Esercizio sul Monitor

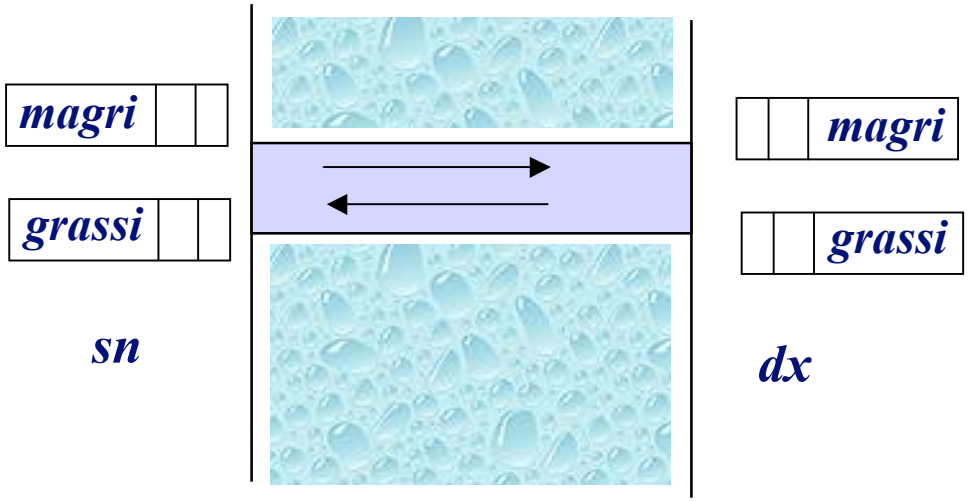
Ponte con utenti grassi e magri
10 Novembre 2009

Ponte con utenti grassi e magri

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 capacita` 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.



➡ *due tipi di thread:*

- ✓ *grassi*
- ✓ *magri*

➡ *una coda per ogni tipo di thread e per ogni direzione*

Progetto del *monitor* `ponte`:

- lo stato del ponte e` definito da:
  - numero magri e di grassi sul ponte (per ogni direzione)
- lo stato e` modificabile dalle operazioni di:
  - accesso: ingresso di un thread nel ponte
  - rilascio: uscita di un thread dal ponte
- l'accesso al monitor deve avvenire in modo mutuamente esclusivo:
  - predispongo un mutex per il controllo della mutua esclusione nell'esecuzione delle operazioni di accesso e di rilascio: `lock`
- i thread grassi e magri si possono sospendere se le condizioni necessarie per l'accesso non sono verificate :
  - una coda per ogni tipo di thread (grasso o magro)e per ogni direzione
- per ispezionare lo stato delle code introduciamo:
  - un contatore dei thread sospesi per ogni tipo di thread (grasso o magro)e per ogni direzione

➡integro il tutto all'interno del tipo struct `ponte`

Grassi & Magri: tipo di dato associato al ponte

```
typedef struct
{
   int nmagri[2]; /* numero magri sul ponte (per ogni dir.)*/
   int ngrassi[2];/* numero grassi sul ponte (per ogni
  dir.)*/
   pthread_mutex_t lock;/*lock associato al"ponte" */
   pthread_cond_t codamagri[2]; /* var. cond. sosp. magri */
   pthread_cond_t codagrassi[2]; /* var. cond. sosp. grassi
  */
   int sospM[2];/* numero di processi magri sospesi*/
   int sospG[2];/* numero di processi grassi sospesi*/
}ponte;
```

Produttore e consumatore

Operazioni sulla *risorsa* `ponte`:

- `init`: inizializzazione del ponte.
- `accessomagri/accessograssi`: operazione eseguita dai thread (grassi/magri) per l'ingresso nel ponte.
- `rilasciomagri/rilasciograssi`: operazione eseguita dai thread (grassi/magri) per l'uscita dal ponte.

Grassi & Magri: soluzione

```
include <stdio.h>
#include <pthread.h>
#define MAX 3 /* max capacita ponte */
#define dx 0 /*costanti di direzione*/
#define sn 1

typedef struct
{
   int nmagri[2]; /* numero magri sul ponte (per ogni dir.)*/
   int ngrassi[2];/* numero grassi sul ponte (per ogni dir.)*/
   pthread_mutex_t lock;/*lock associato al"ponte" */
   pthread_cond_t codamagri[2]; /* var. cond. sosp. magri */
   pthread_cond_t codagrassi[2]; /* var. cond. sosp. grassi */
   int sospM[2];/* numero di processi magri sospesi*/
   int sospG[2];/* numero di processi grassi sospesi*/
}ponte;
```

Grassi & Magri: soluzione

```
/* Inizializzazione del ponte */
void init (ponte *p)
{
   pthread_mutex_init (&p->lock, NULL);
   pthread_cond_init (&p->codamagri[dx], NULL);
   pthread_cond_init (&p->codamagri[sn], NULL);
   pthread_cond_init (&p->codagrassi[dx], NULL);
   pthread_cond_init (&p->codagrassi[sn], NULL);
   p->nmagri[dx]=0;
   p->nmagri[sn]=0;
   p->ngrassi[dx]=0;
   p->ngrassi[sn]=0;
   p->sospM[dx] = 0;
   p->sospM[sn] = 0;
   p->sospG[dx] = 0;
   p->sospG[sn] = 0;
   return;
}
```

```
/*operazioni di utilita`: */
int sulponte(ponte p); /* calcola il num. di persone sul ponte
   */
int altra_dir(int d); /* calcola la direzione opposta a d */

/* Accesso al ponte di un magro in direzione d: */
void accessomagri (ponte *p, int d)
{  pthread_mutex_lock (&p->lock);
   /* controlla le codizioni di accesso:*/
   while ( (sulponte(*p)==MAX) || /* vincolo di capacita` */
           (p->ngrassi[altra_dir(d)]>0) ) /*grassi in
                                            dir. opposta */
   {       p->sospM[d]++;
           pthread_cond_wait (&p->codamagri[d], &p->lock);

   }
   /* entrata: aggiorna lo stato del ponte */
   p->nmagri[d]++;
  pthread_mutex_unlock (&p->lock);
}
```

```
void accessograssi (ponte *p, int d)
{  pthread_mutex_lock (&p->lock);
   /* controlla le codizioni di accesso:*/
   while ( (sulponte(*p)==MAX) ||
           (p->ngrassi[altra_dir(d)]>0)||
           (p->nmagri[altra_dir(d)]>0) ||
           (p->sospM[altra_dir(d)]>0))
   {          p->sospG[d]++;
              pthread_cond_wait (&p->codagrassi[d], &p->lock);
    }
    /* entrata: aggiorna lo stato del ponte */
   p->ngrassi[d]++;
   pthread_mutex_unlock (&p->lock);
}
```

```
void rilasciomagri (ponte *p, int d)
{
   pthread_mutex_lock (&p->lock);
   /* uscita: aggiorna lo stato del ponte */
   p->nmagri[d]--;
   /* risveglio in ordine di priorita` */
   pthread_cond_broadcast (&p->codamagri[altra_dir(d)]);
    p->sospM[altradir(d)]=0;
   pthread_cond_broadcast (&p->codamagri[d]);
    p->sospM[d]=0;
   pthread_cond_broadcast (&p->codagrassi[altra_dir(d)]);
  p->sospG[altradir(d)]=0;
   pthread_cond_broadcast (&p->codagrassi[d]);
  p->sospG[d]=0;
  pthread_mutex_unlock (&p->lock);
}
```

```
void rilasciograssi (ponte *p, int d)
{
   pthread_mutex_lock (&p->lock);

   /* uscita: aggiorna lo stato del ponte */
   p->ngrassi[d]--;
   /* risveglio in ordine di priorita` */
   pthread_cond_broadcast (&p->codamagri[altra_dir(d)]);
   p->sospM[altradir(d)]=0;
   pthread_cond_broadcast (&p->codamagri[d]);
    p->sospM[d]=0;
   pthread_cond_broadcast (&p->codagrassi[altra_dir(d)]);
  p->sospG[altradir(d)]=0;
   pthread_cond_broadcast (&p->codagrassi[d]);
  p->sospG[d]=0;
   pthread_mutex_unlock (&p->lock);
}
```

```
/* Programma di test: genero un numero arbitrario di threa
   magri e
grassi nelle due direzioni */
#define MAXT 20 /* num. max di thread per tipo e per
   direzione */


ponte p;

void *magro (void *arg) /*codice del thread "magro" */
{ int d;
  d=atoi((char *)arg); /*assegno la direzione */
  accessomagri (&p, d);
  /* ATTRAVERSAMENTO: */
  sleep(1);
  rilasciomagri(&p,d);
  return NULL;
}
```

```
void *grasso (void *arg) /*codice del thread "grasso" */
{  int d;
   d=atoi((char *)arg); /*assegno la direzione */
    accessograssi (&p, d);
    sleep(1);
    rilasciograssi(&p,d);
    return NULL;
}

main ()
{
  pthread_t th_M[2][MAXT], th_G[2][MAXT];
  int NMD, NMS, NGD, NGS, i;
  void *retval;

  init (&p);
```

```
/* Creazione threads: */
  printf("\nquanti magri in direzione dx? ");
  scanf("%d", &NMD);
  printf("\nquanti magri in direzione sn? ");
  scanf("%d", &NMS);
  printf("\nquanti grassi in direzione dx? ");
  scanf("%d", &NGD);
  printf("\nquanti grassi in direzione sn? ");
  scanf("%d", &NGS);
  /*CREAZIONE MAGRI IN DIREZIONE DX */
  for (i=0; i<NMD; i++)
          pthread_create (&th_M[dx][i], NULL, magro, "0");
   /*CREAZIONE MAGRI IN DIREZIONE SN */
  for (i=0; i<NMS; i++)
          pthread_create (&th_M[sn][i], NULL, magro, "1");
  /*CREAZIONE GRASSI IN DIREZIONE DX */
  for (i=0; i<NGD; i++)
          pthread_create (&th_G[dx][i], NULL, grasso, "0");
   /*CREAZIONE GRASSI IN DIREZIONE SN */
  for (i=0; i<NGS; i++)
          pthread_create (&th_G[sn][i], NULL, grasso, "1");
```

```
 /* Attesa teminazione threads creati: */

  /*ATTESA MAGRI IN DIREZIONE DX */
  for (i=0; i<NMD; i++)
          pthread_join(th_M[dx][i], &retval);

   /*ATTESA MAGRI IN DIREZIONE SN */
  for (i=0; i<NMS; i++)
          pthread_join(th_M[sn][i], &retval);

  /*ATTESA GRASSI IN DIREZIONE DX */
  for (i=0; i<NGD; i++)
          pthread_join(th_G[dx][i], &retval);

   /*ATTESA GRASSI IN DIREZIONE SN */
  for (i=0; i<NGS; i++)
          pthread_join(th_G[sn][i], &retval);

  return 0;
}
```

```
 /* definizione funzioni utilita`:*/
int sulponte(ponte p) /* calcola il num.di pers.sul ponte */
{
        return p.nmagri[dx]+p.nmagri[sn]+p.ngrassi[dx]+
   p.ngrassi[sn];

}

int altra_dir(int d) /* fornisce la dir. opposta a  d */
{
        if (d==sn) return dx;
        else return sn;
}
```