Esercitazione 4
Go

20 Novembre 2017

Go

Realizzazione di politiche
basate su priorità

Esempio: il ponte

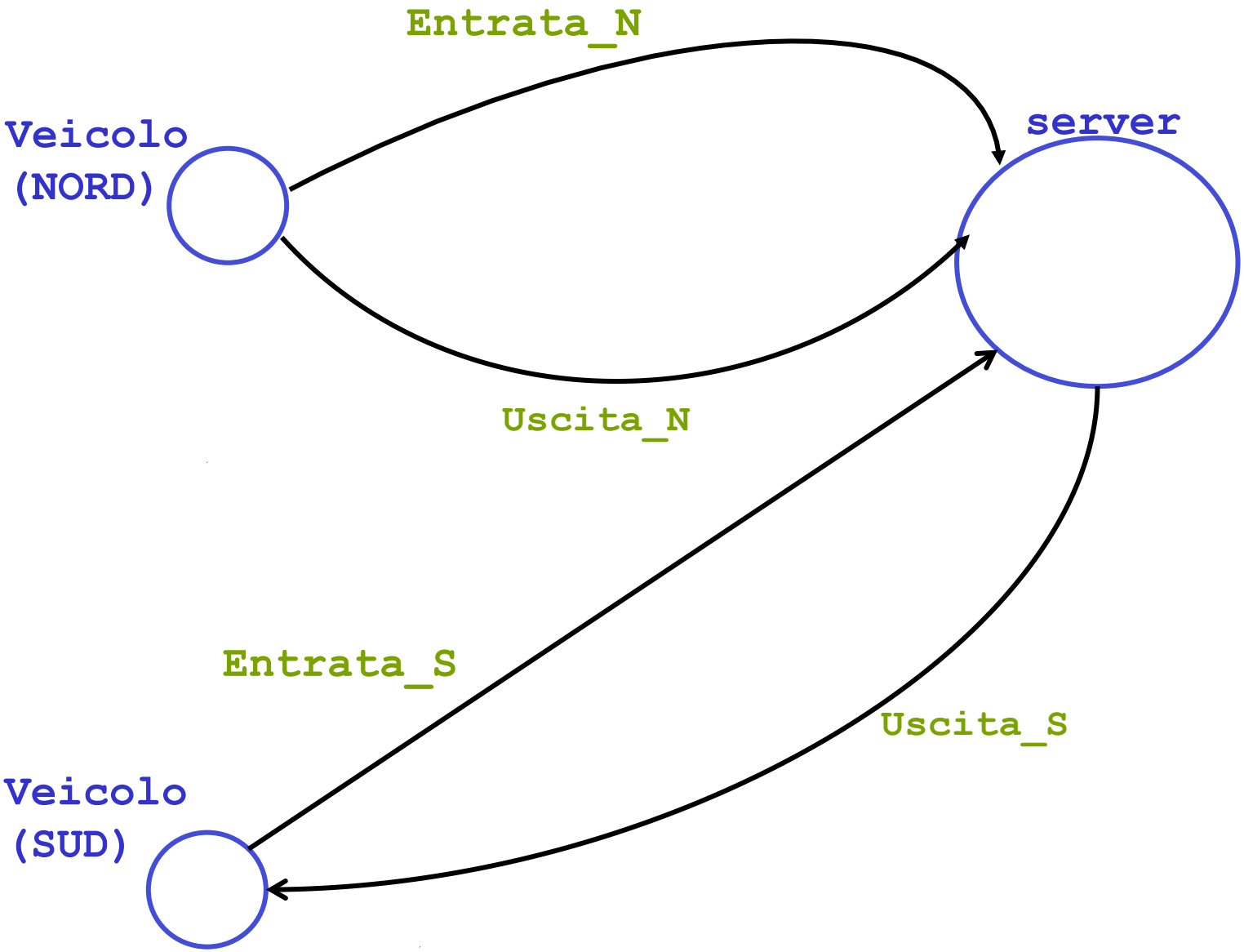
Si consideri un ponte che collega le 2 rive di un fiume (riva Nord, riva Sud).

Il ponte è stretto, pertanto può essere percorso solo a **senso unico alternato**. Ciò significa che un veicolo non può accedere al ponte su di esso vi è almeno un veicolo in direzione opposta.

Inoltre il ponte ha una **capacità limitata a MAX**, che esprime il massimo numero di veicoli che possono attraversarlo contemporaneamente.

Si realizzi un programma GO, nel quale la gestione del ponte sia affidata ad un server. Il server deve controllare accessi e uscite dal ponte rispettando i vincoli dati ed, inoltre, nell'accesso al ponte dia la **priorità** ai veicoli provenienti da NORD.

Impostazione



Impostazione

- **Quali e quanti canali?**
  - Un canale per le richieste di accesso da NORD(veicoli -> server):

```
var entrataN chan int
```

  - Un canale per le richieste di accesso da SUD(veicoli -> server):

```
var entrataS chan int
```

  - Un canale per le richieste di uscita da NORD (veicoli -> server):

```
var uscitaN chan int
```

  - Un canale per le richieste di uscita da SUD (veicoli -> server):

```
var uscitaS chan int
```

Priorità

Un veicolo che accede da sud non può entrare se ci sono processi da nord in attesa.

Problema: come fare per privilegiare le richieste di accesso da NORD?

Go: se un canale è **bufferizzato**, è possibile ottenere il numero di messaggi accodati tramite la funzione **len**:

```
var C=make(chan int, 100)
…
Num_msg:=len(C)
```

Applichiamo al nostro problema:

Quali e quanti canali?

- Due canali **bufferizzati** per le richieste di **accesso** da NORD e da SUD:

```
var entrataN = make(chan int, MAXBUFF)
var entrataS = make(chan int, MAXBUFF)
```

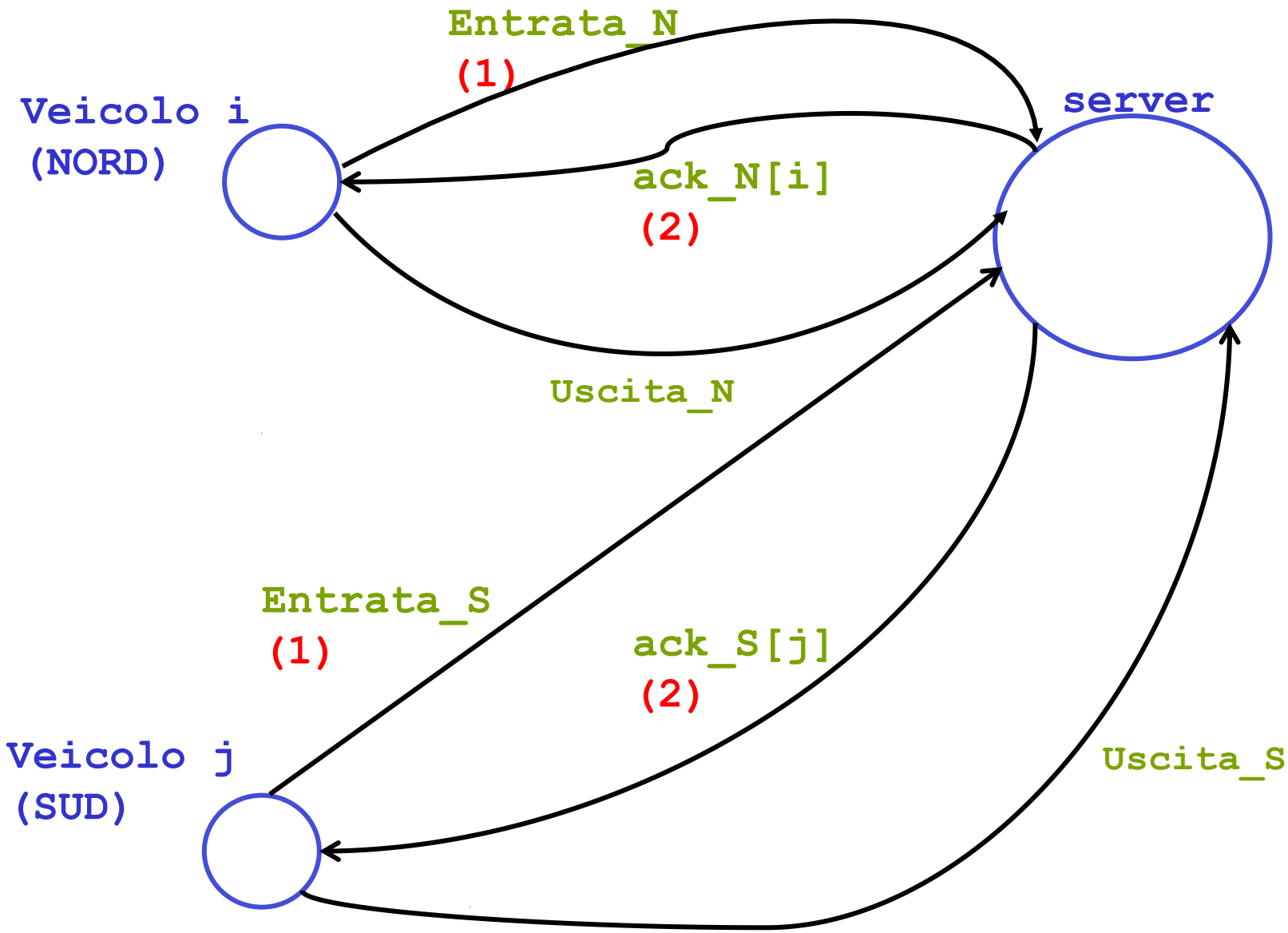
- Due canali **non bufferizzati** per le richieste di **uscita**:

```
var uscitaN = make(chan int)
var uscitaS = make(chan int)
```

- Inoltre, poichè la comunicazione su entrataN e entrataS è di tipo **asincrono**, è necessario prevedere un canale per ogni veicolo (server-> veicolo) per la sincronizzazione server->clienti.

```
var ACK_S [MAXPROC]chan int
var ACK_N [MAXPROC]chan int
```

Impostazione



Soluzione

```
package main
import (
    "fmt"
    "math/rand"
    "time" )
const MAXBUFF = 100
const MAXPROC = 10
const MAX = 3 // capacita'
const N int = 0
const S int = 1
var done = make(chan bool)
var termina = make(chan bool)
var entrataN = make(chan int, MAXBUFF)
var entrataS = make(chan int, MAXBUFF) /
var uscitaN = make(chan int)
var uscitaS = make(chan int)
var ACK_N [MAXPROC]chan int
var ACK_S [MAXPROC]chan int
```

```
func veicolo(myid int, dir int) {
    var tt int
    tt = rand.Intn(5) + 1
    time.Sleep(time.Duration(tt) * time.Second)
    if dir == N {
          entrataN <- myid // send asincrona
          <-ACK_N[myid]    // attesa x sincronizzazione
          tt = rand.Intn(5)
          time.Sleep(time.Duration(tt) * time.Second)
          uscitaN <- myid // send sincrona
    } else {
          entrataS <- myid
          <-ACK_S[myid] // attesa x sincronizzazione
          tt = rand.Intn(5)
          time.Sleep(time.Duration(tt) * time.Second)
          uscitaS <- myid  // send sincrona
    }
    done <- true }
```

```
func server() {
    var contN int = 0
    var contS int = 0
    for {
      select {
          case x := <-when((contN < MAX) && (contS == 0),
                 entrataN): // entrata nord
               contN++
               ACK_N[x] <- 1 // fine "call"
          case x := <-when((contS < MAX) && (contN == 0) &&
                 (len(entrataN) == 0), entrataS): // entrata sud
               contS++
               ACK_S[x] <- 1 // fine"call"
          case x := <-uscitaN:
               contN--
          case x := <-uscitaS:
               contS--
          case <-termina: // quando tutti i processi hanno finito
               done <- true
               return
          }}}
```

```
func main() {
    var VN int
    var VS int
    fmt.Printf("\n quanti veicoli NORD (max %d)? ", MAXPROC)
    fmt.Scanf("%d", &VN)
    fmt.Printf("\n quanti veicoli SUD (max %d)? ", MAXPROC)
    fmt.Scanf("%d", &VS)
    for i := 0; i < VN; i++ {
          ACK_N[i] = make(chan int, MAXBUFF)
    }
    for i := 0; i < VS; i++ {
          ACK_S[i] = make(chan int, MAXBUFF)
    }
    rand.Seed(time.Now().Unix())
    for i := 0; i < VS; i++ {
          go veicolo(i, S)
    }
    for i := 0; i < VN; i++ {
          go veicolo(i, N)
    } // continua
```

```
//..continua
    for i := 0; i < VN+VS; i++ {
            <-done
    }
    termina <- true
    <-done
    fmt.Printf("\n HO FINITO ")
}

func when(b bool, c chan int) chan int {
    if !b {
            return nil
    }
    return c
}

// fine programma
```

Esercizio:

Si consideri un ponte pedonale che collega le due rive di un fiume.

- Al ponte possono accedere due tipi di utenti: utenti **magri** e utenti **grassi**.
- Il ponte ha una **capacità massima** MAX che esprime il numero massimo di persone che possono transitare contemporaneamente su di esso.
- Il ponte e` talmente stretto che il transito di un grasso in una particolare direzione d impedisce l'accesso al ponte di altri utenti (grassi e magri) in direzione opposta a d.

  Realizzare una politica di sincronizzazione delle entrate e delle uscite dal ponte che tenga conto delle specifiche date, e che **favorisca gli utenti magri rispetto a quelli grassi nell'accesso al ponte.**