ECE428 Homework 3

Due: 11:59 p.m. on Wednesday April 3th, 2023

This assignment has 5 questions with 65 points in total. The solutions must be typed, and submitted via Gitlab. However, the diagrams can be hand-drawn. You must acknowledge any sources used to arrive at your solutions, other than the course materials and textbook. All homework assignments are expected to be an individual work, so no collaborations are allowed.

Question 1: Bully Algorithm *[14 points]*

Consider the following modification of Bully algorithm: The initiating node (which is assumed not to fail) sends Election message only to the process with the highest id. If it does not get a response within a timeout, it sends Election message to the process with the second highest id. If after another timeout there is no response, it tries the process with the third highest id, and so on. If no higher numbered processes finally respond, it sends Coordinator message to all the lower-numbered processes.

```
class ModifiedBully:
    def run_election(self, failed_leader = None):
        """
        'failed_leader': previous leader that has failed, to avoid
           sending a message to it
        """
        self.leader = None
        # process pid's in reverse order. self.group
        # is a list of all processes in the group
        for pid in sorted(self.group, reverse=True):
           if pid == failed_leader:
             continue # skip previous leader
           elif pid == self.pid:
             # I am the new leader
             self.leader = self.pid
             for pid2 in self.group:
                if pid2 < self.pid:
                  unicast(pid2, "Coordinator")
             break
           unicast(pid, "Election")
           sleep(TIMEOUT)
           if self.leader is not None:
             break

    def receive_message(self, message):
       if message.contents == "Coordinator":
         self.leader = message.sender
       elif message.contents == "Election":
       # fill in the rest
```

(a) (2 points) Complete the receive_message function (pseudocode is OK).

For the following parts, consider a distributed system with 8 processes that use a modified Bully algorithm for the leader election (including your solution to part (a)). The processes are called $\{ P_1, \ldots, P_8 \}$ with P_i having the PID i. Initially, all 8 processes are alive and P_8 is the leader. Then, P_8 fails, and P_4 detects this, and initiates a new leader election. Assume the one-way message transmission time is T, and the timeout is set using knowledge of T.

(b) (2 points) If no other node fails during the election run, how many total messages will be sent by all the processes in this election run?

(c) (2 points) If no other node fails during the election run, how long will it take for the election to finish?

(d) (2 points) Now assume that immediately after P_4 detects P_8's failure, and initiates the election while P_7 fails. How many total messages will be sent by all processes in this election run?

(e) (2 points) For the above scenario (where P_7 fails right after P_4 initiates the election upon detecting P_8's failure), how long will it take for the election to finish?

(f) (4 points) What are the best and the worst-case turnaround times for Bully algorithm?

Question 2: Leader Election in a Ring *[10 points]*

Consider a system of N processes, $\{ P_1, \ldots, P_N \}$ arranged in a ring. Each process can only communicate to its ring successor, i.e., P_i can only send messages to P_{i+1}, and P_N can only send messages to P_1. We assume that each message is a pair of two 16-bit signed integers, i.e., they can take on values from –32768 to +32767. Assume that $N \leq 1000$, and that there are no failures, and the communication channel delivers all messages correctly and exactly once.

(a) (6 points) In this problem, each process has a 16-bit signed integer zinput x_k ($-32768 \leq x_k \leq 32767$), and an output variable y_k, initialized to None (meaning undecided). A consensus algorithm is designed to calculate the maximum of all the input variables. In other words, at any point, the safety condition is, y_k is either None, or $y_k = max_{\ i \in \{1,\ldots,N\}}\ x_i$. We will adapt the (first version of) ring-based election algorithm for this problem.

- A process P_i initiates the algorithm by sending $(0, x_i)$ to its ring successor (the 0 indicates this is a proposal message).
- When a process P_j receives $(0, x)$ from its ring predecessor:
  - if $x > x_j$, it forwards $(0, x)$ to its successor.
  - if $x < x_j$, it sends while replacing x with x_j ,i.e., send $(0, x_j)$ to its successor.
  - if $x = x_j$, it concludes that $x = x_j$ is the minimum value, sets $y_j = x$ and sends $(1, x)$ to its successor (the 1 indicating it is a decided message).
- When a process P_j receives $(1, x)$ and its y_j is None, it sets $y_j = x$ and forwards $(1, x)$ to its successor. If y_j is not None, it ignores any received messages.

Note that multiple processes may initiate the algorithm simultaneously. Here is a Python implementation:

```
PROPOSAL = 0
DECIDED = 1
X_K = # my input
Y_K = None

def start_consensus():
    # send forwards two integers to the successor in the ring
    send(PROPOSAL, X_K)

# receive two integers from the neighbor
def receive_message(a,b):
    if Y_K is not None:
        continue # ignore messages after decided state
    if a == PROPOSAL:
        if b > X_K: # forward
            send(PROPOSAL, b)
        elif b < X_K: # replace b with X_K
            send(PROPOSAL, X_K)
        else: # b == X_K
            Y_K = b
            send(DECIDED, Y_K)
    elif a == DECIDED:
        # set variable, forward decision
        Y_K = b
        send(DECIDED, b)
```

Does the algorithm described above guarantee safety condition for this problem? If yes, prove how. If not, (i) describe a scenario where safety is violated, and (ii) suggest any modifications to the algorithm that would guarantee the safety condition. You do not need to submit the code but you should have enough details in your modified algorithm for the grader to be able to implement the modification.

(b) (4 points) In this problem, the inputs x_k are all either 0 or 1. In this case, the output variable should be set to represent the majority value. In other words, if y_k is not None, it must be 1 if the majority of processes have input 1, and 0 if the majority of processes have input 0, and 2 if there is a tie. Describe the algorithm for solving this problem (using either Python or a pseudocode). Assume that multiple processes may initiate your algorithm simultaneously.

Question 3: Consensus in Synchronous Systems *[13 points]*

Consider the following algorithm:

- A process P_i initiates the consensus by multicasting a Query message to the group.
- Each process P_j unicasts a reply to P_i with message *Value*(x_j) that includes its input.
- After receiving replies from everyone or a timeout, P_i computes the minimum of all received values (including its own) $y_i = min\ x_j$ and multicasts *Decision*(y_i) to the group.

Assume that we operate in a synchronous system with the maximum one-way delay T for unicast message transmissions including any message processing times. Assume also that unicast channels are reliable.

(a) (3 points) For the multicast of Query message, should R-multicast or B-multicast be used, and why?

(b) (2 points) What should the timeout value be?

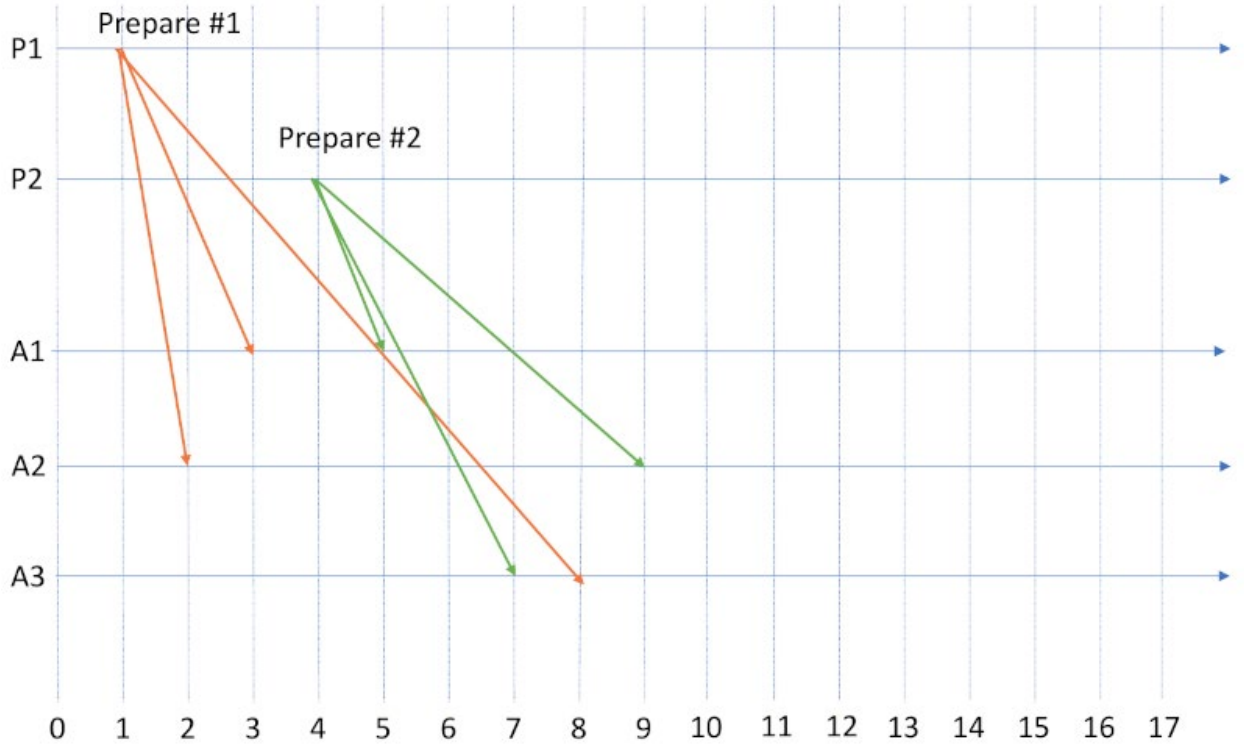
(c) (3 points) For the multicast of Decision message, should R-multicast or B-multicast be used, and why?

(d) (2 points) Assuming no failures, how long will the algorithm take to complete?

(e) (3 points) This algorithm can lead to a safety failure while trying to achieve consensus. Explain how it could happen. (For this part, consider possibility of processes failure.)

Question 4: Understanding Paxos *[14 points]*

Consider a system implementing Paxos with two proposers, *P1* and *P2*, and three acceptors, *A1*, *A2*, and *A3*. Assume that the input value for *P1* is 1, and the input value for *P2* is 2. Answer the following sub-question. (Both sub-questions are unrelated.)



(a)

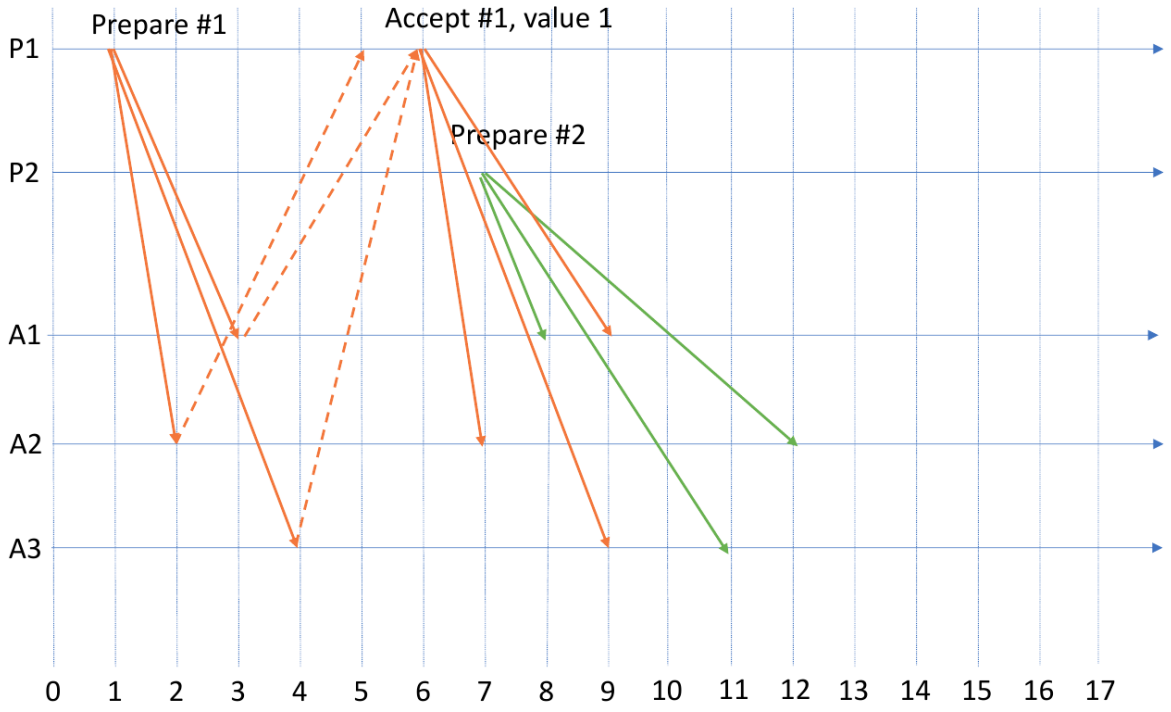
Refer to the figure above. It shows the Prepare messages sent by *P1* and *P2*. The responses (if any) are not shown. Assume that no other proposals are initiated.

i. (1 point) Which processes will reply back to *P1*'s Prepare messages?

ii. (1 point) Which processes will reply back to *P2*'s Prepare messages?

iii. (2 points) Assuming each Promise message take exactly 2 time units, at what time will *P1* send Accept messages? At what time will *P2* send Accept messages?

iv. (2 points) Assuming each Accept message takes at least 1 time unit, and, as above, the Promise messages take exactly 2 time units, is it possible for the Proposal #1 to be accepted by a majority of acceptors? If no, explain why not. If yes, explain how.



(b)

Refer to the figure above. *P1* send Prepare messages at time 1, receives a Promise from all the acceptors, and sends Accept messages at time 6. Meanwhile, *P2* sends Prepare message for the proposal #2 at time 7.

i. (1 point) Which processes will reply to *P2*'s Prepare message?

ii. (1 point) Which processes will accept *P1*'s proposal?

iii. (1 point) Assuming each Promise response for *P2*'s Prepare message takes exactly 2 units, when will *P2* send Accept messages?

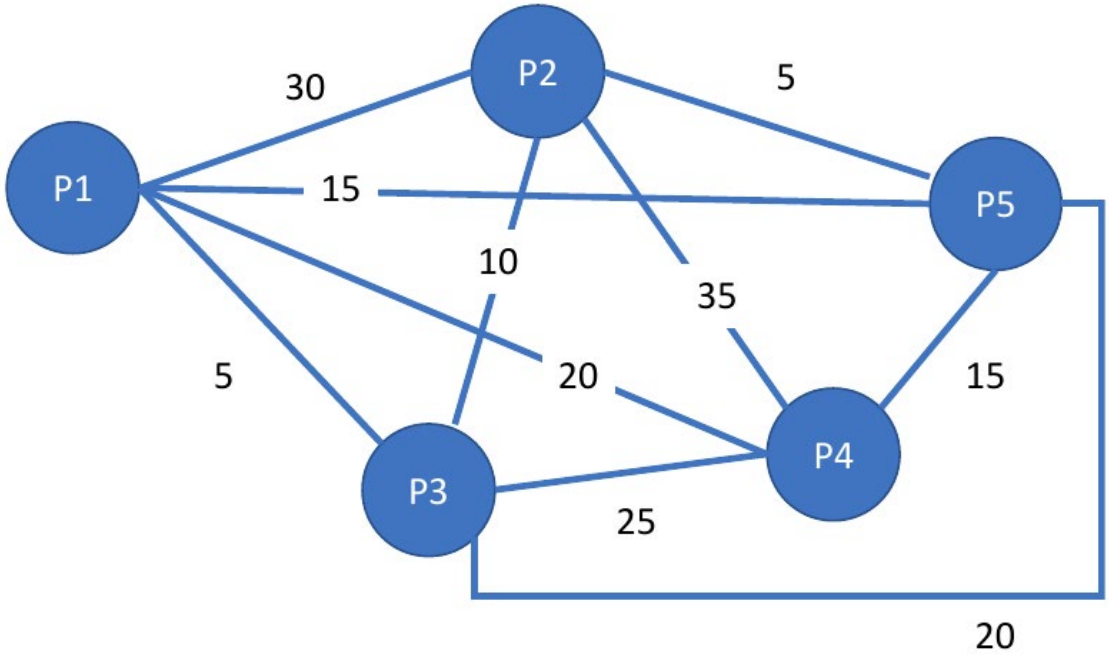
iv. (1 point) Assuming *P2* sends messages at the time stated in the previous parts, which processes will accept *P2*'s proposal?

v. (1 point) What will be the value in *P2*'s proposal?

vi. (3 points) Come up with a scenario where the consensus value changes by switching the arrival time of one message. Complete the diagram showing the timing of the promise messages from the acceptors to *P2* and the accept messages from *P2* back to the acceptors. (For this subpart any messages not shown on the original diagram must take *at least* one time unit, but have no other constraints.)

Question 5: Understanding Raft *[14 points]*

Consider a system of 5 processes, *{ P_1, P_2, P_3, P_4, P_5 }*, implementing the Raft's algorithm for leader election. The one-way delay between each process is specified in the figure below. Assume that processing times can be ignored. Each of the question parts below assumes an independent scenario.



(a) (1 point) Which server is the most likely to be elected leader in any term? No justification is necessary.

(b) (4 points) Suppose *P1* is the leader in term 1. It sends its last heartbeat at time 0 and then fails. Suppose, upon receiving the heartbeat, *P2*, *P3*, *P4*, and *P5* set their timeout values to 50, 75, 100, and 150 ms, respectively. Who will each process vote for as the leader in term 2?

(c) (2 points) Suppose now that *P2, P3, P4, P5* set their timeout values to 150, 100, 75, and 50 ms, respectively. Who will each process vote for as the leader in term 2?

(d) (2 points) Consider the processes logs below, where the number *n* denotes events logged in term *n*. Order the servers from the least up-to-date to the most up-to-date.

P_1 1, 1, 1, 2, 2, 3, 3, 6
P_2 1, 1, 1, 2, 2, 3, 3, 3, 4
P_3 1, 1, 1, 2, 2, 3, 3, 3, 4, 4
P_4 1, 1, 1, 2, 2, 3, 3, 6, 6
P_5 1, 1, 1, 2, 2, 3, 3, 3, 4, 5