2nd Lab - CES 27 - Distributed Programming

September 4th, 2019

Prof. Celso Massaki HIRATA, hirata@ita.br

Prof. JULIANA de Melo Bezerra, juliana@ita.br

Igor Bragaia[1], igor.bragaia@gmail.com

[1]Computer Engineering undergraduate student at Aeronautics Institute of Technology (ITA)

This project aims to implement mutual exclusion through Ricart-Agrawala algorithm using scalar logical clocks from past project. It is the second assignment of Distributed Programming 2019.2 ITA's class.

There are two main codes, one for a process resource management and other to receive messages showing how the resource has been used on this distributed system.

```go
package main

import (
  "fmt"
  "net"
  "os"
  "strconv"
  "time"
  "bufio"
  "encoding/json"
)

type Message struct {
  Time int
  Processor int
  Text string
}

var state string
var counter int
var qty int
var err string
var myPortId int
var myPort string
var nServers int
var CliConn []*net.UDPConn
var ServConn *net.UDPConn
var ResourceConn *net.UDPConn
var logicalClock int
var logicalClockFreeze int
var queue []int

func CheckError(err error) {
  if err != nil {
     fmt.Println("Error: ", err)
     os.Exit(0)
  }
}

func PrintError(err error) {
  if err != nil {
     fmt.Println("Error: ", err)
  }
}
```

```go
func doServerJob() {
  buf := make([]byte, 1024)
  n, _, err := ServConn.ReadFromUDP(buf)
  var receivedMessage Message
  err = json.Unmarshal(buf[:n], &receivedMessage)
  if err != nil {
     fmt.Println("Unmarshal server response failed.")
  }

  if logicalClock < receivedMessage.Time {
     logicalClock = receivedMessage.Time
  }
  logicalClock++

  if receivedMessage.Text == "REPLY" {
     fmt.Printf("[logical clock %d] REPLY from %d\n", logicalClock, receivedMessage.Processor)
     counter++
     if counter == qty {
        state = "HELD"
        fmt.Printf("[logical clock %d] ENTROU NA CS\n", logicalClock)
        useResource()

        state = "RELEASED"
        counter = 0
        fmt.Printf("[logical clock %d] Replying to: ", logicalClock)
        fmt.Println(queue)
        for _, p := range queue {
           doClientJob(p-1, logicalClock, "REPLY")
        }
        queue = []int{}
        fmt.Printf("[logical clock %d] SAIU DA CS\n", logicalClock)
        fmt.Println("*************************************\n")
     }
  } else if receivedMessage.Text == "REQUEST" {
     if state == "HELD" || (state == "WANTED" && logicalClockFreeze < receivedMessage.Time) {
        queue = append(queue, receivedMessage.Processor)
     } else {
        fmt.Printf("[logical clock %d] REPLYING TO %d\n", logicalClock, receivedMessage.Processor)
        doClientJob(receivedMessage.Processor-1, logicalClock, "REPLY")
     }
  }

  if err != nil {
     fmt.Println("Error: ",err)
  }
}

func useResource() {
  msg := Message{
     Time: logicalClock,
     Processor: myPortId,
     Text: "I'm into CS",
  }

  jsonRequest, err := json.Marshal(msg)
  if err != nil {
     fmt.Println("Marshal connection information failed.")
  }

  _, err = ResourceConn.Write(jsonRequest)
  if err != nil {
     fmt.Println(jsonRequest, err)
  }

  time.Sleep(5 * time.Second)
}

func doClientJob(otherProcess int, logicalClock int, text string) {
  Conn := CliConn[otherProcess]
```

```go
  msg := Message{
     Time: logicalClock,
     Processor: myPortId,
     Text: text,
  }

  jsonRequest, err := json.Marshal(msg)
  if err != nil {
     fmt.Println("Marshal connection information failed.")
  }

  _, err = Conn.Write(jsonRequest)
  if err != nil {
     fmt.Println(jsonRequest, err)
  }
}

func initConnections() {
  id, err := strconv.Atoi(os.Args[1])
  myPortId = id
  myPort = os.Args[myPortId+1]
  nServers = len(os.Args) - 2
  qty = len(os.Args) - 3

  ServerAddr, err := net.ResolveUDPAddr("udp",myPort)
  CheckError(err)
  Conn, err := net.ListenUDP("udp", ServerAddr)
  CheckError(err)

  ServConn = Conn

  for i := 0; i < nServers; i++ {
     ServerAddr,err := net.ResolveUDPAddr("udp","127.0.0.1" + os.Args[i+2])
     CheckError(err)
     LocalAddr, err := net.ResolveUDPAddr("udp", "127.0.0.1:0")
     CheckError(err)
     Conn, err := net.DialUDP("udp", LocalAddr, ServerAddr)
     CheckError(err)

     CliConn = append(CliConn, Conn)
  }

  ServerAddr,err = net.ResolveUDPAddr("udp","127.0.0.1:10001")
  CheckError(err)
  LocalAddr, err := net.ResolveUDPAddr("udp", "127.0.0.1:0")
  CheckError(err)
  Conn, err = net.DialUDP("udp", LocalAddr, ServerAddr)
  CheckError(err)

  ResourceConn = Conn
}

func readInput(ch chan string) {
  reader := bufio.NewReader(os.Stdin)
  for {
     text, _, _ := reader.ReadLine()
     ch <- string(text)
  }
}

func main() {
  initConnections()
  defer ServConn.Close()
  logicalClock = 1
  for i := 0; i < nServers; i++ {
     defer CliConn[i].Close()
  }

  state = "RELEASED"
```

```
  ch := make(chan string)
  go readInput(ch)

  for {
     go doServerJob()
     select {
     case text, valid := <-ch:
        if valid {
           if text == "x" {
              state = "WANTED"
              logicalClockFreeze = logicalClock
              for i := 0; i < nServers; i++ {
                 if i != myPortId - 1 {
                    go doClientJob(i, logicalClock, "REQUEST")
                 }
              }
           }
        } else {
           fmt.Println("Channel closed!")
        }
     default:
        time.Sleep(time.Second * 1)
     }
  }
}
```

Code 1: process.go file

```
package main

import (
  "fmt"
  "net"
  "encoding/json"
  "os"
)

type Message struct {
  Time int
  Processor int
  Text string
}

var ServConn *net.UDPConn
var port string

func CheckError(err error) {
  if err != nil {
     fmt.Println("Error: ", err)
     os.Exit(0)
  }
}

func doServerJob() {
  buf := make([]byte, 1024)
  n, _, err := ServConn.ReadFromUDP(buf)
  var receivedMessage Message
  err = json.Unmarshal(buf[:n], &receivedMessage)
  if err != nil {
     fmt.Println("Unmarshal server response failed.")
  }

  fmt.Printf("[logical clock %d] Process %d says %s\n", receivedMessage.Time, receivedMessage.Processor,
receivedMessage.Text)

  if err != nil {
     fmt.Println("Error: ",err)
  }
}

func main() {
```

```go
    Address, err := net.ResolveUDPAddr("udp", ":10001")
    CheckError(err)
    Connection, err := net.ListenUDP("udp", Address)
    ServConn = Connection
    CheckError(err)
    defer ServConn.Close()
    for {
        doServerJob()
    }
}
```

Code 2: sharedResource.go file

In order to validate the algorithm developed, there are two test cases

TEST CASE 1

We aim to simulate three different processes, then we ran three different processes in three different terminals as suggested by the professor. Initially, process 1 tries to get into the Critical Section (CS), then send request message to both process 1 and 2. When process 1 gets all replies, it goes into CS and stay there for 5 seconds. Then process 2 tries to get into the CS, send requests as well and goes in when gets all responses, as seen on the following image. Note that on this test case, Ricart-Agrawala process queues are always empty and for this reason it have not been shown at the simulation.

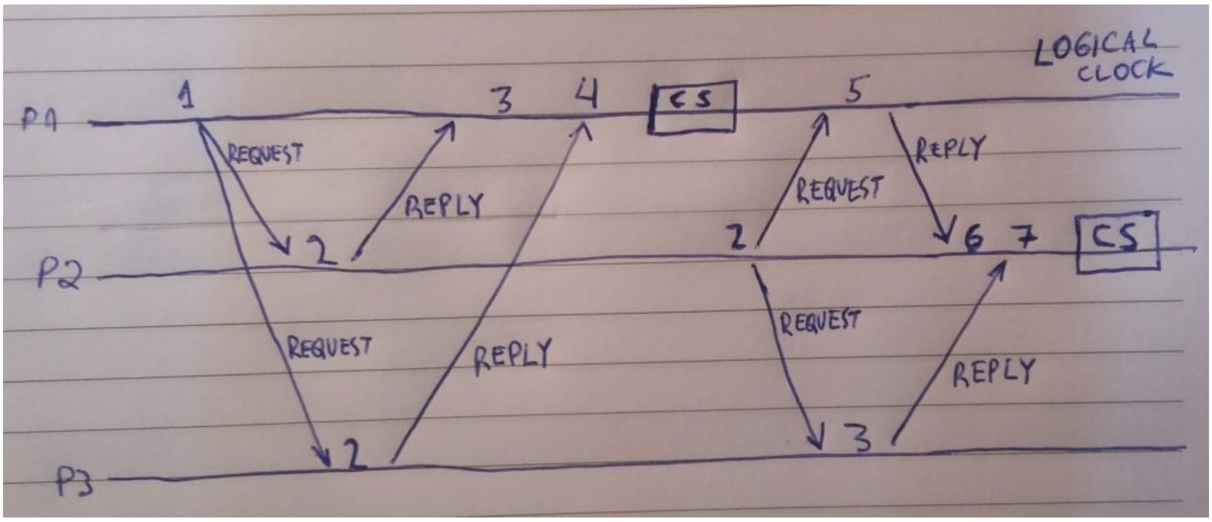


Image 1: expected output from simulation for processes 1, 2 and 3

Results are correct, as seen on the following screenshots that agree with simulation above.

STEP 1

Process 1 requests the shared resource and goes in and out of the CS

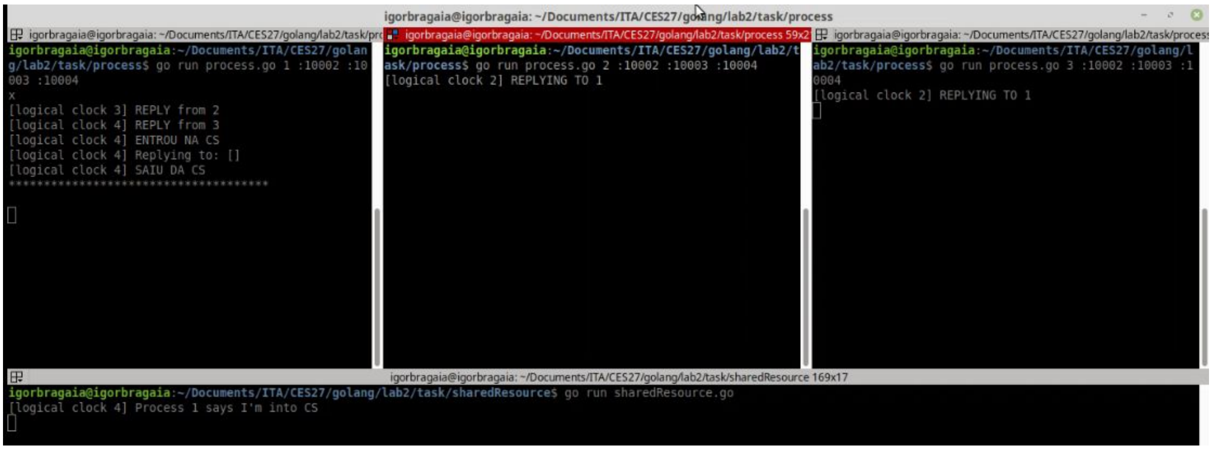


Image 2: screenshot from step 1

STEP 2

Process 2 requests the shared resource and goes in and out of the CS

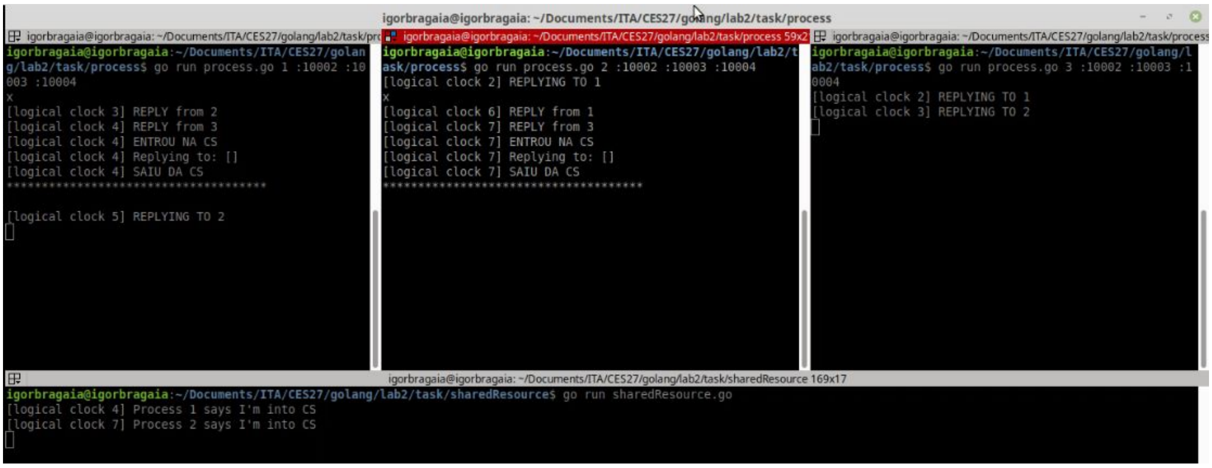


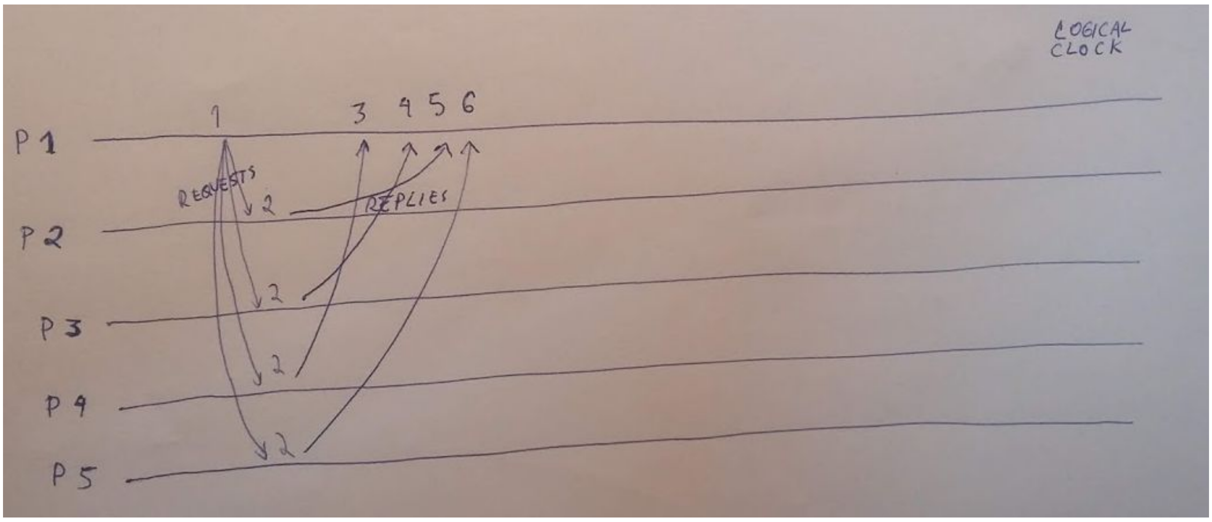
Image 3: screenshot from step 1

TEST CASE 2

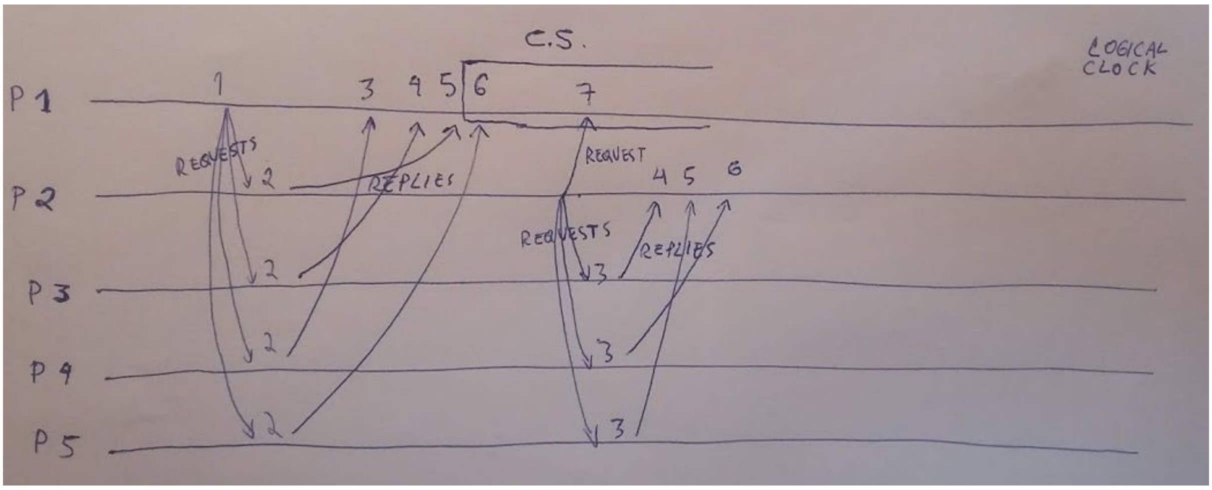
We aim to simulate five different processes, then we ran five different processes in five different terminals as suggested by the professor.

We have 6 steps:

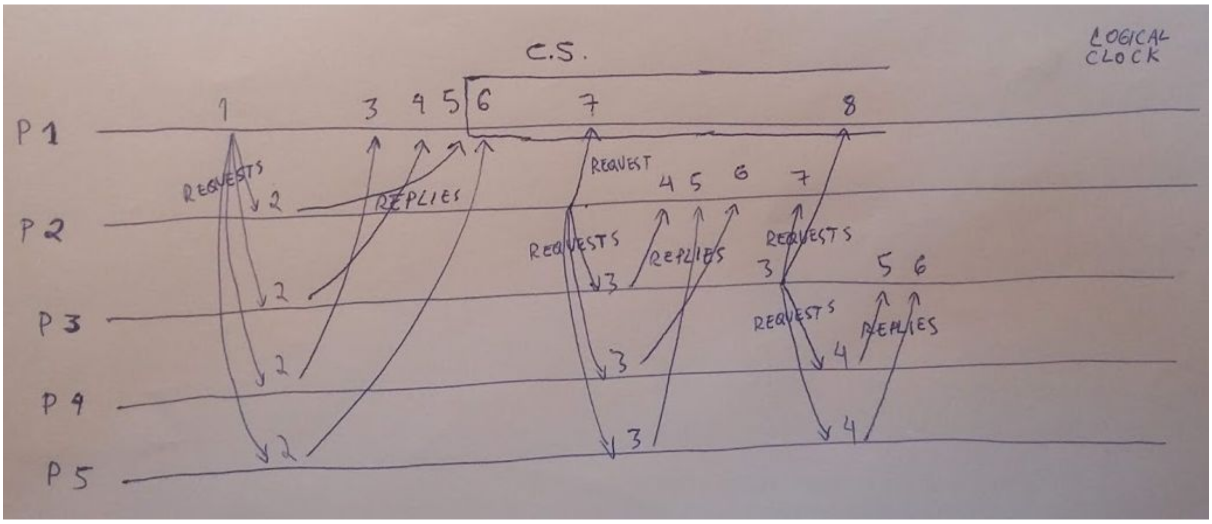
1. Process 1 requests access to CS to all other processes, gets all replies because all of them are idle and then goes to CS. See simulation at the following image.



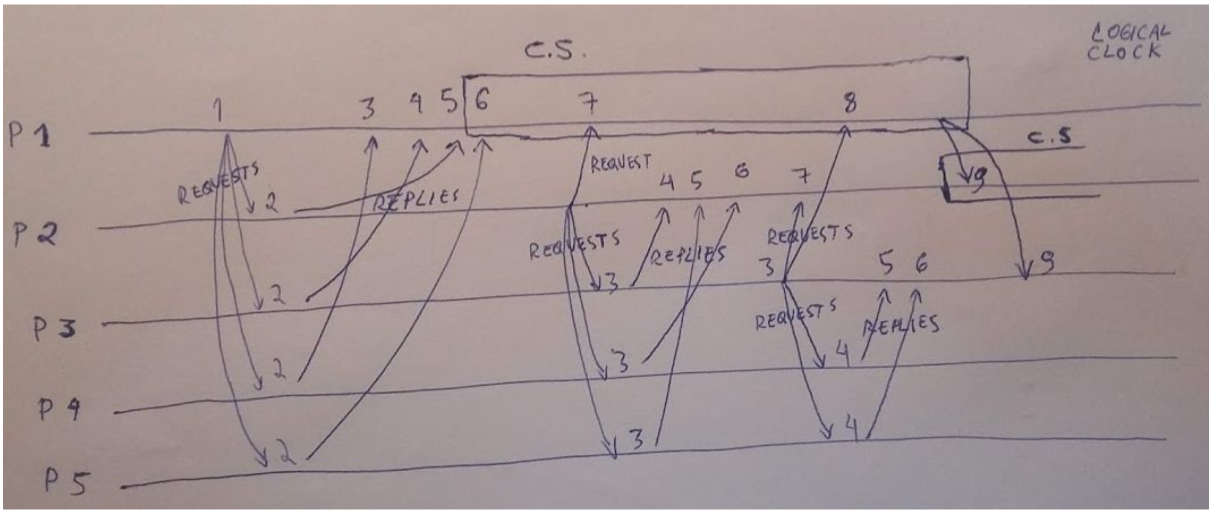
2. Before process 1 exit the CS, process 2 requests access to CS to all other processes, but does not get reply from process 1, which adds process 2 to its reply queue regarding its state HELD. See simulation at the following image.



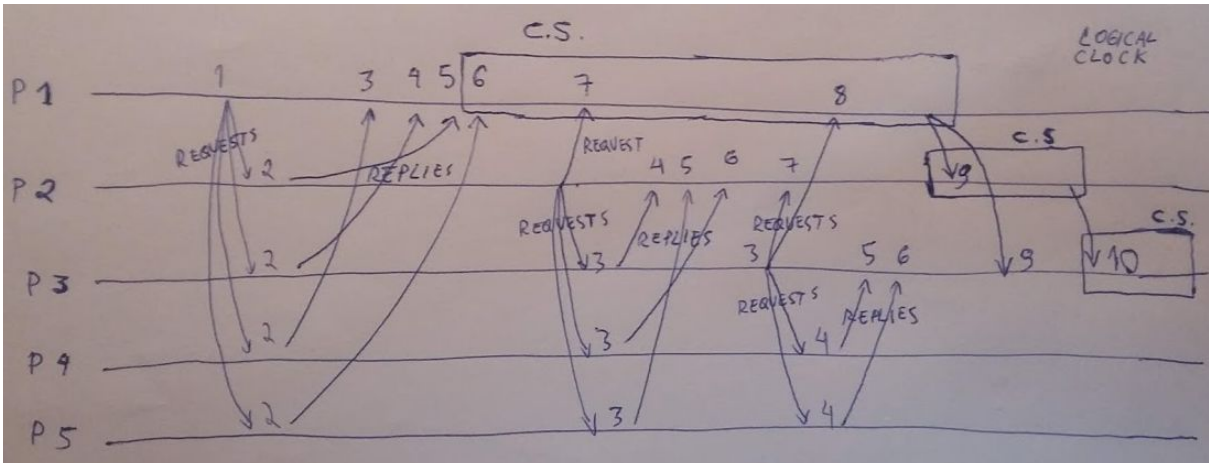
3. Before process 1 exit the CS, process 3 also requests access to CS to all other processes but does not get reply from process 1, which adds process 2 to its reply queue regarding its state HELD. Regarding state WANTED from process 3 and that logical clock for process 3 is larger than process 2, process 2 also adds process 3 to its reply queue. See simulation at the following image.



4. Process 1 exits the CS, then reply processes 2 and 3 that are at its queue. At this moment, process 2 got all necessary replies and then goes to CS. See simulation at the following image.



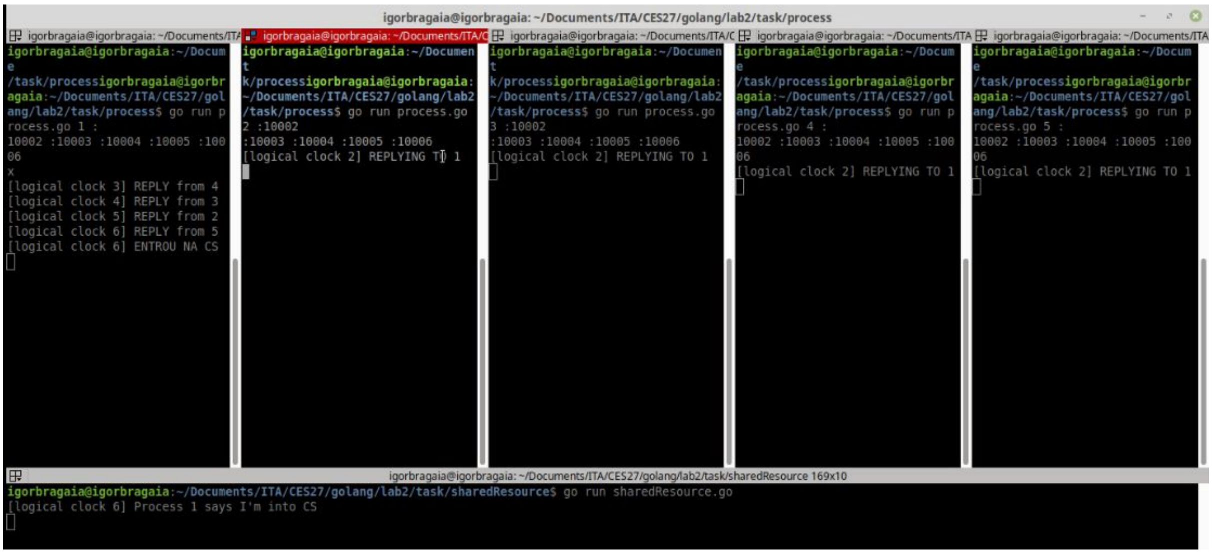
5. Process 2 exits the CS, then reply process 3 that is at its queue. At this moment, process 3 got all necessary replies and then goes to CS. See simulation at the following image.



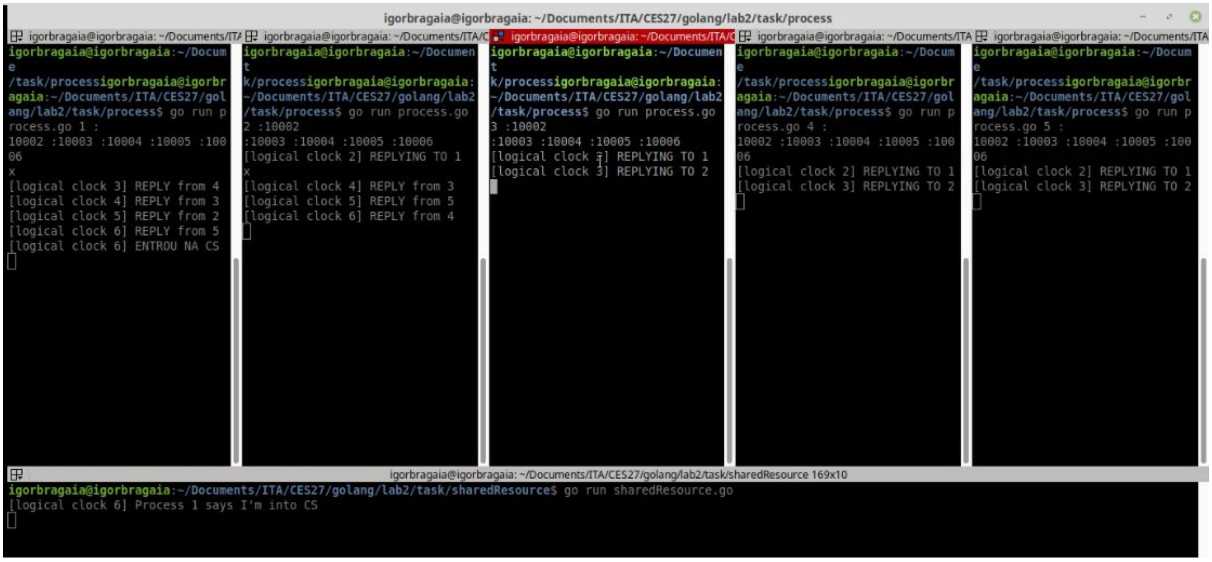
6. Process 3 exits the CS and does not reply any other process due to its empty reply queue.

Note that on this test case, Ricart-Agrawala process queues are not always empty. Finally, results are correct, as seen on the following screenshots that agree with simulation above.

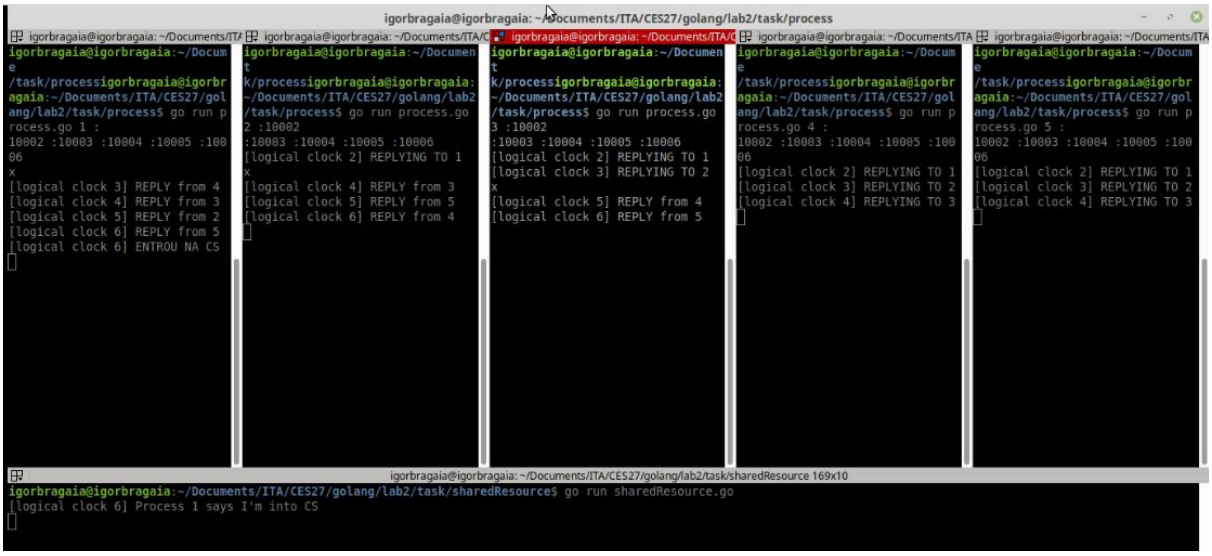
STEP 1



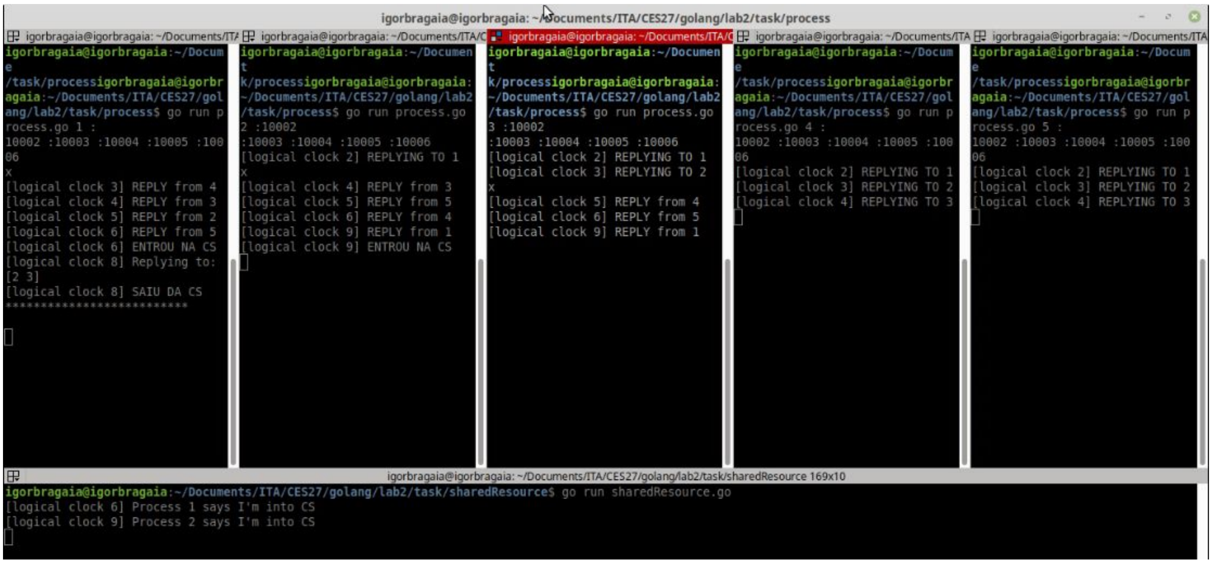
STEP 2



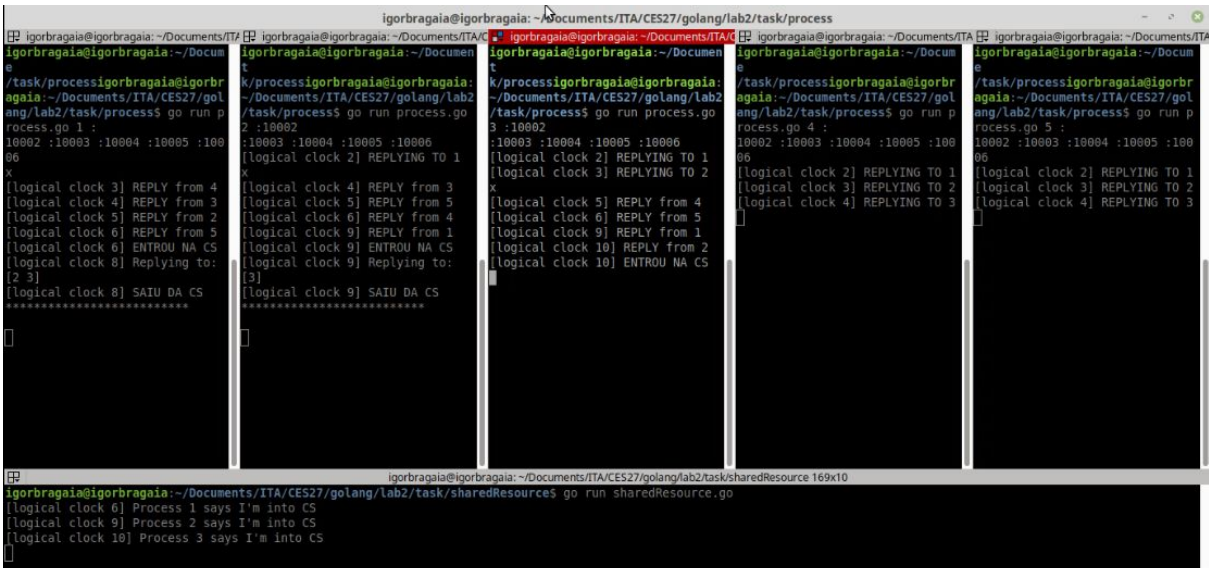
STEP 3



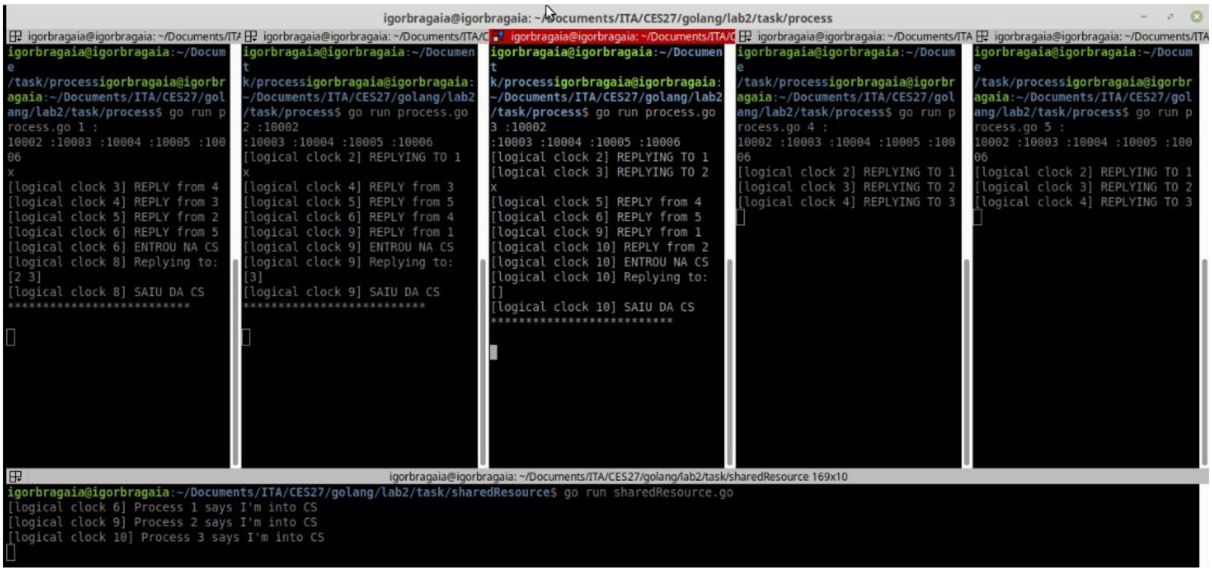
STEP 4



STEP 5



STEP 6


igorbragaia@igorbragaia: ~/Documents/ITA/CES27/golang/lab2/task/process
igorbragaia@igorbragaia:~/Documents/ITA/CES27/golang/lab2/task/process$ go run process.go 1 :10002 :10003 :10004 :10005 :10006
x
[logical clock 3] REPLY from 4
[logical clock 4] REPLY from 3
[logical clock 5] REPLY from 2
[logical clock 6] REPLY from 5
[logical clock 6] ENTROU NA CS
[logical clock 8] Replying to: [2 3]
[logical clock 8] SAIU DA CS
****
igorbragaia@igorbragaia:~/Documents/ITA/CES27/golang/lab2/task/process$ go run process.go 2 :10002 :10003 :10004 :10005 :10006
[logical clock 2] REPLYING TO 1
x
[logical clock 4] REPLY from 3
[logical clock 5] REPLY from 5
[logical clock 6] REPLY from 4
[logical clock 9] REPLY from 1
[logical clock 9] ENTROU NA CS
[logical clock 9] Replying to: [3]
[logical clock 9] SAIU DA CS
****
igorbragaia@igorbragaia:~/Documents/ITA/CES27/golang/lab2/task/process$ go run process.go 3 :10002 :10003 :10004 :10005 :10006
[logical clock 2] REPLYING TO 1
[logical clock 3] REPLYING TO 2
x
[logical clock 5] REPLY from 4
[logical clock 6] REPLY from 5
[logical clock 9] REPLY from 1
[logical clock 10] REPLY from 2
[logical clock 10] ENTROU NA CS
[logical clock 10] Replying to: []
[logical clock 10] SAIU DA CS
****
igorbragaia@igorbragaia:~/Documents/ITA/CES27/golang/lab2/task/process$ go run process.go 4 :10002 :10003 :10004 :10005 :10006
[logical clock 2] REPLYING TO 1
[logical clock 3] REPLYING TO 2
[logical clock 4] REPLYING TO 3
igorbragaia@igorbragaia:~/Documents/ITA/CES27/golang/lab2/task/process$ go run process.go 5 :10002 :10003 :10004 :10005 :10006
[logical clock 2] REPLYING TO 1
[logical clock 3] REPLYING TO 2
[logical clock 4] REPLYING TO 3
igorbragaia@igorbragaia: ~/Documents/ITA/CES27/golang/lab2/task/sharedResource 169x10
igorbragaia@igorbragaia:~/Documents/ITA/CES27/golang/lab2/task/sharedResource$ go run sharedResource.go
[logical clock 6] Process 1 says I'm into CS
[logical clock 9] Process 2 says I'm into CS
[logical clock 10] Process 3 says I'm into CS