Linuxthreads: esercizio

Esercizio

Una società di noleggio di automobili offre ai propri clienti **tre tipi di automobili**: piccole, medie, grandi. Ogni tipo di auto è disponibile in numero **limitato** (*Npiccole, Nmedie, Ngrandi*).

I clienti del noleggio possono essere di due tipi: **convenzionati**, **non convenzionati**.

Ogni cliente richiede una macchina di un particolare tipo (piccola, media, o grande); il noleggio, sulla base della propria disponibilità corrente :

- assegnerà a chi ha richiesto una macchina piccola, preferenzialmente una macchina piccola (se è disponibile), oppure una media (solo se vi è **immediata** disponibilità di medie) .
- assegnerà a chi ha richiesto una media, preferenzialmente una media (se è disponibile), oppure una grande (solo se vi è **immediata** disponibilità di grandi).
- a chi richiede una macchina grande, deve essere comunque assegnata una grande.

..continua

Il cliente, dopo aver **ritirato** la macchina presso la sede del noleggio, la usa per un intervallo di tempo arbitrario, e successivamente procede alla **restituzione** della macchina. Le regole del noleggio prevedono che **ritiro** e **restituzione** delle auto avvengano sempre presso la stessa sede: non è prevista la restituzione in una sede diversa da quella del ritiro.

Per la gestione del noleggio, vale inoltre il seguente vincolo:

- nel ritiro delle auto, i clienti **convenzionati** devono essere prioritari rispetto ai non convenzionati.

Definire una politica di gestione del noleggio, e la si realizzi, utilizzando la libreria pthread.

Progetto della *risorsa* `noleggio`:

- **lo stato del noleggio e`definito da:**
  - **numero auto piccole, medie e grandi disponibili.**
- **lo stato e` modificabile dalle operazioni di:**
  - **richiesta: acquisizione di un'auto da parte di un cliente**
  - **restituzione di un auto da parte di un cliente**
- **il noleggio e`una risorsa da accedere in modo mutuamente esclusivo:**
  - **predispongo un mutex per il controllo della mutua esclusione nell'esecuzione delle operazioni di accesso e di rilascio: `MUX`**
- **ogni cliente e` rappresentato da un thread, caratterizzato dal tipo dell'auto richiesta (P, M, G) e dal tipo del cliente (Convenzionato o No):**
  - **una coda per ogni tipo di cliente (Conv, NonConv) e per ogni tipo di auto (P, M, G).**

➔Incapsulo il tutto all'interno del tipo struct `noleggio`

Tipo associato al noleggio

```
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] */
}ponte;
```

Operazioni sulla *risorsa* `noleggio`:

- **`init(noleggio)`: inizializzazione del noleggio.**
- **`richiesta(noleggio, auto, cliente)`: operazione eseguita da ogni thread per ottenere un'auto a nolo; e` possibile ottenere un'automobile di categoria diversa -> funzione**
- **`restituzione(noleggio, auto)`: operazione eseguita dai thread per restituire un'auto.**

Soluzione

```
#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;
```

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

}
```

```
/* richiesta auto: */
int  richiesta (noleggio *n, int A, int cl)
{ int ottenuta;
   pthread_mutex_lock (&n->MUX);
   /* controlla le codizioni di accesso:*/
   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;
```

```
/* ..continua richiesta: */
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;
}
```

```
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);
}
```

```
/* Programma di test: genero un numero arbitrario di 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) /*codice del "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. %d: ottengo un'auto %d\n\n", pthread_self(),A
   /* USO..: */
  restituzione(&n,A);
  printf("thread C:%d: ho restituito l'auto.\n\n", pthread_self())
  return NULL;
}
```

```
void *clienteNonConv (void *arg)
{ 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. %d: ottengo un'auto %d\n\n", pthread_self(),A );
   /* USO..: */
  restituzione(&n,A);
  printf("thread NC.%d: ho restituito l'auto.\n\n", pthread_self());
  return NULL;
}


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

  init (&n);
```

```
/* Creazione threads: */
/*CREAZIONE convenzionati */
  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");
   }
/*CREAZIONE convenzionati */
  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 creati:*/
  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);
   }
}
```