

Readme and Project Description

I] What is the project?

The project is a Fail Silent replicated token manager with atomic semantics. The replication of servers or nodes is distributed across multiple ports on the machine. Each token acts as a (1, N) register for the read and write requests that it serves. The atomic semantics have been implemented using the read-impose-write-all protocol.

II] How do I run the project?

The project already consists of 2 additional protobuf files that are generated using the following command :

```
protoc --go_out=. --go_opt=paths=source_relative --go-grpc_out=. --go-grpc_opt=paths=source_relative proto/service.proto
```

I have implemented a bash script so that whenever you run `server_bash.sh` using the command `sh server_bash.sh` in git bash, the server and the client run at the same time.

If you want to run them one by one :

For servers :

```
go run server.go -ip 127.0.0.20 -port 8000
```

```
go run server.go -ip 127.0.0.30 -port 8000
```

```
go run server.go -ip 127.0.0.31 -port 8001
```

```
go run server.go -ip 127.0.0.32 -port 8002
```

For client :

```
go run client.go -write -id 101 -name abc -low 0 -mid 10 -high 100 -host 127.0.0.20 -port 8000
```

```
go run client.go -read -id 101 -host 127.0.0.30 -port 8000
```

III] Architecture

Server : There is no single server in this project. The system is a collection of multiple reader and writer nodes. In our case, there is only a single writer and multiple readers for a particular token. The system works in an interconnected manner to serve a read and write request for a token.

The servers can be started at each individual port using the command :

```
go run server.go -ip 127.0.0.20 -port 8000
```

where the flags

-ip is the ip address of the server to start

-port is the port for the server to handle requests.

Client : There is only a single client in our case. The client can issue the following requests :

I] Read : The read request looks something like this :

```
client -read -id 1 -host 127.0.0.20 -port 8000
```

The request is sent out to the server serving at the respective ip:port. Based on the type of request the server at that port handles the request appropriately. Here, The server gets a read request which is resolved using the read-impose-write-all protocol. After resolving this request, a string representation of the token is sent out to the client.

-read indicates the type of request

-id is the unique id for the token to be read

-ip is the server's ip address

-port is the server's port

IV] Write : The write request looks like this :

```
client -write -id 1 -name abc -low 0 -mid 10 -high 100 -ip  
127.0.0.20 -port 8000
```

Similar to the read request above, the corresponding server receiving the request responds to the request using the read-impose-write-all protocol.

-write indicates the type of request

-id represents the unique id of the token

-name, -low, -mid, -high represent the data to be written for the specified token

IV] Documentation

The parameters and their notations:

ctx : Create takes in context from the client as the first parameter.

req : This is the actual request object that the client sends. The fields in this object can be accessed using functions defined in protobuf files.

For example, req.GetId() returns the Id that is passed from the client.

*pb.AckResponse : This pointer refers to the string object that is returned to the client side.

I] Read

```
func (r *TokenServiceServer) Read(ctx context.Context,  
req *pb.IdRequest) (*pb.AckResponse, error)
```

The read function is the endpoint that is hit when the client sends a read request. The read function needs to broadcast a message to the other reader nodes requiring the receiving nodes to send their value of the token along with a timestamp. These values and their corresponding timestamps are compared at the current reader node. If the number of acknowledgements received are greater than $N/2$ where N is the number of nodes in the system then the value with the latest timestamp can be returned to the client after broadcasting this new value to the other nodes.

II] RBroadcast

```
func RBroadcast(id string, node string, ch chan  
pb.RBroadcastReturn, req_id uint64)
```

RBroadcast is the actual goroutine whose instances are created by the read function to broadcast a message. This broadcast is achieved by making an rpc call corresponding to each instance of goroutine. The acknowledgements received are tracked with the help of channel which maintains a count of the number of acknowledgements received.

III] ReadOne

```
func (r *TokenServiceServer) ReadOne(ctx context.Context,  
req *pb.RBroadcastMessage) (*pb.RBroadcastReturn, error)
```

ReadOne is the function that receives these asynchronous broadcast messages. After receiving this broadcast message, the function returns the value of the corresponding token as an acknowledgement.

IV] Write

```
func (r *TokenServiceServer) Write(ctx context.Context,  
req *pb.WriteRequest) (*pb.AckResponse, error)
```

The Write function is the endpoint that is hit when the client issues a write request. As per the read-impose-write-all protocol, the server serving the request writes to the appropriate token and then broadcasts this new token value to the other nodes in the system. The moment the number of acknowledgements received are greater than $N/2$ where N is the number of nodes in the system, The write returns the appropriate written value to the server.

V] WBroadcast

```
func WBroadcast(i int, node string, ack_cnt *int, ch chan  
int, req_id uint64){
```

The WBroadcast function is the goroutine that is instantiated when an asynchronous call needs to be made to other nodes. This WBroadcast function issues rpc calls to each and every node in the system asynchronously. This function dumps all the acknowledgements that it receives in a channel.

VI] WriteOne

```
func (r *TokenServiceServer) WriteOne(ctx context.Context,  
req *pb.WBroadcastMessage) (*pb.WBroadcastReturn, error)
```

WriteOne is a basic write function that responds to the asynchronous rpc calls. The function writes the values broadcasted to the node and returns an acknowledgement.

V] Read-Impose-Write-All protocol

The read-impose-write-all protocol implemented in this project is a means to achieve atomic semantics.

The algorithm works differently for the read and write requests.

FOR READ :

- The server node that receives this read request needs to return the latest value written to the whole system. To ensure this, a read broadcast is initiated to every node in the system.
- This read broadcast requires every system to return the value of the appropriate token that they hold.
- This token value is accompanied by a timestamp.
- Once the number of values or acknowledgements received by the node reaches beyond a certain $N/2$ value where N is the number of nodes in the system, the values associated to the latest timestamp is updated on the current node and then broadcasted to all the nodes again.

FOR WRITE :

- The server node that receives this write request firstly updates the current value of the token along with its timestamp.
- This new value written is then broadcasted across all the nodes in the system.
- Once we receive acknowledgements more than $N/2$ from these broadcasts, we respond to the client with an acknowledgement.

V] Fail Silent Model and it's simulation in the project.

Fail silent model is a failure model in which we cannot detect between a crash failure or omission failure.

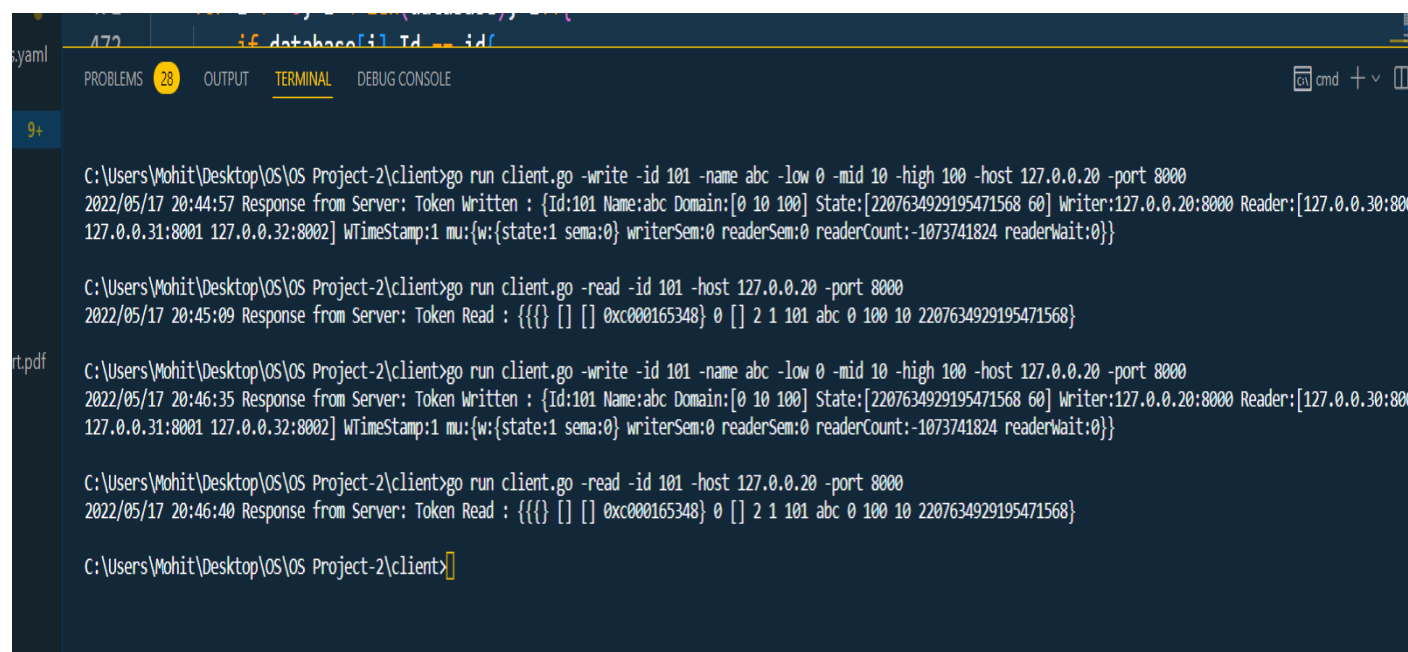
Simulation of Fail Silent model for the READ :

In order to simulate the fail silent model for read, I introduced a sleep function at a particular node in the project. This allowed me to simulate that the server at that port is down.

The sleep function was introduced at the entry point of ReadOne function.

```
if port_address == "127.0.0.32:8002"{
    log.Printf("Sleeping")
    time.Sleep(100 * time.Second)
}
```

As you can see, the sleep was introduced at the server running on 127.0.0.32:8002. This condition simulated a failure of node out of the 3 reader nodes in my project. Despite the failure of the node, the other two reader nodes establish a majority by broadcasting their values to the server. The value with the latest timestamp is updated at the current node and then the server responds to client in a usual manner.



```
C:\Users\Mohit\Desktop\OS Project-2\client>go run client.go -write -id 101 -name abc -low 0 -mid 10 -high 100 -host 127.0.0.20 -port 8000
2022/05/17 20:44:57 Response from Server: Token Written : {Id:101 Name:abc Domain:[0 10 100] State:[2207634929195471568 60] Writer:127.0.0.20:8000 Reader:[127.0.0.30:8000 127.0.0.31:8001 127.0.0.32:8002] WTimeStam:1 mu:{w:{state:1 sema:0} writerSem:0 readerSem:0 readerCount:-1073741824 readerWait:0}}

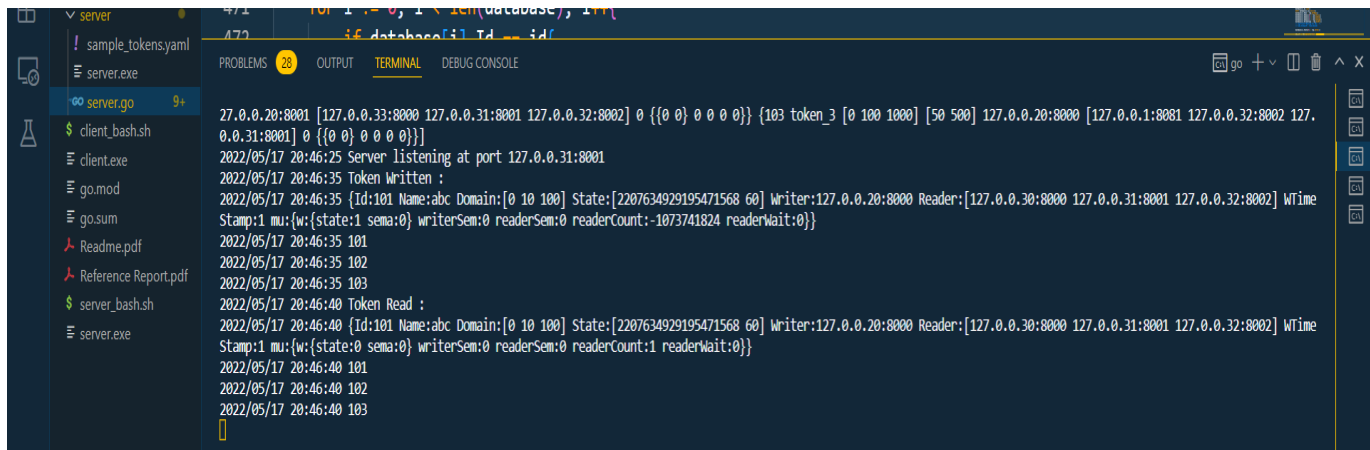
C:\Users\Mohit\Desktop\OS Project-2\client>go run client.go -read -id 101 -host 127.0.0.20 -port 8000
2022/05/17 20:45:09 Response from Server: Token Read : {{{} [] [] 0xc000165348} 0 [] 2 1 101 abc 0 100 10 2207634929195471568}

C:\Users\Mohit\Desktop\OS Project-2\client>go run client.go -write -id 101 -name abc -low 0 -mid 10 -high 100 -host 127.0.0.20 -port 8000
2022/05/17 20:46:35 Response from Server: Token Written : {Id:101 Name:abc Domain:[0 10 100] State:[2207634929195471568 60] Writer:127.0.0.20:8000 Reader:[127.0.0.30:8000 127.0.0.31:8001 127.0.0.32:8002] WTimeStam:1 mu:{w:{state:1 sema:0} writerSem:0 readerSem:0 readerCount:-1073741824 readerWait:0}}

C:\Users\Mohit\Desktop\OS Project-2\client>go run client.go -read -id 101 -host 127.0.0.20 -port 8000
2022/05/17 20:46:40 Response from Server: Token Read : {{{} [] [] 0xc000165348} 0 [] 2 1 101 abc 0 100 10 2207634929195471568}

C:\Users\Mohit\Desktop\OS Project-2\client>
```

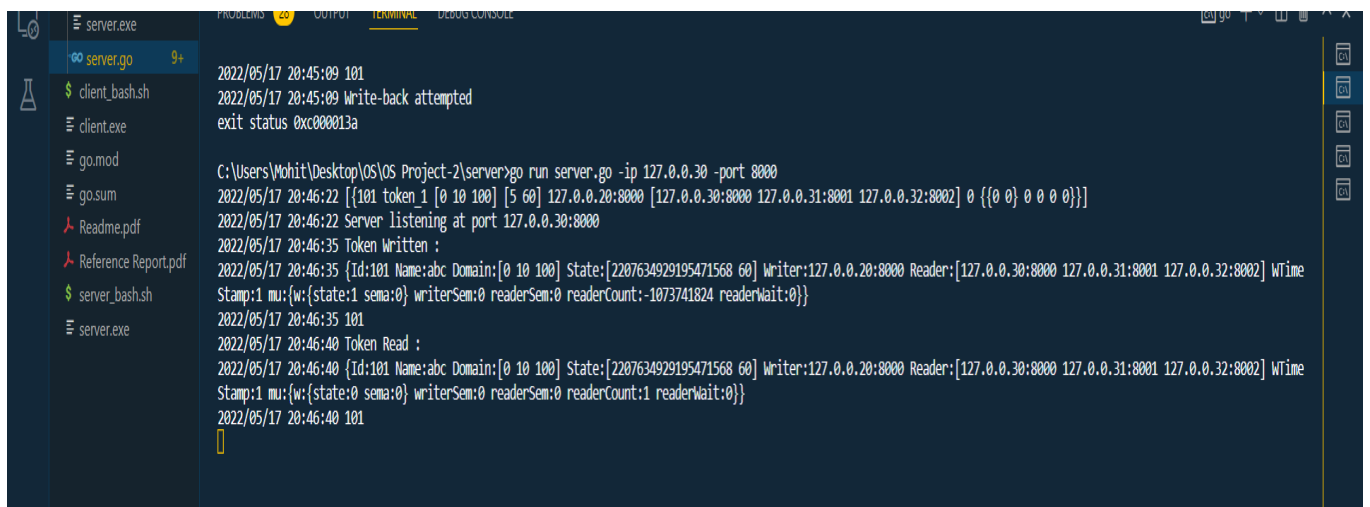
The logs of the sleeping node are documented below :



```
server
! sample_tokens.yaml
server.exe
server.go 9+
$ client_bash.sh
client.exe
go.mod
go.sum
Readme.pdf
Reference Report.pdf
server_bash.sh
server.exe

27.0.0.20:8001 [127.0.0.33:8000 127.0.0.31:8001 127.0.0.32:8002] 0 {{0 0} 0 0 0}} {103 token_3 [0 100 1000] [50 500] 127.0.0.20:8000 [127.0.0.1:8001 127.0.0.32:8002] 127.0.0.31:8001} 0 {{0 0} 0 0 0}}
2022/05/17 20:46:25 Server listening at port 127.0.0.31:8001
2022/05/17 20:46:35 Token Written :
2022/05/17 20:46:35 {Id:101 Name:abc Domain:[0 10 100] State:[2207634929195471568 60] Writer:127.0.0.20:8000 Reader:[127.0.0.30:8000 127.0.0.31:8001 127.0.0.32:8002] WTime Stamp:1 mu:{w:{state:1 sema:0} writerSem:0 readerSem:0 readerCount:-1073741824 readerWait:0}}
2022/05/17 20:46:35 101
2022/05/17 20:46:35 102
2022/05/17 20:46:35 103
2022/05/17 20:46:40 Token Read :
2022/05/17 20:46:40 {Id:101 Name:abc Domain:[0 10 100] State:[2207634929195471568 60] Writer:127.0.0.20:8000 Reader:[127.0.0.30:8000 127.0.0.31:8001 127.0.0.32:8002] WTime Stamp:1 mu:{w:{state:0 sema:0} writerSem:0 readerSem:0 readerCount:1 readerWait:0}}
2022/05/17 20:46:40 101
2022/05/17 20:46:40 102
2022/05/17 20:46:40 103
[]
```

Similarly outputs on all the concerned terminals are as follows :



```
server.exe
server.go 9+
$ client_bash.sh
client.exe
go.mod
go.sum
Readme.pdf
Reference Report.pdf
server_bash.sh
server.exe

2022/05/17 20:45:09 101
2022/05/17 20:45:09 Write-back attempted
exit status 0xc000013a

C:\Users\Mohit\Desktop\OS\OS Project-2\server>go run server.go -ip 127.0.0.30 -port 8000
2022/05/17 20:46:22 [{101 token_1 [0 10 100] [5 60] 127.0.0.20:8000 [127.0.0.30:8000 127.0.0.31:8001 127.0.0.32:8002] 0 {{0 0} 0 0 0}}]
2022/05/17 20:46:22 Server listening at port 127.0.0.30:8000
2022/05/17 20:46:35 Token Written :
2022/05/17 20:46:35 {Id:101 Name:abc Domain:[0 10 100] State:[2207634929195471568 60] Writer:127.0.0.20:8000 Reader:[127.0.0.30:8000 127.0.0.31:8001 127.0.0.32:8002] WTime Stamp:1 mu:{w:{state:1 sema:0} writerSem:0 readerSem:0 readerCount:-1073741824 readerWait:0}}
2022/05/17 20:46:35 101
2022/05/17 20:46:40 Token Read :
2022/05/17 20:46:40 {Id:101 Name:abc Domain:[0 10 100] State:[2207634929195471568 60] Writer:127.0.0.20:8000 Reader:[127.0.0.30:8000 127.0.0.31:8001 127.0.0.32:8002] WTime Stamp:1 mu:{w:{state:0 sema:0} writerSem:0 readerSem:0 readerCount:1 readerWait:0}}
2022/05/17 20:46:40 101
[]
```

```

472         if database[i].Id == id{
PROBLEMS 28 OUTPUT TERMINAL DEBUG CONSOLE
2022/05/17 20:45:09 101
2022/05/17 20:45:09 Write-back attempted
exit status 0xc000013a

C:\Users\Mohit\Desktop\OS\OS Project-2\server>go run server.go -ip 127.0.0.30 -port 8000
2022/05/17 20:46:22 [{101 token_1 [0 10 100] [5 60] 127.0.0.20:8000 [127.0.0.30:8000 127.0.0.31:8001 127.0.0.32:8002] 0 {{0 0} 0 0 0 0}}]
2022/05/17 20:46:22 Server listening at port 127.0.0.30:8000
2022/05/17 20:46:35 Token Written :
2022/05/17 20:46:35 {Id:101 Name:abc Domain:[0 10 100] State:[2207634929195471568 60] Writer:127.0.0.20:8000 Reader:[127.0.0.30:8000 127.0.0.31:8001 127.0.0.32:8002] Stamp:1 mu:{w:{state:1 sema:0} writerSem:0 readerSem:0 readerCount:-1073741824 readerWait:0}}
2022/05/17 20:46:35 101
2022/05/17 20:46:40 Token Read :
2022/05/17 20:46:40 {Id:101 Name:abc Domain:[0 10 100] State:[2207634929195471568 60] Writer:127.0.0.20:8000 Reader:[127.0.0.30:8000 127.0.0.31:8001 127.0.0.32:8002] Stamp:1 mu:{w:{state:0 sema:0} writerSem:0 readerSem:0 readerCount:1 readerWait:0}}
2022/05/17 20:46:40 101

```

```

471         for i := 0; i < len(database); i++){
472             if database[i].Id == id{
PROBLEMS 28 OUTPUT TERMINAL DEBUG CONSOLE
2022/05/17 20:46:35 Value : 1
2022/05/17 20:46:35 Value : 3
2022/05/17 20:46:40 Token Read :
2022/05/17 20:46:40 {Id:101 Name:abc Domain:[0 10 100] State:[2207634929195471568 60] Writer:127.0.0.20:8000 Reader:[127.0.0.30:8000 127.0.0.31:8001 127.0.0.32:8002] Stamp:1 mu:{w:{state:0 sema:0} writerSem:0 readerSem:0 readerCount:1 readerWait:0}}
2022/05/17 20:46:40 101
2022/05/17 20:46:40 103
2022/05/17 20:46:40 WriteBack Triggerred
2022/05/17 20:46:40 Token Written :
2022/05/17 20:46:40 {Id:101 Name:abc Domain:[0 10 100] State:[2207634929195471568 60] Writer:127.0.0.20:8000 Reader:[127.0.0.30:8000 127.0.0.31:8001 127.0.0.32:8002] Stamp:1 mu:{w:{state:1 sema:0} writerSem:0 readerSem:0 readerCount:-1073741824 readerWait:0}}
2022/05/17 20:46:40 101
2022/05/17 20:46:40 103
2022/05/17 20:46:40 Value : 1
2022/05/17 20:46:40 Value : 2

```

Simulation of Fail Silent model for the Write :

```

if port_address == "127.0.0.32:8002"{
    log.Printf("Sleeping...")
    time.Sleep(1000 * time.Second)
}

```

The above code was introduced at the entry point of the WriteOne function to simulate a failure of node.

During the write request, after writing the current node, a broadcast was made to 3 other reader nodes out of which 1 failed. The remaining two nodes responded with an appropriate acknowledgement. This led to majority in the quorum and an appropriate message was sent to the client even if a node failed.


```
exit status 0xc000013a
```

```
C:\Users\Mohit\Desktop\OS\OS Project-2\server>go run server.go -ip 127.0.0.20 -port 8000
```

```
2022/05/17 17:25:28 [{101 token 1 [0 10 100] [5 60] 127.0.0.20:8000 [127.0.0.30:8000 127.0.0.31:8001 127.0.0.32:8002] 0 {{0 0} 0 0 0 0}}  
127.0.0.20:8000 [127.0.0.1:8081 127.0.0.32:8002 127.0.0.31:8001] 0 {{0 0} 0 0 0 0}}]
```

```
2022/05/17 17:25:28 Server listening at port 127.0.0.20:8000
```

```
2022/05/17 17:25:51 Token Written :
```

```
2022/05/17 17:25:51 {Id:101 Name:abc Domain:[0 10 100] State:[2207634929195471568 60] Writer:127.0.0.20:8000 Reader:[127.0.0.30:8000 127.
```

```
Stamp:1 mu:{w:{state:1 sema:0} writerSem:0 readerSem:0 readerCount:-1073741824 readerWait:0}}
```

```
2022/05/17 17:25:51 101
```

```
2022/05/17 17:25:51 103
```

```
2022/05/17 17:25:51 Value : 1
```

```
2022/05/17 17:25:51 Value : 2
```

```
]
```

```
432 log.Printf("Sleeping...")
```

PROBLEMS 28 OUTPUT TERMINAL DEBUG CONSOLE

```
C:\Users\Mohit\Desktop\OS\OS Project-2\client>go run client.go -read -id 101 -host 127.0.0.32 -port 8002
```

```
2022/05/17 17:13:48 Response from Server: Token Read : {{{} [] [] 0xc000131348} 0 [] 1 1 101 abc 0 100 10 2207634929195471568}
```

```
C:\Users\Mohit\Desktop\OS\OS Project-2\client>go run client.go -write -id 101 -name abc -low 0 -mid 10 -high 100 -host 127.0.0.20 -port 8000
```

```
2022/05/17 17:19:08 Response from Server: Token Written : {Id:101 Name:abc Domain:[0 10 100] State:[2207634929195471568 60] Writer:127.0.0.20:8000 Reader:  
127.0.0.31:8001 127.0.0.32:8002] WTimeStamp:1 mu:{w:{state:1 sema:0} writerSem:0 readerSem:0 readerCount:-1073741824 readerWait:0}}
```

```
C:\Users\Mohit\Desktop\OS\OS Project-2\client>go run client.go -write -id 101 -name abc -low 0 -mid 10 -high 100 -host 127.0.0.20 -port 8000
```

```
2022/05/17 17:25:51 Response from Server: Token Written : {Id:101 Name:abc Domain:[0 10 100] State:[2207634929195471568 60] Writer:127.0.0.20:8000 Reader:  
127.0.0.31:8001 127.0.0.32:8002] WTimeStamp:1 mu:{w:{state:1 sema:0} writerSem:0 readerSem:0 readerCount:-1073741824 readerWait:0}}
```

```
C:\Users\Mohit\Desktop\OS\OS Project-2\client>
```

```
431 if port_address == 127.0.0.32:8002 {
```

```
432 log.Printf("Sleeping...")
```

PROBLEMS 28 OUTPUT TERMINAL DEBUG CONSOLE

```
Stamp:1 mu:{w:{state:1 sema:0} writerSem:0 readerSem:0 readerCount:-1073741824 readerWait:0}}
```

```
2022/05/17 17:19:08 101
```

```
exit status 0xc000013a
```

```
C:\Users\Mohit\Desktop\OS\OS Project-2\server>go run server.go -ip 127.0.0.30 -port 8000
```

```
2022/05/17 17:25:34 [{101 token 1 [0 10 100] [5 60] 127.0.0.20:8000 [127.0.0.30:8000 127.0.0.31:8001 127.0.0.32:8002] 0 {{0 0} 0 0 0 0}}]  
2022/05/17 17:25:34 Server listening at port 127.0.0.30:8000
```

```
2022/05/17 17:25:51 Write-back attempted
```

```
2022/05/17 17:25:51 Token Written :
```

```
2022/05/17 17:25:51 {Id:101 Name:abc Domain:[0 10 100] State:[2207634929195471568 60] Writer:127.0.0.20:8000 Reader:[127.0.0.30:8000 127.0.
```

```
Stamp:1 mu:{w:{state:1 sema:0} writerSem:0 readerSem:0 readerCount:-1073741824 readerWait:0}}
```

```
2022/05/17 17:25:51 101
```

```
]
```

```
432      log.Printf("Sleeping...")
PROBLEMS 28 OUTPUT TERMINAL DEBUG CONSOLE
exit status 0xc000013a

C:\Users\Mohit\Desktop\OS\OS Project-2\server>go run server.go -ip 127.0.0.31 -port 8001
2022/05/17 17:25:38 [{101 token_1 [0 10 100] [5 60] 127.0.0.20:8000 [127.0.0.30:8000 127.0.0.31:8001 127.0.0.32:8002] 0 {{0 0} 0 0 0 0}} {102 tok
27.0.0.20:8001 [127.0.0.33:8000 127.0.0.31:8001 127.0.0.32:8002] 0 {{0 0} 0 0 0 0}} {103 token_3 [0 100 1000] [50 500] 127.0.0.20:8000 [127.0.0.1
0.0.31:8001] 0 {{0 0} 0 0 0 0}}]
2022/05/17 17:25:38 Server listening at port 127.0.0.31:8001
2022/05/17 17:25:51 Write-back attempted
2022/05/17 17:25:51 Token Written :
2022/05/17 17:25:51 {Id:101 Name:abc Domain:[0 10 100] State:[2207634929195471568 60] Writer:127.0.0.20:8000 Reader:[127.0.0.30:8000 127.0.0.31:8
Stamp:1 mu:{w:{state:1 sema:0} writerSem:0 readerSem:0 readerCount:-1073741824 readerWait:0}}
2022/05/17 17:25:51 101
2022/05/17 17:25:51 102
2022/05/17 17:25:51 103
[]
```

```
PROBLEMS 28 OUTPUT TERMINAL DEBUG CONSOLE
exit status 0xc000013a

C:\Users\Mohit\Desktop\OS\OS Project-2\server>go run server.go -ip 127.0.0.32 -port 8002
2022/05/17 17:25:41 [{101 token_1 [0 10 100] [5 60] 127.0.0.20:8000 [127.0.0.30:8000 127.0.0.31:8001 127.0.0.32:8002] 0 {{0 0}
27.0.0.20:8001 [127.0.0.33:8000 127.0.0.31:8001 127.0.0.32:8002] 0 {{0 0} 0 0 0 0}} {103 token_3 [0 100 1000] [50 500] 127.0.
0.0.31:8001] 0 {{0 0} 0 0 0 0}}]
2022/05/17 17:25:41 Server listening at port 127.0.0.32:8002
2022/05/17 17:25:51 Sleeping....
[]
```

The above log indicates the sleeping log indicating that the node did crash during the write request.