

File - /Users/chadanlo/go/src/prr-labo2/labo2/network/network.go

```
1 /*
2 -----
3 Lab      : 02
4 File     : network.go
5 Authors  : François Burgener – Tiago P. Quinteiro
6 Date    : 03.12.2019
7
8 Goal     : Network layer for the algorithm of Carvalho et Roucairol
9 -----
10 */
11
12 package network
13
14 import (
15     "PRR-Labo2/labo2/config"
16     "PRR-Labo2/labo2/utlis"
17     "bufio"
18     "bytes"
19     "io"
20     "log"
21     "net"
22     "strconv"
23 )
24
25
26 /*****
27 *          INTERFACE          *
28 *****/
29 type Mutex interface {
30     Req(stamp uint32, id uint16)
31     Ok(stamp uint32, id uint16)
32     Update(value uint32)
33 }
34
35 /*****
36 *          STRUCTURE          *
37 *****/
38 type Network struct {
39     id uint16 //id of our processus
40     nProc uint16 // Number of processus
41     directory map[uint16]net.Conn // map of connection
42     Done chan string // channel to say if the server initialisation is done
43     mutex Mutex //Ref of our mutex
44     Debug bool
45 }
46
47 /*****
48 *          NETWORK METHOD          *
49 *****/
50
51 /**
52 * Method of Network to send a MessageREQ message
53 * @param stamp (logic clock) of the processus
54 * @param id of the processus
55 */
56 func (n *Network) REQ(stamp uint32, id uint16){
57     msg := utlis.InitMessage(stamp,n.id,[]byte(config.MessageREQ))
58     //_, err := n.directory[id].Write(msg)
59     mustCopy(n.directory[id], bytes.NewReader(msg))
60
61     if n.Debug{
62         log.Printf("Network: Send message type:%s stamp:%d id:%d \n", config.MessageREQ,stamp,id)
63     }
64 }
65
66
67 /**
68 * Method of Network to send a MessageREQ message
69 * @param stamp (logic clock) of the processus
70 * @param id of the processus
71 */
72 func (n *Network) OK(stamp uint32, id uint16){
73     msg := utlis.InitMessage(stamp,n.id,[]byte(config.MessageOK))
74
75     mustCopy(n.directory[id], bytes.NewReader(msg))
76
77     if n.Debug{
78         log.Printf("Network: Send message type:%s stamp:%d id:%d \n", config.MessageOK,stamp,id)
79     }
80 }
81
82 /**
83 * Method of Network to send a MessageUPDATE message
84 * @param value to update
85 */
86 func (n *Network) UPDATE(value uint32){
87     for i:=0; i < len(n.directory) + 1; i++){
88         if i != int(n.id){
89             msg := utlis.InitMessageUpdate(value,[]byte(config.MessageUPDATE))
90             mustCopy(n.directory[uint16(i)], bytes.NewReader(msg))
91         }
92     }
93 }
```

File - /Users/chadanlo/go/src/prr-labo2/labo2/network/network.go

```
92         if n.Debug{
93             log.Printf("Network: Send message Update P%d value: %d",i,value)
94         }
95     }
96 }
97 }
98
99
100 /**
101  * Method to init the server and get all connection between processus
102  * @param id of the processus
103  * @param N number of processus
104  * @param mutex ref to mutex
105  */
106 func (n *Network) Init(id uint16,N uint16, mutex Mutex) {
107     log.Printf("Network: Initialisation ")
108     n.directory = make(map[uint16]net.Conn,N)
109     n.Done = make(chan string)
110     n.mutex = mutex
111     n.id = id
112     n.nProc = N
113
114     go func() {
115         n.initAllConn()
116         n.initServ()
117     }()
118
119     <- n.Done
120 }
121
122 // PRIVATE methods -----
123
124 /**
125  * Method to init all dial connection
126  */
127 func (n *Network) initAllConn() {
128     for i:=uint16(0) ; i < n.nProc; i++ {
129         if i != uint16(n.id) {
130             n.initConn(i)
131         }
132     }
133 }
134
135 /**
136  * Method to init a dial connection
137  * @param i id of the processus we want to connect
138  */
139 func (n *Network) initConn(i uint16) {
140     addr := utils.AddressByID(uint16(i))
141     conn, err := net.Dial("tcp", addr)
142
143     if err != nil {
144         log.Printf("Network error : Connection refused with P%d",i)
145     }else{
146         n.directory[uint16(i)] = conn
147         _, err := conn.Write([]byte(strconv.Itoa(int(n.id))))
148         if err != nil{
149             log.Fatalf("Network error: Writing error:", err.Error())
150         }
151
152         if n.Debug{
153             log.Printf("Network : Dial Connection between P%d and P%d\n", n.id, i)
154         }
155
156         go n.handleConn(conn)
157     }
158 }
159
160
161 /**
162  * Method to init a new Network
163  */
164 func (n *Network) initServ(){
165     addr := utils.AddressByID(n.id)
166     listener, err := net.Listen("tcp", addr)
167     if err != nil {
168         log.Fatalf("Network error: Listen error:", err.Error())
169     }
170
171     defer listener.Close()
172
173     for {
174
175         if len(n.directory) == int(n.nProc-1) {
176             n.Done <- "done"
177         }
178
179         conn, err := listener.Accept()
180         if err != nil {
181             log.Fatalf("Network error: Listen accept error:", err.Error())
182         }
```

File - /Users/chadanlo/go/src/prr-labo2/labo2/network/network.go

```
183
184
185     tmp := make([]byte,128)
186     l, err := conn.Read(tmp)
187     if err != nil {
188         log.Fatal("Network error: Reading error:", err.Error())
189     }
190     str := string(tmp[0:l])
191     idConn, err := strconv.Atoi(str)
192     if err != nil {
193         log.Fatal("Network error: Cannot take the id of the processus:", err.Error())
194     }
195
196     log.Println("Network: Serv Connection between P" + strconv.Itoa(int(n.id)) + " and P" + strconv.Itoa(idConn))
197     n.directory[uint16(idConn)] = conn
198
199     go n.handleConn(conn)
200 }
201 }
202
203 /**
204  * Method to read message
205  */
206 func (n *Network)handleConn(conn net.Conn) {
207     for {
208         // Make a buffer to hold incoming data.
209         buf := make([]byte, 32)
210
211         // Read the incoming connection into the buffer.
212         l, err := conn.Read(buf)
213         if err != nil {
214             log.Fatal("Network error: Error reading:", err.Error())
215         }
216
217         s := bufio.NewScanner(bytes.NewReader(buf[0:l]))
218
219         for s.Scan(){
220             n.decodeMessage(s.Bytes())
221         }
222     }
223 }
224 }
225
226 func (n *Network) decodeMessage(bytes []byte) {
227
228     _type := string(bytes[0:3])
229     var stamp uint32
230     var id uint16
231     var value uint32
232
233     if _type == config.MessageUPDATE {
234         value = utils.ConverByteArrayToUint32(bytes[3:7])
235
236         if n.Debug{
237             log.Printf("Network: Decoded message type:%s value:%d",_type,value)
238         }
239     }else if _type == config.MessageOK || _type == config.MessageREQ {
240         stamp = utils.ConverByteArrayToUint32(bytes[3:7])
241         id = utils.ConverByteArrayToUint16(bytes[7:9])
242
243         if n.Debug{
244             log.Printf("Network: Decoded message type:%s stamp:%d id:%d",_type,stamp,id)
245         }
246     }
247
248 }
249
250
251 switch _type {
252 case config.MessageREQ:
253     n.mutex.Req(stamp,id)
254 case config.MessageOK:
255     n.mutex.Ok(stamp,id)
256 case config.MessageUPDATE:
257     n.mutex.Update(value)
258 default:
259     log.Println("Network: Incorrect type message !")
260 }
261 }
262
263 func mustCopy(dst io.Writer, src io.Reader) {
264     if _, err := io.Copy(dst, src); err != nil {
265         log.Fatal(err)
266     }
267 }
```