Esercizio 1

Facendo uso della libreria Pthread, realizzare un programma in cui un thread scrittore, dato un intero N da riga di comando (dove 10<N<=15), scrive in un file nella prima posizione, uno alla volta ed ogni ½ secondo, la sequenza di Fibonacci di ordine N, alternandosi con un thread lettore che legge, uno alla volta dalla prima posizione del file, i numeri scritti dal thread scrittore. Un terzo thread attende la lettura dell' N-esimo intero, quindi stampa a video il messaggio "Operazioni concluse, arrivederci dal thread: tid", attende 5 secondi e termina.

Esercizio 1 Sol.

```
sem_t *sem1,*sem2;
int n;
struct timespec attesa;
attesa.tv_sec = 0;
attesa.tv_nsec = 500000000L;
void *scrittore(void *argc);
void *lettore(void *argc);
void *stampa(void *argc);
int fd;
pthread_mutex_t mutex;
pthread_cond_t cond;
int cnt=-1;
int main(int argc, char **argv){
   if ( argc == 2 ){
      n = atoi(argv[1]); }
   sem_unlink(”/sem1"); sem_unlink(”/sem2");
   sem1 = sem_open(”/sem1",O_CREAT,0777,1);
   sem2 = sem_open(”/sem2",O_CREAT,0777,0);
```

Esercizio 1 Sol.

```
if ( n < 10 || n >= 15 ){
   printf("Errore\n"); return 0;
}
pthread_t thread1,thread2,thread3;
fd = open("numeri",O_RDWR|O_CREAT|O_TRUNC,0777);
pthread_create(&thread1,NULL,scrittore,NULL);
pthread_create(&thread2,NULL,lettore,NULL);
pthread_create(&thread3,NULL,stampa,NULL);

pthread_join(thread1,NULL);
pthread_join(thread2,NULL);
pthread_join(thread3,NULL);
return 0; }
```

Esercizio 1 Sol.

```
void *scrittore(void *argc){
   int i;
   int a; int b;
   int f;
   b = 1; a = 0;
   for(i=0; i<=n; i++){
      sem_wait(sem1);
      f = b+a;
      a = b;
      b = f;
      lseek(fd,0,SEEK_SET);
      write(fd,&b,sizeof(int));
      printf("Ho scritto: %d\n",b);
      nanosleep(&attesa,NULL);
      sem_post(sem2);
   }
}
```

Esercizio 1 Sol.

```
void *lettore(void *argc){
   int i;
   int buff;
   while(1){
      sem_wait(sem2);
      lseek(fd,0,SEEK_SET);
      read(fd,&buff,sizeof(int));
      printf("Ho letto: %d\n",buff);
      sem_post(sem1);
      pthread_mutex_lock(&mutex);
      cnt++;
      if ( cnt == n ){
         pthread_cond_signal(&cond);
         pthread_mutex_unlock(&mutex);
         return;
      }
      pthread_mutex_unlock(&mutex);
   }
}
```

Esercizio 1 Sol.

```
void *stampa(void *argc){
   pthread_mutex_lock(&mutex);
   while ( cnt < n){
      pthread_cond_wait(&cond,&mutex);
   }
   pthread_mutex_unlock(&mutex);
   pthread_t tid = pthread_self();
   printf("Operazioni concluse, arrivederci dal
    thread: %u\n",(unsigned)tid);
}
```

Esercizio 2

- Dato un ponte con una capacità massima MAX che esprime il numero massimo di veicoli che possono transitare contemporaneamente su di esso. Considerata l'ipotesi che sul ponte possono transitare solo due tipi di veicoli: automobili e camion e che il ponte è talmente stretto che il transito di un camion in una particolare direzione impedisce l'accesso al ponte di altri veicoli in direzione opposta.
- Realizzare una politica di sincronizzazione delle entrate e delle uscite dal ponte che tenga conto delle specifiche date e che favorisca le automobili rispetto ai camion nell'accesso al ponte.

Esercizio 2 Sol.

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

typedef struct{
 int auto[2];//numero auto sul ponte (per ogni dir.)
 int camion[2];//numero camion sul ponte (per ogni dir.)
 pthread_mutex_t lock;//lock associato alla risorsa "ponte"
 pthread_cond_t codaauto[2]; // var. cond. sosp. auto
 pthread_cond_t codacamion[2]; // var. cond. sosp. camion
 int sospA[2];// numero di processi auto sospesi
 int sospC[2];// numero di processi camion sospesi
}ponte;
```

Esercizio 2 Sol.

```
/* Inizializzazione del ponte */
void init (ponte *p){
   pthread_mutex_init (&p->lock, NULL);
   pthread_cond_init (&p->codaauto[dx], NULL);
   pthread_cond_init (&p->codaauto[sn], NULL);
   pthread_cond_init (&p->codacamion[dx], NULL);
   pthread_cond_init (&p->codacamion[sn], NULL);
   p->nauto[dx]=0;
   p->nauto[sn]=0;
   p->ncamion[dx]=0;
   p->ncamion[sn]=0;
   p->sospA[dx] = 0;
   p->sospA[sn] = 0;
   p->sospC[dx] = 0;
   p->sospC[sn] = 0;
   return;
}
```

Esercizio 2 Sol.

```
/*operazioni di utilità */
int sulponte(ponte p); /* calcola il num. di veicoli sul ponte */
int altra_dir(int d); /* calcola la direzione opposta a d */
/* Accesso al ponte di un auto in direzione d: */
void accessoauto (ponte *p, int d){
   pthread_mutex_lock (&p->lock);
   /* controlla le codizioni di accesso:*/
   while ((sulponte(*p)==MAX) || /* vincolo di capacita` */
          (p->ncamion[altra_dir(d)]>0) ) /*ci sono camion in direzione opposta */
   {
     p->sospA[d]++;
     pthread_cond_wait (&p->codaauto[d], &p->lock);
     p->sospA[d]--;
   }
   /* entrata: aggiorna lo stato del ponte */
   p->nauto[d]++;
   pthread_mutex_unlock (&p->lock);
}
```

Esercizio 2 Sol.

```
/*Accesso al ponte di un camion in dir.d: */
void accessocamion(ponte *p, int d){
   pthread_mutex_lock (&p->lock);
   /* controlla le codizioni di accesso:*/
   while((sulponte(*p)==MAX) || (p->ncamion[altra_dir(d)]>0) ||
         (p->nauto[altra_dir(d)]>0) || (p->sospA[altra_dir(d)]>0)) /*priorita` alle auto*/
   {
      p->sospC[d]++;
      pthread_cond_wait (&p->codacamion[d], &p->lock);
      p->sospC[d]--;
   }
   /* entrata: aggiorna lo stato del ponte */
   p->ncamion[d]++;
   pthread_mutex_unlock (&p->lock);
}
```

Esercizio 2 Sol.

```
/* Rilascio del ponte di un auto in direzione d: */
void rilascioauto (ponte *p, int d){
   pthread_mutex_lock (&p->lock);
   /* uscita: aggiorna lo stato del ponte */
   p->nauto[d]--;
   /* risveglio in ordine di priorita` */
   pthread_cond_broadcast (&p->codaauto[altra_dir(d)]);
   pthread_cond_broadcast (&p->codaauto[d]);
   pthread_cond_broadcast (&p->codacamion[altra_dir(d)]);
   pthread_cond_broadcast (&p->codacamion[d]);
   //printf("USCITA: auto in direzione %d\n", d);
   pthread_mutex_unlock (&p->lock);
}
```

Esercizio 2 Sol.

```
/* Rilascio del ponte di un camion in direzione d: */
void rilasciocamion (ponte *p, int d){
   pthread_mutex_lock (&p->lock);
   /* uscita: aggiorna lo stato del ponte */
   p->ncamion[d]--;
   /* risveglio in ordine di priorita` */
   pthread_cond_broadcast (&p->codaauto[altra_dir(d)]);
   pthread_cond_broadcast (&p->codaauto[d]);
   pthread_cond_broadcast (&p->codacamion[altra_dir(d)]);
   pthread_cond_broadcast (&p->codacamion[d]);
   pthread_mutex_unlock (&p->lock);
}
```

Esercizio 2 Sol.

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

ponte p;

void *auto (void *arg) /*codice del thread «auto" */
{
   int d;
   d = atoi((char *)arg); /*assegno la direzione */
   accessoauto (&p, d);
   /* ATTRAVERSAMENTO: */
   printf("Auto in dir %d: sto attraversando..\n", d);
   sleep(1);
   rilascioauto(&p,d);
   return NULL;
}
```

Esercizio 2 Sol.

```
void *camion (void *arg) /*codice del thread "camion" */
{
   int d;
   d = atoi((char *)arg); /*assegno la direzione */
   accessocamion(&p, d);
   sleep(1);
   printf("Camion in dir %d: sto attraversando\n", d);
   rilasciocamion(&p,d);
   return NULL;
}

main (){
   pthread_t th_A[2][MAXT], th_C[2][MAXT];
   int NAD, NAS, NCD, NCS, i;
   void *retval;
   init (&p);
```

Esercizio 2 Sol.

```
/* Creazione threads: */
printf("\nquante auto in direzione dx? ");
scanf("%d", &NAD);
printf("\nquante auto in direzione sn? ");
scanf("%d", &NAS);
printf("\nquanti camion in direzione dx? ");
scanf("%d", &NCD);
printf("\nquanti camion in direzione sn? ");
scanf("%d", &NCS);
/*CREAZIONE CAMION IN DIREZIONE DX */
for (i=0; i<NcD; i++)
   pthread_create (&th_C[dx][i], NULL, camion, "0");
/*CREAZIONE CAMION IN DIREZIONE SN */
for (i=0; i<NGS; i++)
   pthread_create (&th_C[sn][i], NULL, camion, "1");
/*CREAZIONE AUTO IN DIREZIONE DX */
for (i=0; i<NMD; i++)
   pthread_create (&th_A[dx][i], NULL, auto, "0");
/*CREAZIONE AUTO IN DIREZIONE SN */
for (i=0; i<NMS; i++)
   pthread_create (&th_A[sn][i], NULL, auto, "1");
```

Esercizio 2 Sol.

```
  /* Attesa teminazione threads creati: */
   /*ATTESA AUTO IN DIREZIONE DX */
   for (i=0; i<NAD; i++)
      pthread_join(th_A[dx][i], &retval);
   /*ATTESA AUTO IN DIREZIONE SN */
   for (i=0; i<NAS; i++)
      pthread_join(th_A[sn][i], &retval);
   /*ATTESA CAMION IN DIREZIONE DX */
   for (i=0; i<NCD; i++)
      pthread_join(th_C[dx][i], &retval);
   /*ATTESA CAMION IN DIREZIONE SN */
   for (i=0; i<NCS; i++)
      pthread_join(th_C[sn][i], &retval);
   return 0;
}
```

Esercizio 2 Sol.

```
/* definizione funzioni utilita`:*/
int sulponte(ponte p){
   return p.nauto[dx] + p.ncamion[dx] +
    p.nauto[sn]+ p.ncamion[sn];
}

int altra_dir(int d){
   if (d==sn) return dx;
   else return sn;
}
```