DistributedSystemsPSets50.041

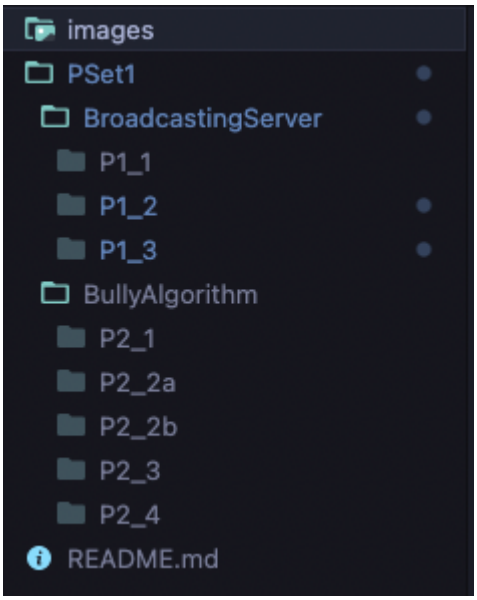
Assignment 1

Toh Kai Feng

1004581

Introduction

This is the following file structure used:



1. To run questions 1_1 to 1_3, replace "--Question--" with either "P1_1","P1_2","P1_3" in the following command:

```
go run -race PSet1/BroadcastingServer/--Question--/main.go
```

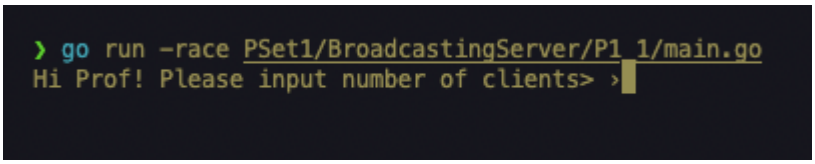
2. To run questions 2_1 to 2_4, replace "--Question--" with either "P2_1","P2_2a", "P2_2b","P2_3", "P2_4" in the following command:

```
 go run -race PSet1/BullyAlgorithm/--Question--/main.go
```

Question 1

Part 1

This would be the prompt when the program is ran:



This would be the output after entering input:

```
❯ go run -race PSet1/BroadcastingServer/P1_1/main.go
Hi Prof! Please input number of clients> 5
"5" looks like a number. Creating "5" clients. Press ENTER again to stop processes
Client 2 has sent [0 0 1 0 0]
[0 0 1 0 0] received from Client 2
Client 4 has sent [0 0 0 0 1]
[0 0 0 0 1] received from Client 4
Client 1 has sent [0 1 0 0 0]
[0 1 0 0 0] received from Client 1
Client 0 has sent [1 0 0 0 0]
[1 0 0 0 0] received from Client 0
Starting to broadcast message from Server
[0 0 1 0 0] received from server by client 3
Event Log: Server sent [0 0 1 0 0] to Client 0
[0 0 1 0 0] received from server by client 4
[0 0 1 0 0] received from server by client 0
Event Log: Server sent [0 0 1 0 0] to Client 1
[0 0 1 0 0] received from server by client 1
Event Log: Server sent [0 0 1 0 0] to Client 3
Event Log: Server sent [0 0 1 0 0] to Client 4
[0 0 2 0 0] received from Client 2
Client 2 has sent [0 0 2 0 0]
Client 3 has sent [0 0 0 1 0]
[0 0 0 1 0] received from Client 3
Starting to broadcast message from Server
[1 0 0 0 0] received from server by client 2
Event Log: Server sent [1 0 0 0 0] to Client 1
[1 0 0 0 0] received from server by client 1
[1 0 0 0 0] received from server by client 4
Event Log: Server sent [1 0 0 0 0] to Client 2
[1 0 0 0 0] received from server by client 3
Event Log: Server sent [1 0 0 0 0] to Client 3
Event Log: Server sent [1 0 0 0 0] to Client 4
Starting to broadcast message from Server
[0 1 0 0 0] received from server by client 0
[0 1 0 0 0] received from server by client 2
Event Log: Server sent [0 1 0 0 0] to Client 0
[0 1 0 0 0] received from server by client 3
Event Log: Server sent [0 1 0 0 0] to Client 2
[0 1 0 0 0] received from server by client 4
Event Log: Server sent [0 1 0 0 0] to Client 3
Event Log: Server sent [0 1 0 0 0] to Client 4
Starting to broadcast message from Server
[0 0 0 1 0] received from server by client 4
[0 0 0 1 0] received from server by client 1
[0 0 0 1 0] received from server by client 0
Event Log: Server sent [0 0 0 1 0] to Client 0
[0 0 0 1 0] received from server by client 2
Event Log: Server sent [0 0 0 1 0] to Client 1
Event Log: Server sent [0 0 0 1 0] to Client 2
Event Log: Server sent [0 0 0 1 0] to Client 4
Starting to broadcast message from Server
[0 0 0 0 1] received from server by client 3
Event Log: Server sent [0 0 0 0 1] to Client 0
[0 0 0 0 1] received from server by client 1
[0 0 0 0 1] received from server by client 0
[0 0 0 0 1] received from server by client 2
Event Log: Server sent [0 0 0 0 1] to Client 1
Event Log: Server sent [0 0 0 0 1] to Client 2
Event Log: Server sent [0 0 0 0 1] to Client 3
```

What we are looking for are:

1. the Clients are randomly sending messages (in the form of arrays)
2. Server received the messages immediately "[0 0 1 0 0] received from Client 2"

3. Server does not broadcast the messages in order (broadcasted [1,0,0,0,0] from client 0 before [0,1,0,0,0] from client 1 even though server receives message from client 1 first)

Implementation details: This is achieve with a asynchronous call of my "broadcast" function, where the function would sleep for a random delay before sending the message.

Part 2

This would be the prompt when the program is ran:

```
❯ go run -race PSet1/BroadcastingServer/P1_2/main.go
Hi Prof! Please input number of clients>
```

This would be the output after entering input:

```
[0 4 0 0 0] received from server by client 3
[0 4 0 0 0] received from server by client 2
[0 4 0 0 0] received from server by client 4
[0 4 0 0 0] received from server by client 0
Messages to be read are[{[0 1 0 0 0] 1 1} {[0 0 0 0 1] 4 3} {[0 2 0 0 0] 1 2} {[0 3 0 0 0] 1 6} {[0 0 0 0 2] 4 6} {[0 0 0 1 0
] 3 7} {[0 4 0 0 0] 1 9}]
Total Order for Client 0:
Clock: 1, Message: [0 1 0 0 0]
Clock: 2, Message: [0 2 0 0 0]
Clock: 3, Message: [0 0 0 0 1]
Clock: 6, Message: [0 3 0 0 0]
Clock: 6, Message: [0 0 0 0 2]
Clock: 7, Message: [0 0 0 1 0]
Clock: 9, Message: [0 4 0 0 0]
Messages to be read are[{[0 0 0 0 1] 4 3} {[1 0 0 0 0] 0 3} {[2 0 0 0 0] 0 4} {[3 0 0 0 0] 0 7} {[0 0 0 0 2] 4 6} {[0 0 0 1 0
] 3 7}]
Total Order for Client 1:
Clock: 3, Message: [0 0 0 0 1]
Clock: 3, Message: [1 0 0 0 0]
Clock: 4, Message: [2 0 0 0 0]
Clock: 6, Message: [0 0 0 0 2]
Clock: 7, Message: [3 0 0 0 0]
Clock: 7, Message: [0 0 0 1 0]
Client 2 has sent [0 0 1 0 0]
[0 0 1 0 0] received from Client 2
Client 0 has sent [4 0 0 0 0]
[4 0 0 0 0] received from Client 0
Messages to be read are[{[0 1 0 0 0] 1 1} {[0 0 0 0 1] 4 3} {[0 2 0 0 0] 1 2} {[1 0 0 0 0] 0 3} {[2 0 0 0 0] 0 4} {[3 0 0 0 0
] 0 7} {[0 3 0 0 0] 1 6} {[0 0 0 0 2] 4 6} {[0 4 0 0 0] 1 9}]
Total Order for Client 3:
Clock: 1, Message: [0 1 0 0 0]
Clock: 2, Message: [0 2 0 0 0]
Clock: 3, Message: [0 0 0 0 1]
Clock: 3, Message: [1 0 0 0 0]
Clock: 4, Message: [2 0 0 0 0]
Clock: 6, Message: [0 3 0 0 0]
Clock: 6, Message: [0 0 0 0 2]
Clock: 7, Message: [3 0 0 0 0]
Clock: 9, Message: [0 4 0 0 0]
Messages to be read are[{[0 1 0 0 0] 1 1} {[0 2 0 0 0] 1 2} {[1 0 0 0 0] 0 3} {[2 0 0 0 0] 0 4} {[3 0 0 0 0] 0 7} {[0 3 0 0 0
] 1 6} {[0 0 0 1 0] 3 7} {[0 4 0 0 0] 1 9}]
Total Order for Client 4:
Clock: 1, Message: [0 1 0 0 0]
Clock: 2, Message: [0 2 0 0 0]
Clock: 3, Message: [1 0 0 0 0]
Clock: 4, Message: [2 0 0 0 0]
Clock: 6, Message: [0 3 0 0 0]
Clock: 7, Message: [3 0 0 0 0]
Clock: 7, Message: [0 0 0 1 0]
Clock: 9, Message: [0 4 0 0 0]
Starting to broadcast message from Server
[4 0 0 0 0] received from server by client 1
[4 0 0 0 0] received from server by client 3
[4 0 0 0 0] received from server by client 2
```

This would be the output after an ENTER keypress to end the program:

```
--------- CLIENT 0 REPORT ------------
Total Order for Client 0:
Clock: 1, Message: [0 1 0 0 0]
Clock: 2, Message: [0 2 0 0 0]
Clock: 3, Message: [0 0 0 0 1]
Clock: 6, Message: [0 3 0 0 0]
Clock: 6, Message: [0 0 0 0 2]
Clock: 7, Message: [0 0 0 1 0]
Clock: 9, Message: [0 4 0 0 0]
Clock: 13, Message: [0 0 1 0 0]
-------- END OF CLIENT 0 REPORT ------
0 : Terminating

--------- CLIENT 2 REPORT ------------
Total Order for Client 2:
Clock: 1, Message: [0 1 0 0 0]
Clock: 2, Message: [0 2 0 0 0]
Clock: 3, Message: [0 0 0 0 1]
Clock: 3, Message: [1 0 0 0 0]
Clock: 4, Message: [2 0 0 0 0]
Clock: 6, Message: [0 3 0 0 0]
Clock: 6, Message: [0 0 0 0 2]
Clock: 7, Message: [3 0 0 0 0]
Clock: 7, Message: [0 0 0 1 0]
Clock: 9, Message: [0 4 0 0 0]
Clock: 12, Message: [4 0 0 0 0]
-------- END OF CLIENT 2 REPORT ------
2 : Terminating

--------- CLIENT 3 REPORT ------------
Total Order for Client 3:
Clock: 1, Message: [0 1 0 0 0]
Clock: 2, Message: [0 2 0 0 0]
Clock: 3, Message: [0 0 0 0 1]
Clock: 3, Message: [1 0 0 0 0]
Clock: 4, Message: [2 0 0 0 0]
Clock: 6, Message: [0 3 0 0 0]
Clock: 6, Message: [0 0 0 0 2]
Clock: 7, Message: [3 0 0 0 0]
Clock: 9, Message: [0 4 0 0 0]
Clock: 12, Message: [4 0 0 0 0]
Clock: 13, Message: [0 0 1 0 0]
-------- END OF CLIENT 3 REPORT ------
3 : Terminating

--------- CLIENT 4 REPORT ------------
Total Order for Client 4:
Clock: 1, Message: [0 1 0 0 0]
Clock: 2, Message: [0 2 0 0 0]
Clock: 3, Message: [1 0 0 0 0]
Clock: 4, Message: [2 0 0 0 0]
Clock: 6, Message: [0 3 0 0 0]
Clock: 7, Message: [3 0 0 0 0]
Clock: 7, Message: [0 0 0 1 0]
Clock: 9, Message: [0 4 0 0 0]
Clock: 12, Message: [4 0 0 0 0]
Clock: 13, Message: [0 0 1 0 0]
-------- END OF CLIENT 4 REPORT ------
4 : Terminating

--------- CLIENT 1 REPORT ------------
Total Order for Client 1:
```

In addition to what we have seen in P1_1, What we are looking for are:

1. regular reports of the total order whereby the clients sort the messages to be read based on the logical clock attached to each message.

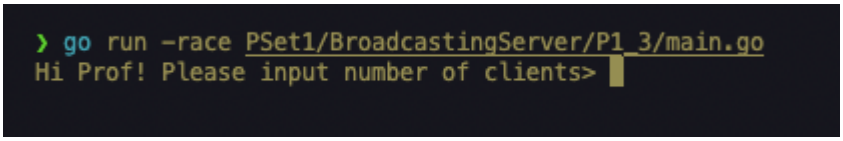
2. The final report generated an ENTER keypress is sent to end the program to summarize the order of messages
3. The messages in the report are sorted based on the logical clock of the message.

Implementation details:

1. Each client would increment their logical clock before sending a message and after receiving the message
2. When a client receives a message, the client would compute the max of the message's clock and its logical clock before incrementing.

Part 3

This would be the prompt when the program is ran:



This would be the output after entering input:

```
Client 4 has sent [5 5 6 5 11 14]
Starting to broadcast message from Server
Event Log: Server sent [0 0 3 0 0] to Clients
[0 0 3 0 0] received from server by client 0
[0 0 3 0 0] received from server by client 4
[0 0 3 0 0] received from server by client 1
[0 0 3 0 0] received from server by client 3
Starting to broadcast message from Server
Event Log: Server sent [0 0 0 0 3] to Clients
[0 0 0 0 3] received from server by client 1
[0 0 0 0 3] received from server by client 3
[0 0 0 0 3] received from server by client 2
[0 0 0 0 3] received from server by client 0
Client 3 has sent [5 5 6 12 6 15]
[0 0 0 2 0] received from Client 3
Starting to broadcast message from Server
Event Log: Server sent [0 0 0 0 4] to Clients
[0 0 0 0 4] received from server by client 1
[0 0 0 0 4] received from server by client 3
[0 0 0 0 4] received from server by client 2
[0 0 0 0 4] received from server by client 0
Starting to broadcast message from Server
[0 0 0 2 0] received from server by client 0
[0 0 0 2 0] received from server by client 4
[0 0 0 2 0] received from server by client 2
[0 0 0 2 0] received from server by client 1
Event Log: Server sent [0 0 0 2 0] to Clients
[4 0 0 0 0] received from Client 0
Client 0 has sent [14 5 6 12 11 24]
Client 4 has sent [5 5 6 12 14 24]
[0 0 0 0 5] received from Client 4
[0 0 4 0 0] received from Client 2
Client 2 has sent [5 5 14 12 11 24]
Messages to be read are[{[1 0 0 0 0] 0 [1 0 0 0 0 1]} {[0 0 0 0 1] 4 [1 0 0 0 2 3]} {[0 0 1 0 0] 2 [1 0 2 0 2 4]} {[2 0 0 0 0
] 0 [2 0 2 0 2 5]} {[0 0 0 1 0] 3 [2 0 2 5 2 9]} {[0 0 0 0 2] 4 [2 0 5 5 5 11]} {[3 0 0 0 0] 0 [5 5 6 5 5 14]} {[0 0 2 0 0] 2
 [2 0 5 5 2 10]} {[0 0 3 0 0] 2 [2 0 6 5 5 12]} {[0 0 0 0 3] 4 [5 5 6 5 6 15]} {[0 0 0 0 4] 4 [5 5 6 5 11 21]} {[0 0 0 2 0] 3
 [5 5 6 12 11 24]}]
Total Order for Client 1:
Clock: [1 0 0 0 0 1], Message: [1 0 0 0 0]
Clock: [1 0 0 0 2 3], Message: [0 0 0 0 1]
Clock: [1 0 2 0 2 4], Message: [0 0 1 0 0]
Clock: [2 0 2 0 2 5], Message: [2 0 0 0 0]
Clock: [2 0 2 5 2 9], Message: [0 0 0 1 0]
Clock: [2 0 5 5 2 10], Message: [0 0 2 0 0]
Clock: [2 0 5 5 5 11], Message: [0 0 0 0 2]
Clock: [2 0 6 5 5 12], Message: [0 0 3 0 0]
Clock: [5 5 6 5 5 14], Message: [3 0 0 0 0]
Clock: [5 5 6 5 6 15], Message: [0 0 0 0 3]
Clock: [5 5 6 5 11 21], Message: [0 0 0 0 4]
Clock: [5 5 6 12 11 24], Message: [0 0 0 2 0]
Starting to broadcast message from Server
Event Log: Server sent [4 0 0 0 0] to Clients
[4 0 0 0 0] received from server by client 2
[4 0 0 0 0] received from server by client 3
[4 0 0 0 0] received from server by client 4
```

This would be the output after an ENTER keypress to end the program:

```
Clock: [5 5 6 5 6 15], Message: [0 0 0 0 3]
Clock: [5 5 6 5 11 21], Message: [0 0 0 0 4]
Clock: [5 5 6 12 11 24], Message: [0 0 0 2 0]
Clock: [14 5 6 12 14 28], Message: [0 0 0 0 5]
Clock: [14 5 14 12 14 29], Message: [0 0 4 0 0]
Clock: [14 5 16 12 14 32], Message: [0 0 5 0 0]
Clock: [14 5 16 12 17 33], Message: [0 0 0 0 6]
Clock: [14 16 16 12 17 34], Message: [0 2 0 0 0]
Clock: [14 16 19 12 17 38], Message: [0 0 6 0 0]
Clock: [20 16 19 12 20 41], Message: [0 0 0 0 7]
-------- END OF CLIENT 0 REPORT ------
0 : Terminating

--------- CLIENT 2 REPORT ------------
Total Order for Client 2:
Clock: [1 0 0 0 0 1], Message: [1 0 0 0 0]
Clock: [1 0 0 0 2 3], Message: [0 0 0 0 1]
Clock: [2 0 2 0 2 5], Message: [2 0 0 0 0]
Clock: [2 0 2 5 2 9], Message: [0 0 0 1 0]
Clock: [2 0 5 5 5 11], Message: [0 0 0 0 2]
Clock: [2 5 6 5 5 13], Message: [0 1 0 0 0]
Clock: [5 5 6 5 5 14], Message: [3 0 0 0 0]
Clock: [5 5 6 5 6 15], Message: [0 0 0 0 3]
Clock: [5 5 6 5 11 21], Message: [0 0 0 0 4]
Clock: [5 5 6 12 11 24], Message: [0 0 0 2 0]
Clock: [14 5 6 12 11 27], Message: [4 0 0 0 0]
Clock: [14 5 6 12 14 28], Message: [0 0 0 0 5]
Clock: [14 5 16 12 17 33], Message: [0 0 0 0 6]
Clock: [14 16 16 12 17 34], Message: [0 2 0 0 0]
Clock: [20 16 19 12 17 40], Message: [5 0 0 0 0]
Clock: [20 16 19 12 20 41], Message: [0 0 0 0 7]
-------- END OF CLIENT 2 REPORT ------
2 : Terminating

--------- CLIENT 3 REPORT ------------
Total Order for Client 3:
Clock: [1 0 0 0 0 1], Message: [1 0 0 0 0]
Clock: [1 0 0 0 2 3], Message: [0 0 0 0 1]
Clock: [1 0 2 0 2 4], Message: [0 0 1 0 0]
Clock: [2 0 2 0 2 5], Message: [2 0 0 0 0]
Clock: [2 0 5 5 2 10], Message: [0 0 2 0 0]
Clock: [2 0 5 5 5 11], Message: [0 0 0 0 2]
Clock: [2 0 6 5 5 12], Message: [0 0 3 0 0]
Clock: [2 5 6 5 5 13], Message: [0 1 0 0 0]
Clock: [5 5 6 5 5 14], Message: [3 0 0 0 0]
Clock: [5 5 6 5 6 15], Message: [0 0 0 0 3]
Clock: [5 5 6 5 11 21], Message: [0 0 0 0 4]
Clock: [14 5 6 12 11 27], Message: [4 0 0 0 0]
Clock: [14 5 6 12 14 28], Message: [0 0 0 0 5]
Clock: [14 5 14 12 14 29], Message: [0 0 4 0 0]
Clock: [14 5 16 12 14 32], Message: [0 0 5 0 0]
Clock: [14 5 16 12 17 33], Message: [0 0 0 0 6]
Clock: [14 16 16 12 17 34], Message: [0 2 0 0 0]
Clock: [14 16 19 12 17 38], Message: [0 0 6 0 0]
Clock: [20 16 19 12 17 40], Message: [5 0 0 0 0]
Clock: [20 16 19 12 20 41], Message: [0 0 0 0 7]
-------- END OF CLIENT 3 REPORT ------
3 : Terminating
Program Ended
```

In addition to what we have seen in P1_2, What we are looking for are:

1. The messages contain a vector clock instead of a integer clock.
2. The messages in the report are sorted based on the logical vector clock of the message.

Implementation details:

1. I had to create a function to compare two vectors where it was not need in P1_2 when comparing integers
2. I also had to create a function to perform the Max() of two vectors

Question 2

Part 1

There are two options available for part 1: (1) Best case and (2) Worst case

This would be the output when option (b) is selected for the best case scenario:

```
❯ go run -race PSet1/BullyAlgorithm/P2_1/main.go
Hi Prof! Please best(b) or worst (w) case> b
best case scenario selected
3 : regular ping checks
3 : Regular ping timeout. Checking for machine failure
3 : machine 4 failure detected
3 : machine 4 is coordinator? true
3 : starting election
3 : election succeeeded. Starting broadcast
0 : new coordinator is  3
2 : new coordinator is  3
1 : new coordinator is  3

program has ended
```

This would be the output when option (w) is selected for the worst case scenario:

```
❯ go run -race PSet1/BullyAlgorithm/P2_1/main.go
Hi Prof! Please best(b) or worst (w) case> w
worst case scenario selected
0 : regular ping checks
0 : Regular ping timeout. Checking for machine failure
0 : machine 4 failure detected
0 : machine 4 is coordinator? true
0 : starting election
2 : coordinator request from 0 received
2 : Rejecting coordinator request from 0 received
2 : starting election
0 : Rejection. new coordinator is temporarily set to 2
1 : coordinator request from 0 received
1 : Rejecting coordinator request from 0 received
1 : starting election
3 : coordinator request from 0 received
3 : Rejecting coordinator request from 0 received
3 : starting election
0 : Rejection. new coordinator is temporarily set to 1
0 : Rejection. new coordinator is temporarily set to 3
2 : coordinator request from 1 received
2 : Rejecting coordinator request from 1 received
2 : starting election
1 : Rejection. new coordinator is temporarily set to 2
3 : coordinator request from 2 received
3 : Rejecting coordinator request from 2 received
3 : starting election
3 : coordinator request from 1 received
3 : Rejecting coordinator request from 1 received
3 : starting election
3 : coordinator request from 2 received
3 : Rejecting coordinator request from 2 received
3 : starting election
2 : Rejection. new coordinator is temporarily set to 3
2 : Rejection. new coordinator is temporarily set to 3
1 : Rejection. new coordinator is temporarily set to 3
2 : Election failed.
3 : election succeeeded. Starting broadcast
0 : Election failed.
2 : Election failed.
2 : new coordinator is  3
1 : new coordinator is  3
0 : new coordinator is  3
3 : election succeeeded. Starting broadcast
3 : election succeeeded. Starting broadcast
3 : election succeeeded. Starting broadcast
0 : new coordinator is  3
1 : new coordinator is  3
1 : new coordinator is  3
0 : new coordinator is  3
0 : new coordinator is  3
2 : new coordinator is  3
2 : new coordinator is  3
2 : new coordinator is  3
1 : new coordinator is  3
0 : regular ping checks
3 : ping from 0 received
0 : ping acknowledgement from 3 received

program has ended
```

What we are looking for are: [best case]

1. machine 4 (id=3) detects that machine 5 (id=4) is down and triggers an election

2. no rejection message is sent to machine 4, hence it results in a successful election. This enables machine 4 to broadcast to the rest of the machines that machine 4(id=3) is the new coordinator [worst case]
3. machine 1 (id=0) detects that machine 5 (id=4) is down and triggers an election.
4. All other machines that are alive and with higher ids would reply with a rejection message and trigger an election on their own. This would continue until machine 4(id=3) encounters a successful election since no other machine can reject machine 4.

Implementation details:

1. Regular ping messages are sent to the coordinator. for this part, I have specified the machine to send the ping message to the coordinator to replicate the best and worst cases.
2. Ping messages and election procedures each have a time out. A "select" statement handles each timeout to determine what to do next. If ping messages and election requests receive their respective responses before the timeout, no further action would be triggerd.
3. A message handler would help the machine determine what to reply and whether the machine itself needs to initiate an election.

Part 2a (death of coordinator before completion of broadcast)

To create this scenario, I inserted a stopping point before the broadcast loop could end. I then self the machine state to "DOWN" which prevents the machine from replying. This can be seen from line 143-149:

```
132         case <-electionTimeoutChannel: //timeout handler
133             //election request time - check whether self.Coordinator is overriden
134             if self.IsInElection {
135                 if self.Coordinator == self.Id {
136                     // election succeeded - start broadcasting
137                     fmt.Printf("%v : election succeeeded. Starting broadcast\n", self.Id)
138                     for i := 0; i < self.NumberOfMachines; i++ {
139                         if i == self.Id {
140                             continue //no need to broadcast to self
141                         }
142                         self.Channels[i] <- Message{Sender: self.Id, Type: NewCoordinator}
143                         if i == 2 && self.Id == numberOfMachines-2 {
144                             //random failure when announcing
145                             //machine 0,1,2 will know that machine 4 being the coordinator but not machine 3.
146                             fmt.Printf("%v : dying before broadcasting to machine %v\n", self.Id, i+1)
147                             self.IsDown = true
148                             break
149                         }
150                     }
151                     self.IsInElection = false
152                 }
153                 if self.Coordinator != self.Id {
154                     //Election failed do nothing
155                     fmt.Printf("%v : Election failed. \n", self.Id)
156                 }
157             }
```

Thus the expected output is this:

```
4 : election succeeeded. Starting broadcast
4 : dying before broadcasting to machine 3
2 : new coordinator is  4
1 : new coordinator is  4
0 : new coordinator is  4
3 : Election failed.
3 : Election failed.
3 : Election failed.
```

```
3 : Election failed.
2 : regular ping checks
0 : regular ping checks
2 : Regular ping timeout. Checking for machine failure
2 : machine 4 failure detected
2 : machine 4 is coordinator? true
2 : machine 5 failure detected
2 : machine 5 is coordinator? false
2 : starting election
3 : coordinator request from 2 received
3 : Rejecting coordinator request from 2 received
3 : starting election
2 : Rejection. new coordinator is temporarily set to 3
1 : regular ping checks
0 : Regular ping timeout. Checking for machine failure
0 : machine 4 failure detected
0 : machine 4 is coordinator? true
0 : machine 5 failure detected
0 : machine 5 is coordinator? false
0 : starting election
1 : coordinator request from 0 received
2 : coordinator request from 0 received
2 : Rejecting coordinator request from 0 received
0 : Rejection. new coordinator is temporarily set to 2
3 : coordinator request from 0 received
3 : Rejecting coordinator request from 0 received
2 : starting election
1 : Rejecting coordinator request from 0 received
3 : starting election
0 : Rejection. new coordinator is temporarily set to 3
0 : Rejection. new coordinator is temporarily set to 1
1 : starting election
2 : coordinator request from 1 received
2 : Rejecting coordinator request from 1 received
2 : starting election
1 : Rejection. new coordinator is temporarily set to 2
3 : coordinator request from 2 received
3 : Rejecting coordinator request from 2 received
3 : starting election
2 : Rejection. new coordinator is temporarily set to 3
3 : coordinator request from 1 received
3 : Rejecting coordinator request from 1 received
3 : starting election
3 : coordinator request from 2 received
3 : Rejecting coordinator request from 2 received
3 : starting election
2 : Rejection. new coordinator is temporarily set to 3
1 : Rejection. new coordinator is temporarily set to 3
3 : election succeeeded. Starting broadcast
1 : new coordinator is  3
2 : Election failed.
0 : new coordinator is  3
2 : new coordinator is  3
1 : Regular ping timeout. Checking for machine failure
1 : machine 4 failure detected
1 : machine 4 is coordinator? false
1 : machine 5 failure detected
1 : machine 5 is coordinator? false
3 : regular ping checks
1 : regular ping checks
3 : ping from 1 received
1 : ping acknowledgement from 3 received

program has ended
```

what we are looking for are:

1. after machine 4 has declared itself as the coordinator to machine 0,1,2, machine 4 enters a downstate.
2. This would cause machines 0,1 and 2 to trigger an election since their regular ping messages to the coordinator would fail when no replies are received.
3. The new election would result in machine 3 with the highest Id to be the coordinator

Part 2b (death of non-coordinator before completion of broadcast)'

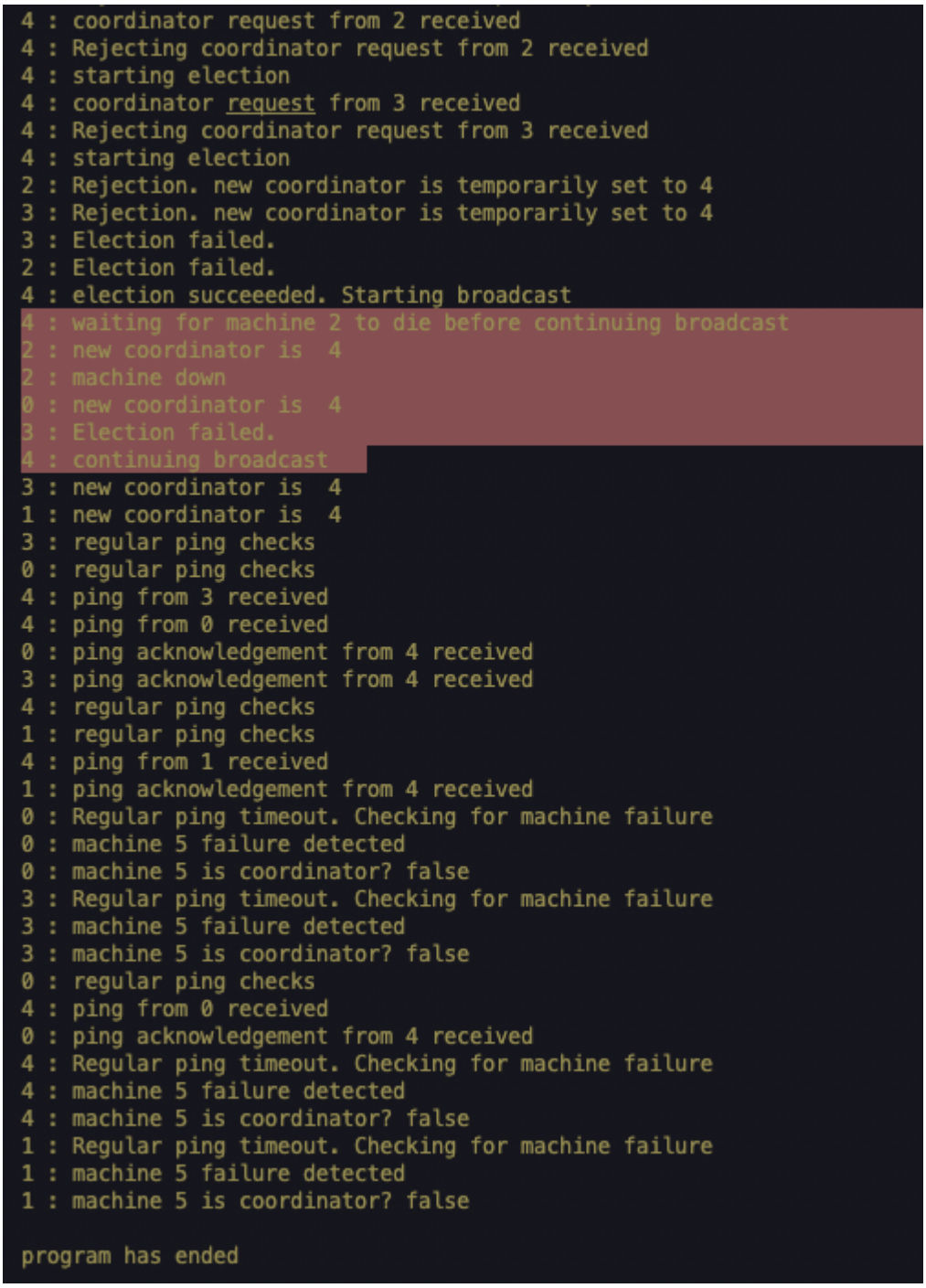
To create this scenario I used a WaitGroup to wait for a node to die before continuing with broadcast. This can be seen from line 150-152:

```
136     case <-electionTimeoutChannel: //timeout handler
137         //election request time - check whether self.Coordinator is overriden
138         if self.IsInElection {
139             if self.Coordinator == self.Id {
140                 // election succeeded - start broadcasting
141                 fmt.Printf("%v : election succeeeded. Starting broadcast\n", self.Id)
142                 for i := 0; i < self.NumberOfMachines; i++ {
143                     if i == self.Id {
144                         continue //no need to broadcast to self
145                     }
146                     self.Channels[i] <- Message{Sender: self.Id, Type: NewCoordinator}
147                     if i == 2 && self.Id == numberOfMachines-2 {
148                         //random failure when announcing
149                         //machine 0,1,2 will know that machine 4 being the coordinator but not machine 3.
150                         (*self.WaitGroup).Add(1)
151                         fmt.Printf("%v : waiting for machine %v to die before continuing broadcast\n", self.Id, i)
152                         (*self.WaitGroup).Wait()
153                         fmt.Printf("%v : continuing broadcast\n", self.Id)
154                     }
155                 }
156                 self.IsInElection = false
157             }
158             if self.Coordinator != self.Id {
159                 //Election failed do nothing
160                 fmt.Printf("%v : Election failed. \n", self.Id)
161             }
162         }
```

As well as on line 199-203:

```
196                     fmt.Printf("%v : Rejection. new coordinator is temporarily set to %v\n",
197
198                 case NewCoordinator: // set coordinator to sender
199                     self.Coordinator = msg.Sender
200                     fmt.Printf("%v : new coordinator is  %v\n", self.Id, self.Coordinator)
201                     if self.Id == 2 && msg.Sender == self.NumberOfMachines-2 {
202                         self.IsDown = true
203                         fmt.Printf("%v : machine down\n", self.Id)
204                         (*self.WaitGroup).Done()
205                     }
206                     self.IsInElection = false
207                 }
208             }
209         }
210 }
```

The expected result is therefore:



what we are looking for are:

1. machine 4 waits for machine 2 to die before finishing broadcast
2. This would cause any action to be triggered since machine 2 is not a coordinator and does not receive any ping requests

Part 3

Part 3 is a variant of part 1, except there is no specified sender. Thus all machines are pinging the failed coordinator, and triggering elections concurrently.

The output would suggest that the protocol is robust against concurrently election requests:

```
❯ go run -race PSet1/BullyAlgorithm/P2_3/main.go
Press enter to start. Press enter again to stop.
0 : regular ping checks
1 : regular ping checks
2 : regular ping checks
3 : regular ping checks
0 : Regular ping timeout. Checking for machine failure
0 : machine 4 failure detected
0 : machine 4 is coordinator? true
0 : starting election
3 : coordinator request from 0 received
3 : Rejecting coordinator request from 0 received
3 : starting election
1 : coordinator request from 0 received
1 : Rejecting coordinator request from 0 received
1 : starting election
2 : coordinator request from 0 received
2 : Rejecting coordinator request from 0 received
2 : starting election
2 : coordinator request from 1 received
2 : Rejecting coordinator request from 1 received
3 : coordinator request from 1 received
1 : Rejection. new coordinator is temporarily set to 2
2 : starting election
3 : Rejecting coordinator request from 1 received
0 : Rejection. new coordinator is temporarily set to 3
3 : starting election
1 : Rejection. new coordinator is temporarily set to 3
0 : Rejection. new coordinator is temporarily set to 1
0 : Rejection. new coordinator is temporarily set to 2
3 : coordinator request from 2 received
3 : Rejecting coordinator request from 2 received
3 : starting election
2 : Rejection. new coordinator is temporarily set to 3
3 : coordinator request from 2 received
3 : Rejecting coordinator request from 2 received
3 : starting election
2 : Rejection. new coordinator is temporarily set to 3
1 : Election failed.
3 : election succeeeded. Starting broadcast
0 : Election failed.
3 : election succeeeded. Starting broadcast
0 : new coordinator is  3
3 : election succeeeded. Starting broadcast
0 : new coordinator is  3
3 : election succeeeded. Starting broadcast
0 : new coordinator is  3
0 : new coordinator is  3
2 : Election failed.
1 : new coordinator is  3
2 : Election failed.
1 : new coordinator is  3
2 : new coordinator is  3
1 : new coordinator is  3
2 : new coordinator is  3
1 : new coordinator is  3
2 : new coordinator is  3
2 : new coordinator is  3
2 : regular ping checks
3 : ping from 2 received
2 : ping acknowledgement from 3 received

program has ended
```

what we are looking for are:

1. multiple elections are started
2. New coordinator for all machines end up as machine 3 with the highest Id

Part 4

I have made it such that any new machine joining would initiate an election. Thus this would inform all other machines that a new machine with a possibly higher Id has joined.

The code changes can be found here in line 102-110 where an election is always started when a machine joins:

```
92  func machine(self MachineData) {
93      //check state => if Down, don't respond to messages
94      ...ation(self.Id+self.NumberOfMachines) * time.Second) //used for regular ping chec
95       var pingTimeoutChannel chan int  ...n int, 5)
96      pingTimeoutChannel := make(chan int, 2)
97
98      machinesStillAlive := make([]bool, self.NumberOfMachines)
99      for i := 0; i < self.NumberOfMachines; i++ {
100         machinesStillAlive[i] = true
101     }
102     //start election
103     fmt.Printf("%v : newly joined. starting election\n", self.Id)
104     self.IsInElection = true
105     for i := 0; i < self.NumberOfMachines; i++ {
106         if i <= self.Id {
107             continue //only ask machines of high id
108         }
109         self.Channels[i] <- Message{Sender: self.Id, Type: CoordinatorRequest}
110     }
111     self.Coordinator = self.Id //self elect, a reply would override this before timeout
112     go start_timeout(electionTimeoutChannel, self.Timeout)
113     for {
114         select {
115         case <-self.Terminate:
```

This would be an example output:

```
4 : election succeeeded. Starting broadcast
1 : new coordinator is  4
1 : new coordinator is  4
1 : new coordinator is  4
0 : new coordinator is  4
0 : new coordinator is  4
3 : new coordinator is  4
0 : new coordinator is  4
3 : new coordinator is  4
3 : new coordinator is  4
0 : regular ping checks
4 : ping from 0 received
0 : ping acknowledgement from 4 received
1 : regular ping checks
4 : ping from 1 received
1 : ping acknowledgement from 4 received
3 : regular ping checks
4 : ping from 3 received
3 : ping acknowledgement from 4 received
0 : Regular ping timeout. Checking for machine failure
2 : newly joined. starting election
5 : newly joined. starting election
4 : coordinator request from 2 received
4 : Rejecting coordinator request from 2 received
4 : starting election
5 : coordinator request from 3 received
5 : Rejecting coordinator request from 3 received
5 : starting election
```

```
2 : coordinator request from 0 received
2 : Rejecting coordinator request from 0 received
2 : starting election
3 : coordinator request from 2 received
5 : coordinator request from 0 received
3 : Rejecting coordinator request from 2 received
5 : Rejecting coordinator request from 0 received
5 : starting election
0 : Rejection. new coordinator is temporarily set to 2
0 : Rejection. new coordinator is temporarily set to 5
3 : starting election
3 : Rejection. new coordinator is temporarily set to 5
3 : coordinator request from 2 received
3 : Rejecting coordinator request from 2 received
3 : starting election
4 : coordinator request from 2 received
4 : Rejecting coordinator request from 2 received
4 : starting election
4 : coordinator request from 3 received
4 : Rejecting coordinator request from 3 received
4 : starting election
4 : coordinator request from 3 received
4 : Rejecting coordinator request from 3 received
4 : starting election
3 : Rejection. new coordinator is temporarily set to 4
3 : Rejection. new coordinator is temporarily set to 4
2 : coordinator request from 1 received
2 : Rejecting coordinator request from 1 received
2 : starting election
2 : coordinator request from 1 received
2 : Rejecting coordinator request from 1 received
2 : starting election
3 : coordinator request from 2 received
3 : Rejecting coordinator request from 2 received
3 : starting election
2 : new coordinator is  4
2 : new coordinator is  4
2 : new coordinator is  4
2 : new coordinator is  4
2 : new coordinator is  4
3 : coordinator request from 2 received
2 : new coordinator is  4
2 : new coordinator is  4
2 : new coordinator is  4
2 : Rejection. new coordinator is temporarily set to 4
2 : Rejection. new coordinator is temporarily set to 3
2 : Rejection. new coordinator is temporarily set to 3
2 : Rejection. new coordinator is temporarily set to 4
4 : coordinator request from 2 received
2 : Rejection. new coordinator is temporarily set to 3
4 : Rejecting coordinator request from 2 received
5 : coordinator request from 1 received
5 : Rejecting coordinator request from 1 received
3 : Rejecting coordinator request from 2 received
2 : Rejection. new coordinator is temporarily set to 4
4 : starting election
2 : Rejection. new coordinator is temporarily set to 3
1 : Rejection. new coordinator is temporarily set to 2
5 : starting election
1 : Rejection. new coordinator is temporarily set to 2
3 : starting election
1 : Rejection. new coordinator is temporarily set to 5
4 : coordinator request from 2 received
4 : Rejecting coordinator request from 2 received
4 : starting election
5 : coordinator request from 3 received
```

what we are looking for are:

1. Elections are started after machine 2 and 5 arrived
2. eventually machine 5 with the highest Id is elected as the coordinator.