Esercizio

Si consideri l'ufficio di relazioni con il pubblico (URP) di una grande città.
L'ufficio è costituito da N sportelli, attraverso i quali è in grado di fornire al pubblico 2 tipi di prestazione:

- Informazioni turistiche (TUR)
- Informazioni su eventi (EVE)

Ogni sportello può eseguire un servizio alla volta (di qualunque tipo).
Per semplicità, si assuma che ogni utente richieda **un solo servizio alla volta.**
Si assuma inoltre che la permanenza di un utente allo sportello abbia una **durata non trascurabile.**
L'accesso degli utenti agli sportelli è regolato dai seguenti vincoli:

- l'erogazione di un servizio a un utente presuppone l'acquisizione di uno sportello libero da parte dell'utente richiedente.

Riguardo all'ordine delle richieste servite, si adotti un criterio basato su **priorità dinamica** e cioè:

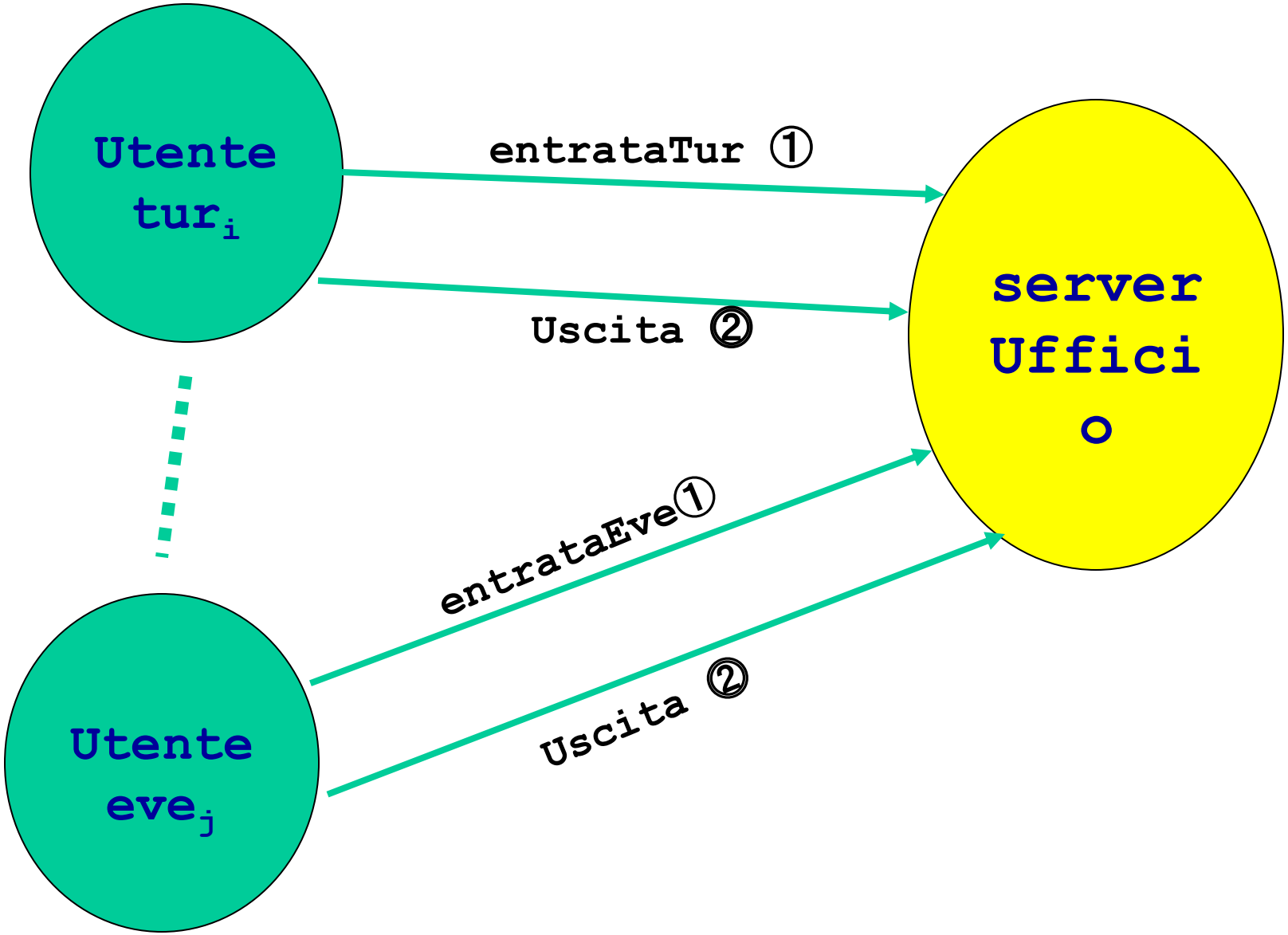
1) Inizialmente la priorità è assegnata alle richieste di tipo **TUR;**

2) **dopo aver servito K richieste TUR** (K è una costante data), la priorità viene invertita, quindi diventano prioritarie le richieste di tipo **EVE.**

3) Analogamente, **dopo aver servito K richieste** di tipo **EVE,** la priorità viene ancora invertita, e verrà data la precedenza a richieste di tipo **TUR,** e così via , ricominciando dal punto 1.

Realizzare un'applicazione nel linguaggio **Ada,** nella quale **clienti e ufficio** siano

Impostazione soluzione



SOLUZIONE IN GO

```
package main

import (
        "fmt"
        "math/rand"
        "time"
)
const MAXBUFF       = 100 //max channel buffer
const MAXPROC       = 10 //max numero processi
const N       = 5 // numero sportelli
const K       =3
const LIBERO = 0
const OCCUPATO = 1
const TUR     = 0
const EVE     = 1
```

```
//definizione canali
var done  = make(chan bool)
var termina    = make(chan bool)

//Channel
var entrataTUR = make(chan int, MAXBUFF) //
necessità di accodamento per priorità
var entrataEVE = make(chan int, MAXBUFF) //
necessità di accodamento per priorità
var uscita     = make(chan int)

//ACK (1 per ogni cliente)
var ACK_TUR [MAXPROC]chan int
var ACK_EVE [MAXPROC]chan int
```

```
func utente(myid int, typeT int) {
      var tt int
      var sp int
      tt = rand.Intn(5) + 1
      time.Sleep(time.Duration(tt) * time.Second)
      switch typeT{
      case TUR:
            entrataTUR<-myid
            sp=<-ACK_TUR[myid]
            tt= rand.Intn(5)+2
            time.Sleep(time.Duration(tt)*time.Second)
            uscita<-sp
      case EVE:
            entrataEVE<-myid
            modalita EVE \n", myid)
            sp=<-ACK_EVE[myid]
            tt=rand.Intn(5)+2
            time.Sleep(time.Duration(tt)*time.Second)
            uscita<-sp
      }
    done<-true }
```

```
func ufficio(){ // SERVER
        var contK = 0
        var contSp int = 0
        var modalita int = TUR
        var sportelli[N] int

        for i:=0; i<N; i++ {
        sportelli[i]=LIBERO
        }
        // continua...
```

```
for {
        select {
        case x := <-when( (modalita==TUR && contSp<N) ||
        (modalita!=TUR && len(entrataEVE)==0 && contSp<N),
        entrataTUR):
                contSp++
                if modalita==TUR {
                        contK++
                        if contK==K { // inversione prio
                                contK=0
                                modalita=EVE
                        }
                }
                sp:= sportelloLibero(sportelli)
                sportelli[sp]= OCCUPATO
                ACK_TUR[x] <- sp // termine "call» TUR
```

```
    case x := <-when((modalita==EVE && contSp<N)||
        (modalita!=EVE && len(entrataTUR)==0 && contSp<N) ,
        entrataEVE):
                contSp++
                if modalita==EVE {
                        contK++
                        if contK==K { // INVERSIONE PRIO
                                contK=0
                                modalita=TUR
                        }
                }
                sp:= sportelloLibero(sportelli)
                sportelli[sp]= OCCUPATO
                ACK_EVE[x] <- sp // termine "call"
        case x := <-uscita:
                contSp--
                sportelli[x]=LIBERO
        case <-termina: // quando tutti i processi hanno finito
                done <- true
                return
        }//select
}} // FINE SERVER
```

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

func sportelloLibero(sportelli[N] int) int{
      for i:=0;i<N; i++ {
            if sportelli[i]==LIBERO {
                  return i
            }
      }
      return -1
}
```

```
func main() {
    var V1 int
    var V2 int
    fmt.Printf("\nQuanti Thread TUR (max %d)? ", MAXPROC)
    fmt.Scanf("%d", &V1)
    fmt.Printf("\nQuanti Thread EVE (max %d)? ", MAXPROC)
    fmt.Scanf("%d", &V2)
     for i := 0; i < V1; i++ {
           ACK_TUR[i] = make(chan int, MAXBUFF)
    }
     for i := 0; i < V2; i++ {
           ACK_EVE[i] = make(chan int, MAXBUFF)
    }
    rand.Seed(time.Now().Unix())
    go ufficio() //Go routine server
    for i := 0; i < V1; i++ {
           go utente(i,TUR) //Go routine client
    }
     for i := 0; i < V2; i++ {
           go utente(i,EVE)
    }
```

```
        //attesa terminazione
        for i := 0; i < V1+V2; i++ {
                <-done
        }

         termina <- true
        <-done
        fmt.Printf("\n[main]HO FINITO!!!   \n")
}
```