CES-27

5ª ATIVIDADE

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

Objetivo
Implementar o algoritmo de detecção de *deadlock* distribuído de Misra-Chandy-Haas para modelo OR.

Algoritmo:

```
package main

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

//Variáveis globais interessantes para o processo
var err string
var myPort string //porta do meu servidor
var id int //numero do meu processo
var nServers int //qtde de outros processo
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)
var dependences []int
var num []int
var wait []bool
var latest []int
var engager []int

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() {
        id,    = strconv.Atoi(os.Args[1])
        fmt.Println("-------------------------------------------")
        fmt.Println("Conventions: S = sending, R = receiving\n\tExample: S1 means \"sending to 1\"")
        fmt.Println("\t\t R3 means \"receiving from 1\"")
        fmt.Println("-------------------------------------------")
        fmt.Println("My id:", id)

        firstPort := 2 // essa variavel encontra a primeira porta no array de argumentos
        for { // loop pra encontrar o firstPort
                if strings.HasPrefix(os.Args[firstPort], ":") {
                        break
                } else {
                        p, _ := strconv.Atoi(os.Args[firstPort])
                        dependences = append(dependences, p)
                        firstPort++
                }
        }

        myPort = os.Args[firstPort+id-1]
        nServers = len(os.Args) - firstPort - 1
        // o firstPort tira o nome do prog, o num do meu proc e as relacoes; o 1 tira meu processo

        //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)

        //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:=firstPort; i<firstPort+nServers+1; i++ {
                if i!=firstPort+id-1 {
                        ServAddr,err = net.ResolveUDPAddr("udp","127.0.0.1"+os.Args[i])
                        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++
                        // definir os parametros latest, engager, wait e num
                        num = append(num, 0)
                        wait = append(wait, false)
                        latest = append(latest, 0)
                        engager = append(engager, -1)
                }
        }
        fmt.Println("Connections initialized")
        fmt.Println("-----------------------------------------")
}

func doServerJob() {
        //Ler (uma vez somente) da conexão UDP a mensagem
        buf := make([]byte, 1024)
        n,_,err := ServConn.ReadFromUDP(buf)
        CheckError(err)
        if string(buf[0:n]) == "D" { // caso tenha ocorrido um deadlock
                os.Exit(0)
        }
        data := string(buf[0:n])
        msgType := string(data[0])
        str := "[" + data[2:] + "]"
        var ints []int
    json.Unmarshal([]byte(str),&ints) // converter do formato byte pra ints

    // inicializacao das variáveis
        i := ints[0]
        m :=  ints[1]
        j := ints[2]
        k :=  ints[3]

        fmt.Print("R", j, "\t--->\t", data, "\n")

        if msgType == "Q" { // mensagem do tipo Query
                if m > latest[i] {
                        latest[i] = m
                        engager[i] = j
                        wait[i] = true
                        num[i] = len(dependences)
                        for _,r := range dependences {
                                sendQuery(i,m,k,r)
                        }
                } else if wait[i] && m==latest[i] {
                        sendReply(i,m,k,j)
                }
        } else if msgType == "R" { // mensagem do tipo Reply
                if wait[i] && m==latest[i] {
                        num[i]--
                        if num[i]==0 {
                                if i==k {
                                        fmt.Println("\nDEADLOCK") // deadlock detectado
                                        fmt.Println("\nClosing channels...")
                                        for u:=1; u<nServers+2; u++ { // fechar todos os processos
                                                if u!=id {
                                                        doClientJob(u, "D")
                                                }
                                        }
                                        os.Exit(0)
                                } else {
                                        sendReply(i,m,k,engager[i])
                                }
                        }
                }
        }
        time.Sleep(time.Second * 1)
}

func doClientJob(otherProcess int, data string) { // entrar na secao critica
        buf := []byte(data)
        if otherProcess > id { // o vetor CliConn "pula" o processo id, por isso o --
                otherProcess--
        }
        _, err := CliConn[otherProcess-1].Write(buf) //envio os dados pelo canal
        PrintError(err)
        time.Sleep(time.Second * 1)
}

func sendQuery(i int, m int, j int, k int) { // enviar uma Query
        data := "Q," + strconv.Itoa(i) + "," + strconv.Itoa(m) + "," + strconv.Itoa(j) + "," +
strconv.Itoa(k)
        fmt.Print("S", k, "\t--->\t", data, "\n")
        go doClientJob(k, data)
}

func sendReply(i int, m int, k int, j int) { // enviar um Reply
```

```
        data := "R," + strconv.Itoa(i) + "," + strconv.Itoa(m) + "," + strconv.Itoa(k) + "," +
strconv.Itoa(j)
        fmt.Print("S", j, "\t--->\t", data, "\n")
        go doClientJob(j, data)
}

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; i++ {
                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 x, valid := <-ch:
                                if valid && x == "start" {
                                        latest[id]++
                                        m := latest[id]
                                        wait[id] = true
                                        num[id] = len(dependences)
                                        for _,k := range dependences {
                                                sendQuery(id,m,id,k)
                                        }
                                } 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. No caso em que o grafo é como o da Figura 1, os resultados estão na Figura 2.

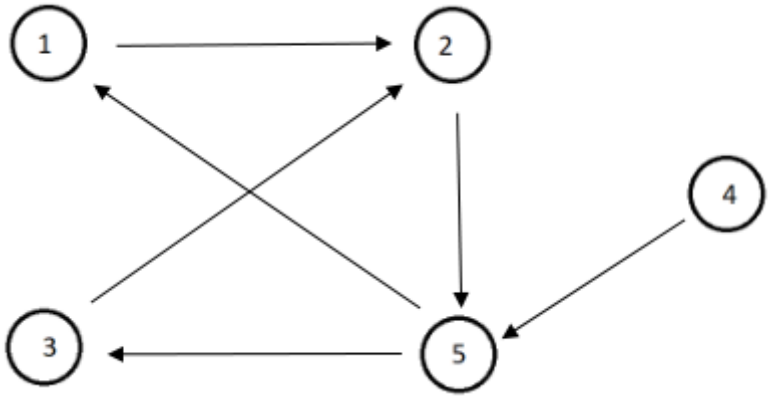

Figura 1: Grafo de dependências entre processos WFG para o primeiro teste.

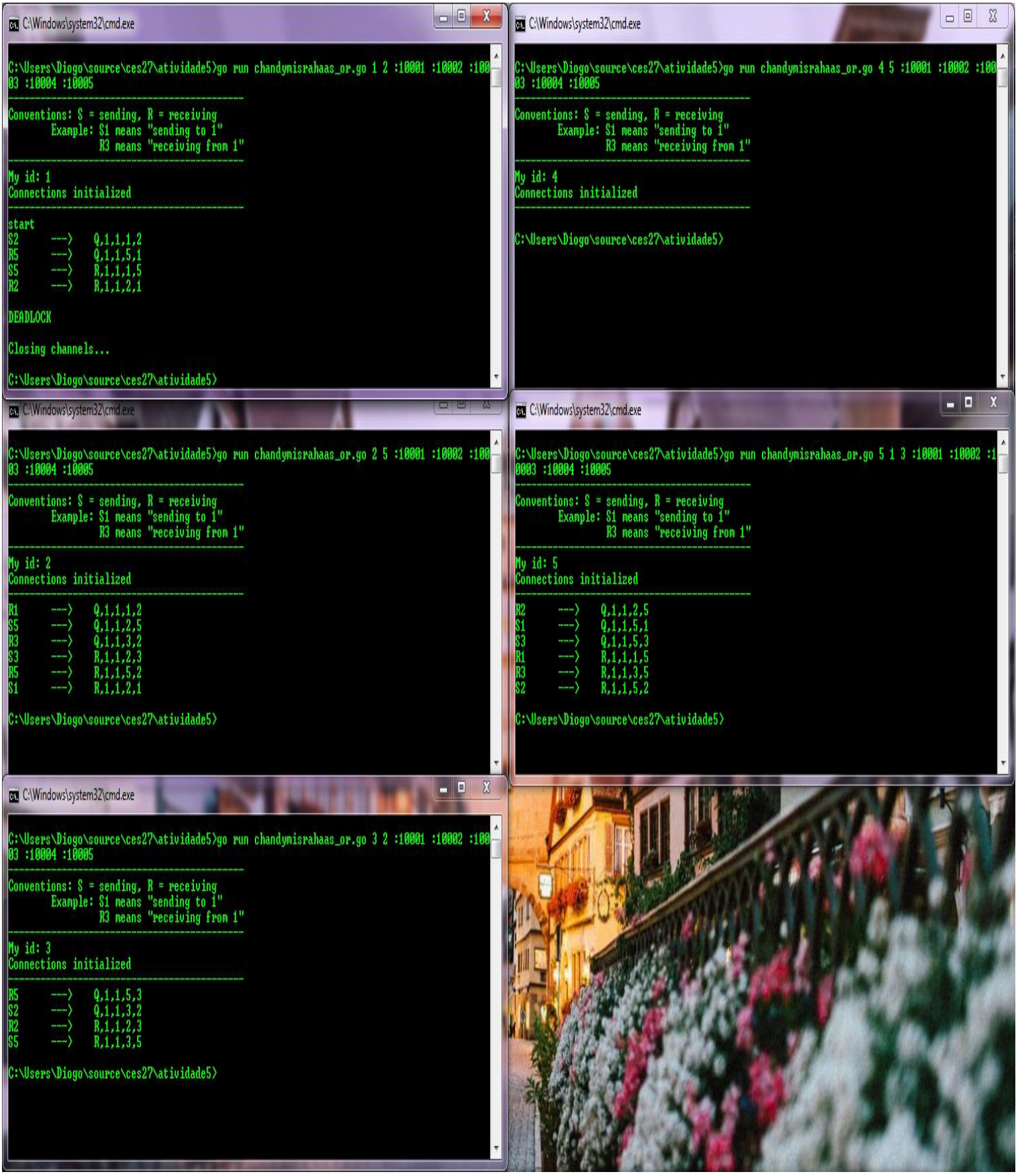

Figura 2: Resultado para o grafo da Figura 1.

O resultado concorda com o esperado.

2. No caso em que o grafo é como o da Figura 3, os resultados estão na Figura 4.

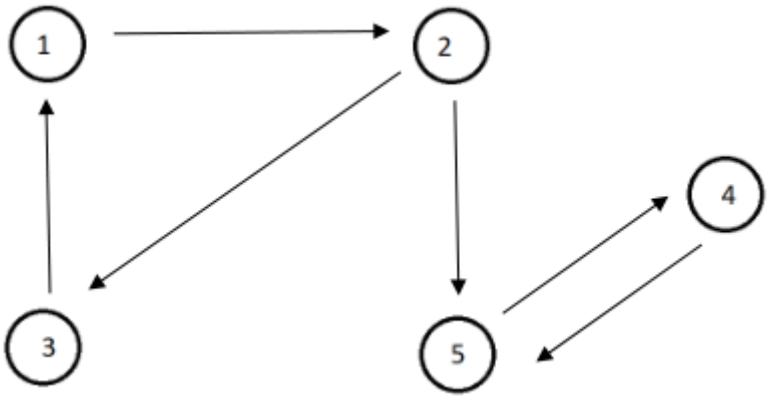

Figura 3: Grafo de dependências entre processos WFG para o segundo teste.

Figura 4: Resultado para o grafo da Figura 3.

O resultado concorda com o esperado.