

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

```
1 /*
2 -----
3 Lab      : 03
4 File     : manager.go
5 Authors  : François Burgener – Tiago P. Quinteiro
6 Date    : 10.12.19
7
8 Goal     : Implements the manager for the bully algorithm of Chang and Roberts
9 -----
10 */
11 package manager
12
13 import (
14     "log"
15 )
16
17 /**
18  * ENUM declaration of the states
19  */
20 const (
21     NOTIFICATION = iota
22     RESULT
23 )
24
25 /**
26  * Interface wanted for the Network
27  */
28 type Network interface {
29     EmitNotif(map[uint16]uint16)
30     EmitResult(uint16, map[uint16]bool)
31 }
32
33 /**
34  * private utility struct
35  * to send through channels
36  * a result message
37  */
38 type resultMessage struct {
39     id uint16
40     visitedResult map[uint16]bool
41 }
42
43 /**
44  * Manager class
45  */
46 type Manager struct {
47     N uint16
48     me uint16
49     aptitude uint16
50     state uint8
51     elected uint16
52     asked bool
53     debug bool
54     network Network
55     chanAskElection chan bool
56     chanGiveElection chan uint16
57     chanNotification chan map[uint16]uint16
58     chanResult chan resultMessage
59     chanAsk chan bool
60 }
61
62 /**
63  * Constructor
64  * @param N number of Processes
65  * @param me id of this Process
66  * @param aptitude the aptitude of this Process
67  * @param network a struct which represents the network layer
68  */
69 func (m *Manager) Init(N uint16, me uint16, aptitude uint16, network Network) {
70     log.Println("Manager : Initialization of the manager")
71     m.N = N
72     m.me = me
73     m.aptitude = aptitude
74     m.network = network
75     m.state = RESULT
76     m.asked = false
77
78     //Channels
79     m.chanAskElection = make(chan bool)
80     m.chanGiveElection = make(chan uint16)
81     m.chanNotification = make(chan map[uint16]uint16)
82     m.chanResult = make(chan resultMessage)
83     m.chanAsk = make(chan bool)
84
85     // Debug
86     m.debug = true
87
88     go m.handler()
89 }
90
91 /**
```

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

```
92  * Once Init, this handler will treat incoming requests
93  * from Task and Network
94  */
95 func (m *Manager) handler() {
96     for {
97         select {
98             case <- m.chanAskElection:
99                 m.handleElection()
100             case notifMap := <- m.chanNotification:
101                 m.handleNotification(notifMap)
102             case resultMessage := <- m.chanResult:
103                 m.handleResult(resultMessage)
104             case m.asks = <- m.chanAsk:
105             default:
106                 if m.state == RESULT && m.asks {
107                     if m.debug {
108                         log.Println("Manager : Send elected processus")
109                     }
110                     m.asks = false
111                     m.chanGiveElection <- m.elected
112                 }
113             }
114         }
115     }
116 }
117 // API for network
118
119 /**
120  * Submits a Notification message to manager from network
121  */
122 func (m *Manager) SubmitNotification(notifMap map[uint16]uint16) {
123     m.chanNotification <- notifMap
124 }
125
126 /**
127  * Submits a result message to manager from network
128  */
129 func (m *Manager) SubmitResult(id uint16, resultMap map[uint16]bool) {
130     m.chanResult <- resultMap{
131         id: id,
132         visitedResult: resultMap,
133     }
134 }
135
136 // API for Task
137
138 /**
139  * Tells manager to start an election
140  */
141 func (m *Manager) RunElection() {
142     m.chanAskElection <- true
143 }
144
145 /**
146  * Get the elected id
147  */
148 func (m *Manager) GetElected() uint16 {
149     m.chanAsk <- true
150     return <- m.chanGiveElection
151 }
152
153 // Privates
154
155 /**
156  * Runs an election
157  */
158 func (m *Manager) handleElection() {
159     l := m.createNewMap()
160     m.sendNotification(l)
161 }
162
163 /**
164  * Handles a Notification request
165  * @param notifMap map of id and aptitudes
166  */
167 func (m *Manager) handleNotification(notifMap map[uint16]uint16) {
168     if m.debug {
169         log.Println("Manager : Received NOTIFICATION ")
170     }
171
172     _, isInside := notifMap[m.me] // Test if I'm here
173     if isInside {
174         m.elected = findMax(notifMap)
175         m.sendResult()
176     } else {
177         notifMap[m.me] = m.aptitude // Add myself in map
178         m.sendNotification(notifMap)
179     }
180 }
181
182 /**
```

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

```
183 * Handles a Result request
184 * @param resultMessage
185 */
186 func (m *Manager) handleResult(resultMessage resultMessage) {
187     if m.debug {
188         log.Println("Manager : Received RESULT, new boss is ", resultMessage.id)
189     }
190
191     i := resultMessage.id
192     resultMap := resultMessage.visitedResult
193
194     _, isInside := resultMap[m.me] // Test if I'm here
195     if isInside {
196         // Nothing to do ~\_(ツ)_/~
197     } else if m.state == RESULT && m.elected != i {
198         l := m.createNewMap()
199
200         m.sendNotification(l)
201     } else if m.state == NOTIFICATION {
202         m.elected = i
203         m.sendResult()
204     }
205 }
206
207 /**
208 * Calls network and emit notification
209 * @param map of ids and aptitudes
210 */
211 func (m *Manager) sendNotification(_map map[uint16]uint16) {
212     m.network.EmitNotif(_map)
213     m.state = NOTIFICATION
214 }
215
216 /**
217 * Calls network and emit result
218 */
219 func (m *Manager) sendResult() {
220     resultMap := make(map[uint16]bool)
221     resultMap[m.me] = true
222
223     m.network.EmitResult(m.elected, resultMap)
224     m.state = RESULT
225 }
226
227 /**
228 *
229 */
230 func (m *Manager) createNewMap() map[uint16]uint16 {
231     l := make(map[uint16]uint16)
232     l[m.me] = m.aptitude
233     return l
234 }
235
236 /**
237 * Utility function to find max
238 * @param m Map where you want to find max
239 */
240 func findMax (m map[uint16]uint16) uint16 {
241     var id, max uint16 = 0, 0
242
243     for key, val := range m {
244         if val > max {
245             max = val
246             id = key
247         }
248     }
249
250     return id
251 }
```