Esercizio E5.1

Impostazione

1. Quali processi?
    a. Proprietario del Castello (guida)
    b. Visitatori Bambini
    c. Visitatori Adulti

2. Quale struttura per i processi ?
   Sia C la struttura dati che contiene i dati relativi al castello:

Proprietario:
```
while(1) {
 iniziovisita(&C);
<visita>
finevisita(&C);
}
```

Bambino:
```
bigliettoeattesa(&C,B);
<visita>
```

Adulto:
```
bigliettoeattesa(&C,A);
<visita>
```

3. Definizione del monitor **castello**:

Dati:

```
typedef struct{
            pthread_mutex_t lock; /* mutex per la mutua esclusione nell'accesso al monitor*/
            pthread_cond_t coda[3]; /*code bambini(0), adulti (1) e proprietario(2) */
            int sosp[3]; /*numero dei processi sospesi nelle 3 code */
            int prossima; /* tipo della prossima visita (0: bambini; 1:adulti) */
            int NumeroVisita; /* numero d'ordine della visita corrente */
          } castello;
```

Operazioni (entry):

`iniziovisita(castello *c);`
operazione eseguita dal thread proprietario per iniziare una nuova visita guidata; puo` comportare attesa, se il gruppo della prossima visita non e` ancora completo.

`finevisita(castello *c);`
operazione eseguita dal thread proprietario concludere una visita guidata e decidere il tipo della prossima visita.

`bigliettoeattesa(castello *c, int tipo);`
operazione eseguita da ogni visitatore (il parametro tipo stabilisce se si tratta di adulto o bambino) per aggregarsi al proprio gruppo e attendere l'avvio della visita.

Soluzione:


```
/* soluzione  castello*/
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#define TIPOBAMBINO 0
#define TIPOADULTO 1
#define TIPOPROPRIETARIO 2
#define PMAX 5
#define MAXT 15

typedef struct{
            pthread_mutex_t lock;
            pthread_cond_t coda[3];
            int sosp[3];
            int prossima; /* 0: bambini; 1:adulti */
            int NumeroVisita;
            } castello;

castello C;

void iniziovisita(castello *c)
{     int next,i;
      pthread_mutex_lock (&c->lock);
      next=c->prossima;
      /* controlla le codizioni di accesso:*/
```

```
      if ( (c->sosp[next])<PMAX) /*il gruppo non e` completo */
      {        c->sosp[2]++;
              pthread_cond_wait (&c->coda[2], &c->lock);
              c->sosp[2]--;
      }
      /* la visita parte */
      for (i=0; i< PMAX; i++) pthread_cond_signal(&c->coda[next]);
      printf("\n\n PROPRIETARIO: inizio visita numero %d per
          visitatori di tipo %d\n", ++(c->NumeroVisita), next);
      pthread_mutex_unlock (&c->lock);
}

void finevisita(castello *c)
{     pthread_mutex_lock (&c->lock);
      /* decisione prossima visita:*/
      if (c->sosp[TIPOADULTO]>c->sosp[TIPOBAMBINO])
            c->prossima=TIPOADULTO;
      else  c->prossima=TIPOBAMBINO;
      printf("Proprietario:  prossima visita: %d\n", c->prossima);
      pthread_mutex_unlock (&c->lock);
}

void bigliettoeattesa(castello *c, int t)
{      pthread_mutex_lock (&c->lock);
       c->sosp[t]++;
       pthread_cond_wait (&c->coda[t], &c->lock);
      c->sosp[t]--;
       printf("Visitatore di tipo %d: inizio la visita!\n", t);
      pthread_mutex_unlock (&c->lock);
}

void *thread_Proprietario(void * arg) /*thread proprietario*/
{   while(1)
    {
        iniziovisita(&C);
        sleep(2); /* durata visita... */
        finevisita(&C);
    }
    pthread_exit (0);
}

void *thread_Bambino(void * arg) /*thread bambino*/
{
      bigliettoeattesa(&C,TIPOBAMBINO);
      sleep(2);
      printf("un bambino ha finito il giro!\n");
}

void *thread_Adulto(void * arg) /*thread adulto*/
{
      bigliettoeattesa(&C,TIPOADULTO);
      sleep(2);
      printf("un adulto ha finito il giro!\n");
}
```

```
void init (castello *p)
{
  pthread_mutex_init (&p->lock, NULL);
  pthread_cond_init (&p->coda[0], NULL);
  pthread_cond_init (&p->coda[1], NULL);
  pthread_cond_init (&p->coda[2], NULL);
  p->sosp[0] = 0;
  p->sosp[1] = 0;
  p->sosp[2] = 0;
  p->prossima = TIPOBAMBINO;
  p->NumeroVisita=0;
 }
```

```
/* programma di test */
main()
{
  pthread_t th_A[MAXT], th_B[MAXT], th_P;
  int NB, NA, i;
  void *retval;

  init (&C);
/* Creazione threads: */
  printf("\nquanti bambini ? ");
  scanf("%d", &NB);
  printf("\nquanti adulti? ");
  scanf("%d", &NA);

 /*Creazione bambini */
  for (i=0; i<NB; i++)
          pthread_create (&th_B[i], NULL, thread_Bambino, NULL);

  /* Creazione adulti */
  for (i=0; i<NA; i++)
          pthread_create (&th_A[i], NULL, thread_Adulto, NULL);

  pthread_create (&th_P, NULL, thread_Proprietario, NULL);

 /* Attesa teminazione threads creati: */

  for (i=0; i<NA; i++)
          pthread_join(th_A[i], &retval);
  for (i=0; i<NB; i++)
          pthread_join(th_B[i], &retval);
   pthread_join(th_P, &retval);

  return 0;
}
```