CES-27

4ª ATIVIDADE

Professores: Juliana de Melo Bezerra
Vitor Curtis
Aluno: Dennys Leandro Agostini Rocha

Objetivo
Trabalhar com algoritmo de exclusão mútua para sistemas distribuidos.

Algoritmo:

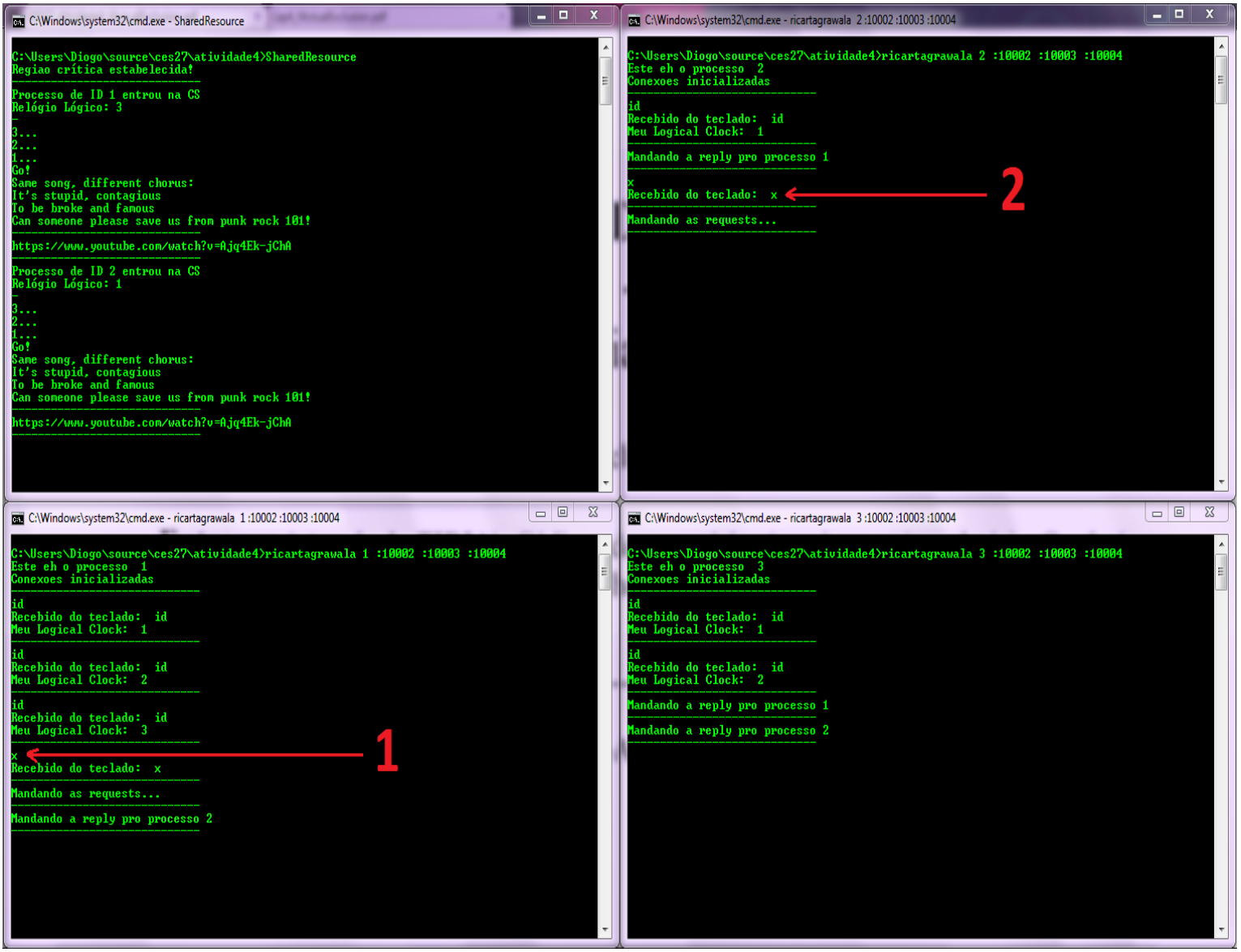
```
package main

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

//Variáveis globais interessantes para o processo
var err string
var myPort string //porta do meu servidor
var myProcess int //numero do meu processo
var nServers int //qtde de outros processo
var state string
var nReplies int // numero de replicas apos solicitacao da CS
var CliConn []*net.UDPConn //vetor com conexões para os servidores dos outros processos
var ServConn *net.UDPConn //conexão do meu servidor (onde recebo mensagens dos outros processos)

//Estruturas para o processo
type Message struct { //crio a estrutura para o vector timestamp (aqui chamado de VM)
        Id int // meu P_i
        Text string
        LogicalClock int // meu T
}

var DataReceived Message
var DataSent Message
var Data Message
var queue []Message

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

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

func readInput(ch chan string) {
        // Non-blocking async routine to listen for terminal input
        reader := bufio.NewReader(os.Stdin)
        for {
                text, _, _ := reader.ReadLine()
                ch <- string(text)
        }
}

func initConnections() {
        myProcess, _ = strconv.Atoi(os.Args[1])
        fmt.Println("Este eh o processo ", myProcess)
        myPort = os.Args[myProcess+1]
        nServers = len(os.Args) - 3
        /*Esse 3 tira o nome (no caso Process), o numero do (meu) processo e tira a porta que é
        minha. As demais portas são dos outros processos*/

        state = "RELEASED"
        nReplies = 0
        Data.Id = myProcess
        Data.Text = "DEBOAS"
        Data.LogicalClock = 0
        DataSent.Id = myProcess
        DataSent.Text = "REPLY"

        //Outros códigos para deixar ok a conexão do meu servidor
        ServAddr,err := net.ResolveUDPAddr("udp","127.0.0.1"+myPort)
        CheckError(err)
        ServConn, err = net.ListenUDP("udp", ServAddr)
        CheckError(err)
        CliConn = make([]*net.UDPConn, nServers+1)

        //Outros códigos para deixar ok as conexões com os servidores dos outros processos
        j:=0 //esse j eh apenas para "pular" o i correspondente ao meu servidor
        for i:=0; i<nServers+1; i++ {
                if i!=myProcess-1 {
                        ServAddr,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)
                        CliConn[j], err = net.DialUDP("udp", LocalAddr, ServAddr)
                        CheckError(err)
                        j++
                }
        }
        // Se conectar ao SharedResource
        ServAddr,err = net.ResolveUDPAddr("udp","127.0.0.1"+":10001")
        CheckError(err)
        LocalAddr, err := net.ResolveUDPAddr("udp","127.0.0.1:0")
        CheckError(err)
        CliConn[nServers], err = net.DialUDP("udp", LocalAddr, ServAddr)
        fmt.Println("Conexoes inicializadas")
        fmt.Println("------------------------------")
}

func sendReply() {
        fmt.Print("Mandando a reply ")
        DataSent.LogicalClock=Data.LogicalClock
        fmt.Print("pro processo ", DataReceived.Id, "\n")
        jsonRequest, err := json.Marshal(DataSent) //reescrevo os dados por meio do json
        CheckError(err)
        x:=DataReceived.Id
        if DataReceived.Id>myProcess {
                x=DataReceived.Id-1
        }
        _, err = CliConn[x-1].Write(jsonRequest) //envio os dados reescritos pelo canal
        PrintError(err)
        fmt.Println("------------------------------")
}

func doServerJob() {
        //Ler (uma vez somente) da conexão UDP a mensagem
        buf := make([]byte, 1024)
        n, ,err := ServConn.ReadFromUDP(buf)
        CheckError(err)

        err = json.Unmarshal(buf[:n], &DataReceived) //interpreto por meio do json e passo pra
estrutura de dados
        PrintError(err)
        time.Sleep(2*time.Second)
        if DataReceived.Text == "REQUEST" {
                if state=="HELD" || (state=="WANTED" && Data.LogicalClock <=
DataReceived.LogicalClock) {
                        if Data.LogicalClock == DataReceived.LogicalClock &&
Data.Id>DataReceived.Id {
                                sendReply()
                        } else {
                                queue = append(queue, DataReceived) // coloca a mensagem na fila
                        }
                } else {
                        sendReply()
                }
        } else if DataReceived.Text == "REPLY" {
                nReplies++
        }

}

func doClientJob() { // entrar na secao critica
        state = "WANTED"
        // mandando as requests
        fmt.Println("Mandando as requests...")
        Data.Text = "REQUEST"
        jsonRequest, err := json.Marshal(Data) //reescrevo os dados por meio do json
        CheckError(err)

        time.Sleep(time.Second)
        for i:=0; i<nServers; i++ {
                _, err = CliConn[i].Write(jsonRequest) //envio os dados reescritos pelo canal
                PrintError(err)
        }
        fmt.Println("------------------------------")

        for {
                if nReplies == nServers {
                        state = "HELD"
                        break
                }
        }
        Data.Text = "https://www.youtube.com/watch?v=Ajq4Ek-jChA\n"
        jsonRequest, err = json.Marshal(Data) //reescrevo os dados por meio do json
        CheckError(err)
         , err = CliConn[nServers].Write(jsonRequest) //envio os dados reescritos pelo canal
        PrintError(err)
        time.Sleep(16*time.Second)
        // Aqui ja esta fora da secao critica
        state = "RELEASED"
```

```
        Data.Text = "DEBOAS"
        nReplies = 0
        for len(queue)>0 {
                DataReceived.Id = queue[0].Id
                queue = queue[1:]
                sendReply()
        }
}

func main() {
        initConnections()
        //O fechamento de conexões devem ficar aqui, assim só fecha conexão quando a main morrer
        defer ServConn.Close()
        for i := 0; i < nServers+1; i++ { // o +1 eh pro SharedResource
                defer CliConn[i].Close()
        }
        //Todo Process fará a mesma coisa: ouvir msg e mandar infinitos i's para os outros processos
        ch := make(chan string)
        for {
                go readInput(ch)
                //Server
                go doServerJob()
                // When there is a request (from stdin). Do it!
                select {
                        case msgTerminal, valid := <-ch:
                                if valid && msgTerminal=="x" && state!="HELD" && state!="WANTED" {
// ñ pode estar/esperar CS
                                        fmt.Println("Recebido do teclado: ", msgTerminal)
                                        fmt.Println("------------------------------")
                                        go doClientJob()
                                } else if valid && msgTerminal=="id" {
                                        fmt.Println("Recebido do teclado: ", msgTerminal)
                                        Data.LogicalClock++
                                        fmt.Println("Meu Logical Clock: ", Data.LogicalClock)
                                        fmt.Println("------------------------------")
                                } else {
                                        fmt.Println("Channel closed!")
                                }
                        default:
                                // Do nothing in the non-blocking approach.
                                time.Sleep(time.Second * 1)
                }
                // Wait a while
                time.Sleep(time.Second * 1)
        }
}
```

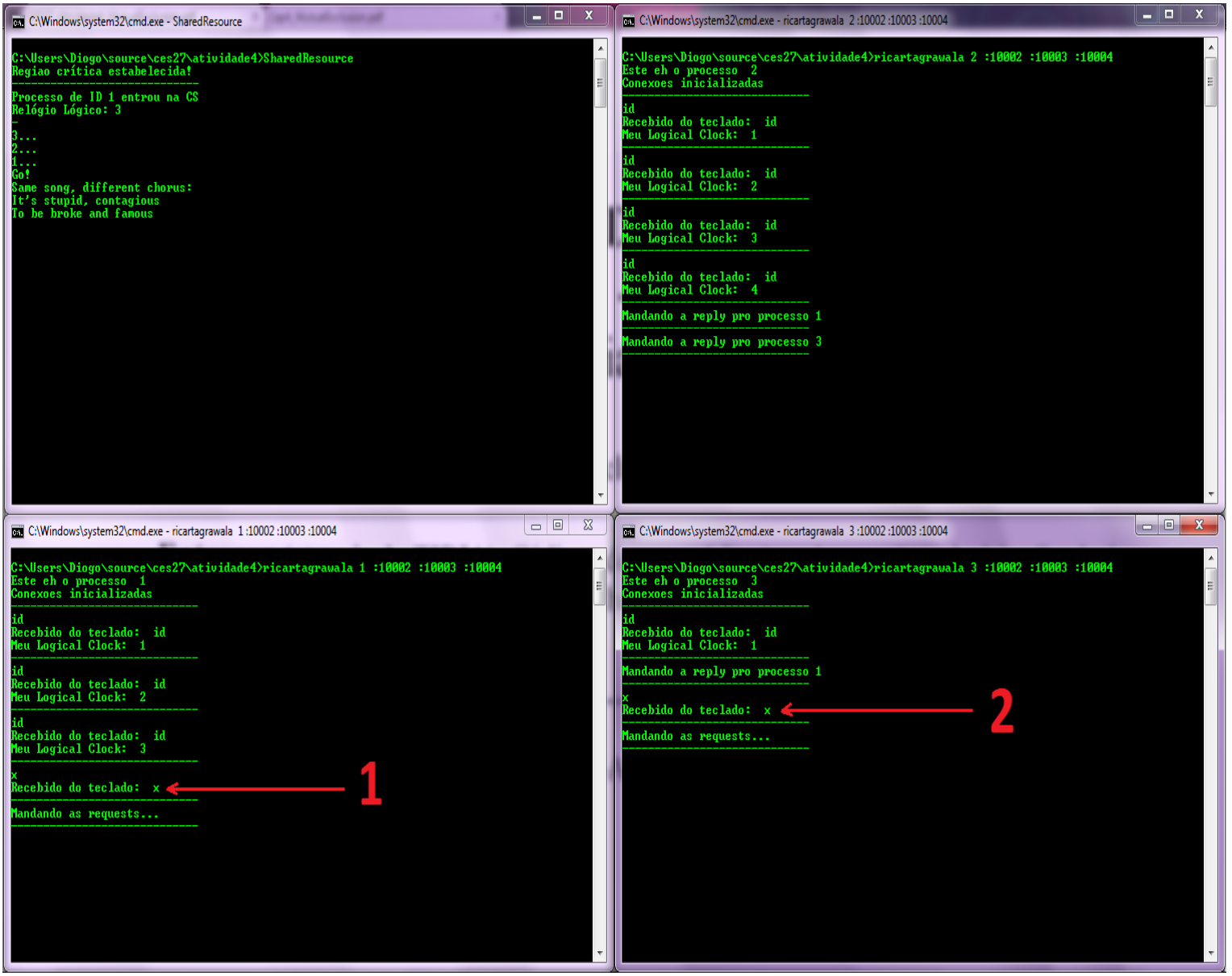
Resultados:

1. Caso em que não existem filas: incrementei os logical clocks dos três processos e fiz os processos 1 e 2 entrarem na CS, sem filas.

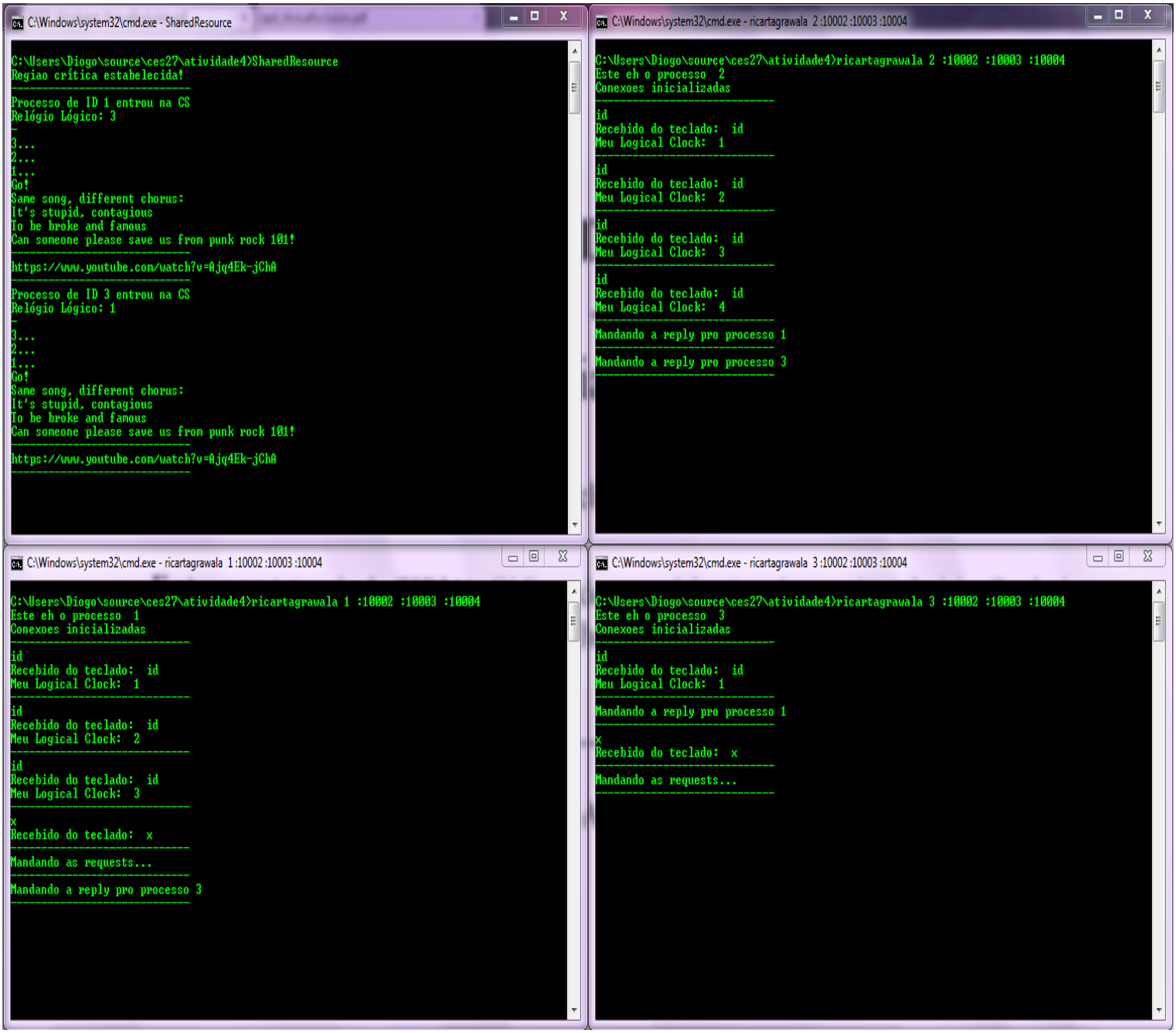


2. Caso em que existem filas: incrementei os logical clocks dos três processos e fiz o processos 1 entrar na CS. Enquanto ele estava na CS, fiz o processo 3 solicitar a entrada.

Note que, na primeira Figura, enquanto o processo 1 está na CS ele não envia replica para o processo 3, que solicitou entrada na CS enquanto o processo 1 a ocupa.



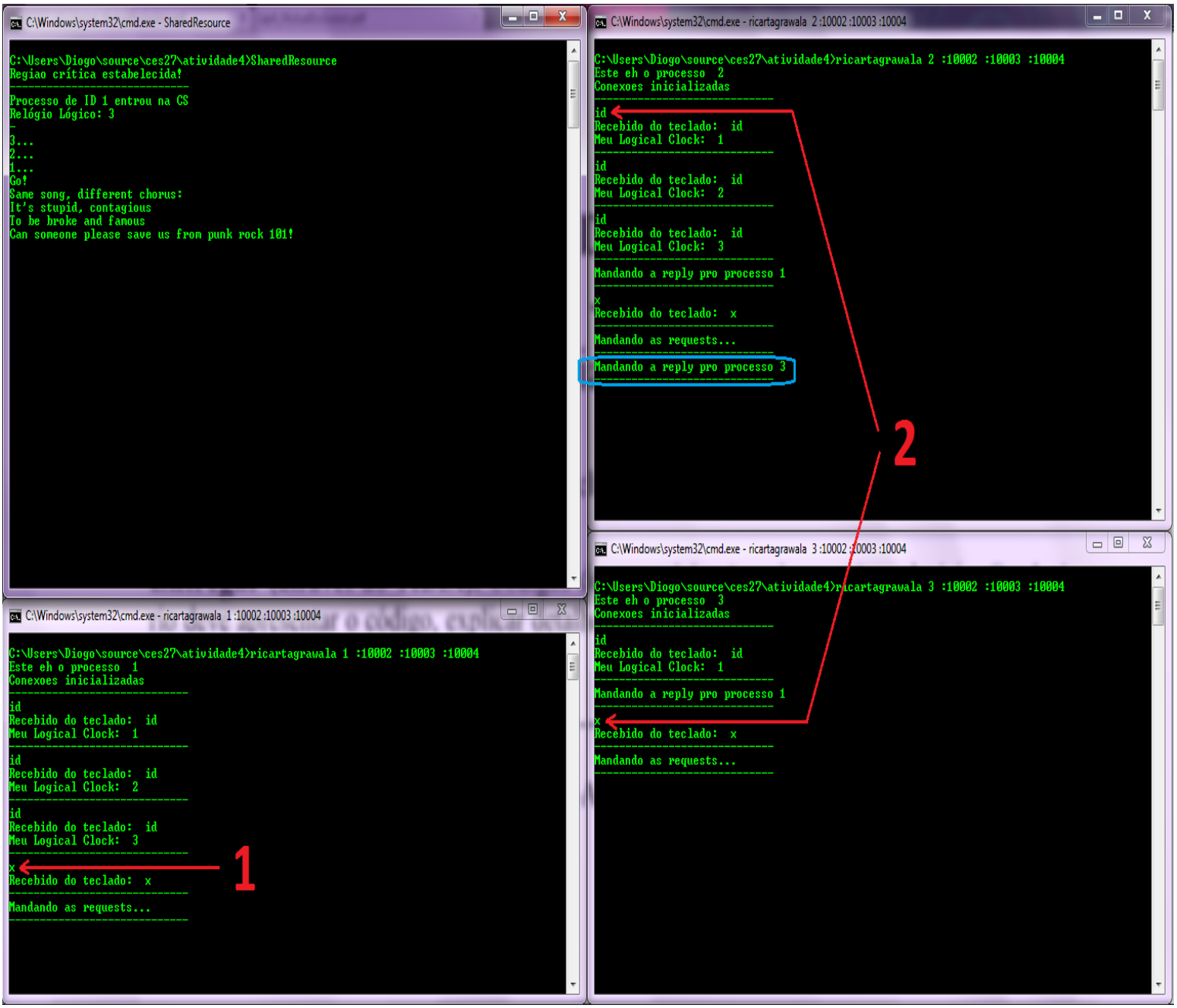
Note agora pela segunda Figura que após o processo 1 deixar a CS ele envia a replica para o processo 3, que então estará livre para ocupar a região.



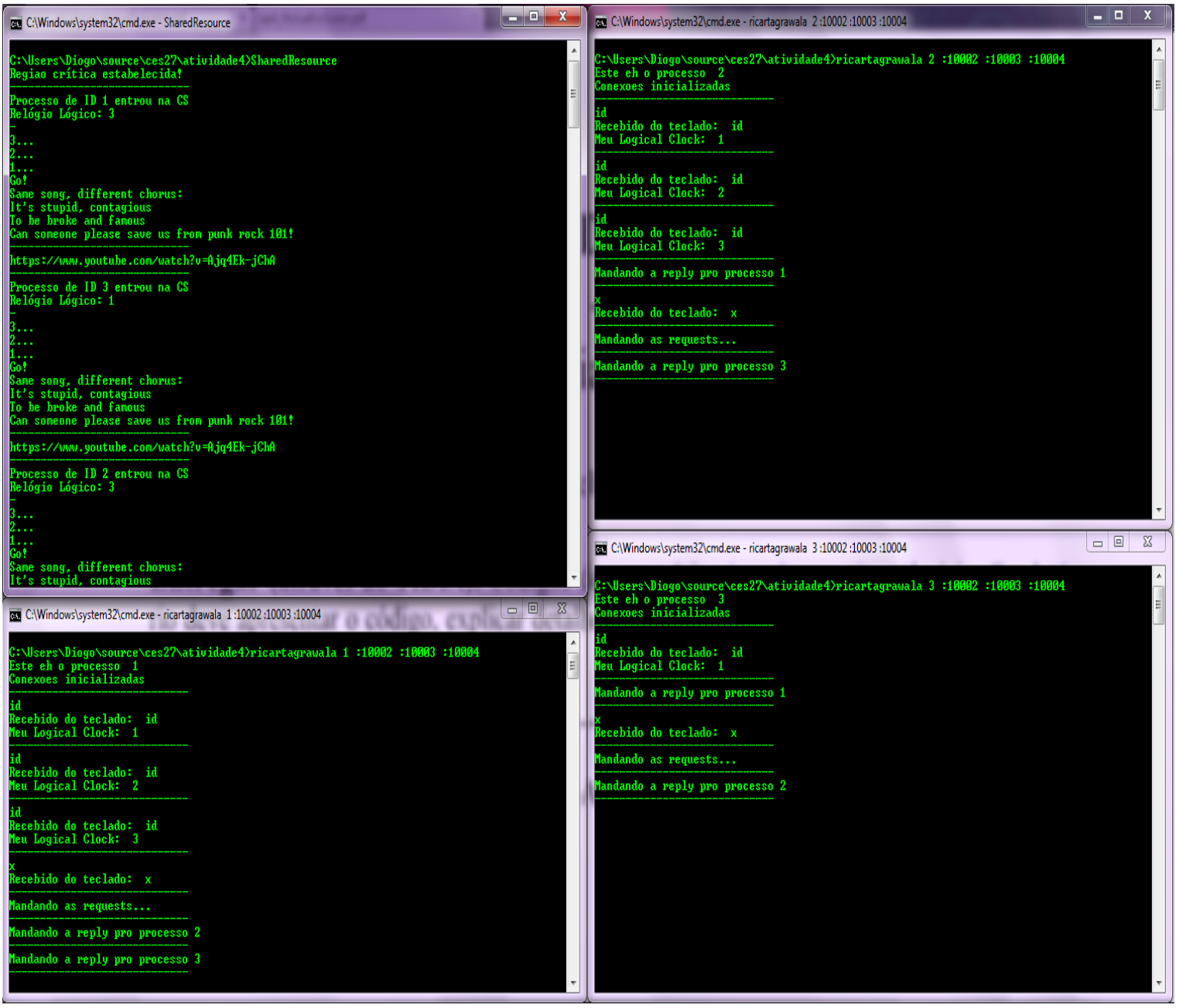
3. Caso em que existem filas e a prioridade é dada pelo valor do logical clock: incrementei os logical clocks dos três processos, os dois ultimos com valores diferentes, e fiz o processo 1 entrar na CS. Enquanto ele estava na CS, fiz os processos 2 e 3 solicitarem a entrada.

Note que, devido ao processo 3 ter logical clock menor que o processo 2, este entrará na CS logo após o processo 1 a deixar.

Na primeira Figura, fica evidente o fato do processo 2 enviar sua replica ao processo 3 por ter maior logical clock. Os processos 1 e 3 só enviarão suas replicas ao processo 2 quando saírem da região crítica, assim como o processo 3 só receberá a réplica do processo 1 quando este deixar a região.

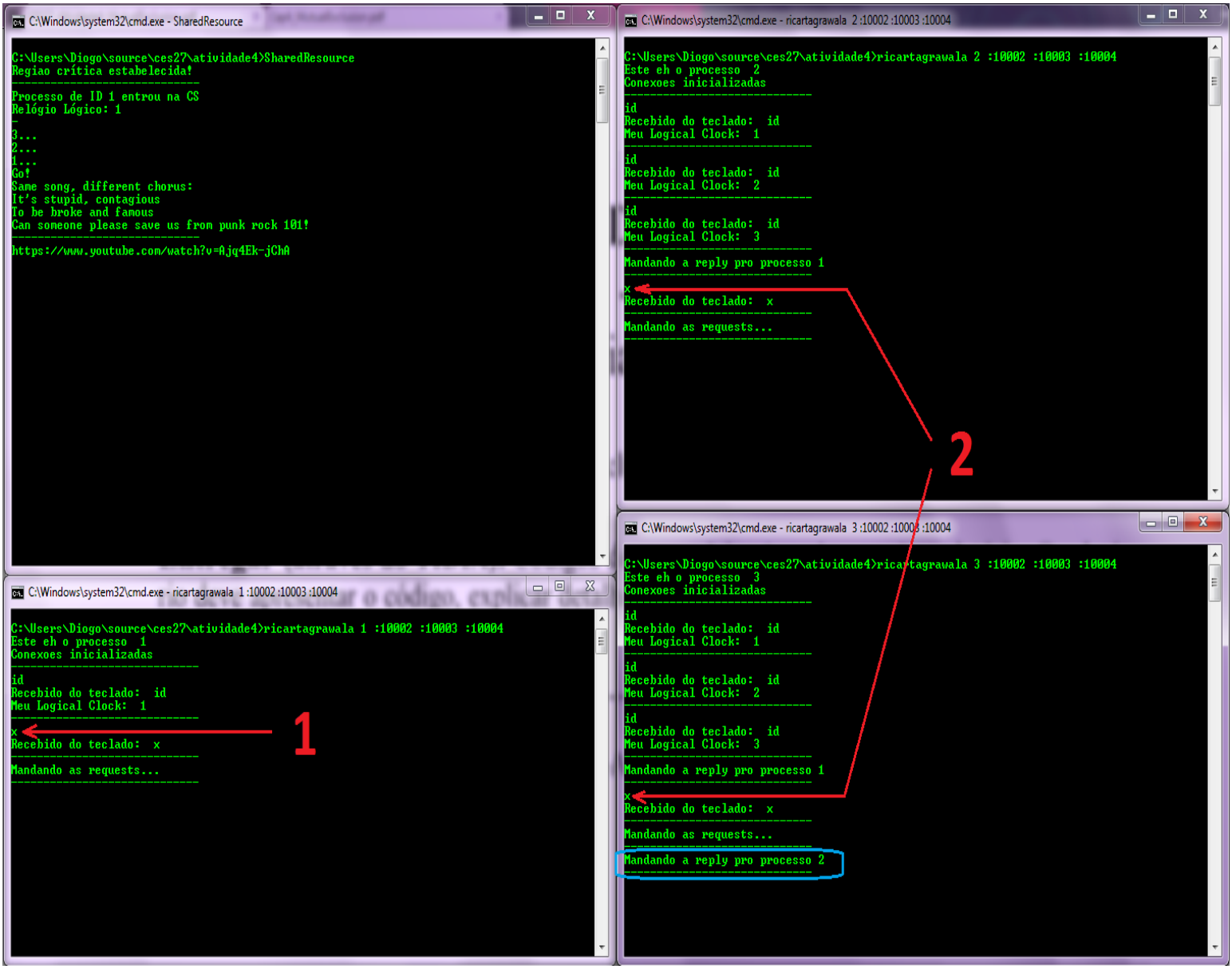


Na segunda Figura é possível ver que, logo após o processo 3 deixar a CS, este envia sua replica para o processo 2, que finalmente acessa a CS.

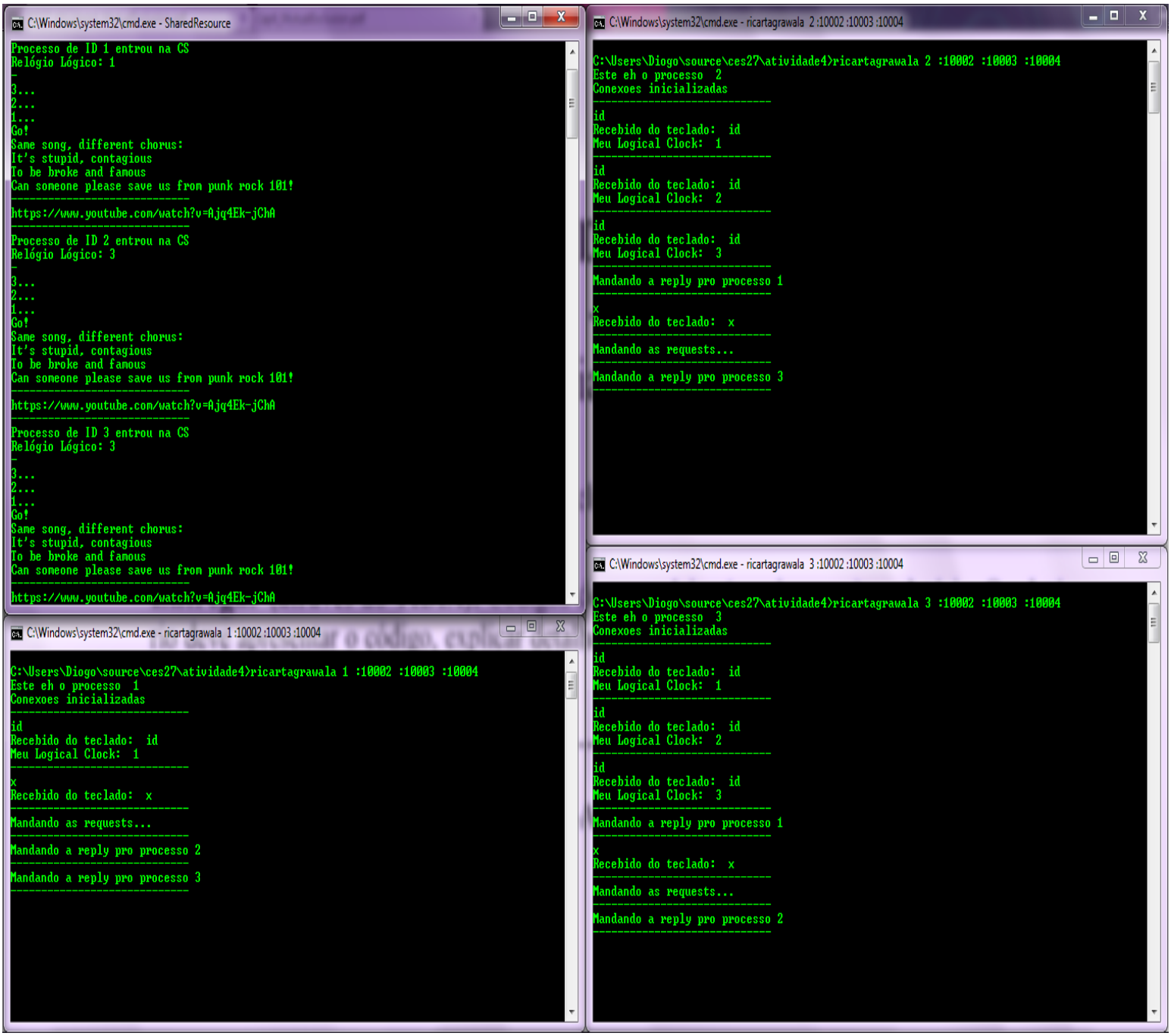


4. Caso em que existem filas e a prioridade é dada pelo id do processo, já que os logical clocks são iguais: incrementei os logical clocks dos três processos, os dois ultimos com valores iguais, e fiz o processo 1 entrar na CS primeiro.

Pela primeira Figura é possível notar que o processo 3, por ter mesmo logical clock que o processo 2 porém id menor, envia uma replica para ele, de tal forma que o processo 2 entrará primeiro na CS após o processo 1 finalizar.



Pela segunda figura, é possível ver que quando o processo 2 sai da CS, este envia uma replica ao processo 3 que estava aguardando.



5. Caso em que todos tem mesmo logical clock e disputam a CS.
Como vê-se na Figura, a ordem foi estabelecida corretamente (segundo o id): primeiro o processo 1 entrou na CS, seguido do processo 2 e por ultimo o processo 3.