NATIONAL UNIVERSITY OF SINGAPORE

SCHOOL OF COMPUTING

SEMESTER 2 (2021/2022)

FINAL ASSESSMENT FOR

CS4231: Parallel and Distributed Algorithms

27 APR 2022 **09:00am to 11:00am**

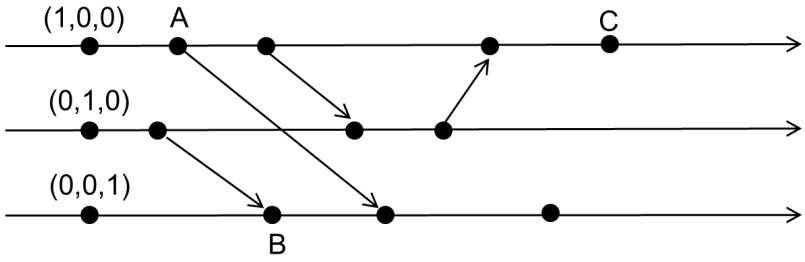
INSTRUCTIONS

1. This assessment contains **TWELVE** problems and comprises **SIX** printed pages, including this page.
2. Answer **ALL** questions within the space provided in this booklet.
3. This is an **Open Book** assessment.
4. This assessment should be completed by each student individually, without help from anyone else or from the Internet.
5. When answering the questions, you can invoke lemmas/theorems that we have already covered in class, unless the question prohibits this. When you do so, you should first state the lemmas/theorems that you wish to invoke, and then explain how you want to invoke it in your context. You CANNOT invoke lemmas/theorems/results from anywhere else (i.e., those not covered in class). Answering the questions will not require you to draw any diagram. But you still can draw diagram in Word if you prefer. You are NOT allowed to insert a photo or scanned image/text into the Microsoft word file. You can use standard C/Java mathematical operators. For example, "*" means multiplication, and "^" means exponentiation. You can also define additional functions. For example, you can define the function sqrt(x) to return the square root of x.
6. At the end of the assessment, you should name the Word file using your student number (e.g., "A0123456X.doc") and then upload the Word file to Luminus. The submission MUST be in Word format. You are NOT allowed to upload a photo or scanned copy of the Word document --- such submissions will get zero mark. You should submit once and ONLY ONCE. If you want to update, you should delete your old submission and then submit a new one.
7. Write your student number (e.g., "A0123456") below. Do not write your name.

STUDENT NO: _____

Problem 1. (0 marks)
Have you double-checked that you typed your student number correctly on the first page?
Answer (yes/no): _____

Problem 2. (6 marks)



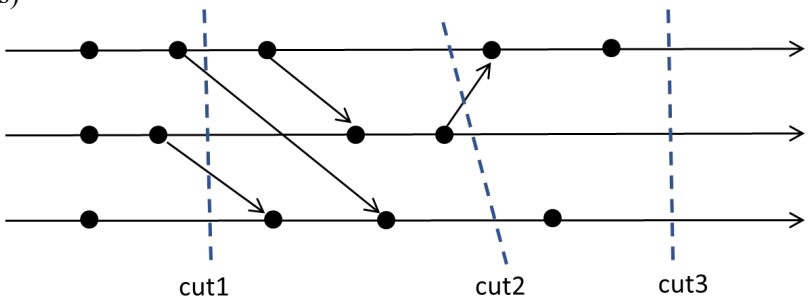
Consider the above execution of a certain distributed system. Imagine that we maintain vector clocks in the system so that each event gets a vector clock value. The vector clock value of the first event on each process is already indicated in the figure.

The vector clock value of the event A should be _____. (No further explanation needed.)

The vector clock value of the event B should be _____. (No further explanation needed.)

The vector clock value of the event C should be _____. (No further explanation needed.)

Problem 3. (6 marks)



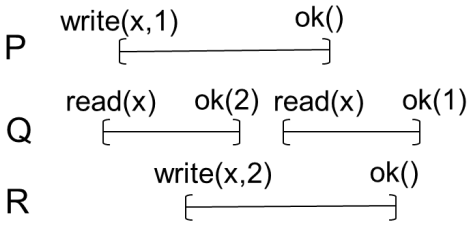
Consider the above execution of a certain distributed system. Here cut1, cut2, and cut3 each correspond to a consistent global snapshot.

How many messages are "on-the-fly" in the consistent global snapshot corresponding to cut1? _____

How many messages are "on-the-fly" in the consistent global snapshot corresponding to cut2? _____

How many messages are "on-the-fly" in the consistent global snapshot corresponding to cut3? _____

Problem 4. (6 marks)



in the above, x is a shared register with initial value being 0

Consider the above history of a certain execution of a certain parallel system with 3 processes P, Q, and R.

Does x in this particular history satisfy the definition of **atomic register**? (answer "yes"/"no"): _____

Does x in this particular history satisfy the definition of **regular register**? (answer "yes"/"no"): _____

Does x in this particular history satisfy the definition of **safe register**? (answer "yes"/"no"): _____

Problem 5. (4 marks)
Alice is trying to implement the following parallel algorithm for two processes P0 and P1 (in the code, obj1 is a shared (Java-style) monitor object):

P0's code
… (some arbitrary code)
synchronized (obj1) { obj1.wait(); }
… (some arbitrary code)

P1's code
… (some arbitrary code)
synchronized (obj1) { obj1.notify(); }
… (some arbitrary code)

But Alice's parallel system only supports semaphores, and not monitors. So Alice modifies the algorithm to be (in the code, obj2 is a shared semaphore object, and the red part is the modified part):

P0's code
… (some arbitrary code)
obj2.P();
… (some arbitrary code)

P1's code
… (some arbitrary code)
obj2.V();
… (some arbitrary code)

Explain why Alice's modification can potentially cause the modified algorithm to behave differently from the original algorithm. You should explicitly describe a scenario where the difference arises.

Problem 6. (4 marks)
Alice has a distributed algorithm for 3 nodes P1, P2, and P3. Assume that each node has a perfect physical clock with no error. The algorithm starts at time t, and ends at time t+99. (All time values in this problem are in "seconds".) At time t+10, t+20, t+30, …, t+90, each node in this algorithm will send a message to each of the remaining 2 nodes (which will be 2 separate messages). We assume that the time needed to do such sending is negligible. This means that there are total 9*2*3 = 54 messages in the execution. Assume that message propagation delay is always exactly 3 second, and assume a message is always delivered immediately upon receipt. Let S be the set of messages received by P3. For two different messages m1 and m2 in S, we say that m2 is after m1 in "causal order", if and only if the send event of m1 happened before the send event of m2. Let m be the message sent by P1 to P3 at time t+10. How many messages in S are after m in "causal order"? Justify your answer.

Problem 7. (4 marks)
Consider the Chang-Roberts algorithm for leader election on rings. Assume that the ring has n nodes, where n >= 100. We learned in class that the best-case message complexity for this algorithm is 2n-1, while the worst-case message complexity is n*(n+1)/2. Construct a ring such that the message complexity of this algorithm, when running on this ring, is 4n+x. Here you are allowed to pick x to be any single (positive or negative) value that you like, as long as x is a constant that does not depend on n. You answer should be general, and should cover all n values. You need to i) clearly describe your construction, ii) indicate your chosen x value, and ii) explain why the message complexity is 4n+x.

Problem 8. (6 marks)
Consider the distributed consensus problem where nodes are reliable, communication channels can drop messages arbitrarily, and the timing model is synchronous. Each process has a binary input value. On every process P, there is now a new magical device that, at the end of round 2, tells P which of those messages sent by P in round 1 and round 2 were(was) dropped by the communication channel. Within this new context, design a **deterministic algorithm that takes total 2 rounds**, for solving the distributed consensus problem **for 2 processes**. In your answer, you should i) describe high-level intuitions on how your algorithm works, ii) give complete pseudo-code for your algorithm, and iii) give rigorous proof showing that your algorithm achieves all the properties needed (i.e., termination, agreement, validity). In particular, validity here means that i) if all nodes start with 0, then decision should be 0; ii) if all nodes start with 1 and if no message is lost throughout the execution, then decision should be 1; iii) otherwise nodes are allowed to decide on anything except that they should still satisfy the agreement property. If you feel that it is impossible to design such a protocol, then provide an impossibility proof.

Problem 9. (6 marks)
Consider the self-stabilizing spanning tree construction algorithm learned in class. Imagine that we run this algorithm on a ring of n nodes, where n >= 99, n is odd, and (n-5)/2 is also odd. The nodes have ids 1, 2, 3, …, n, clockwise along the ring. Node 1 is the tree root. Imagine that the algorithm has already stabilized, in the sense that the spanning tree has been constructed and the states (i.e., values of the **parent** and **dist** variables) of the nodes no longer change any more. Now the topology changes as following: A new edge is created to connect the node with id 1 and the node with id (n-1)/2, and a second new edge is created to connect the node with id 1 and the node with id (n+5)/2. After adding these two new edges, the algorithm will eventually construct a new spanning tree. A node is called "affected" if the final value of its local **dist** variable in the new spanning tree is different from before (i.e., in the old spanning tree). How many nodes are "affected"? Justify your answer.

Problem 10. (6 marks)
Consider the setting under which the FLP theorem is proved. Let n denote the total number of nodes. In our class, the FLP theorem was proved via a contradiction. Namely, we assumed there exists some (black-box) deterministic algorithm A for solving distributed consensus, and eventually obtained a contradiction. The first step in our proof was the following lemma:

Lemma 1: For any deterministic algorithm A for solving distributed consensus, there must exist a bivalent initial state.

Recall that there are 2^n initial states. Alice hopes to prove the following strengthened version of the lemma:

Alice's Lemma: For any deterministic algorithm A for solving distributed consensus, there must exist at least x bivalent initial state(s).

Obviously, Alice's Lemma holds for $x=1$. But Alice wants to prove her lemma for as large an x as possible. What is the maximum value of x, for which you can prove Alice's Lemma? (If your maximum x is not a function of n, then simply write that value. Otherwise you may write your maximum x as a function of n.)
Your answer: My maximum x is _____.

Justify your answer by proving Alice's Lemma for this maximum x value that you have indicated. You do NOT need to show that it is impossible to further increase x. Hint: If you prove Alice's Lemma by finding x bivalent initial states, then you need to explicitly prove that these x states are all distinct (namely, no duplicates).

Problem 11. (6 marks)
Consider the distributed consensus problem where nodes can experience crash failures, all communication channels are reliable, and the timing model is asynchronous. Let n denote the total number of nodes, and let f denote the maximum number of crash failures. We know that $n >= 8$ and $f = 3$. Let $x(1)$ through $x(n)$ denote the binary input values to the n nodes, respectively. Define $y = x(1)+x(2)+x(3)+...+x(n)$. It is guaranteed that y is an even number. In such a context, do you feel you can design a deterministic algorithm for solving distributed consensus? Justify your answer by:

- EITHER providing a rigorous impossibility proof
- OR i) describe high-level intuitions on how your algorithm works, ii) give complete pseudo-code for your algorithm, and iii) give rigorous proof showing that your algorithm achieves all the properties needed (i.e., termination, agreement, validity)

In case you feel that your answer depends on the value of n, you may explain when you can design an algorithm and when you cannot, and then justify your answer accordingly.

Problem 12. (6 marks)
Consider the distributed consensus problem where the timing model is synchronous, nodes may experience byzantine failures, and communication channels are reliable. Let n be the total number of processes, and f be the **exact number** of processes that experience byzantine failures. In class, we learned the following algorithm, which works for n = 4f + 1 (or f = (n-1)/4):

```
Code for Process i:
1.      V[1..n] = 0; V[i] = my input;
2.      for (k = 1; k ≤ f+1; k++) {    // (f+1) phases
3.              send V[i] to all processes;
4.              set V[1..n] to be the n values received;
5.              if (value x occurs (> n/2) times in V) proposal = x;
6.              else proposal = 0;

7.              if (k==i) send proposal to all; // I am coordinator
8.              receive coordinatorProposal from the coordinator

9.              if (value y occurs (> n/2 + f) times in V) V[i] = y;
10.             else V[i] = coordinatorProposal;
11.     }
12.     decide on V[i];
```

Alice is now studying a novel *weak-adversary* model, where a byzantine node will no longer deviate from the algorithm arbitrarily. Instead, now a byzantine node will still run the prescribed algorithm, except that whenever the algorithm requires the node to send a message, the byzantine node will replace each bit in the message with a new random bit (a random bit is 0 with half probability and 1 with the remaining half probability), and then send the message. Each random bit is independent of all other events in the systems (such as values of other random bits, message propagation delay, and node input values). As an example, with this weak-adversary model, at Line 3 of the above algorithm, a byzantine node will send a (potentially different) random bit to each process. (Note that Line 3 involves n messages, each with a single bit. A byzantine node will use a new random bit for each such message. Hence the messages to different nodes may be different.) Line 7 is similar. Under this weak-adversary model, Alice wants to **modify** the above algorithm so that it can tolerate a larger f value (i.e., larger than (n-1)/4). Alice wants the modified algorithm to always terminate within $O(n^2)$ rounds. Alice further wants that the modified algorithm, with at least $1-(1/(n^2))$ probability, guarantees the standard Agreement and Validity requirement (as defined in class) for byzantine consensus. With the remaining $1/(n^2)$ probability, Agreement and Validity can both be violated. **We are only concerned with n values that are sufficiently large, and we only care about the largest f value that we can tolerate for a given n. To rule out trivial solutions, we only consider f values that are in [1, n-10].**

Following are some common probability inequalities, which may or may not be useful for you:

- Consider random events X(1) through X(k), for k >= 1. Regardless of any (potential) correlation among these events, the probability that none of them happening is at least 1-Pr[X(1)]- Pr[X(2)] - … - Pr[X(k)].
- Consider independent binary random variables X(1) through X(k), for k >= 1, where Pr[X(1)=1] = Pr[X(2)=1] = … = Pr[X(k)=1] = 0.5. Define Y=X(1)+X(2)+ … +X(k). Then for all real number c where 0 < c < 1, we have Pr[Y <= (1-c)*0.5*k] <= exp(-c*c*k/6) and Pr[Y >= (1+c)*0.5*k] <= exp(-c*c*k/6).

Please help Alice to achieve her goal. You should first indicate the largest f, as a function n, for which you have a correct algorithm. This largest f value should be within the range of [1, n-10]. Since we only care about the largest f, you should write your answer for example, as f = (n/2) - 15 instead of f <= (n/2) - 15. The mark you get will depend on **how large** your largest f value is, so you should try to tolerate as large an f value as you can. Next clearly explain how to **modify** the above algorithm (if needed, you may refer to the line numbers), and rigorously prove that your modified algorithm achieves the desired properties under such n and f.

END-OF-ASSESSMENT