Esercizio E5.2

Impostazione

1. Quali processi?
   a. clienti convenzionati
   b. clienti non convenzionati

2. Quale struttura per i processi ?

Convenzionato:
```
A=richiesta(nolo, Auto, Conv);
<usa l'auto A>
restituzione(nolo, A);
```

Non Convenzionato:
```
A=richiesta nolo, Auto, Non Conv)
      <usa l'auto A>
      restituzione (nolo, A);
```

3. Definizione del monitor **noleggio**:

Dati:

```
typedef struct
{ int disp[3]; /* numero auto disp.  (per ogni cat.)*/
  pthread_mutex_t MUX; /*mutex */
  pthread_cond_t Q[3][2]; /* Q[tipoauto][tipocliente] */
}noleggio;
```

Operazioni

`richiesta(noleggio, auto, cliente);`
operazione eseguita da ogni thread per ottenere un'auto a nolo; è possibile ottenere un'automobile di categoria diversa .

`restituzione(noleggio, auto);`
operazione eseguita dai thread per restituire un'auto.

Soluzione:

```
/* soluzione  noleggio*/
include <stdio.h>
#include <pthread.h>
#define TotP 10 /* numero totale auto piccole */
#define TotM 10 /* numero totale auto medie */
#define TotG 10 /* numero totale auto grandi */
#define P 0 /*indice auto piccole*/
#define M 1 /*indice auto medie*/
#define G 2 /*indice auto grandi*/
#define Conv 0
#define NonConv 1

typedef struct
            { int disp[3]; /* numero auto disp.(per cat.)*/
              pthread_mutex_t MUX;  /*mutex */
```

```
            pthread_cond_t Q[3][2]; /* Q[tipoauto][tipocliente] */
          }noleggio;

/* funzione richiesta auto: */
int  richiesta (noleggio *n, int A, int cl)
{     int ottenuta;
      pthread_mutex_lock (&n->MUX);
      switch(A){
      case P:     if ((n->disp[P]==0)&&(n->disp[M]==0))
            {     while(!n->disp[P])
                        pthread_cond_wait (&n->Q[P][cl], &n->MUX);
                  ottenuta=P;
                  }
                  else if (n->disp[P]!=0) ottenuta=P;
                  else  ottenuta=M;
                  break;
      case M:     if ((n->disp[M]==0)&&(n->disp[G]==0))
            {      while(!n->disp[M])
                         pthread_cond_wait (&n->Q[M][cl], &n->MUX);
                   ottenuta=M;
                  }
                  else if (n->disp[M]!=0) ottenuta=M;
                  else  ottenuta=G;
                  break;
      case G:     while (n->disp[G]==0)
                  pthread_cond_wait(&n->Q[G][cl],&n->MUX);
            ottenuta=G;
                  break;
                {
      n->disp[ottenuta]--;
      pthread_mutex_unlock (&n->MUX);
      return ottenuta;
}

/* procedure restituzione auto: */
void restituzione (noleggio *n, int A)
{
     pthread_mutex_lock (&n->MUX);
     /*  aggiorna lo stato del noleggio */
     n->disp[A]++;
     /* risveglio in ordine di priorita` */
     pthread_cond_signal (&n->Q[A][Conv]);
     pthread_cond_signal (&n->Q[A][NonConv]);
     pthread_mutex_unlock (&n->MUX);
}

void init (noleggio *n)
{     int i, j;
      pthread_mutex_init (&n->MUX, NULL);
      for (i=0; i<2; i++)
            for (j=0; j<3; j++)
                  pthread_cond_init (&n->Q[i][j], NULL);
      n->disp[P]=TotP;
      n->disp[M]=TotM;
      n->disp[G]=TotG;
```

```
      return;
}
```

/* Programma di test: genero MAXT thread convenzionati e non convenzionati per ogni tipo di auto */

```
#define MAXT 50 /* num. di thread per tipo e per auto */
noleggio n;

void *clienteConv (void *arg) /*cliente "convenzionato"*/
{ int A, i;
  printf("thread C. %d: richiedo un'auto di tipo %s\n\n",
                 pthread_self(),(char *)arg);
  A=atoi((char *)arg); /*A auto richiesta */
  A=richiesta (&n, A, Conv);
  printf(" thread C.%lu: ottengo un'auto %d\n\n",
           pthread_self(),A);

  sleep(2); /* Uso.. */

  restituzione(&n,A);
  printf("thread C:%lu: ho restituito l'auto.\n\n",
           pthread_self());
  return NULL;
}


void *clienteNonConv (void *arg) /* cliente non convenzionato*/
{ int A, i;
  printf("thread NC. %d: richiedo un'auto di tipo %s\n\n",
                 pthread_self(),(char *)arg);
  A=atoi((char *)arg); /*A auto richiesta */
  A=richiesta (&n, A, NonConv);
  printf(" thread NC. %lu: ottengo un'auto %d\n\n",
                 pthread_self(),A );
  sleep(2); /* Uso.. */
  restituzione(&n,A);
  printf("thread NC.%lu: ho restituito l'auto.\n\n",
                 pthread_self());
  return NULL;
}
```

/* programma di test: */

```
main ()
{  pthread_t th_C[3][MAXT], th_NC[3][MAXT];
  int i,j;
  void *retval;
  init (&n);
```

/*Creazione thread: */

```
  for (i=0; i<MAXT; i++)
      {     pthread_create (&th_C[0][i], NULL, clienteConv, "0");
            pthread_create (&th_C[1][i], NULL, clienteConv, "1");
            pthread_create (&th_C[2][i], NULL, clienteConv, "2");
```

```
      }
  for (i=0; i<MAXT; i++)
      {     pthread_create (&th_NC[0][i], NULL, clienteNonConv, "0");
            pthread_create (&th_NC[1][i], NULL, clienteNonConv, "1");
            pthread_create (&th_NC[2][i], NULL, clienteNonConv, "2");
      }
/* Attesa teminazione threads :*/
  for (i=0; i<MAXT; i++)
      {     pthread_join (&th_C[0][i], &retval);
            pthread_join (&th_C[1][i], &retval);
            pthread_join (&th_C[2][i], &retval);
      }
      for (i=0; i<MAXT; i++)
      {     pthread_join (&th_NC[0][i], &retval);
            pthread_join (&th_NC[1][i], &retval);
            pthread_join (&th_NC[2][i], &retval);
      }
}
```