.compareAndSet(0, 2);
System.out.println(r.get());
```

A. 0

B. 1

C. 2

D. Compiler error

(h) Which of the following RMW (*read-modify-write*) register methods is an identity function? (1)

A. `getAndSet(...)`

B. `getAndIncrement()`

C. `set(...)`

D. `get()`

(i) Suppose that thread C executes line 4 in the code snippet below and suspends itself. After some time, an earlier thread A modifies the `property` value to `true` and signals all waiting threads. What is the next line that thread C will execute when it is awakened? (1)

```
1 mutex.lock();
2 try {
3   while(!property)
4     condition.await();
5   count++; // some implementation
6 } finally {
7   mutex.unlock();
8 }
```

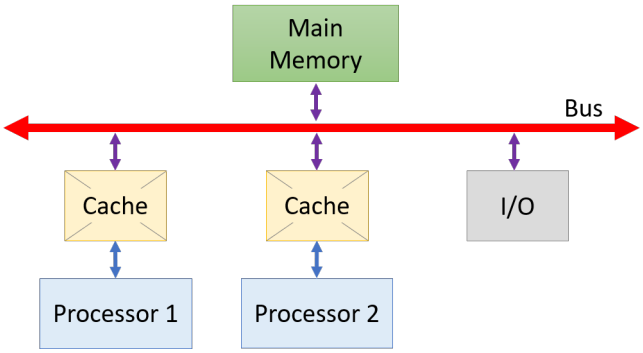
A. line 1

B. line 3

C. line 5

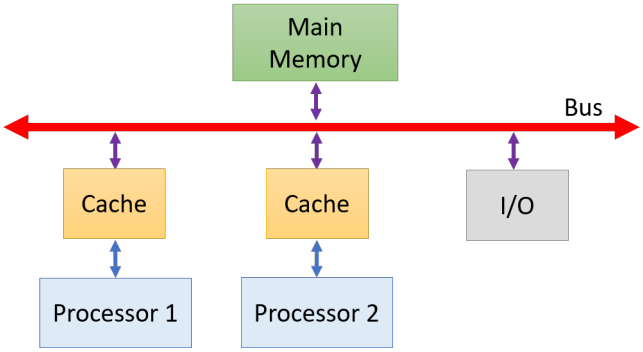
D. line 7

(j) The processors' respective caches in the diagram below are empty. Suppose that Processor 2 executes an operation that requires it to read data from some memory location. Which of the following options will supply Processor 2 with the required data? (1)



A. Processor 1's cache
B. Processor 2's cache
C. Input/Output processor
D. Main memory

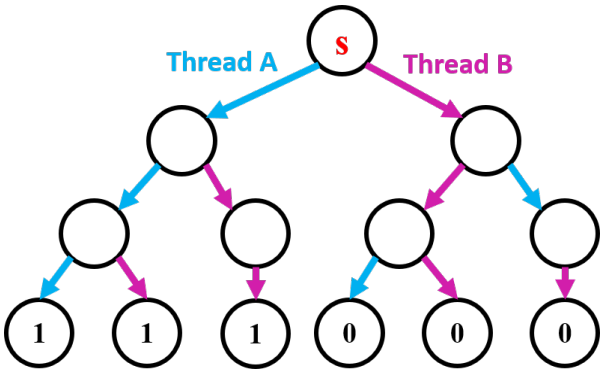
(k) Suppose that the main memory and Processor 1's cache memory have the latest version of data `x`. (1)



If Processor 2 executes an instruction to read data `x`, but cannot find it in its cache, which of the following options will supply Processor 2 with data `x`?

A. Processor 1's cache
B. Processor 2's cache
C. Input/Output processor
D. Main memory

(l) The figure below shows the execution tree of a binary consensus protocol for two threads. Thread A (in *blue*) proposes the value 1 and Thread B (in *purple*) proposes the value 0. (1)



Which of the following options contains a list of all protocol states that are applicable to node s.

A. initial state
B. initial state, bivalent state
C. initial state, univalent state
D. initial state, bivalent state, critical state

Long questions

Chapter 4

2. Consider the implementation of a regular MRSW register class below.

```
1  public class RegMRSWRegister implements Register<Byte> {
2    private static int RANGE = Byte.MAX_VALUE - Byte.MIN_VALUE + 1;
3    boolean[] r_bit = new boolean[RANGE];
4
5    public RegMRSWRegister(int capacity) {
6      for (int i=1; i < r_bit.length; i++)
7        r_bit[i] = false;
8      r_bit[0] = true;
9    }
10
11   public void write(Byte x) {
12     r_bit[x] = true;
13     for (int i=x-1; i >= 0; i--)
14       r_bit[i] = false;
15   }
16
17   public Byte read() {
18     for (int i=0; i < RANGE; i++)
19       if (r_bit[i])
20         return i;
21     return -1;   // impossible
22   }
23 }
```

(a) Suppose the lines of the `write(...)` function are reordered as follows: (2)

```
public void write(Byte x) {
  for (int i=x-1; i >= 0; i--)
    r_bit[i] = false;
  r_bit[x] = true;
}
```

Provide a brief analysis of what could go wrong during a `read()` call that overlaps with a `write()` call?

(b) If the `write(...)` function were to be changed to the following: (2)

```
public void write(Byte x) {
  r_bit[x] = true;
  for (int i=0; i <= x-1; i++)
    r_bit[i] = false;
}
```

Provide a brief analysis of what could go wrong during a `read()` call that overlaps with a `write()` call?

Chapter 5

3. **Concurrency and Consensus**

(a) What are the advantages of wait-free concurrent objects (e.g. WaitFreeQueue) over lock-based concurrency objects (e.g. LockedQueue)? (2)

(b) Study the code below of an adapted queue consensus protocol. (3)

```
1  public class AdaptedQueueConsensus<T> extends ConsensusProtocol<T> {
2    Queue queue;
3    private static final int WIN = 0;
4    private static final int LOSE = 1;
5    private volatile int winner = -1;
6
7    public AdaptedQueueConsensus() {
8      queue = new Queue();
9      queue.enq(WIN);
10     queue.enq(LOSE);
11     queue.enq(LOSE);
12   }
13
14   public propose(T value) {
15     super.propose(value);
16   }
17
18   public T decide(T value) {
19     int status = queue.deq();
20     int i = ThreadID.get();
21
22     if (status == WIN) {
23       winner = i;
24       return proposed[i];
25     }
26     else
```

```
27          return proposed[winner];
28      }
29  }
```

Discuss whether this protocol successfully solves a three-thread consensus.

(c) Consider the following hypothetical scenario. In its drive to building a "smart city", the Gautrain management board upgrades its train system into a **distributed** automatic driverless system. The automatic driving system uses defined time rules to control departures and movements between stations. Each train has its own clock, which may suffer from significant clock drifts after a period of time (i.e. the clock gradually desynchronizes from a reference clock). To reduce chances of collisions, the trains need to synchronize their clocks periodically though a consensus protocol. (6)

Implement a consensus protocol called `ClockConcensus` (with a capacity of n concurrent threads) with the following rules:

- Each train (represented by a thread) proposes its current clock's time through the `propose(...)` method call. Assume that time is represented as a **long integer** value.
- The agreed-upon time is determined through:
  - The "majority-wins" decision, with the most frequently occurring proposed time amongst contending threads being the deciding / winning value.
  - If all trains propose distinct values, the first train (thread) to call `decide(...)` wins.
- Assume that all threads would have proposed their values before any thread calls `decide(...)`.
- To aid with your implementation, assume that the function
  `public static Long computeMode(Long[] arr);`
  has been defined elsewhere and can be called directly from your class implementation. The `computeMode` function computes and returns the most common number in the given input array. If the input array contains no duplicate data points, the function returns -1.
- There is no need to display output or logging statements.

Chapter 7

4. **Spin Locks**

(a) Suppose that a `Counter` object with an initial value of 0 uses a Peterson lock to enforce mutual exclusion on the `Counter`'s `getAndIncrement()` method. After 5000 concurrent executions of the `getAndIncrement()` method, it is possible for the `Counter` object's final value to be below 5000. Explain the cause of this discrepancy and what measures can be implemented to prevent this? (3)

(b) Discuss the concept of an "invalidation storm" with respect to the TAS and TTAS locks (i.e. what it is and how it occurs)? In your discussion, explain the effect that the invalidation storm has on the performance of the two locks, and mention the mechanisms that the TTAS lock has in place to reduce the effect of invalidation storms. (5)

Chapter 8

5. MongoDB uses a readers-writer lock that allows concurrent readers shared access to a database or collection, and an exclusive locking mode for write operations. The code below shows a hypothetical implementation of MongoDB's `insert(...)` and `count()` functions. (4)

```
public class Collection {
  ArrayList<Document> documents;
  ReadWriteLock rwLock;

  public Collection() {
    documents = new ArrayList<Document>();
    rwLock = new SimpleReadWriteLock();
  }

  // Inserts a new document into the database collection
  public void insert(Document newDoc) {
    documents.add(newDoc);
  }

  // Returns the number of documents in this collection
  public int count() {
    return documents.size();
  }
}
```

Make use of the Readers-Writers lock object `rwLock` to enforce appropriate thread access to the `insert(...)` and `count()` functions. (*Note:* you only need to write code for the two functions)

6. The code segment below is of the inner read lock of the `SimpleReadWriteLock` class.

```
1  private class ReadLock implements Lock {
2    public void lock() {
3      lock.lock();
4      try {
5        while (writer) {
6          condition.await();
7        }
8        readers++;
9      } finally {
10       lock.unlock();
11     }
12   }
13
14   public void unlock() {
15     lock.lock();
16     try {
17       readers--;
18       if (readers == 0) {
19         condition.signalAll();
20       }
21     } finally {
22       lock.unlock();
23     }
24   }
25 }
```

   (a) If the `ReadLock` class allows multiple reader threads to access the critical section, what is the purpose of the `lock()` method call in line 3? (1)

   (b) Use semaphore concepts to restrict the number of readers that can access the critical section at the same time to some `capacity` value. You only need to adapt the `lock()` function of the `ReadLock` class. If you define extra class variables, state the scope of the variables (in other words, where the variables are declared). (4)

7. Consider the code below for the inner `ReadLock` and `WriteLock` classes of the **Fair Readers-Writers Lock** and answer the questions that follow.

```
1  private class ReadLock implements Lock {
2    public void lock() {
3      lock.lock();
4      try {
5        while (writer) {
6          condition.await();
7        }
8        readAcquires++;
9      } finally {
10       lock.unlock();
11     }
12   }
13
14   public void unlock() {
15     lock.lock();
16     try {
17       readReleases++;
18       if (readAcquires == readReleases) {
19         condition.signalAll();
20       }
21     } finally {
22       lock.unlock();
23     }
24   }
25 }
26
27 private class WriteLock implements Lock {
28   public void lock() {
29     lock.lock();
30     try {
31       while (writer) {
32         condition.await();
33       }
34       writer = true;
35       while (readAcquires != readReleases) {
36         condition.await();
```

```
37        }
38      } finally {
39        lock.unlock();
40      }
41    }
42
43    public void unlock() {
44      writer = false;
45      condition.signalAll();
46    }
47  }
```

(a) What "fairness" does the Fair Readers-Writers Lock provide? (2)

(b) Explain the role of the `readAcquires` and `readReleases` variables in providing fairness. That is, how are the variables used to help achieve fairness? (2)

(c) Why is a lock not used in the `WriteLock`'s `unlock()` function? (2)

(d) In order to reduce unnecessary contention from awakening both reader and writer threads when there is a waiting writer thread, how would you go about adapting the `ReadLock` and `WriteLock` classes to allow reader and writer threads to only awaken the necessary threads? (10)

If you define extra class variables, state the scope of the variables (in other words, where the variables are declared). Give meaningful names to any variables that you declare.