DSC Homework 1

Tey Shi Ying (1003829)

Make sure you are in the correct directory before running the cmds as stated below.

PA 1.1

In your cmd line, type go run task1-1.go.

Run the function testOnePointOne() in your main function .

This will start up the creation of 5 clients and a server whom use two []chan int to communicate.

The Client sends its id to the server as seen in the first line of the print statement below. Upon receiving the message from the client, the server prints the second line and calls a goroutine to broadcast the message it receives from client 2 to the four other clients. The goroutine prints the third line of the print statement below.

```
Client 2 sent 2 to server
Server receiving 2 from client 2
serVer msg is running for send_client 2
```

As the goroutine iteratively sends the broadcast message from the server to the four other clients, it prints the following statement.

```
Server is Sending 3 on behalf of client 3 to client 2
```

After the client has sent the first message, the client has been waiting to receive messages from the server. Upon receiving message from the server, it will print the statement below.

```
Client receives 3 from Server 0
```

While the client waits for messages, it will print the updated broadcast messages it has received up till then (this is just for checking, not an implementation required by the question.)

```
Client 4 has received [0 0 0 0 4]
```

Eventually within your logs you will receive the following information about the client which means that the client has all the broadcast messages that the server sent from other clients.

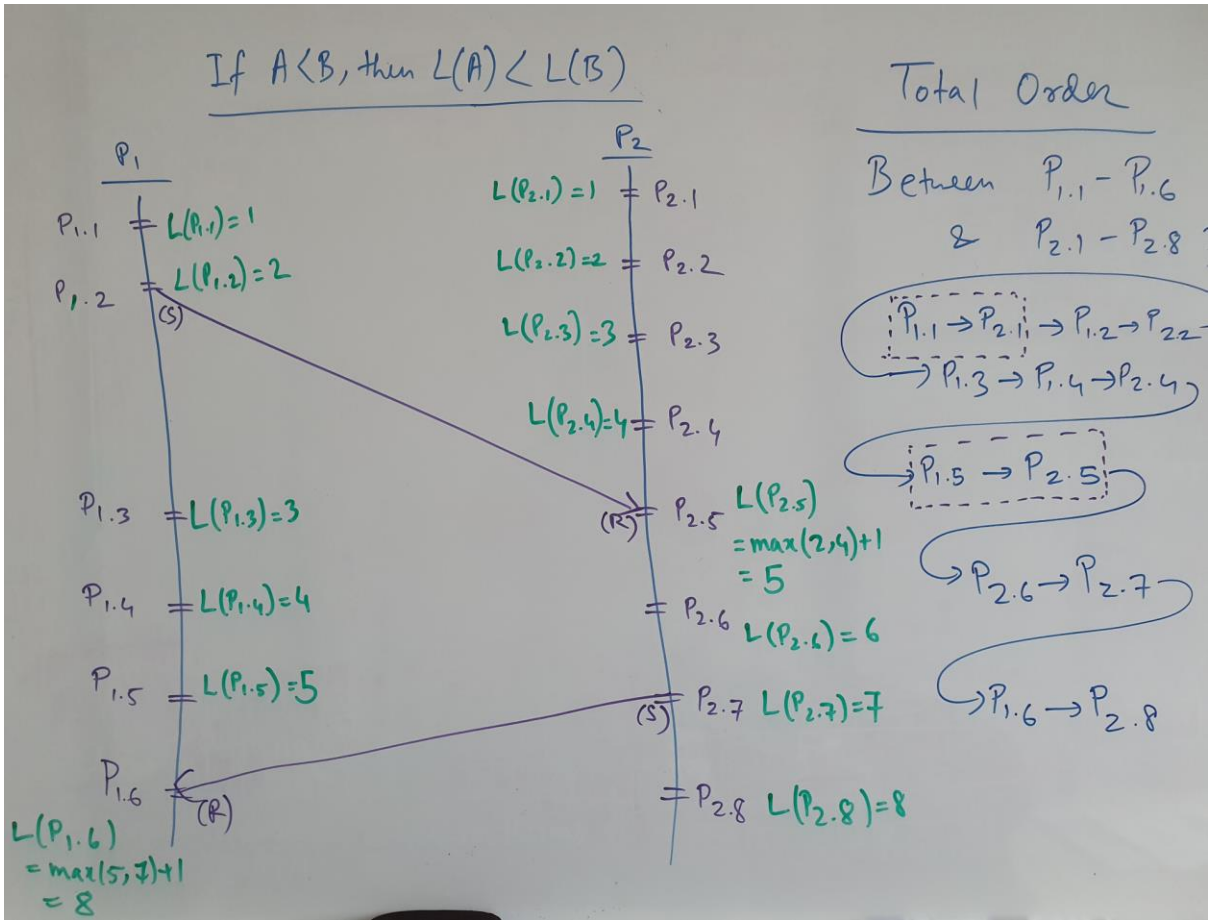
```
Client 1 has received [0 1 2 3 4]
Client 2 has received [0 1 2 3 4]
Client 3 has received [0 1 2 3 4]
Client 0 has received [0 1 2 3 4]
timeout

Client 4 has received [0 1 2 3 4]
```

PA 1.2

In your cmd line, type go run task1-2.go.

I try to implement total order as follows.




```
logs: [1.1,S.3 S.4,1.? 0.1,S.5 S.6,0.? 2.1,S.7 S.8,2.?]
```

Each log event is delimited by a blankspace. There are two types of log event recorded in the logs, you can differentiate by seeing which element is S present.

An example of one log event where client -> server is 1.1,S.3 which represents that client 1 at time 1 sent a message to Server whom received the message at time 3.

An example of one log event where server -> client is S.4,1.? Which represents that server at time 4 sent a message to client 1 whose local clock time is unknown.

```
total order logs: [1.0 1.1 S.0 S.1 0.0 0.1 0.2 0.3 0.4 S.0 S.1 S.2 S.3 S.4 0.0 0.1 S.0 S.1 0.0 0.1 0.2 0.3 0.4 0.5 0.6 S.0 S.1 S.2 S.3 S.4 S.5 S.6 2.0 2.1 S.0 S.1 0.0 0.1 0.2 0.3 0.4
 0.5 0.6 0.7 0.8 S.0 S.1 S.2 S.3 S.4 S.5 S.6 S.7 S.8]
```

When the time comes to print the total_order_logs, I sort the order of the logs. I create a vector to keep track of clients time that is last read by the server to determine the total order. All local clock timings up to the point at which the event occurs can happen synchronously hence I can for loop the local clock timings for the total order from the last_seen_clock_value_of_previous_message to

last_seen_clock_value_of_current_message. I do this for the server and client in order to generate a possible total ordering as shown in the ppt slide.

PA1.3

In your cmd line, type go run task1-3.go.

I started 3 clients. I created a struct clock_for_all that consisted of [no_client] [no_client]chan [no_client]int where each client has a private channel to send a vector clock which is of type [no_client] int to another client. Each client continuously reads its own input channels in clock_for_all[client_id]. If it receives a message, it would check for causality violation by checking if the incoming client's clock time in the vector clock message received is less than the incoming client's clock time of the client's own vector clock.

When a client would like to send a message to another client, it updates its local vector clock and starts a goroutine to async send messages. It is important to use a goroutine function as we do not want to introduce deadlock – In the case that the receiving channel be blocked, the client is also stuck waiting to send message and does not read its own input channels.

An implementation that I have done to induce causality violation is to simulate time delay in the connection from Client 1 to Client 2 by creating a significant time delay in the goroutine function before Client 1 sends a copy of its vector Clock to Client 2. Meanwhile in parallel, any amount of messages that happens to form a closed loop to update the vector clock of Client 2 with information from Client 1 would result in an updated vector clock of Client 2 that would be greater than the vector clock message from Client 1 when it arrives at Client 2. This induces a causality Violation.

```
Client 2 receives [39 33 33] from Client 0. Client clock compares incoming vector clock [39 33 33] and localClock [38 33 36].
SLEEPEDDDDDD. NOW CHECKKK: Client 1 sends message [14 14 9] to client 2
Client 2 receives [14 14 9] from Client 1. Client clock compares incoming vector clock [14 14 9] and localClock [39 33 37].
CAUSALITY VIOLATION: client 2 received vector_clock [14 14 9] from Client 1 when local clock is [39 33 37]
```

PA2.1

In your cmd line, type go run task2-1.go. This will run the best case.

In the best case, I callElection() on the node with the biggest id.

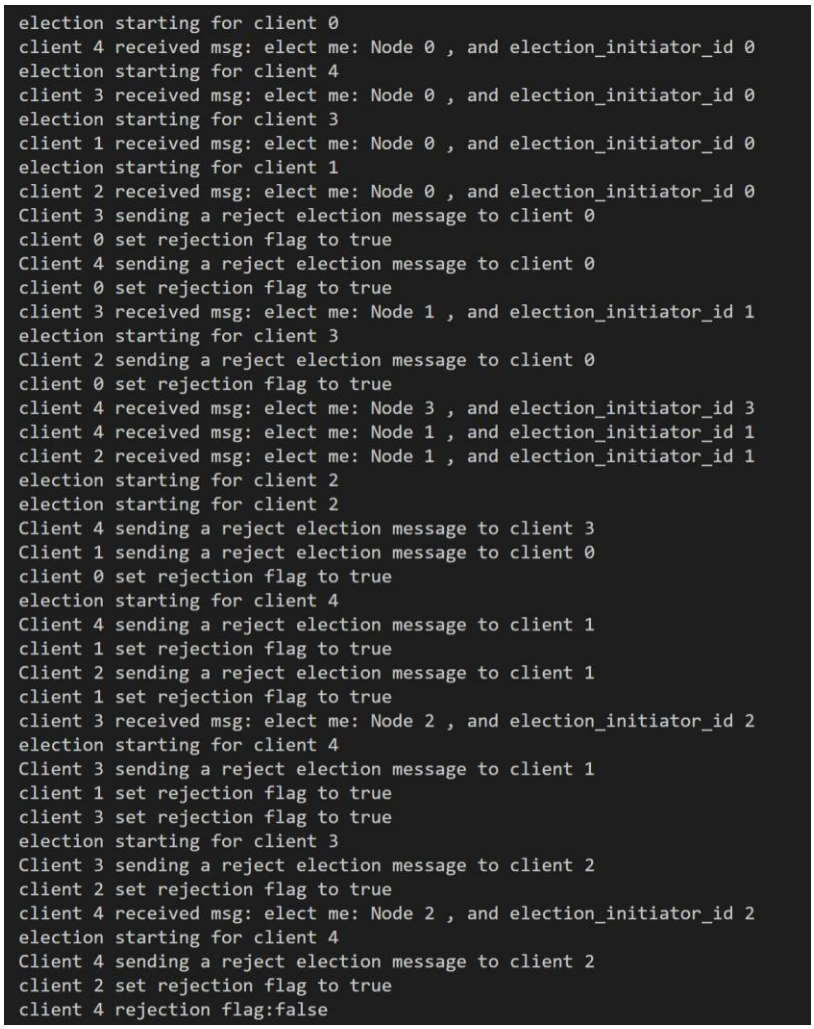
```
election starting for client 4
client 4 rejection flag:false
Client 3 has set the coordinator id to 4
Client 0 has set the coordinator id to 4
Client 4 has set the coordinator id to 4
Client 1 has set the coordinator id to 4
Client 2 has set the coordinator id to 4
```

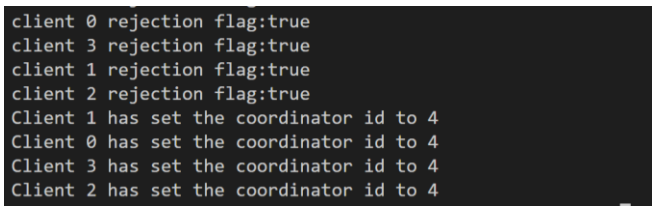
I also implemented a pingCoordinatorAlive message such that nodes can restart the election should the coordinator go down, under the assumption that if the coordinator does not respond to the pingCoordinatorAlive messages sent by clients before timeout, it means that the coordinator node is down.

To run the worst case, type go run task2-1b.go.

In the worst case, I callElection() on the node with the smallest id, which will trigger the callElection() process on all other nodes in the system. These nodes will wait for a period of time to receive reject messages from nodes with a higher id than them, if the period of time elapsed and they still have not received any reject message, this node assumes it is the node with the highest id and declares

itself as the coordinator node to all the other nodes in a broadcast message. The other nodes then update the value of their coordinator id.





PA 2.2

To run type go run task2-2.go.

I simulated a scenario where after 3000ms I sent an instruction to kill off the coordinator node. Then a re-election was held and the node with the highest id (excluding the previous coordinator node's id) was chosen as the coordinator.

```
election starting for client 4
client 4 rejection flag:false
Client 0 has set the coordinator id to 4
Client 3 has set the coordinator id to 4
Sent message to KILL coordinator node
Client 1 has set the coordinator id to 4
Client 2 has set the coordinator id to 4
Coordinator did not respond before timeout. Client 0 restarting election
election starting for client 0
Coordinator did not respond before timeout. Client 3 restarting election
election starting for client 3
Coordinator did not respond before timeout. Client 2 restarting election
election starting for client 2
client 1 received msg: elect me: Node 0 , and election_initiator_id 0
```

Eventually all of the elections will conclude with the selected coordinator node as the node with the highest id(with the exception of the previous coordinator node.)

```
Client 3 sending a reject election message to client 1
client 1 set rejection flag to true
client 0 rejection flag:true
client 1 rejection flag:true
client 3 rejection flag:false
client 2 rejection flag:true
Client 0 has set the coordinator id to 3
Client 2 has set the coordinator id to 3
Client 1 has set the coordinator id to 3
```

PA 2.3

To run type go run task2-3.go.

I started election for node 0 and the node with the highest id concurrently. As expected, the coordinator node was the node with the highest id.

```
election starting for client 0
election starting for client 4
client 1 received msg: elect me: Node 0 , and election_initiator_id 0
election starting for client 1
client 4 received msg: elect me: Node 0 , and election_initiator_id 0
client 4 received msg: elect me: Node 1 , and election_initiator_id 1
client 3 received msg: elect me: Node 0 , and election_initiator_id 0
client 3 received msg: elect me: Node 1 , and election_initiator_id 1
client 4 sending a reject election message to client 0
client 0 set rejection flag to true
election starting for client 4
election starting for client 4
client 4 sending a reject election message to client 1
client 1 set rejection flag to true
client 1 sending a reject election message to client 0
client 0 set rejection flag to true
client 3 sending a reject election message to client 1
election starting for client 3
client 2 received msg: elect me: Node 0 , and election_initiator_id 0
client 2 received msg: elect me: Node 1 , and election_initiator_id 1
election starting for client 2
client 1 set rejection flag to true
client 4 received msg: elect me: Node 3 , and election_initiator_id 3
client 4 sending a reject election message to client 3
client 3 sending a reject election message to client 0
client 0 set rejection flag to true
election starting for client 3
client 2 sending a reject election message to client 1
client 1 set rejection flag to true
client 4 received msg: elect me: Node 2 , and election_initiator_id 2
```

```
Client 4 sending a reject election message to client 3
Client 3 sending a reject election message to client 0
client 0 set rejection flag to true
election starting for client 3
Client 2 sending a reject election message to client 1
client 1 set rejection flag to true
client 4 received msg: elect me: Node 2 , and election_initiator_id 2
election starting for client 4
election starting for client 4
election starting for client 2
client 3 received msg: elect me: Node 2 , and election_initiator_id 2
client 3 set rejection flag to true
Client 4 sending a reject election message to client 2
client 2 set rejection flag to true
Client 2 sending a reject election message to client 0
client 0 set rejection flag to true
Client 3 sending a reject election message to client 2
election starting for client 3
client 2 set rejection flag to true
client 2 rejection flag:true
client 3 rejection flag:true
client 1 rejection flag:true
client 4 rejection flag:false
client 0 rejection flag:true
Client 0 has set the coordinator id to 4
Client 3 has set the coordinator id to 4
Client 2 has set the coordinator id to 4
Client 1 has set the coordinator id to 4
```