

File - /Users/chadanlo/go/src/prr-labo3/labo3/network/network.go

```
1 /*
2 -----
3 Lab      : 03
4 File     : network.go
5 Authors  : François Burgener – Tiago P. Quinteiro
6 Date    : 10.12.19
7
8 Goal     : Network layer for the algorithm of chang and robert (bully)
9 -----
10 */
11
12 package network
13
14 import (
15     "bufio"
16     "bytes"
17     "log"
18     "net"
19     "pr-r-labo3/labo3/config"
20     "pr-r-labo3/labo3/network/messages"
21     "pr-r-labo3/labo3/utlis"
22     "time"
23 )
24
25 type Manager interface {
26     SubmitNotification(notifMap map[uint16]uint16)
27     SubmitResult(id uint16, resultMap map[uint16]bool)
28 }
29
30 type Network struct {
31     id uint16
32     N  uint16
33     manager Manager
34     Debug bool
35 }
36
37 /**
38  * Method to init our Network
39  * @param id of the processus
40  * @param N number of processus
41  */
42 func (n *Network) Init(id uint16, N uint16, manager Manager) {
43     log.Println("Network : Initialization of the network")
44     n.id = id
45     n.N = N
46     n.manager = manager
47
48     go func() {
49         n.initServ()
50     }()
51 }
52
53 /**
54  * Method to init our udp server
55  */
56 func (n *Network) initServ() {
57     addr := utlis.AddressByID(n.id)
58     conn, err := net.ListenPacket("udp", addr)
59     if err != nil {
60         log.Fatal("Network error: Initialisation failed", err)
61     }
62     defer conn.Close()
63
64     n.handleConn(conn)
65 }
66
67 /**
68  * Method to init our Network
69  * @param id of the processus
70  * @param N number of processus
71  */
72 func (n *Network) handleConn(conn net.PacketConn) {
73     buf := make([]byte, 1024)
74     for {
75         l, cliAddr, err := conn.ReadFrom(buf)
76         if err != nil {
77             log.Fatal("Network error: Reading error ", err)
78         }
79         s := bufio.NewScanner(bytes.NewReader(buf[0:l]))
80         for s.Scan() {
81             buf := s.Bytes()
82             n.emitACK(conn, cliAddr)
83             n.decodeMessage(buf)
84         }
85     }
86 }
87
88 /**
89  * Method to emit a notification
90  * @param _map with all processus and this aptitude
91  */
```

File - /Users/chadanlo/go/src/prr-labo3/labo3/network/network.go

```
92 func (n *Network) EmitNotif(_map map[uint16]uint16){
93     notif := messages.MessageNotif{_map}
94     msg := utils.EncodeMessageNotif(notif)
95     buf := utils.InitMessage([]byte(config.NotifMessage),msg)
96     n.emit(buf)
97
98     if n.Debug{
99         log.Println("Network : Emit notification : ",_map)
100     }
101 }
102
103 /**
104  * Method to emit a result
105  * @param id processus who is elected
106  * @param _map of processus who send the result
107  */
108 func (n *Network) EmitResult(id uint16,_map map[uint16]bool){
109     result := messages.MessageResult{id,_map}
110     msg := utils.EncodeMessageResult(result)
111     buf := utils.InitMessage([]byte(config.ResultMessage),msg)
112     n.emit(buf)
113
114     if n.Debug{
115         log.Println("Network : Emit result : id-",id," map-",_map)
116     }
117 }
118
119 /**
120  * Method to emit a ACK
121  * @param conn conn of the client
122  * @param cliAddr address of the client
123  */
124 func (n *Network) emitACK(conn net.PacketConn, cliAddr net.Addr) {
125     ack := messages.Message{n.id}
126     msg := utils.EncodeMessage(ack)
127     buf := utils.InitMessage([]byte(config.AckMessage),msg)
128
129     if _, err := conn.WriteTo(buf, cliAddr); err != nil {
130         log.Fatal("Network error: Writing error ",err)
131     }
132
133     if n.Debug{
134         log.Println("Network : Emit ACK")
135     }
136 }
137
138 /**
139  * Method to emit an ECHO
140  * @param id of the processus we want to send
141  * @return true if we received an ACK, false otherwise
142  */
143 func (n *Network) EmitEcho(id uint16) bool {
144     channel := make(chan bool, 1) // channel to know if we received an ACK
145     echo := messages.Message{n.id}
146     msg := utils.EncodeMessage(echo)
147     buf := utils.InitMessage([]byte(config.EchoMessage),msg)
148
149     if n.Debug {
150         log.Println("Network : Emit ECHO : ",n.id)
151     }
152
153     go n.emitById(buf,id,channel)
154
155     select {
156     case <-channel: //We received an ACK
157         return true
158     case <-time.After(config.TIME_OUT): // Timeout
159         log.Println("Network : Timeout")
160         return false
161     }
162 }
163
164 /**
165  * Method to emit a message of our next processus (Id + 1) with we can we try another (id + 2) ect
166  * @param msg we want to send
167  */
168 func (n *Network) emit(msg []byte) {
169     for i:= n.id; i < n.N + n.id; i++){
170
171         id := (i + 1) % n.N // id of the next processus
172         channel := make(chan bool, 1) // channel to know if we received an ACK
173         receivedACK := false //Boolean to stop the loop if we received an ACK
174
175         //Emit message to the next processus
176         n.emitById(msg,id,channel)
177
178         select {
179         case receivedACK = <-channel: //We received an ACK
180         case <-time.After(config.TIME_OUT): // Timeout
```

File - /Users/chadanlo/go/src/prr-labo3/labo3/network/network.go

```
183         log.Println("Network : Timeout")
184         continue
185     }
186
187     //If we received an ACK, we stop the loop
188     if receivedACK{
189         break
190     }
191 }
192 }
193
194 /**
195  * Method to emit an message
196  * @param msg we want to send
197  * @param id of the processus we want to send
198  * @param channel to say if we received ACK
199  */
200 func (n *Network) emitById(msg []byte, id uint16, channel chan bool) {
201     add := utils.AddressByID(id)
202     addr, err := net.ResolveUDPAddr("udp", add)
203     if err != nil {
204         log.Printf("The processus %d is not alive ", id)
205     }
206
207     conn, err := net.DialUDP("udp", nil, addr)
208     if err != nil {
209         log.Println("Network error: Error dial", err.Error())
210     }
211
212     _, err = conn.Write(msg)
213     if err != nil {
214         log.Fatal("Network error: Writing error ", err)
215     }
216
217     go n.readACK(conn, channel)
218 }
219 }
220
221 /**
222  * Method to read an ACK message
223  * @param conn to read the ack
224  * @param channel to say if we received ACK
225  */
226 func (n *Network) readACK(conn net.Conn, channel chan bool){
227     // Make a buffer to hold incoming data.
228     buf := make([]byte, 1024)
229
230     // Read the incoming connection into the buffer.
231     l, err := conn.Read(buf)
232     if err != nil {
233         log.Println("Network error: Error reading", err.Error()) //TODO Check
234     }
235
236     s := bufio.NewScanner(bytes.NewReader(buf[0:l]))
237
238     for s.Scan(){
239         buf := s.Bytes()
240         if string(buf[0:3]) == config.AckMessage{
241             msg := utils.DecodeMessage(buf[3:])
242
243             channel <- true
244
245             if n.Debug{
246                 log.Println("Decode : ", string(buf[0:3]), "-", msg.Id)
247             }
248         }
249     }
250 }
251 }
252
253 /**
254  * Method to read decode a message
255  * @param buf array of byte we want to decode
256  */
257 func (n *Network) decodeMessage(buf []byte) {
258     _type := string(buf[0:3])
259
260     switch _type {
261     case config.EchoMessage:
262         msg := utils.DecodeMessage(buf[3:])
263
264         if n.Debug{
265             log.Println("Decode", _type, "-", msg.Id)
266         }
267     case config.ResultMessage:
268         msg := utils.DecodeMessageResult(buf[3:])
269
270         if n.Debug{
271             log.Println("Decode", _type, "-", msg.Id, "-", msg.Map)
272         }
273     }
```

File - /Users/chadanlo/go/src/prr-labo3/labo3/network/network.go

```
274
275     n.manager.SubmitResult(msg.Id,msg.Map)
276 case config.NotifMessage:
277     msg := utils.DecodeMessageNotif(buf[3:])
278
279     if n.Debug{
280         log.Println("Decode",_type,"-",msg.Map)
281     }
282
283     n.manager.SubmitNotification(msg.Map)
284 default:
285     log.Println("Network: Incorrect type message !")
286 }
287 }
288
289
290
```