POLITECNICO DI MILANO

Esercizi di mutua esclusione e sincronizzazione (2)

Alessandro A. Nacci
alessandro.nacci@polimi.it

ACSO

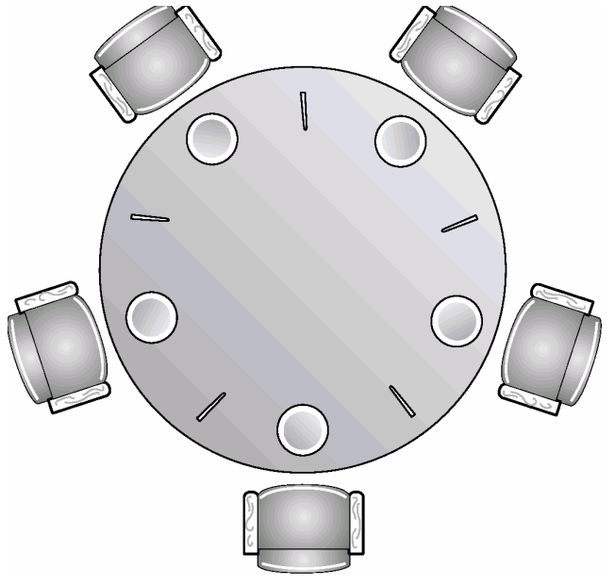
2014/2014


START

Esercizio 13

Il problema dei cinque filosofi

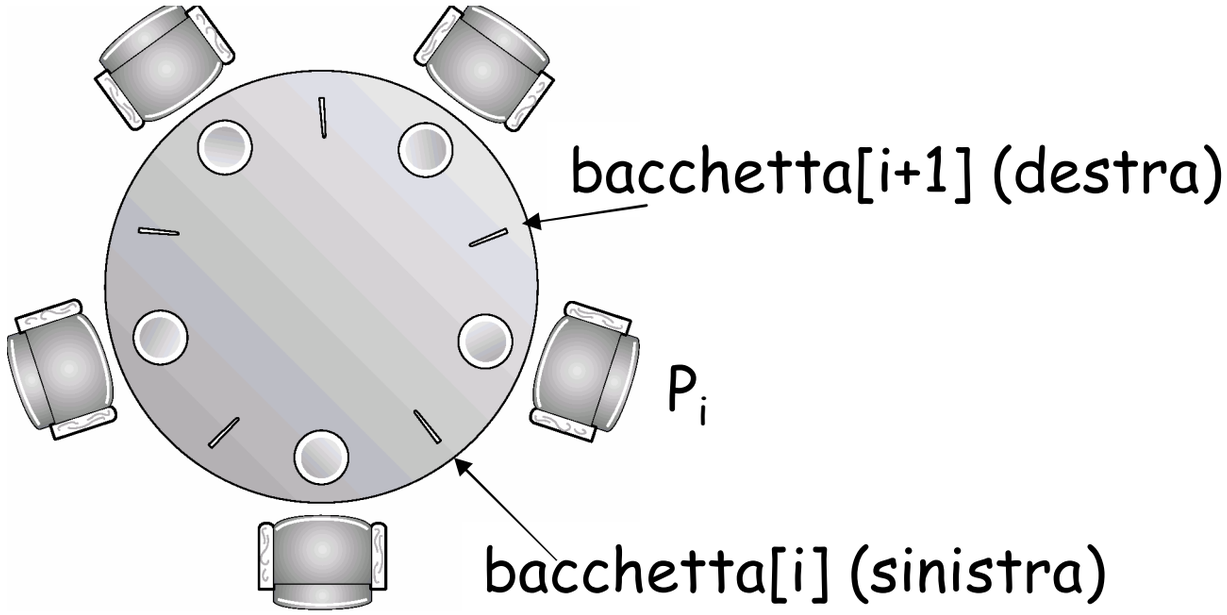
- Cinque filosofi passano la vita pensando e mangiando, attorno ad una tavola rotonda. Al centro della tavola vi è una zuppiera di riso e la tavola è apparecchiata con cinque bacchette.
- Quando un filosofo pensa non interagisce con i colleghi. Quando gli viene fame tenta di impossessarsi delle bacchette che stanno alla sua destra ed alla sua sinistra. Il filosofo può prendere una sola bacchetta per volta.
- Quando un filosofo affamato possiede due bacchette contemporaneamente, mangia. Terminato il pasto, appoggia le bacchette e ricomincia a pensare.



Esercizio 13

Il problema dei cinque filosofi

- Si può rappresentare ciascuna bacchetta con un semaforo.
- Ogni filosofo tenta di afferrare una bacchetta con un'operazione di *wait* e la rilascia eseguendo post



Il problema dei cinque filosofi

Soluzione 1

- Variabili condivise:
  semaphore bacchetta[5]; // tutti gli elementi inizializzati a 1
- Filosofo *i* :

```
do {
   wait(bacchetta[ i ])          <- Prende bacch. sinistra
   wait(bacchetta[(i+1) % 5])    <- Prende bacch. destra
      ...
      mangia                     } Sezione critica
      ...
   post (bacchetta[ i ]);        <- lascia bacch. sinistra
   post (bacchetta[(i+1) % 5]);  <- lascia bacch. destra
      ...
      pensa
      ...
} while (1);
```

Il problema dei cinque filosofi

Soluzione 1: deadlock

se tutti i filosofi hanno fame contemporaneamente e prendono la bacchetta alla loro sinistra nessuno puo' prendere la bacchetta alla sua destra

(DEADLOCK)

Possibile soluzione

Ogni filosofo, dopo aver preso la prima forchetta, verifica se la seconda e' disponibile. In caso contrario posa anche la prima

Il problema dei cinque filosofi

Soluzione 2

In questo esercizio la notazione per i mutex è differente:
lock -> wait
unlock -> post

Dato l'i-esimo filosofo…

```
do {

   while(not.entrambe){
      prendi( i );
      if (b[(i+1) % 5] > 0) {
            prendi( (i+1) % 5 );
            entrambe = true
      } else{
            posa( i );
      }
   }

            ...
            mangia
            ...

      posa( i );
      posa( (i+1) % 5 );
      entrambe = false;

            ...
            pensa
            ...
} while (1);
```

Nel caso del filosofo #5, con %5 prendo la prima forchetta (è una tavola rotonda!)

```
void prendi(int i){
            wait( &mutex );
            wait( &bacchetta[ i ] )
            b[ i ] --;
            signal( &mutex);
}


void posa(int i){
            wait( &mutex );
            b[ i ] ++;
            post( &bacchetta[ i ] );
            post( &mutex );
}
```

b[5] array di interi condiviso che "replica" il contenuto di bacchetta[5]

Il problema dei cinque filosofi

Soluzione 2: *starvation*

c'e il rischio che tutti i filosofi prendono la bacchetta sinistra,
vedono che la destra e' occupata e posano di nuovo la sinistra.
(STARVATION)

Possibile soluzione

Ogni filosofo, prima di mangiare si assicura di aver preso entrambe le bacchette

Il problema dei cinque filosofi
Deadlock & Starvation

- Deadlock: insieme di processi bloccati (su una istruzione wait), ognuno dei quali in attesa che si libera una risorsa assegnata in uso esclusivo di un altro processo

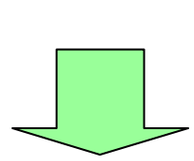
Starvation: insieme di processi in esecuzione che tentano ripetutamente senza successo di accedere ad una risorsa assegnata in uso esclusivo di un altro processo

Il problema dei cinque filosofi

Soluzione 3: inefficiente

```
do {
                    ...
                    pensa
                    ...
          wait(&mutex);
          wait(bacchetta[ i ])
          wait(bacchetta[(i+1) % 5])
                    ...
                    mangia
                                        ...
          post(bacchetta[ i ]);
          post(bacchetta[(i+1) % 5]);
          post(&mutex);
                    ...
                    pensa
                    ...
} while (1);
```

Proteggere le
"istruzioni critiche"



1 solo filosofo alla
volta puo' mangiare!!

Il problema dei cinque filosofi

Soluzione 4

Stato di un generico filosofo (si poteva fare con una enum)

```
#define N                5                /* number of philosophers */
#define LEFT             (i+N−1)%N        /* number of i's left neighbor */
#define RIGHT            (i+1)%N          /* number of i's right neighbor */
#define THINKING         0                /* philosopher is thinking */
#define HUNGRY           1                /* philosopher is trying to get forks */
#define EATING           2                /* philosopher is eating */
typedef int semaphore;                    /* semaphores are a special kind of int */
int state[N];                             /* array to keep track of everyone's state */
semaphore mutex = 1;                      /* mutual exclusion for critical regions */
semaphore s[N];                           /* one semaphore per philosopher */

void philosopher(int i)                   /* i: philosopher number, from 0 to N−1 */
{
    while (TRUE) {                        /* repeat forever */
        think( );                         /* philosopher is thinking */
        take_forks(i);                    /* acquire two forks or block */
        eat( );                           /* yum-yum, spaghetti */
        put_forks(i);                     /* put both forks back on table */
    }
}
```

Il problema dei cinque filosofi

Soluzione 4

```
void test(int i) {
        if(state[i]==HUNGRY && state[LEFT]!=EATING && state[RIGHT]!=EATING){
                state[i] = EATING;              // comunico agli altri che inizio a mangiare
                post ( &s[i] );                 // posso mangiare: semaforo +1
        }
}

void take_forks(int i) {
        wait( &mutex );
        state[ i ]=HUNGRY;                      // sto per prendere le forchette
        test(i);
        post ( &mutex );
        wait( &s[ i ] );                        // inizio a mangiare.
}

void put_forks(int i) {
        wait( &mutex );
        state[ i] = THINKING;                   // comunico che ho finito di mangiare
        test(LEFT);                             // semaforo a sinistra +1
        test(RIGHT);                            // semaforo a destra +1
        post ( &mutex );
}
```

Il problema dei cinque filosofi

Soluzione 4: osservazioni

- state[i]=0 i=0,..,4 s[i]=0 i=0,..,4
- Il semaforo e' relativo al filosofo e non alla forchetta
- La funzione test vede se i vicini stanno mangiando. In caso contrario dichiara che inizia a mangiare e alza il semaforo (abilita il filosofo a mangiare)
- Un filosofo in attesa sulla wait in take_forks, sara' sbloccato da un filosofo vicino che esegue test in put_forks

Tratto da: http://wpage.unina.it/lapegna/lucidi/SOmodA/lezione7-bis.pdf

Esercizio 14

Trova l'errore

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

int conto = 0;

void * th_fun1(void *arg){
    conto++;
    printf("th_fun1 ha depositato sul conto - saldo: %d\n", conto);
    return NULL;
}
void * th_fun2(void *arg){
    conto++;
    printf("th_fun2 ha depositato sul conto - saldo: %d\n", conto);
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;

    pthread_create(&th1, NULL, th_fun1, NULL);
    pthread_create(&th2, NULL, th_fun2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

Esercizio 14

Trova l'errore

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

int conto = 0;

void * th_fun1(void *arg){
    conto++;
    printf("th_fun1 ha depositato sul conto - saldo: %d\n", conto);
    return NULL;
}
void * th_fun2(void *arg){
    conto++;
    printf("th_fun2 ha depositato sul conto - saldo: %d\n", conto);
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;

    pthread_create(&th1, NULL, th_fun1, NULL);
    pthread_create(&th2, NULL, th_fun2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

Il problema consiste nel fatto che i due thread operano sulla variabile conto con una sequenza lettura/scrittura (operatore ++) che può rendere inconsistente lo stato della variabile stessa.

COME SI RISOLVE?

Esercizio 14

Trova l'errore

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

int conto = 0;

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void * th_fun1(void *arg){
    int temp,i;
    pthread_mutex_lock(&lock);
    printf( "th_fun1: saldo del conto prima del deposito: %i\n", conto );
    conto++;
    pthread_mutex_unlock(&lock);
    printf( "th_fun1 ha depositato sul conto - saldo: %d\n", conto );
    return NULL;
}
void * th_fun2(void *arg){
    int temp,i;
    pthread_mutex_lock(&lock);
    printf( "th_fun2: saldo del conto prima del deposito: %i\n", conto );
    conto++;
    printf( "th_fun2 ha depositato sul conto - saldo: %d\n", conto );
    pthread_mutex_unlock(&lock);
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;

    pthread_create(&th1, NULL, th_fun1, NULL);
    pthread_create(&th2, NULL, th_fun2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

Esercizio 14

Trova l'errore

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

int conto = 0;

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

void * th_fun1(void *arg){
    int temp,i;
    pthread_mutex_lock(&lock);
    printf( "th_fun1: saldo del conto prima del deposito: %i\n", conto );
    conto++;
    pthread_mutex_unlock(&lock);
    printf( "th_fun1 ha depositato sul conto - saldo: %d\n", conto );
    return NULL;
}
void * th_fun2(void *arg){
    int temp,i;
    pthread_mutex_lock(&lock);
    printf( "th_fun2: saldo del conto prima del deposito: %i\n", conto );
    conto++;
    printf( "th_fun2 ha depositato sul conto - saldo: %d\n", conto );
    pthread_mutex_unlock(&lock);
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;

    pthread_create(&th1, NULL, th_fun1, NULL);
    pthread_create(&th2, NULL, th_fun2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

SI UTILIZZANO I MUTEX!

Esercizio 15

Trova l'errore (2)

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

char carattere = '\x0';

void * th_fun1(void *arg){
    printf("Letto %c\n", carattere);
    return NULL;
}

void * th_fun2(void *arg){
    int i;
    carattere = (rand()%93)+33;
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;

    pthread_create(&th1, NULL, th_fun1, NULL);
    pthread_create(&th2, NULL, th_fun2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

Esercizio 15

Trova l'errore (2)

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

char carattere = '\x0';

void * th_fun1(void *arg){
    printf("Letto %c\n", carattere);
    return NULL;
}

void * th_fun2(void *arg){
    int i;
    carattere = (rand()%93)+33;
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;

    pthread_create(&th1, NULL, th_fun1, NULL);
    pthread_create(&th2, NULL, th_fun2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

Il problema sta nel fatto che *th_fun2* dovrebbe essere eseguita sempre prima di *th_fun1* in modo tale da leggere un carattere valido dal buffer.

COME SI RISOLVE?

Esercizio 15

Trova l'errore (2)

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

char carattere = '\x0';

sem_t semaforo;

void * th_fun1(void *arg){
    int i;
    sem_wait(&semaforo);
    printf("Letto %c\n", carattere);
    return NULL;
}
void * th_fun2(void *arg){
    int i;
    carattere = (rand()%93)+33;
    sem_post(&semaforo);
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;
    srand( time( NULL ) );
    sem_init(&semaforo, 0, 0);

    pthread_create(&th1, NULL, th_fun1, NULL);
    pthread_create(&th2, NULL, th_fun2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

Esercizio 15

Trova l'errore (2)

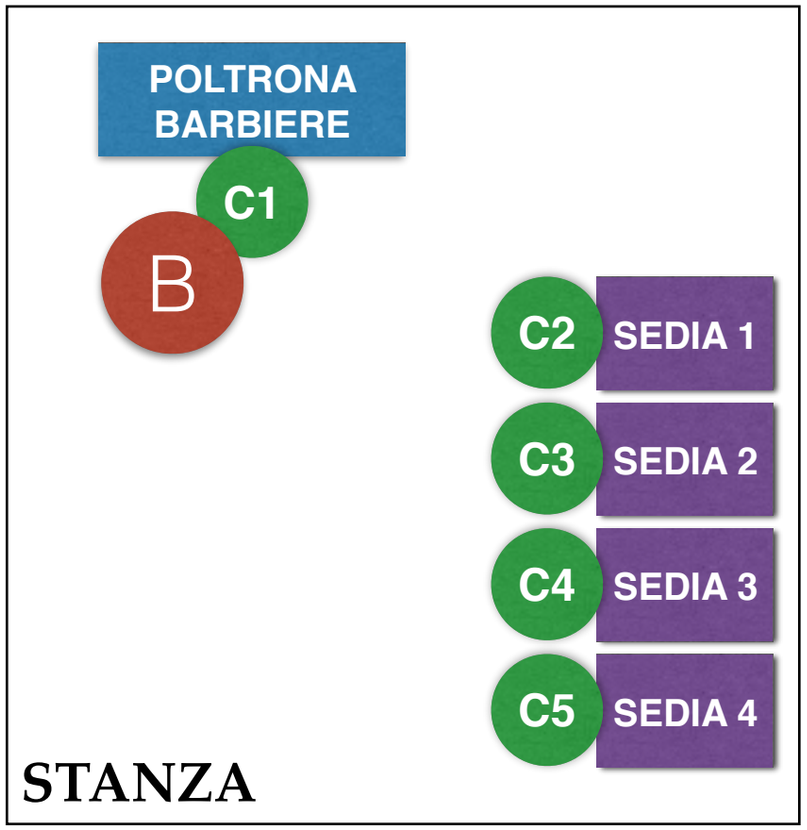
```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

char carattere = '\x0';

sem_t semaforo;

void * th_fun1(void *arg){
    int i;
    sem_wait(&semaforo);
    printf("Letto %c\n", carattere);
    return NULL;
}
void * th_fun2(void *arg){
    int i;
    carattere = (rand()%93)+33;
    sem_post(&semaforo);
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;
    srand( time( NULL ) );
    sem_init(&semaforo, 0, 0);

    pthread_create(&th1, NULL, th_fun1, NULL);
    pthread_create(&th2, NULL, th_fun2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

SI UTILIZZANO I SEMAFORI!

Esercizio 16
Dal barbiere

Negozio del barbiere: il barbiere dorme fino a che non ci sono clienti: all'arrivo di un cliente esso può:

1. svegliare il barbiere (nel caso stesse dormendo)
2. sedersi ed aspettare che il barbiere abbia finito con il cliente che sta radendo al momento
3. se tutte le sedie della sala di attesa sono occupate, il cliente se ne va



Implementare un programma che simula il comportamento appena descritto.

Si fa l'ipotesi che la particolare implementazione dei semafori sblocchi i thread secondo l'ordine di chiamata della *sem_wait*.

Esercizio 16
Dal barbiere

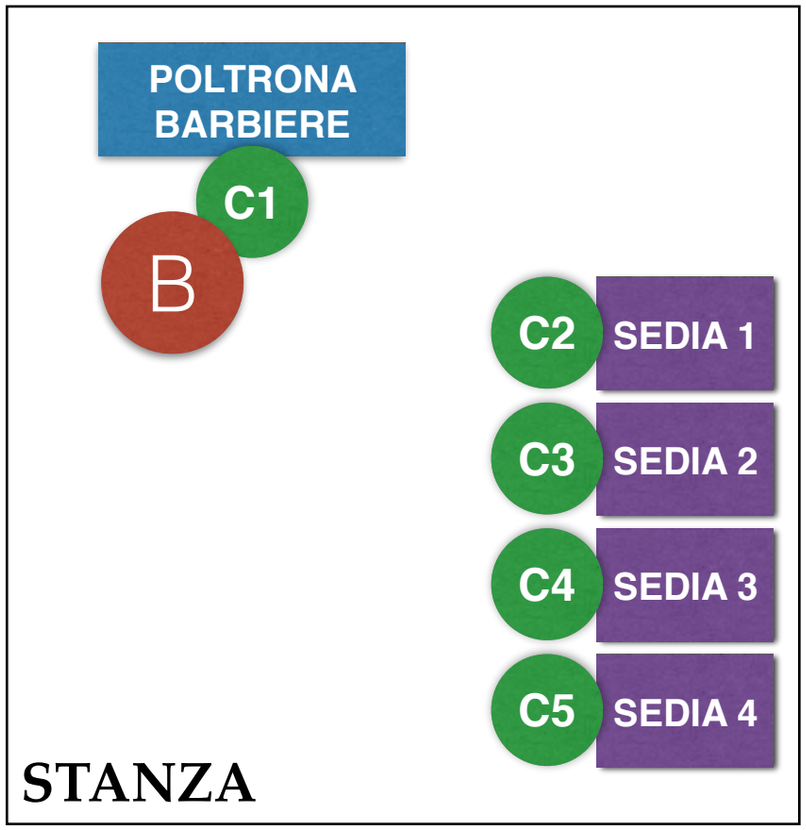
Ci è utile avere due tipi di notifiche:

1. Quando un cliente arriva
2. Quando il barbiere è disponibile

Inoltre ci serve avere:

1. Sapere quante sedie abbiamo
2. Sapere il numero di clienti nella nostra simulazione
3. Sapere quante sedie sono disponibili

Infine, dobbiamo tener presente che la stanza è una risorsa condivisa tra più attori.



Esercizio 16
Dal barbiere

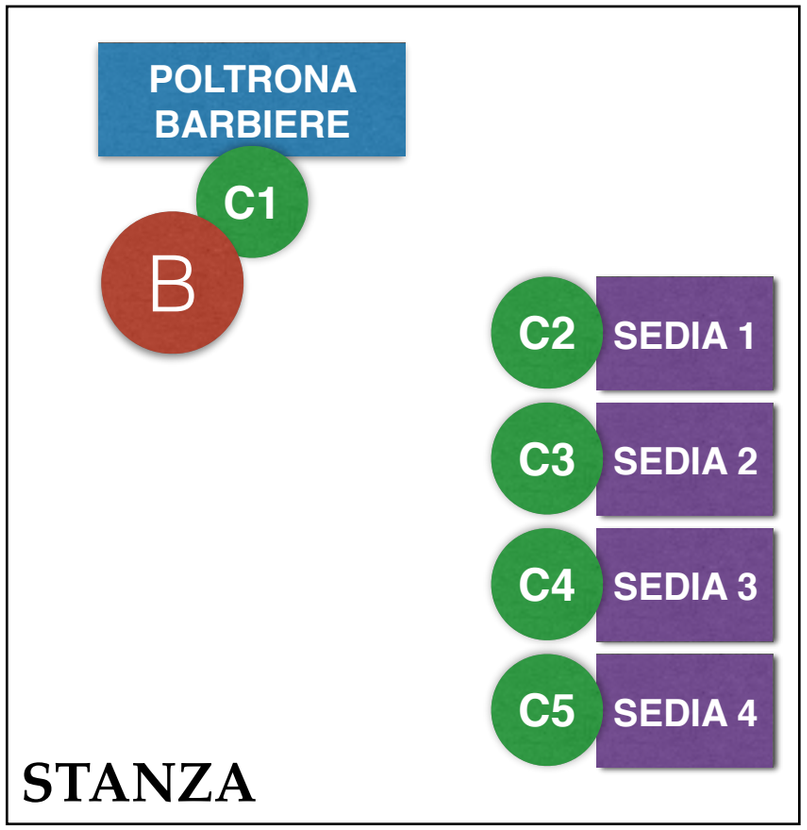
Ci è utile avere due tipi di notifiche:

1. Quando un cliente arriva
2. Quando il barbiere è disponibile

Inoltre ci serve avere:

1. Sapere quante sedie abbiamo
2. Sapere il numero di clienti nella nostra simulazione
3. Sapere quante sedie sono disponibili

Infine, dobbiamo tener presente che la stanza è una risorsa condivisa tra più attori.




```c
#define NUMERO_DI_SEDIE 5
#define NUMERO_DI_CLIENTI 20

sem_t sem_barbiere_disponibile, sem_cliente_arrivato;
pthread_mutex_t stanza = PTHREAD_MUTEX_INITIALIZER;
int sedie_disponibili = NUMERO_DI_SEDIE;
```

Dal barbiere
Il barbiere

```
void * barbiere( void * arg )
{
    while( 1 )
    {
        //IL BARBIERE DORME ASPETTANDO L'ARRIVO DI UN CLIENTE
        sem_wait( &sem_cliente_arrivato );
        //IL BARBIERE SI SVEGLIA E FA ACCOMODARE IL CLIENTE
        sleep(1) /* TEMPO DI SERVIZIO */
        sem_post( &sem_barbiere_disponibile );
        //IL BARBIERE E' PRONTO A FARE LA BARBA
    }
    return NULL;
}
```

Dal barbiere
Il cliente

```
void * cliente( void * arg )
{
    pthread_mutex_lock( &stanza );
    if( sedie_disponibili > 0 )
    {
        //IL CLIENTE SI SIEDE IN SALA D'ATTESA
        sedie_disponibili--;
        pthread_mutex_unlock( &stanza );
        //IL CLIENTE AVVISA IL BARBIERE DELLA SUA PRESENZA
        sem_post( &sem_cliente_arrivato );
        //IL CLIENTE ASPETTA CHE IL BARBIERE SIA DISPONIBILE
        sem_wait( &sem_barbiere_disponibile );
        pthread_mutex_lock( &stanza );
        //IL CLIENTE SI ALZA DALLA SEDIA E SI ACCOMODA SULLA POLTRONA DEL BARBIERE
        sedie_disponibili++;
        pthread_mutex_unlock( &stanza );
        //FINALMENTE IL CLIENTE RIESCE A FARSI LA BARBA
    }
    else
    {
        //IL CLIENTE ESCE DAL NEGOZIO DEL BARBIERE
        pthread_mutex_unlock( &stanza );
    }
    return NULL;
}
```

Dal barbiere
Il main()

```
int main( int argc, char **argv )
{
    pthread_t th_clienti[ NUMERO_DI_CLIENTI ];
    pthread_t th_barbiere;
    int i;

    sem_init( &sem_barbiere_disponibile, 0, 0 );
    sem_init( &sem_cliente_arrivato, 0, 0 );

    pthread_create( &th_barbiere, NULL, barbiere, NULL );
    for( i = 0; i < NUMERO_DI_CLIENTI; i++ )
        pthread_create( &th_clienti[ i ], NULL, cliente, NULL );

    for( i = 0; i < NUMERO_DI_CLIENTI; i++ )
        pthread_join( th_clienti[ i ], NULL );

    pthread_join( th_barbiere, NULL );

    return 0;
}
```

Esercizio 17

Sequenze di esecuzione

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutexA = PTHREAD_MUTEX_INITIALIZER;

void * th_fun1(void *arg){
    pthread_mutex_lock(&mutexA);
    printf("th_fun1 - 1\n");
    pthread_mutex_unlock(&mutexA);
    printf("th_fun1 - 2\n");
    return NULL;
}
void * th_fun2(void *arg){
    pthread_mutex_lock(&mutexA);
    printf("th_fun2 - 1\n");
    printf("th_fun2 - 2\n");
    pthread_mutex_unlock(&mutexA);
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;

    pthread_create(&th1, NULL, th_fun1, NULL);
    pthread_create(&th2, NULL, th_fun2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

Esercizio 17

Sequenze di esecuzione

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutexA = PTHREAD_MUTEX_INITIALIZER;

void * th_fun1(void *arg){
    pthread_mutex_lock(&mutexA);
    printf("th_fun1 - 1\n");
    pthread_mutex_unlock(&mutexA);
    printf("th_fun1 - 2\n");
    return NULL;
}
void * th_fun2(void *arg){
    pthread_mutex_lock(&mutexA);
    printf("th_fun2 - 1\n");
    printf("th_fun2 - 2\n");
    pthread_mutex_unlock(&mutexA);
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;

    pthread_create(&th1, NULL, th_fun1, NULL);
    pthread_create(&th2, NULL, th_fun2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

Dato il programma, stampare tutte i possibili output, supponendo che l'istruzione printf di stampa a schermo sia atomica (ciò supponendo che una volta iniziata la stampa dell'argomento di una printf) essa termini prima che nel sistema venga eseguita un'altra direttiva *printf*.

Esercizio 17

Sequenze di esecuzione

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutexA = PTHREAD_MUTEX_INITIALIZER;

void * th_fun1(void *arg){
    pthread_mutex_lock(&mutexA);
    printf("th_fun1 - 1\n");
    pthread_mutex_unlock(&mutexA);
    printf("th_fun1 - 2\n");
    return NULL;
}
void * th_fun2(void *arg){
    pthread_mutex_lock(&mutexA);
    printf("th_fun2 - 1\n");
    printf("th_fun2 - 2\n");
    pthread_mutex_unlock(&mutexA);
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;

    pthread_create(&th1, NULL, th_fun1, NULL);
    pthread_create(&th2, NULL, th_fun2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

1. *th1* acuiqsice il lock sul `mutexA` prima di *th2*: viene stampato “th_fun1 - 1”. “th_fun2 - 1” e “th_fun2 - 2” vengono sicuramente stampati dopo “th_fun1 - 1” e in un’orine imprecisato rispetto a “th_fun1 - 2”. In poche parole tutte le sequenze sono valide purchè “th_fun1 - 1” venga stampato prima di “th_fun2 - 1” e “th_fun2 - 2”. Le possibili sequenze di esecuzione in questo caso sono:

th_fun1 - 1 th_fun1 - 2 th_fun2 - 1 th_fun2 - 2
th_fun1 - 1 th_fun2 - 1 th_fun1 - 2 th_fun2 - 2
th_fun1 - 1 th_fun2 - 1 th_fun2 - 2 th_fun1 - 2

Esercizio 17

Sequenze di esecuzione

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutexA = PTHREAD_MUTEX_INITIALIZER;

void * th_fun1(void *arg){
    pthread_mutex_lock(&mutexA);
    printf("th_fun1 - 1\n");
    printf("th_fun1 - 2\n");
    return NULL;
}
void * th_fun2(void *arg){
    pthread_mutex_lock(&mutexA);
    printf("th_fun2 - 1\n");
    printf("th_fun2 - 2\n");
    pthread_mutex_unlock(&mutexA);
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;

    pthread_create(&th1, NULL, th_fun1, NULL);
    pthread_create(&th2, NULL, th_fun2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

2. *th2* acuiqsice il lock sul `mutexA` prima di *th1*: vengono stampati “th_fun2 - 1” e “th_fun2 - 2”. “th_fun1 - 1” viene sicuramente stampato dopo di essi; “th_fun1 - 2” segue. In questo caso esiste solo una sequenza di esecuzione:

```
th_fun2 - 1
th_fun2 - 2
th_fun1 - 1
th_fun1 - 2
```

In parole povere, il mutex serve per evitare che “th_fun1 - 1” venga stampata tra “th_fun2 - 1” e “th_fun2 - 2”.

Esercizio 18

Race conditions

Identificare le race-conditions presenti nel seguente programma e modificare quest'ultimo in modo tale da risolvere il seguente problema:

Due persone (P1 e P2) diverse devono maneggiare due conti in banca (C1 e C2): P1 deve depositare 10 in C1 e prelevare 20 da C2; P2 deve depositare 40 in C1 e 20 in C2. Notare che i due conti in banca devono poter essere maneggiati separatamente.

Risolvere il problema sostituendo a '?' `lock` o `unlock` e a '____' il nome del mutex relativo.

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutexA = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutexB = PTHREAD_MUTEX_INITIALIZER;

int conto1 = 100;
int conto2 = 100;

void * persona1(void *arg){
    pthread_mutex_?(&____);
    conto1 += 10;
    pthread_mutex_?(&____);
    pthread_mutex_?(&____);
    conto2 -= 20;
    pthread_mutex_?(&____);
    return NULL;
}

void * persona2(void *arg){
    pthread_mutex_lock(&____);
    conto2 += 20;
    pthread_mutex_unlock(&____);
    pthread_mutex_?(&____);
    conto1 += 40;
    pthread_mutex_?(&____);
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;

    pthread_create(&th1, NULL, persona1, NULL);
    pthread_create(&th2, NULL, persona2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

Esercizio 18

Race conditions

Identificare le race-conditions presenti nel seguente programma e modificare quest'ultimo in modo tale da risolvere il seguente problema:

Due persone (P1 e P2) diverse devono maneggiare due conti in banca (C1 e C2): P1 deve depositare 10 in C1 e prelevare 20 da C2; P2 deve depositare 40 in C1 e 20 in C2. Notare che i due conti in banca devono poter essere maneggiati separatamente.

Risolvere il problema sostituendo a '?' `lock` o `unlock` e a '____' il nome del mutex relativo.

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutexA = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutexB = PTHREAD_MUTEX_INITIALIZER;

int conto1 = 100;
int conto2 = 100;

void * persona1(void *arg){
    pthread_mutex_?(&____); mutexA
    conto1 += 10;
    pthread_mutex_?(&____);
    pthread_mutex_?(&____);
    conto2 -= 20;
    pthread_mutex_?(&____);
    return NULL;
}
```

```c
void * persona2(void *arg){
    pthread_mutex_lock(&____);
    conto2 += 20;
    pthread_mutex_unlock(&____);
    pthread_mutex_?(&____);
    conto1 += 40;
    pthread_mutex_?(&____);
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;

    pthread_create(&th1, NULL, persona1, NULL);
    pthread_create(&th2, NULL, persona2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

Esercizio 18

Race conditions

Identificare le race-conditions presenti nel seguente programma e modificare quest'ultimo in modo tale da risolvere il seguente problema:

Due persone (P1 e P2) diverse devono maneggiare due conti in banca (C1 e C2): P1 deve depositare 10 in C1 e prelevare 20 da C2; P2 deve depositare 40 in C1 e 20 in C2. Notare che i due conti in banca devono poter essere maneggiati separatamente.

Risolvere il problema sostituendo a '?' `lock` o `unlock` e a '____' il nome del mutex relativo.

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutexA = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutexB = PTHREAD_MUTEX_INITIALIZER;

int conto1 = 100;
int conto2 = 100;

void * persona1(void *arg){
    pthread_mutex_?(&____); mutexA
    conto1 += 10;
    pthread_mutex_?(&____); mutexA
    pthread_mutex_?(&____);
    conto2 -= 20;
    pthread_mutex_?(&____);
    return NULL;
}
```

```c
void * persona2(void *arg){
    pthread_mutex_lock(&____);
    conto2 += 20;
    pthread_mutex_unlock(&____);
    pthread_mutex_?(&____);
    conto1 += 40;
    pthread_mutex_?(&____);
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;

    pthread_create(&th1, NULL, persona1, NULL);
    pthread_create(&th2, NULL, persona2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

Esercizio 18

Race conditions

Identificare le race-conditions presenti nel seguente programma e modificare quest'ultimo in modo tale da risolvere il seguente problema:

Due persone (P1 e P2) diverse devono maneggiare due conti in banca (C1 e C2): P1 deve depositare 10 in C1 e prelevare 20 da C2; P2 deve depositare 40 in C1 e 20 in C2. Notare che i due conti in banca devono poter essere maneggiati separatamente.

Risolvere il problema sostituendo a '?' `lock` o `unlock` e a '____' il nome del mutex relativo.

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutexA = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutexB = PTHREAD_MUTEX_INITIALIZER;

int conto1 = 100;
int conto2 = 100;

void * persona1(void *arg){
    pthread_mutex_?(&____); mutexA
    conto1 += 10;
    pthread_mutex_?(&____); mutexA
    pthread_mutex_?(&____); mutexB
    conto2 -= 20;
    pthread_mutex_?(&____);
    return NULL;
}
```

```c
void * persona2(void *arg){
    pthread_mutex_lock(&____);
    conto2 += 20;
    pthread_mutex_unlock(&____);
    pthread_mutex_?(&____);
    conto1 += 40;
    pthread_mutex_?(&____);
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;

    pthread_create(&th1, NULL, persona1, NULL);
    pthread_create(&th2, NULL, persona2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

Esercizio 18

Race conditions

Identificare le race-conditions presenti nel seguente programma e modificare quest'ultimo in modo tale da risolvere il seguente problema:

Due persone (P1 e P2) diverse devono maneggiare due conti in banca (C1 e C2): P1 deve depositare 10 in C1 e prelevare 20 da C2; P2 deve depositare 40 in C1 e 20 in C2. Notare che i due conti in banca devono poter essere maneggiati separatamente.

Risolvere il problema sostituendo a '?' `lock` o `unlock` e a '____' il nome del mutex relativo.

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutexA = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutexB = PTHREAD_MUTEX_INITIALIZER;

int conto1 = 100;
int conto2 = 100;

void * persona1(void *arg){
    pthread_mutex_?(&____); mutexA
    conto1 += 10;
    pthread_mutex_?(&____); mutexA
    pthread_mutex_?(&____); mutexB
    conto2 -= 20;
    pthread_mutex_?(&____); mutexB
    return NULL;
}
```

```c
void * persona2(void *arg){
    pthread_mutex_lock(&____);
    conto2 += 20;
    pthread_mutex_unlock(&____);
    pthread_mutex_?(&____);
    conto1 += 40;
    pthread_mutex_?(&____);
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;

    pthread_create(&th1, NULL, persona1, NULL);
    pthread_create(&th2, NULL, persona2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

Esercizio 18

Race conditions

Identificare le race-conditions presenti nel seguente programma e modificare quest'ultimo in modo tale da risolvere il seguente problema:

Due persone (P1 e P2) diverse devono maneggiare due conti in banca (C1 e C2): P1 deve depositare 10 in C1 e prelevare 20 da C2; P2 deve depositare 40 in C1 e 20 in C2. Notare che i due conti in banca devono poter essere maneggiati separatamente.

Risolvere il problema sostituendo a '?' `lock` o `unlock` e a '____' il nome del mutex relativo.

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutexA = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutexB = PTHREAD_MUTEX_INITIALIZER;

int conto1 = 100;
int conto2 = 100;

void * persona1(void *arg){
    pthread_mutex_?(&____); mutexA
    conto1 += 10;
    pthread_mutex_?(&____); mutexA
    pthread_mutex_?(&____); mutexB
    conto2 -= 20;
    pthread_mutex_?(&____); mutexB
    return NULL;
}
```

```c
void * persona2(void *arg){
    pthread_mutex_lock(&____);mutexB
    conto2 += 20;
    pthread_mutex_unlock(&____);
    pthread_mutex_?(&____);
    conto1 += 40;
    pthread_mutex_?(&____);
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;

    pthread_create(&th1, NULL, persona1, NULL);
    pthread_create(&th2, NULL, persona2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

Esercizio 18

Race conditions

Identificare le race-conditions presenti nel seguente programma e modificare quest'ultimo in modo tale da risolvere il seguente problema:

Due persone (P1 e P2) diverse devono maneggiare due conti in banca (C1 e C2): P1 deve depositare 10 in C1 e prelevare 20 da C2; P2 deve depositare 40 in C1 e 20 in C2. Notare che i due conti in banca devono poter essere maneggiati separatamente.

Risolvere il problema sostituendo a '?' `lock` o `unlock` e a '____' il nome del mutex relativo.

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutexA = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutexB = PTHREAD_MUTEX_INITIALIZER;

int conto1 = 100;
int conto2 = 100;

void * persona1(void *arg){
    pthread_mutex_?(&____); mutexA
    conto1 += 10;
    pthread_mutex_?(&____); mutexA
    pthread_mutex_?(&____); mutexB
    conto2 -= 20;
    pthread_mutex_?(&____); mutexB
    return NULL;
}

void * persona2(void *arg){
    pthread_mutex_lock(&____); mutexB
    conto2 += 20;
    pthread_mutex_unlock(&____); mutexB
    pthread_mutex_?(&____);
    conto1 += 40;
    pthread_mutex_?(&____);
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;

    pthread_create(&th1, NULL, persona1, NULL);
    pthread_create(&th2, NULL, persona2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

Esercizio 18

Race conditions

Identificare le race-conditions presenti nel seguente programma e modificare quest'ultimo in modo tale da risolvere il seguente problema:

Due persone (P1 e P2) diverse devono maneggiare due conti in banca (C1 e C2): P1 deve depositare 10 in C1 e prelevare 20 da C2; P2 deve depositare 40 in C1 e 20 in C2. Notare che i due conti in banca devono poter essere maneggiati separatamente.

Risolvere il problema sostituendo a '?' `lock` o `unlock` e a '____' il nome del mutex relativo.

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutexA = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutexB = PTHREAD_MUTEX_INITIALIZER;

int conto1 = 100;
int conto2 = 100;

void * persona1(void *arg){
    pthread_mutex_?(&____); mutexA
    conto1 += 10;
    pthread_mutex_?(&____); mutexA
    pthread_mutex_?(&____); mutexB
    conto2 -= 20;
    pthread_mutex_?(&____); mutexB
    return NULL;
}
```

```c
void * persona2(void *arg){
    pthread_mutex_lock(&____);mutexB
    conto2 += 20;
    pthread_mutex_unlock(&____);mutexB
    pthread_mutex_?(&____);mutexA
    conto1 += 40;
    pthread_mutex_?(&____);
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;

    pthread_create(&th1, NULL, persona1, NULL);
    pthread_create(&th2, NULL, persona2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

Esercizio 18

Race conditions

Identificare le race-conditions presenti nel seguente programma e modificare quest'ultimo in modo tale da risolvere il seguente problema:

Due persone (P1 e P2) diverse devono maneggiare due conti in banca (C1 e C2): P1 deve depositare 10 in C1 e prelevare 20 da C2; P2 deve depositare 40 in C1 e 20 in C2. Notare che i due conti in banca devono poter essere maneggiati separatamente.

Risolvere il problema sostituendo a '?' `lock` o `unlock` e a '____' il nome del mutex relativo.

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutexA = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutexB = PTHREAD_MUTEX_INITIALIZER;

int conto1 = 100;
int conto2 = 100;

void * persona1(void *arg){
    pthread_mutex_?(&____); mutexA
    conto1 += 10;
    pthread_mutex_?(&____); mutexA
    pthread_mutex_?(&____); mutexB
    conto2 -= 20;
    pthread_mutex_?(&____); mutexB
    return NULL;
}

void * persona2(void *arg){
    pthread_mutex_lock(&____); mutexB
    conto2 += 20;
    pthread_mutex_unlock(&____); mutexB
    pthread_mutex_?(&____); mutexA
    conto1 += 40;
    pthread_mutex_?(&____); mutexA
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;

    pthread_create(&th1, NULL, persona1, NULL);
    pthread_create(&th2, NULL, persona2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

Esercizio 18
Race conditions

La race condition esiste per il fatto che entrambi i conti correnti sono modificati in parallelo da entrambe le persone. Per correggere il problema è necessario proteggere l'accesso ai conti in banca con due mutex, uno per ogni conto corrente.

```c
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t mutexA = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mutexB = PTHREAD_MUTEX_INITIALIZER;

int conto1 = 100;
int conto2 = 100;

void * persona1(void *arg){
    pthread_mutex_?(&____); mutexA
    conto1 += 10;
    pthread_mutex_?(&____); mutexA
    pthread_mutex_?(&____); mutexB
    conto2 -= 20;
    pthread_mutex_?(&____); mutexB
    return NULL;
}
```

```c
void * persona2(void *arg){
    pthread_mutex_lock(&____);mutexB
    conto2 += 20;
    pthread_mutex_unlock(&____);mutexB
    pthread_mutex_?(&____);mutexA
    conto1 += 40;
    pthread_mutex_?(&____);mutexA
    return NULL;
}

int main(int argc, char * argv[]){
    pthread_t th1, th2;

    pthread_create(&th1, NULL, persona1, NULL);
    pthread_create(&th2, NULL, persona2, NULL);
    pthread_join(th1, NULL);
    pthread_join(th2, NULL);

    return 0;
}
```

Temi di Esame

ATTENZIONE!

Il testo degli esercizi sarà poco visibile

SCARICALO ORA

http://goo.gl/sBdVTT

Temi d’Esame

1 marzo 2011

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto indicando lo stato di esistenza della variabile locale, così:

- se la variabile certamente esiste, si scriva ESISTE;
- se certamente non esiste, si scriva NON ESISTE;
- e se può essere esistente o inesistente, si
- scriva PUÒ ESISTERE.

Ogni casella della tabella va riempita in uno dei tre modi (non va lasciata vuota).
Si badi bene alla colonna condizione: con subito dopo statement X si chiede lo stato che la variabile assume tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**variabile locale**	
	luggage in **V1**	*luggage* in **V2**
subito dopo stat. **C**		
subito dopo stat. **D** in **V2**		
subito dopo stat. **E**		

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

void * gatekeeper (void * arg) {                    /* funzione di thread   */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

void * visitor (void * bag) {                       /* funzione di thread   */

    int luggage = 0;                                /* variabile locale     */

    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

int main (int argc, char * argv [ ]) {              /* thread principale    */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto indicando lo stato di esistenza della variabile locale, così:

- se la variabile certamente esiste, si scriva ESISTE;
- se certamente non esiste, si scriva NON ESISTE;
- e se può essere esistente o inesistente, si
- scriva PUÒ ESISTERE.

Ogni casella della tabella va riempita in uno dei tre modi (non va lasciata vuota).
Si badi bene alla colonna condizione: con subito dopo statement X si chiede lo stato che la variabile assume tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**variabile locale**	
	luggage in **V1**	*luggage* in **V2**
subito dopo stat. **C**	*PUÒ ESISTERE*	
subito dopo stat. **D** in **V2**		
subito dopo stat. **E**		

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

void * gatekeeper (void * arg) {                    /* funzione di thread  */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

void * visitor (void * bag) {                       /* funzione di thread  */

    int luggage = 0;                                /* variabile locale    */

    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

int main (int argc, char * argv [ ]) {              /* thread principale   */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto indicando lo stato di esistenza della variabile locale, così:

- se la variabile certamente esiste, si scriva ESISTE;
- se certamente non esiste, si scriva NON ESISTE;
- e se può essere esistente o inesistente, si
- scriva PUÒ ESISTERE.

Ogni casella della tabella va riempita in uno dei tre modi (non va lasciata vuota).
Si badi bene alla colonna condizione: con subito dopo statement X si chiede lo stato che la variabile assume tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**variabile locale**	
	luggage in **V1**	*luggage* in **V2**
subito dopo stat. **C**	*PUÒ ESISTERE*	*PUÒ ESISTERE*
subito dopo stat. **D** in **V2**		
subito dopo stat. **E**		

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

void * gatekeeper (void * arg) {                    /* funzione di thread  */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

void * visitor (void * bag) {                       /* funzione di thread  */

    int luggage = 0;                                /* variabile locale    */

    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

int main (int argc, char * argv [ ]) {              /* thread principale   */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto indicando lo stato di esistenza della variabile locale, così:

- se la variabile certamente esiste, si scriva ESISTE;
- se certamente non esiste, si scriva NON ESISTE;
- e se può essere esistente o inesistente, si
- scriva PUÒ ESISTERE.

Ogni casella della tabella va riempita in uno dei tre modi (non va lasciata vuota).
Si badi bene alla colonna condizione: con subito dopo statement X si chiede lo stato che la variabile assume tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**variabile locale**	
	luggage in **V1**	*luggage* in **V2**
subito dopo stat. **C**	*PUÒ ESISTERE*	*PUÒ ESISTERE*
subito dopo stat. **D** in **V2**	*PUÒ ESISTERE*	
subito dopo stat. **E**		

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

void * gatekeeper (void * arg) {                    /* funzione di thread   */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

void * visitor (void * bag) {                       /* funzione di thread   */

    int luggage = 0;                                /* variabile locale     */

    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

int main (int argc, char * argv [ ]) {              /* thread principale    */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto indicando lo stato di esistenza della variabile locale, così:

- se la variabile certamente esiste, si scriva ESISTE;
- se certamente non esiste, si scriva NON ESISTE;
- e se può essere esistente o inesistente, si
- scriva PUÒ ESISTERE.

Ogni casella della tabella va riempita in uno dei tre modi (non va lasciata vuota).
Si badi bene alla colonna condizione: con subito dopo statement X si chiede lo stato che la variabile assume tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**variabile locale**	
	luggage in **V1**	*luggage* in **V2**
subito dopo stat. **C**	*PUÒ ESISTERE*	*PUÒ ESISTERE*
subito dopo stat. **D** in **V2**	*PUÒ ESISTERE*	*ESISTE*
subito dopo stat. **E**		

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

void * gatekeeper (void * arg) {                    /* funzione di thread   */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

void * visitor (void * bag) {                       /* funzione di thread   */

    int luggage = 0;                                /* variabile locale     */

    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

int main (int argc, char * argv [ ]) {              /* thread principale    */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto indicando lo stato di esistenza della variabile locale, così:

- se la variabile certamente esiste, si scriva ESISTE;
- se certamente non esiste, si scriva NON ESISTE;
- e se può essere esistente o inesistente, si
- scriva PUÒ ESISTERE.

Ogni casella della tabella va riempita in uno dei tre modi (non va lasciata vuota).
Si badi bene alla colonna condizione: con subito dopo statement X si chiede lo stato che la variabile assume tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**variabile locale**	
	luggage in **V1**	*luggage* in **V2**
subito dopo stat. **C**	*PUÒ ESISTERE*	*PUÒ ESISTERE*
subito dopo stat. **D** in **V2**	*PUÒ ESISTERE*	*ESISTE*
subito dopo stat. **E**	*NON ESISTE*	

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

void * gatekeeper (void * arg) {                    /* funzione di thread   */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

void * visitor (void * bag) {                       /* funzione di thread   */

    int luggage = 0;                                /* variabile locale     */

    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

int main (int argc, char * argv [ ]) {              /* thread principale    */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto indicando lo stato di esistenza della variabile locale, così:

- se la variabile certamente esiste, si scriva ESISTE;
- se certamente non esiste, si scriva NON ESISTE;
- e se può essere esistente o inesistente, si
- scriva PUÒ ESISTERE.

Ogni casella della tabella va riempita in uno dei tre modi (non va lasciata vuota).
Si badi bene alla colonna condizione: con subito dopo statement X si chiede lo stato che la variabile assume tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**variabile locale**	
	luggage in **V1**	*luggage* in **V2**
subito dopo stat. **C**	*PUÒ ESISTERE*	*PUÒ ESISTERE*
subito dopo stat. **D** in **V2**	*PUÒ ESISTERE*	*ESISTE*
subito dopo stat. **E**	*NON ESISTE*	*PUÒ ESISTERE*

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

void * gatekeeper (void * arg) {                    /* funzione di thread   */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

void * visitor (void * bag) {                       /* funzione di thread   */

    int luggage = 0;                                /* variabile locale     */

    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

int main (int argc, char * argv [ ]) {              /* thread principale    */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto, indicando i valori delle variabili globali (sempre esistenti)

Il valore della variabile va indicato così:

- intero,carattere,stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quindi
- il semaforo può avere valore positivo o nullo (non negativo)

Si badi bene alla colonna condizione: con subito dopo statement X si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	variabili globali	
	inside	*served*
subito dopo stat. **A**		
subito dopo stat. **B**		
subito dopo stat. **C**		
subito dopo stat. **D** in **V1**		

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

void * gatekeeper (void * arg) {                    /* funzione di thread  */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

void * visitor (void * bag) {                       /* funzione di thread  */

    int luggage = 0;                                /* variabile locale    */

    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

int main (int argc, char * argv [ ]) {              /* thread principale   */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto, indicando i valori delle variabili globali (sempre esistenti)

Il valore della variabile va indicato così:

- intero,carattere,stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quindi
- il semaforo può avere valore positivo o nullo (non negativo)

Si badi bene alla colonna condizione: con subito dopo statement X si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**variabili globali**	
	inside	*served*
subito dopo stat. **A**	*0*	
subito dopo stat. **B**		
subito dopo stat. **C**		
subito dopo stat. **D** in **V1**		

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

void * gatekeeper (void * arg) {                    /* funzione di thread  */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

void * visitor (void * bag) {                       /* funzione di thread  */

    int luggage = 0;                                /* variabile locale    */

    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

int main (int argc, char * argv [ ]) {              /* thread principale   */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto, indicando i valori delle variabili globali (sempre esistenti)

Il valore della variabile va indicato così:

- intero,carattere,stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quindi
- il semaforo può avere valore positivo o nullo (non negativo)

Si badi bene alla colonna condizione: con subito dopo statement X si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	variabili globali	
	inside	*served*
subito dopo stat. **A**	*0*	*0*
subito dopo stat. **B**		
subito dopo stat. **C**		
subito dopo stat. **D** in **V1**		

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

void * gatekeeper (void * arg) {                    /* funzione di thread  */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

void * visitor (void * bag) {                       /* funzione di thread  */

    int luggage = 0;                                /* variabile locale    */

    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

int main (int argc, char * argv [ ]) {              /* thread principale   */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto, indicando i valori delle variabili globali (sempre esistenti)

Il valore della variabile va indicato così:

- intero,carattere,stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quindi
- il semaforo può avere valore positivo o nullo (non negativo)

Si badi bene alla colonna condizione: con subito dopo statement X si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	variabili globali	
	inside	*served*
subito dopo stat. **A**	*0*	*0*
subito dopo stat. **B**	*0 / 1*	
subito dopo stat. **C**		
subito dopo stat. **D** in **V1**		

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

void * gatekeeper (void * arg) {                    /* funzione di thread  */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

void * visitor (void * bag) {                       /* funzione di thread  */

    int luggage = 0;                                /* variabile locale    */

    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

int main (int argc, char * argv [ ]) {              /* thread principale   */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto, indicando i valori delle variabili globali (sempre esistenti)

Il valore della variabile va indicato così:

- intero,carattere,stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quindi
- il semaforo può avere valore positivo o nullo (non negativo)

Si badi bene alla colonna condizione: con subito dopo statement X si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	variabili globali	
	inside	*served*
subito dopo stat. **A**	*0*	*0*
subito dopo stat. **B**	*0 / 1*	*1*
subito dopo stat. **C**		
subito dopo stat. **D** in **V1**		

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

void * gatekeeper (void * arg) {                    /* funzione di thread   */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

void * visitor (void * bag) {                       /* funzione di thread   */

    int luggage = 0;                                /* variabile locale     */

    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

int main (int argc, char * argv [ ]) {              /* thread principale    */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto, indicando i valori delle variabili globali (sempre esistenti)

Il valore della variabile va indicato così:

- intero,carattere,stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quindi
- il semaforo può avere valore positivo o nullo (non negativo)

Si badi bene alla colonna condizione: con subito dopo statement X si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**variabili globali**	
	inside	*served*
subito dopo stat. **A**	*0*	*0*
subito dopo stat. **B**	*0 / 1*	*1*
subito dopo stat. **C**	*1 / 2*	
subito dopo stat. **D** in **V1**		

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

void * gatekeeper (void * arg) {                    /* funzione di thread  */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

void * visitor (void * bag) {                       /* funzione di thread  */

    int luggage = 0;                                /* variabile locale    */

    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

int main (int argc, char * argv [ ]) {              /* thread principale   */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto, indicando i valori delle variabili globali (sempre esistenti)

Il valore della variabile va indicato così:

- intero,carattere,stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quindi
- il semaforo può avere valore positivo o nullo (non negativo)

Si badi bene alla colonna condizione: con subito dopo statement X si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**variabili globali**	
	inside	*served*
subito dopo stat. **A**	*0*	*0*
subito dopo stat. **B**	*0 / 1*	*1*
subito dopo stat. **C**	*1 / 2*	*2*
subito dopo stat. **D** in **V1**		

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

void * gatekeeper (void * arg) {                    /* funzione di thread  */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

void * visitor (void * bag) {                       /* funzione di thread  */

    int luggage = 0;                                /* variabile locale    */

    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

int main (int argc, char * argv [ ]) {              /* thread principale   */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto, indicando i valori delle variabili globali (sempre esistenti)

Il valore della variabile va indicato così:

- intero,carattere,stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quindi
- il semaforo può avere valore positivo o nullo (non negativo)

Si badi bene alla colonna condizione: con subito dopo statement X si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	variabili globali	
	inside	*served*
subito dopo stat. **A**	*0*	*0*
subito dopo stat. **B**	*0 / 1*	*1*
subito dopo stat. **C**	*1 / 2*	*2*
subito dopo stat. **D** in **V1**	*0 / 1*	

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

void * gatekeeper (void * arg) {                    /* funzione di thread   */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

void * visitor (void * bag) {                       /* funzione di thread   */

    int luggage = 0;                                /* variabile locale     */

    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

int main (int argc, char * argv [ ]) {              /* thread principale    */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto, indicando i valori delle variabili globali (sempre esistenti)

Il valore della variabile va indicato così:

- intero,carattere,stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quindi
- il semaforo può avere valore positivo o nullo (non negativo)

Si badi bene alla colonna condizione: con subito dopo statement X si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	variabili globali	
	inside	*served*
subito dopo stat. **A**	*0*	*0*
subito dopo stat. **B**	*0 / 1*	*1*
subito dopo stat. **C**	*1 / 2*	*2*
subito dopo stat. **D** in **V1**	*0 / 1*	*0 / 1 / 2*

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

void * gatekeeper (void * arg) {                    /* funzione di thread   */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

void * visitor (void * bag) {                       /* funzione di thread   */

    int luggage = 0;                                /* variabile locale     */

    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

int main (int argc, char * argv [ ]) {              /* thread principale    */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Il sistema può andare in stallo (deadlock). Qui si indichino gli statement dove si bloccano i thread incorsi in stallo (non necessariamente tutti):

GK

V1

V2

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

-------------------------------------------------------------------------
void * gatekeeper (void * arg) {                    /* funzione di thread  */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

-------------------------------------------------------------------------
void * visitor (void * bag) {                       /* funzione di thread  */

    int luggage = 0;                                /* variabile locale    */
    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

-------------------------------------------------------------------------
int main (int argc, char * argv [ ]) {              /* thread principale   */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
-------------------------------------------------------------------------
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Il sistema può andare in stallo (deadlock). Qui si indichino gli statement dove si bloccano i thread incorsi in stallo (non necessariamente tutti):

GK
pthread_mutex_lock *(&door)*

V1

V2

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

void * gatekeeper (void * arg) {                    /* funzione di thread   */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

void * visitor (void * bag) {                       /* funzione di thread   */

    int luggage = 0;                                /* variabile locale     */
    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

int main (int argc, char * argv [ ]) {              /* thread principale    */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Il sistema può andare in stallo (deadlock). Qui si indichino gli statement dove si bloccano i thread incorsi in stallo (non necessariamente tutti):

GK
pthread_mutex_lock *(&door)*

V1
terminato

V2

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

void * gatekeeper (void * arg) {                    /* funzione di thread   */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

void * visitor (void * bag) {                       /* funzione di thread   */

    int luggage = 0;                                /* variabile locale     */
    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

int main (int argc, char * argv [ ]) {              /* thread principale    */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Il sistema può andare in stallo (deadlock). Qui si indichino gli statement dove si bloccano i thread incorsi in stallo (non necessariamente tutti):

GK
pthread_mutex_lock *(&door)*

V1
terminato

V2
sem_wait *(&inside)*

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

void * gatekeeper (void * arg) {                    /* funzione di thread   */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

void * visitor (void * bag) {                       /* funzione di thread   */

    int luggage = 0;                                /* variabile locale     */
    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

int main (int argc, char * argv [ ]) {              /* thread principale    */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di martedì 1 marzo 2011

esercizio n. 1 – thread e parallelismo

Il sistema può andare in stallo (deadlock). Qui si indichino gli statement dove si bloccano i thread incorsi in stallo (non necessariamente tutti):

GK
pthread_mutex_lock *(&door)*

V1
terminato

V2
sem_wait *(&inside)*

Andamento: GK esegue sem_post; V1 esegue sem_wait e termina; V2 effettua il lock di door si blocca su sem_wait; GK si blocca sul lock di door; ora V2 e GK sono in stallo; si possono scambiare i ruoli di V1 e V2.

```
/* variabili globali                                                    */
pthread_mutex_t door = PTHREAD_MUTEX_INITIALIZER;
sem_t inside;
int served = 0;

void * gatekeeper (void * arg) {                    /* funzione di thread   */

    printf ("Service started.\n");                  /* statement A */
    sem_post (&inside);
    served = served + 1;                            /* statement B */
    pthread_mutex_lock (&door);
    sem_post (&inside);
    served = served + 1;                            /* statement C */
    pthread_mutex_unlock (&door);
    printf ("Service finished.\n");
    return NULL;

} /* gatekeeper */

void * visitor (void * bag) {                       /* funzione di thread   */

    int luggage = 0;                                /* variabile locale     */
    pthread_mutex_lock (&door);
    sem_wait (&inside);
    luggage = (int) bag;                            /* statement D */
    pthread_mutex_unlock (&door);
    return NULL;

} /* visitor */

int main (int argc, char * argv [ ]) {              /* thread principale    */

    pthread_t gk, v1, v2;
    int bag1 = 4;
    int bag2 = 8;

    sem_init (&inside, 0, 0);
    pthread_create (&gk, NULL, &gatekeeper, NULL);
    pthread_create (&v1, NULL, &visitor, (void *) bag1);
    pthread_create (&v2, NULL, &visitor, (void *) bag2);
    pthread_join (gk);
    pthread_join (v1);                              /* statement E */
    pthread_join (v2);

} /* main */
```

Temi d’Esame

3 settembre 2010

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di venerdì 3 settembre 2010

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili globali del programma		
	`data_out [0]`	`data_out [1]`	`p`
subito dopo le statement **A** del thread th [0]			
subito dopo le statement **B** del thread th [1]			
subito dopo le statement **C** del thread th [0]			

```
/* constants                                                          */
#define  DATA_SIZE 2

/* global variables                                                   */
char p = 0;
int  data_in [DATA_SIZE] = { 3, 7 };
int  data_out [DATA_SIZE];

/* global mutex and semaphore variables                               */
sem_t           sems [DATA_SIZE];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

/* thread function                                                    */
void * th_fun (void * arg) {
   /* local variable                                                  */
   int r = (int) arg;
   /* data_out is updated to the rotated contents of data_in          */
   if (r == 0) data_out [0] = data_in [DATA_SIZE - 1];   /* stat. A  */
   else data_out [r] = data_in [r - 1];                  /* stat. B  */
   pthread_mutex_lock (&mutex);
   p++;                                                  /* stat. C  */
   pthread_mutex_unlock (&mutex);
   sem_wait (&sems [r]);                                 /* stat. D  */
   return 0;
} /* th_fun */

/* main thread                                                        */
int main (int argc, char ** argv) {
   /* local variable                                                  */
   int i;
   /* local thread id variable                                        */
   pthread_t th [DATA_SIZE];
   for (i = 0; i < DATA_SIZE; i++) {
      sem_init (&sems [i], 0, 0);
      pthread_create (&th [i], NULL, th_fun, (void *) i);
   } /* end for */
   /* wait cycle                                                      */
   while (p < DATA_SIZE);
   for (i = 0; i < DATA_SIZE; i++) sem_post (&sems [i]);  /* stat. E */
   for (i = 0; i < DATA_SIZE; i++) pthread_join (th [i], NULL);
} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di venerdì 3 settembre 2010

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili globali del programma		
	`data_out [0]`	`data_out [1]`	`p`
subito dopo le statement **A** del thread th [0]	*7*		
subito dopo le statement **B** del thread th [1]			
subito dopo le statement **C** del thread th [0]			

```
/* constants                                                          */
#define  DATA_SIZE 2

/* global variables                                                   */
char p = 0;
int  data_in [DATA_SIZE] = { 3, 7 };
int  data_out [DATA_SIZE];

/* global mutex and semaphore variables                               */
sem_t           sems [DATA_SIZE];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

/* thread function                                                    */
void * th_fun (void * arg) {
   /* local variable                                                  */
   int r = (int) arg;
   /* data_out is updated to the rotated contents of data_in          */
   if (r == 0) data_out [0] = data_in [DATA_SIZE - 1];   /* stat. A  */
   else data_out [r] = data_in [r - 1];                  /* stat. B  */
   pthread_mutex_lock (&mutex);
   p++;                                                  /* stat. C  */
   pthread_mutex_unlock (&mutex);
   sem_wait (&sems [r]);                                 /* stat. D  */
   return 0;
} /* th_fun */

/* main thread                                                        */
int main (int argc, char ** argv) {
   /* local variable                                                  */
   int i;
   /* local thread id variable                                        */
   pthread_t th [DATA_SIZE];
   for (i = 0; i < DATA_SIZE; i++) {
      sem_init (&sems [i], 0, 0);
      pthread_create (&th [i], NULL, th_fun, (void *) i);
   } /* end for */
   /* wait cycle                                                      */
   while (p < DATA_SIZE);
   for (i = 0; i < DATA_SIZE; i++) sem_post (&sems [i]);  /* stat. E */
   for (i = 0; i < DATA_SIZE; i++) pthread_join (th [i], NULL);
} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di venerdì 3 settembre 2010

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili globali del programma		
	`data_out [0]`	`data_out [1]`	`p`
subito dopo le statement **A** del thread th [0]	*7*	*X / 3*	
subito dopo le statement **B** del thread th [1]			
subito dopo le statement **C** del thread th [0]			

```
/* constants                                                          */
#define  DATA_SIZE 2

/* global variables                                                   */
char p = 0;
int  data_in [DATA_SIZE] = { 3, 7 };
int  data_out [DATA_SIZE];

/* global mutex and semaphore variables                               */
sem_t           sems [DATA_SIZE];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

/* thread function                                                    */
void * th_fun (void * arg) {
   /* local variable                                                  */
   int r = (int) arg;
   /* data_out is updated to the rotated contents of data_in          */
   if (r == 0) data_out [0] = data_in [DATA_SIZE - 1];   /* stat. A  */
   else data_out [r] = data_in [r - 1];                  /* stat. B  */
   pthread_mutex_lock (&mutex);
   p++;                                                  /* stat. C  */
   pthread_mutex_unlock (&mutex);
   sem_wait (&sems [r]);                                 /* stat. D  */
   return 0;
} /* th_fun */

/* main thread                                                        */
int main (int argc, char ** argv) {
   /* local variable                                                  */
   int i;
   /* local thread id variable                                        */
   pthread_t th [DATA_SIZE];
   for (i = 0; i < DATA_SIZE; i++) {
      sem_init (&sems [i], 0, 0);
      pthread_create (&th [i], NULL, th_fun, (void *) i);
   } /* end for */
   /* wait cycle                                                      */
   while (p < DATA_SIZE);
   for (i = 0; i < DATA_SIZE; i++) sem_post (&sems [i]);  /* stat. E */
   for (i = 0; i < DATA_SIZE; i++) pthread_join (th [i], NULL);
} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di venerdì 3 settembre 2010

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili globali del programma		
	`data_out [0]`	`data_out [1]`	`p`
subito dopo le statement **A** del thread th [0]	*7*	*X / 3*	*0 / 1*
subito dopo le statement **B** del thread th [1]			
subito dopo le statement **C** del thread th [0]			

```
/* constants                                                          */
#define  DATA_SIZE 2

/* global variables                                                   */
char p = 0;
int  data_in [DATA_SIZE] = { 3, 7 };
int  data_out [DATA_SIZE];

/* global mutex and semaphore variables                               */
sem_t           sems [DATA_SIZE];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

/* thread function                                                    */
void * th_fun (void * arg) {
   /* local variable                                                  */
   int r = (int) arg;
   /* data_out is updated to the rotated contents of data_in          */
   if (r == 0) data_out [0] = data_in [DATA_SIZE - 1];   /* stat. A  */
   else data_out [r] = data_in [r - 1];                  /* stat. B  */
   pthread_mutex_lock (&mutex);
   p++;                                                  /* stat. C  */
   pthread_mutex_unlock (&mutex);
   sem_wait (&sems [r]);                                 /* stat. D  */
   return 0;
} /* th_fun */

/* main thread                                                        */
int main (int argc, char ** argv) {
   /* local variable                                                  */
   int i;
   /* local thread id variable                                        */
   pthread_t th [DATA_SIZE];
   for (i = 0; i < DATA_SIZE; i++) {
      sem_init (&sems [i], 0, 0);
      pthread_create (&th [i], NULL, th_fun, (void *) i);
   } /* end for */
   /* wait cycle                                                      */
   while (p < DATA_SIZE);
   for (i = 0; i < DATA_SIZE; i++) sem_post (&sems [i]);  /* stat. E */
   for (i = 0; i < DATA_SIZE; i++) pthread_join (th [i], NULL);
} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di venerdì 3 settembre 2010

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili globali del programma		
	`data_out [0]`	`data_out [1]`	`p`
subito dopo le statement **A** del thread th [0]	*7*	*X / 3*	*0 / 1*
subito dopo le statement **B** del thread th [1]	*X / 7*		
subito dopo le statement **C** del thread th [0]			

```
/* constants                                                          */
#define  DATA_SIZE 2

/* global variables                                                   */
char p = 0;
int  data_in [DATA_SIZE] = { 3, 7 };
int  data_out [DATA_SIZE];

/* global mutex and semaphore variables                               */
sem_t           sems [DATA_SIZE];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

/* thread function                                                    */
void * th_fun (void * arg) {
   /* local variable                                                  */
   int r = (int) arg;
   /* data_out is updated to the rotated contents of data_in          */
   if (r == 0) data_out [0] = data_in [DATA_SIZE - 1];   /* stat. A  */
   else data_out [r] = data_in [r - 1];                  /* stat. B  */
   pthread_mutex_lock (&mutex);
   p++;                                                  /* stat. C  */
   pthread_mutex_unlock (&mutex);
   sem_wait (&sems [r]);                                 /* stat. D  */
   return 0;
} /* th_fun */

/* main thread                                                        */
int main (int argc, char ** argv) {
   /* local variable                                                  */
   int i;
   /* local thread id variable                                        */
   pthread_t th [DATA_SIZE];
   for (i = 0; i < DATA_SIZE; i++) {
      sem_init (&sems [i], 0, 0);
      pthread_create (&th [i], NULL, th_fun, (void *) i);
   } /* end for */
   /* wait cycle                                                      */
   while (p < DATA_SIZE);
   for (i = 0; i < DATA_SIZE; i++) sem_post (&sems [i]);  /* stat. E */
   for (i = 0; i < DATA_SIZE; i++) pthread_join (th [i], NULL);
} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di venerdì 3 settembre 2010

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili globali del programma		
	`data_out [0]`	`data_out [1]`	`p`
subito dopo le statement **A** del thread th [0]	*7*	*X / 3*	*0 / 1*
subito dopo le statement **B** del thread th [1]	*X / 7*	*3*	
subito dopo le statement **C** del thread th [0]			

```
/* constants                                                          */
#define  DATA_SIZE 2

/* global variables                                                   */
char p = 0;
int  data_in [DATA_SIZE] = { 3, 7 };
int  data_out [DATA_SIZE];

/* global mutex and semaphore variables                               */
sem_t           sems [DATA_SIZE];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

/* thread function                                                    */
void * th_fun (void * arg) {
   /* local variable                                                  */
   int r = (int) arg;
   /* data_out is updated to the rotated contents of data_in          */
   if (r == 0) data_out [0] = data_in [DATA_SIZE - 1];   /* stat. A  */
   else data_out [r] = data_in [r - 1];                  /* stat. B  */
   pthread_mutex_lock (&mutex);
   p++;                                                  /* stat. C  */
   pthread_mutex_unlock (&mutex);
   sem_wait (&sems [r]);                                 /* stat. D  */
   return 0;
} /* th_fun */

/* main thread                                                        */
int main (int argc, char ** argv) {
   /* local variable                                                  */
   int i;
   /* local thread id variable                                        */
   pthread_t th [DATA_SIZE];
   for (i = 0; i < DATA_SIZE; i++) {
      sem_init (&sems [i], 0, 0);
      pthread_create (&th [i], NULL, th_fun, (void *) i);
   } /* end for */
   /* wait cycle                                                      */
   while (p < DATA_SIZE);
   for (i = 0; i < DATA_SIZE; i++) sem_post (&sems [i]);  /* stat. E */
   for (i = 0; i < DATA_SIZE; i++) pthread_join (th [i], NULL);
} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di venerdì 3 settembre 2010

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili globali del programma		
	`data_out [0]`	`data_out [1]`	`p`
subito dopo le statement **A** del thread th [0]	*7*	*X / 3*	*0 / 1*
subito dopo le statement **B** del thread th [1]	*X / 7*	*3*	*0 / 1*
subito dopo le statement **C** del thread th [0]			

```
/* constants                                                          */
#define  DATA_SIZE 2

/* global variables                                                   */
char p = 0;
int  data_in [DATA_SIZE] = { 3, 7 };
int  data_out [DATA_SIZE];

/* global mutex and semaphore variables                               */
sem_t           sems [DATA_SIZE];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

/* thread function                                                    */
void * th_fun (void * arg) {
   /* local variable                                                  */
   int r = (int) arg;
   /* data_out is updated to the rotated contents of data_in          */
   if (r == 0) data_out [0] = data_in [DATA_SIZE - 1];   /* stat. A  */
   else data_out [r] = data_in [r - 1];                  /* stat. B  */
   pthread_mutex_lock (&mutex);
   p++;                                                  /* stat. C  */
   pthread_mutex_unlock (&mutex);
   sem_wait (&sems [r]);                                 /* stat. D  */
   return 0;
} /* th_fun */

/* main thread                                                        */
int main (int argc, char ** argv) {
   /* local variable                                                  */
   int i;
   /* local thread id variable                                        */
   pthread_t th [DATA_SIZE];
   for (i = 0; i < DATA_SIZE; i++) {
      sem_init (&sems [i], 0, 0);
      pthread_create (&th [i], NULL, th_fun, (void *) i);
   } /* end for */
   /* wait cycle                                                      */
   while (p < DATA_SIZE);
   for (i = 0; i < DATA_SIZE; i++) sem_post (&sems [i]);  /* stat. E */
   for (i = 0; i < DATA_SIZE; i++) pthread_join (th [i], NULL);
} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di venerdì 3 settembre 2010

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili globali del programma		
	`data_out [0]`	`data_out [1]`	`p`
subito dopo le statement **A** del thread th [0]	*7*	*X / 3*	*0 / 1*
subito dopo le statement **B** del thread th [1]	*X / 7*	*3*	*0 / 1*
subito dopo le statement **C** del thread th [0]	*7*		

```
/* constants                                                          */
#define  DATA_SIZE 2

/* global variables                                                   */
char p = 0;
int  data_in [DATA_SIZE] = { 3, 7 };
int  data_out [DATA_SIZE];

/* global mutex and semaphore variables                               */
sem_t           sems [DATA_SIZE];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

/* thread function                                                    */
void * th_fun (void * arg) {
   /* local variable                                                  */
   int r = (int) arg;
   /* data_out is updated to the rotated contents of data_in          */
   if (r == 0) data_out [0] = data_in [DATA_SIZE - 1];   /* stat. A  */
   else data_out [r] = data_in [r - 1];                  /* stat. B  */
   pthread_mutex_lock (&mutex);
   p++;                                                  /* stat. C  */
   pthread_mutex_unlock (&mutex);
   sem_wait (&sems [r]);                                 /* stat. D  */
   return 0;
} /* th_fun */

/* main thread                                                        */
int main (int argc, char ** argv) {
   /* local variable                                                  */
   int i;
   /* local thread id variable                                        */
   pthread_t th [DATA_SIZE];
   for (i = 0; i < DATA_SIZE; i++) {
      sem_init (&sems [i], 0, 0);
      pthread_create (&th [i], NULL, th_fun, (void *) i);
   } /* end for */
   /* wait cycle                                                      */
   while (p < DATA_SIZE);
   for (i = 0; i < DATA_SIZE; i++) sem_post (&sems [i]);  /* stat. E */
   for (i = 0; i < DATA_SIZE; i++) pthread_join (th [i], NULL);
} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di venerdì 3 settembre 2010

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili globali del programma		
	`data_out [0]`	`data_out [1]`	`p`
subito dopo le statement **A** del thread th [0]	*7*	*X / 3*	*0 / 1*
subito dopo le statement **B** del thread th [1]	*X / 7*	*3*	*0 / 1*
subito dopo le statement **C** del thread th [0]	*7*	*X / 3*	

```
/* constants                                                          */
#define  DATA_SIZE 2

/* global variables                                                   */
char p = 0;
int  data_in [DATA_SIZE] = { 3, 7 };
int  data_out [DATA_SIZE];

/* global mutex and semaphore variables                               */
sem_t           sems [DATA_SIZE];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

/* thread function                                                    */
void * th_fun (void * arg) {
   /* local variable                                                  */
   int r = (int) arg;
   /* data_out is updated to the rotated contents of data_in          */
   if (r == 0) data_out [0] = data_in [DATA_SIZE - 1];   /* stat. A   */
   else data_out [r] = data_in [r - 1];                  /* stat. B   */
   pthread_mutex_lock (&mutex);
   p++;                                                  /* stat. C   */
   pthread_mutex_unlock (&mutex);
   sem_wait (&sems [r]);                                 /* stat. D   */
   return 0;
} /* th_fun */

/* main thread                                                        */
int main (int argc, char ** argv) {
   /* local variable                                                  */
   int i;
   /* local thread id variable                                        */
   pthread_t th [DATA_SIZE];
   for (i = 0; i < DATA_SIZE; i++) {
      sem_init (&sems [i], 0, 0);
      pthread_create (&th [i], NULL, th_fun, (void *) i);
   } /* end for */
   /* wait cycle                                                      */
   while (p < DATA_SIZE);
   for (i = 0; i < DATA_SIZE; i++) sem_post (&sems [i]);  /* stat. E */
   for (i = 0; i < DATA_SIZE; i++) pthread_join (th [i], NULL);
} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di venerdì 3 settembre 2010

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili globali del programma		
	`data_out [0]`	`data_out [1]`	`p`
subito dopo le statement **A** del thread th [0]	*7*	*X / 3*	*0 / 1*
subito dopo le statement **B** del thread th [1]	*X / 7*	*3*	*0 / 1*
subito dopo le statement **C** del thread th [0]	*7*	*X / 3*	*1 / 2*

```
/* constants                                                          */
#define  DATA_SIZE 2

/* global variables                                                   */
char p = 0;
int  data_in [DATA_SIZE] = { 3, 7 };
int  data_out [DATA_SIZE];

/* global mutex and semaphore variables                               */
sem_t           sems [DATA_SIZE];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

/* thread function                                                    */
void * th_fun (void * arg) {
   /* local variable                                                  */
   int r = (int) arg;
   /* data_out is updated to the rotated contents of data_in          */
   if (r == 0) data_out [0] = data_in [DATA_SIZE - 1];   /* stat. A   */
   else data_out [r] = data_in [r - 1];                  /* stat. B   */
   pthread_mutex_lock (&mutex);
   p++;                                                  /* stat. C   */
   pthread_mutex_unlock (&mutex);
   sem_wait (&sems [r]);                                 /* stat. D   */
   return 0;
} /* th_fun */

/* main thread                                                        */
int main (int argc, char ** argv) {
   /* local variable                                                  */
   int i;
   /* local thread id variable                                        */
   pthread_t th [DATA_SIZE];
   for (i = 0; i < DATA_SIZE; i++) {
      sem_init (&sems [i], 0, 0);
      pthread_create (&th [i], NULL, th_fun, (void *) i);
   } /* end for */
   /* wait cycle                                                      */
   while (p < DATA_SIZE);
   for (i = 0; i < DATA_SIZE; i++) sem_post (&sems [i]);  /* stat. E */
   for (i = 0; i < DATA_SIZE; i++) pthread_join (th [i], NULL);
} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di venerdì 3 settembre 2010

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili locali dei thread	
	r in th [0]	r in th [1]
subito dopo lo statement **D** del thread th [1]		
subito dopo lo statement **E** nella prima iterazione del ciclo for del thread main		

```
/* constants                                                          */
#define  DATA_SIZE 2

/* global variables                                                   */
char p = 0;
int  data_in [DATA_SIZE] = { 3, 7 };
int  data_out [DATA_SIZE];

/* global mutex and semaphore variables                               */
sem_t           sems [DATA_SIZE];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

/* thread function                                                    */
void * th_fun (void * arg) {
   /* local variable                                                  */
   int r = (int) arg;
   /* data_out is updated to the rotated contents of data_in          */
   if (r == 0) data_out [0] = data_in [DATA_SIZE - 1];   /* stat. A  */
   else data_out [r] = data_in [r - 1];                  /* stat. B  */
   pthread_mutex_lock (&mutex);
   p++;                                                  /* stat. C  */
   pthread_mutex_unlock (&mutex);
   sem_wait (&sems [r]);                                 /* stat. D  */
   return 0;
} /* th_fun */

/* main thread                                                        */
int main (int argc, char ** argv) {
   /* local variable                                                  */
   int i;
   /* local thread id variable                                        */
   pthread_t th [DATA_SIZE];
   for (i = 0; i < DATA_SIZE; i++) {
      sem_init (&sems [i], 0, 0);
      pthread_create (&th [i], NULL, th_fun, (void *) i);
   } /* end for */
   /* wait cycle                                                      */
   while (p < DATA_SIZE);
   for (i = 0; i < DATA_SIZE; i++) sem_post (&sems [i]);  /* stat. E */
   for (i = 0; i < DATA_SIZE; i++) pthread_join (th [i], NULL);
} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di venerdì 3 settembre 2010

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili locali dei thread	
	r in th [0]	r in th [1]
subito dopo lo statement **D** del thread th [1]	*O / NE*	
subito dopo lo statement **E** nella prima iterazione del ciclo for del thread main		

```
/* constants                                                          */
#define  DATA_SIZE 2

/* global variables                                                   */
char p = 0;
int  data_in [DATA_SIZE] = { 3, 7 };
int  data_out [DATA_SIZE];

/* global mutex and semaphore variables                               */
sem_t           sems [DATA_SIZE];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

/* thread function                                                    */
void * th_fun (void * arg) {
   /* local variable                                                  */
   int r = (int) arg;
   /* data_out is updated to the rotated contents of data_in          */
   if (r == 0) data_out [0] = data_in [DATA_SIZE - 1];   /* stat. A  */
   else data_out [r] = data_in [r - 1];                  /* stat. B  */
   pthread_mutex_lock (&mutex);
   p++;                                                  /* stat. C  */
   pthread_mutex_unlock (&mutex);
   sem_wait (&sems [r]);                                 /* stat. D  */
   return 0;
} /* th_fun */

/* main thread                                                        */
int main (int argc, char ** argv) {
   /* local variable                                                  */
   int i;
   /* local thread id variable                                        */
   pthread_t th [DATA_SIZE];
   for (i = 0; i < DATA_SIZE; i++) {
      sem_init (&sems [i], 0, 0);
      pthread_create (&th [i], NULL, th_fun, (void *) i);
   } /* end for */
   /* wait cycle                                                      */
   while (p < DATA_SIZE);
   for (i = 0; i < DATA_SIZE; i++) sem_post (&sems [i]);  /* stat. E */
   for (i = 0; i < DATA_SIZE; i++) pthread_join (th [i], NULL);
} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di venerdì 3 settembre 2010

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili locali dei thread	
	r in th [0]	r in th [1]
subito dopo lo statement **D** del thread th [1]	*O / NE*	
subito dopo lo statement **E** nella prima iterazione del ciclo for del thread main		*1*

```
/* constants                                                          */
#define  DATA_SIZE 2

/* global variables                                                   */
char p = 0;
int  data_in [DATA_SIZE] = { 3, 7 };
int  data_out [DATA_SIZE];

/* global mutex and semaphore variables                               */
sem_t           sems [DATA_SIZE];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

/* thread function                                                    */
void * th_fun (void * arg) {
   /* local variable                                                  */
   int r = (int) arg;
   /* data_out is updated to the rotated contents of data_in          */
   if (r == 0) data_out [0] = data_in [DATA_SIZE - 1];   /* stat. A  */
   else data_out [r] = data_in [r - 1];                  /* stat. B  */
   pthread_mutex_lock (&mutex);
   p++;                                                  /* stat. C  */
   pthread_mutex_unlock (&mutex);
   sem_wait (&sems [r]);                                 /* stat. D  */
   return 0;
} /* th_fun */

/* main thread                                                        */
int main (int argc, char ** argv) {
   /* local variable                                                  */
   int i;
   /* local thread id variable                                        */
   pthread_t th [DATA_SIZE];
   for (i = 0; i < DATA_SIZE; i++) {
      sem_init (&sems [i], 0, 0);
      pthread_create (&th [i], NULL, th_fun, (void *) i);
   } /* end for */
   /* wait cycle                                                      */
   while (p < DATA_SIZE);
   for (i = 0; i < DATA_SIZE; i++) sem_post (&sems [i]);  /* stat. E */
   for (i = 0; i < DATA_SIZE; i++) pthread_join (th [i], NULL);
} /* main */
```

Temi d’Esame

19 novembre 2010

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di venerdì 19 novembre 2010

Esercizio n. 1 – modello thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili globali		
	well	sem	
subito dopo stat. **A** in visitor [0]			
subito dopo stat. **B** in fisher			
subito dopo stat. **C** in main			

```
int well = 0;
pthread_mutex_t cover = PTHREAD_MUTEX_INITIALIZER;
sem_t sem;
```

```
void * toss (void * arg) {
   int coin = (int) arg;
   pthread_mutex_lock (&cover);
   well = well + coin;                                   /* statement A */
   pthread_mutex_unlock (&cover);
   coin = 0;
   sem_post (&sem);
   return NULL;
} /* end toss */
```

```
void * fish (void * arg) {
   int coin = (int) arg;
   sem_wait (&sem);
   pthread_mutex_lock (&cover);
   well = well - coin;
   pthread_mutex_unlock (&cover);
   coin = 2 * coin;                                      /* statement B */
   return NULL;
} /* end fish */
```

```
void main (...) {
   pthread_t visitor [2], fisher;
   sem_init (&sem, 0, 0);
   pthread_create (&visitor [0], NULL, &toss, (void *) 5);
   pthread_create (&visitor [1], NULL, &toss, (void *) 5);
   sem_wait (&sem);
   pthread_create (&fisher, NULL, &fish, (void *) 1);
   pthread_join (visitor [0], NULL);
   pthread_join (visitor [1], NULL);                     /* statement C */
   pthread_join (fisher);
   return;
} /* end main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di venerdì 19 novembre 2010

Esercizio n. 1 – modello thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili globali		
	well	sem	
subito dopo stat. **A** in visitor [0]	*5 / 10*		
subito dopo stat. **B** in fisher			
subito dopo stat. **C** in main			

```
int well = 0;
pthread_mutex_t cover = PTHREAD_MUTEX_INITIALIZER;
sem_t sem;
```

```
void * toss (void * arg) {
   int coin = (int) arg;
   pthread_mutex_lock (&cover);
   well = well + coin;                                   /* statement A */
   pthread_mutex_unlock (&cover);
   coin = 0;
   sem_post (&sem);
   return NULL;
} /* end toss */
```

```
void * fish (void * arg) {
   int coin = (int) arg;
   sem_wait (&sem);
   pthread_mutex_lock (&cover);
   well = well - coin;
   pthread_mutex_unlock (&cover);
   coin = 2 * coin;                                      /* statement B */
   return NULL;
} /* end fish */
```

```
void main (...) {
   pthread_t visitor [2], fisher;
   sem_init (&sem, 0, 0);
   pthread_create (&visitor [0], NULL, &toss, (void *) 5);
   pthread_create (&visitor [1], NULL, &toss, (void *) 5);
   sem_wait (&sem);
   pthread_create (&fisher, NULL, &fish, (void *) 1);
   pthread_join (visitor [0], NULL);
   pthread_join (visitor [1], NULL);                     /* statement C */
   pthread_join (fisher);
   return;
} /* end main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di venerdì 19 novembre 2010

Esercizio n. 1 – modello thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili globali		
	well	sem	
subito dopo stat. **A** in visitor [0]	*5 / 10*	*0 / 1*	
subito dopo stat. **B** in fisher			
subito dopo stat. **C** in main			

```
int well = 0;
pthread_mutex_t cover = PTHREAD_MUTEX_INITIALIZER;
sem_t sem;
```

```
void * toss (void * arg) {
   int coin = (int) arg;
   pthread_mutex_lock (&cover);
   well = well + coin;                                   /* statement A */
   pthread_mutex_unlock (&cover);
   coin = 0;
   sem_post (&sem);
   return NULL;
} /* end toss */
```

```
void * fish (void * arg) {
   int coin = (int) arg;
   sem_wait (&sem);
   pthread_mutex_lock (&cover);
   well = well - coin;
   pthread_mutex_unlock (&cover);
   coin = 2 * coin;                                      /* statement B */
   return NULL;
} /* end fish */
```

```
void main (...) {
   pthread_t visitor [2], fisher;
   sem_init (&sem, 0, 0);
   pthread_create (&visitor [0], NULL, &toss, (void *) 5);
   pthread_create (&visitor [1], NULL, &toss, (void *) 5);
   sem_wait (&sem);
   pthread_create (&fisher, NULL, &fish, (void *) 1);
   pthread_join (visitor [0], NULL);
   pthread_join (visitor [1], NULL);                     /* statement C */
   pthread_join (fisher);
   return;
} /* end main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di venerdì 19 novembre 2010

Esercizio n. 1 – modello thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili globali		
	well	sem	
subito dopo stat. **A** in visitor [0]	*5 / 10*	*0 / 1*	
subito dopo stat. **B** in fisher	*9*		
subito dopo stat. **C** in main			

```
int well = 0;
pthread_mutex_t cover = PTHREAD_MUTEX_INITIALIZER;
sem_t sem;
```

```
void * toss (void * arg) {
   int coin = (int) arg;
   pthread_mutex_lock (&cover);
   well = well + coin;                                  /* statement A */
   pthread_mutex_unlock (&cover);
   coin = 0;
   sem_post (&sem);
   return NULL;
} /* end toss */
```

```
void * fish (void * arg) {
   int coin = (int) arg;
   sem_wait (&sem);
   pthread_mutex_lock (&cover);
   well = well - coin;
   pthread_mutex_unlock (&cover);
   coin = 2 * coin;                                     /* statement B */
   return NULL;
} /* end fish */
```

```
void main (...) {
   pthread_t visitor [2], fisher;
   sem_init (&sem, 0, 0);
   pthread_create (&visitor [0], NULL, &toss, (void *) 5);
   pthread_create (&visitor [1], NULL, &toss, (void *) 5);
   sem_wait (&sem);
   pthread_create (&fisher, NULL, &fish, (void *) 1);
   pthread_join (visitor [0], NULL);
   pthread_join (visitor [1], NULL);                    /* statement C */
   pthread_join (fisher);
   return;
} /* end main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di venerdì 19 novembre 2010

Esercizio n. 1 – modello thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili globali		
	well	sem	
subito dopo stat. **A** in visitor [0]	*5 / 10*	*0 / 1*	
subito dopo stat. **B** in fisher	*9*	*0*	
subito dopo stat. **C** in main			

```
int well = 0;
pthread_mutex_t cover = PTHREAD_MUTEX_INITIALIZER;
sem_t sem;
```

```
void * toss (void * arg) {
   int coin = (int) arg;
   pthread_mutex_lock (&cover);
   well = well + coin;                                  /* statement A */
   pthread_mutex_unlock (&cover);
   coin = 0;
   sem_post (&sem);
   return NULL;
} /* end toss */
```

```
void * fish (void * arg) {
   int coin = (int) arg;
   sem_wait (&sem);
   pthread_mutex_lock (&cover);
   well = well - coin;
   pthread_mutex_unlock (&cover);
   coin = 2 * coin;                                     /* statement B */
   return NULL;
} /* end fish */
```

```
void main (...) {
   pthread_t visitor [2], fisher;
   sem_init (&sem, 0, 0);
   pthread_create (&visitor [0], NULL, &toss, (void *) 5);
   pthread_create (&visitor [1], NULL, &toss, (void *) 5);
   sem_wait (&sem);
   pthread_create (&fisher, NULL, &fish, (void *) 1);
   pthread_join (visitor [0], NULL);
   pthread_join (visitor [1], NULL);                    /* statement C */
   pthread_join (fisher);
   return;
} /* end main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di venerdì 19 novembre 2010

Esercizio n. 1 – modello thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili globali		
	well	sem	
subito dopo stat. **A** in visitor [0]	*5 / 10*	*0 / 1*	
subito dopo stat. **B** in fisher	*9*	*0*	
subito dopo stat. **C** in main	*10 / 9*		

```
int well = 0;
pthread_mutex_t cover = PTHREAD_MUTEX_INITIALIZER;
sem_t sem;
```

```
void * toss (void * arg) {
   int coin = (int) arg;
   pthread_mutex_lock (&cover);
   well = well + coin;                                  /* statement A */
   pthread_mutex_unlock (&cover);
   coin = 0;
   sem_post (&sem);
   return NULL;
} /* end toss */
```

```
void * fish (void * arg) {
   int coin = (int) arg;
   sem_wait (&sem);
   pthread_mutex_lock (&cover);
   well = well - coin;
   pthread_mutex_unlock (&cover);
   coin = 2 * coin;                                     /* statement B */
   return NULL;
} /* end fish */
```

```
void main (...) {
   pthread_t visitor [2], fisher;
   sem_init (&sem, 0, 0);
   pthread_create (&visitor [0], NULL, &toss, (void *) 5);
   pthread_create (&visitor [1], NULL, &toss, (void *) 5);
   sem_wait (&sem);
   pthread_create (&fisher, NULL, &fish, (void *) 1);
   pthread_join (visitor [0], NULL);
   pthread_join (visitor [1], NULL);                    /* statement C */
   pthread_join (fisher);
   return;
} /* end main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di venerdì 19 novembre 2010

Esercizio n. 1 – modello thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili globali		
	well	sem	
subito dopo stat. **A** in visitor [0]	*5 / 10*	*0 / 1*	
subito dopo stat. **B** in fisher	*9*	*0*	
subito dopo stat. **C** in main	*10 / 9*	*1 / 0*	

```
int well = 0;
pthread_mutex_t cover = PTHREAD_MUTEX_INITIALIZER;
sem_t sem;
```

```
void * toss (void * arg) {
   int coin = (int) arg;
   pthread_mutex_lock (&cover);
   well = well + coin;                                   /* statement A */
   pthread_mutex_unlock (&cover);
   coin = 0;
   sem_post (&sem);
   return NULL;
} /* end toss */
```

```
void * fish (void * arg) {
   int coin = (int) arg;
   sem_wait (&sem);
   pthread_mutex_lock (&cover);
   well = well - coin;
   pthread_mutex_unlock (&cover);
   coin = 2 * coin;                                      /* statement B */
   return NULL;
} /* end fish */
```

```
void main (...) {
   pthread_t visitor [2], fisher;
   sem_init (&sem, 0, 0);
   pthread_create (&visitor [0], NULL, &toss, (void *) 5);
   pthread_create (&visitor [1], NULL, &toss, (void *) 5);
   sem_wait (&sem);
   pthread_create (&fisher, NULL, &fish, (void *) 1);
   pthread_join (visitor [0], NULL);
   pthread_join (visitor [1], NULL);                     /* statement C */
   pthread_join (fisher);
   return;
} /* end main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di venerdì 19 novembre 2010

Esercizio n. 1 – modello thread e parallelismo

Analizzare ora le variabili locali e dire se esistono con certezza o meno oppure se si è certe della fatto che non esistono.

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili locali dei thread	
	coin in visitor [0]	coin in fisher
subito dopo stat. **A** in visitor [0]		
subito dopo stat. **B** in fisher		
subito dopo stat. **C** in main		

```
int well = 0;
pthread_mutex_t cover = PTHREAD_MUTEX_INITIALIZER;
sem_t sem;
```

```
void * toss (void * arg) {
   int coin = (int) arg;
   pthread_mutex_lock (&cover);
   well = well + coin;                                   /* statement A */
   pthread_mutex_unlock (&cover);
   coin = 0;
   sem_post (&sem);
   return NULL;
} /* end toss */
```

```
void * fish (void * arg) {
   int coin = (int) arg;
   sem_wait (&sem);
   pthread_mutex_lock (&cover);
   well = well - coin;
   pthread_mutex_unlock (&cover);
   coin = 2 * coin;                                      /* statement B */
   return NULL;
} /* end fish */
```

```
void main (...) {
   pthread_t visitor [2], fisher;
   sem_init (&sem, 0, 0);
   pthread_create (&visitor [0], NULL, &toss, (void *) 5);
   pthread_create (&visitor [1], NULL, &toss, (void *) 5);
   sem_wait (&sem);
   pthread_create (&fisher, NULL, &fish, (void *) 1);
   pthread_join (visitor [0], NULL);
   pthread_join (visitor [1], NULL);                     /* statement C */
   pthread_join (fisher);
   return;
} /* end main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di venerdì 19 novembre 2010

Esercizio n. 1 – modello thread e parallelismo

Analizzare ora le variabili locali e dire se esistono con certezza o meno oppure se si è certe della fatto che non esistono.

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili locali dei thread	
	coin in visitor [0]	coin in fisher
subito dopo stat. **A** in visitor [0]	*esiste con certezza*	
subito dopo stat. **B** in fisher		
subito dopo stat. **C** in main		

```
int well = 0;
pthread_mutex_t cover = PTHREAD_MUTEX_INITIALIZER;
sem_t sem;
```

```
void * toss (void * arg) {
   int coin = (int) arg;
   pthread_mutex_lock (&cover);
   well = well + coin;                                   /* statement A */
   pthread_mutex_unlock (&cover);
   coin = 0;
   sem_post (&sem);
   return NULL;
} /* end toss */
```

```
void * fish (void * arg) {
   int coin = (int) arg;
   sem_wait (&sem);
   pthread_mutex_lock (&cover);
   well = well - coin;
   pthread_mutex_unlock (&cover);
   coin = 2 * coin;                                      /* statement B */
   return NULL;
} /* end fish */
```

```
void main (...) {
   pthread_t visitor [2], fisher;
   sem_init (&sem, 0, 0);
   pthread_create (&visitor [0], NULL, &toss, (void *) 5);
   pthread_create (&visitor [1], NULL, &toss, (void *) 5);
   sem_wait (&sem);
   pthread_create (&fisher, NULL, &fish, (void *) 1);
   pthread_join (visitor [0], NULL);
   pthread_join (visitor [1], NULL);                     /* statement C */
   pthread_join (fisher);
   return;
} /* end main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di venerdì 19 novembre 2010

Esercizio n. 1 – modello thread e parallelismo

Analizzare ora le variabili locali e dire se esistono con certezza o meno oppure se si è certe della fatto che non esistono.

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili locali dei thread	
	coin in visitor [0]	coin in fisher
subito dopo stat. **A** in visitor [0]	*esiste con certezza*	*(può non esistere)*
subito dopo stat. **B** in fisher		
subito dopo stat. **C** in main		

```
int well = 0;
pthread_mutex_t cover = PTHREAD_MUTEX_INITIALIZER;
sem_t sem;
```

```
void * toss (void * arg) {
   int coin = (int) arg;
   pthread_mutex_lock (&cover);
   well = well + coin;                                   /* statement A */
   pthread_mutex_unlock (&cover);
   coin = 0;
   sem_post (&sem);
   return NULL;
} /* end toss */
```

```
void * fish (void * arg) {
   int coin = (int) arg;
   sem_wait (&sem);
   pthread_mutex_lock (&cover);
   well = well - coin;
   pthread_mutex_unlock (&cover);
   coin = 2 * coin;                                      /* statement B */
   return NULL;
} /* end fish */
```

```
void main (...) {
   pthread_t visitor [2], fisher;
   sem_init (&sem, 0, 0);
   pthread_create (&visitor [0], NULL, &toss, (void *) 5);
   pthread_create (&visitor [1], NULL, &toss, (void *) 5);
   sem_wait (&sem);
   pthread_create (&fisher, NULL, &fish, (void *) 1);
   pthread_join (visitor [0], NULL);
   pthread_join (visitor [1], NULL);                     /* statement C */
   pthread_join (fisher);
   return;
} /* end main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di venerdì 19 novembre 2010

Esercizio n. 1 – modello thread e parallelismo

Analizzare ora le variabili locali e dire se esistono con certezza o meno oppure se si è certe della fatto che non esistono.

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili locali dei thread	
	coin in visitor [0]	coin in fisher
subito dopo stat. **A** in visitor [0]	*esiste con certezza*	*(può non esistere)*
subito dopo stat. **B** in fisher	*(può non esistere)*	
subito dopo stat. **C** in main		

```
int well = 0;
pthread_mutex_t cover = PTHREAD_MUTEX_INITIALIZER;
sem_t sem;
```

```
void * toss (void * arg) {
   int coin = (int) arg;
   pthread_mutex_lock (&cover);
   well = well + coin;                                   /* statement A */
   pthread_mutex_unlock (&cover);
   coin = 0;
   sem_post (&sem);
   return NULL;
} /* end toss */
```

```
void * fish (void * arg) {
   int coin = (int) arg;
   sem_wait (&sem);
   pthread_mutex_lock (&cover);
   well = well - coin;
   pthread_mutex_unlock (&cover);
   coin = 2 * coin;                                      /* statement B */
   return NULL;
} /* end fish */
```

```
void main (...) {
   pthread_t visitor [2], fisher;
   sem_init (&sem, 0, 0);
   pthread_create (&visitor [0], NULL, &toss, (void *) 5);
   pthread_create (&visitor [1], NULL, &toss, (void *) 5);
   sem_wait (&sem);
   pthread_create (&fisher, NULL, &fish, (void *) 1);
   pthread_join (visitor [0], NULL);
   pthread_join (visitor [1], NULL);                     /* statement C */
   pthread_join (fisher);
   return;
} /* end main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di venerdì 19 novembre 2010

Esercizio n. 1 – modello thread e parallelismo

Analizzare ora le variabili locali e dire se esistono con certezza o meno oppure se si è certe della fatto che non esistono.

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili locali dei thread	
	coin in visitor [0]	coin in fisher
subito dopo stat. **A** in visitor [0]	*esiste con certezza*	*(può non esistere)*
subito dopo stat. **B** in fisher	*(può non esistere)*	*esiste con certezza*
subito dopo stat. **C** in main		

```
int well = 0;
pthread_mutex_t cover = PTHREAD_MUTEX_INITIALIZER;
sem_t sem;
```

```
void * toss (void * arg) {
   int coin = (int) arg;
   pthread_mutex_lock (&cover);
   well = well + coin;                                   /* statement A */
   pthread_mutex_unlock (&cover);
   coin = 0;
   sem_post (&sem);
   return NULL;
} /* end toss */
```

```
void * fish (void * arg) {
   int coin = (int) arg;
   sem_wait (&sem);
   pthread_mutex_lock (&cover);
   well = well - coin;
   pthread_mutex_unlock (&cover);
   coin = 2 * coin;                                      /* statement B */
   return NULL;
} /* end fish */
```

```
void main (...) {
   pthread_t visitor [2], fisher;
   sem_init (&sem, 0, 0);
   pthread_create (&visitor [0], NULL, &toss, (void *) 5);
   pthread_create (&visitor [1], NULL, &toss, (void *) 5);
   sem_wait (&sem);
   pthread_create (&fisher, NULL, &fish, (void *) 1);
   pthread_join (visitor [0], NULL);
   pthread_join (visitor [1], NULL);                     /* statement C */
   pthread_join (fisher);
   return;
} /* end main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di venerdì 19 novembre 2010

Esercizio n. 1 – modello thread e parallelismo

Analizzare ora le variabili locali e dire se esistono con certezza o meno oppure se si è certe della fatto che non esistono.

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili locali dei thread	
	coin in visitor [0]	coin in fisher
subito dopo stat. **A** in visitor [0]	*esiste con certezza*	*(può non esistere)*
subito dopo stat. **B** in fisher	*(può non esistere)*	*esiste con certezza*
subito dopo stat. **C** in main	*(non esiste)*	

```
int well = 0;
pthread_mutex_t cover = PTHREAD_MUTEX_INITIALIZER;
sem_t sem;
```

```
void * toss (void * arg) {
   int coin = (int) arg;
   pthread_mutex_lock (&cover);
   well = well + coin;                                   /* statement A */
   pthread_mutex_unlock (&cover);
   coin = 0;
   sem_post (&sem);
   return NULL;
} /* end toss */
```

```
void * fish (void * arg) {
   int coin = (int) arg;
   sem_wait (&sem);
   pthread_mutex_lock (&cover);
   well = well - coin;
   pthread_mutex_unlock (&cover);
   coin = 2 * coin;                                      /* statement B */
   return NULL;
} /* end fish */
```

```
void main (...) {
   pthread_t visitor [2], fisher;
   sem_init (&sem, 0, 0);
   pthread_create (&visitor [0], NULL, &toss, (void *) 5);
   pthread_create (&visitor [1], NULL, &toss, (void *) 5);
   sem_wait (&sem);
   pthread_create (&fisher, NULL, &fish, (void *) 1);
   pthread_join (visitor [0], NULL);
   pthread_join (visitor [1], NULL);                     /* statement C */
   pthread_join (fisher);
   return;
} /* end main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di venerdì 19 novembre 2010

Esercizio n. 1 – modello thread e parallelismo

Analizzare ora le variabili locali e dire se esistono con certezza o meno oppure se si è certe della fatto che non esistono.

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

condizione	variabili locali dei thread	
	coin in visitor [0]	coin in fisher
subito dopo stat. **A** in visitor [0]	*esiste con certezza*	*(può non esistere)*
subito dopo stat. **B** in fisher	*(può non esistere)*	*esiste con certezza*
subito dopo stat. **C** in main	*(non esiste)*	*(può non esistere)*

```
int well = 0;
pthread_mutex_t cover = PTHREAD_MUTEX_INITIALIZER;
sem_t sem;
```

```
void * toss (void * arg) {
   int coin = (int) arg;
   pthread_mutex_lock (&cover);
   well = well + coin;                                   /* statement A */
   pthread_mutex_unlock (&cover);
   coin = 0;
   sem_post (&sem);
   return NULL;
} /* end toss */
```

```
void * fish (void * arg) {
   int coin = (int) arg;
   sem_wait (&sem);
   pthread_mutex_lock (&cover);
   well = well - coin;
   pthread_mutex_unlock (&cover);
   coin = 2 * coin;                                      /* statement B */
   return NULL;
} /* end fish */
```

```
void main (...) {
   pthread_t visitor [2], fisher;
   sem_init (&sem, 0, 0);
   pthread_create (&visitor [0], NULL, &toss, (void *) 5);
   pthread_create (&visitor [1], NULL, &toss, (void *) 5);
   sem_wait (&sem);
   pthread_create (&fisher, NULL, &fish, (void *) 1);
   pthread_join (visitor [0], NULL);
   pthread_join (visitor [1], NULL);                     /* statement C */
   pthread_join (fisher);
   return;
} /* end main */
```

Temi d’Esame

4 luglio 2011

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di lunedì 4 luglio 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto indicando lo stato di esistenza della variabile locale nell'istante di tempo specificato da ciascuna condizione, così: se la variabile certamente esiste, si scriva ESISTE; se certamente non esiste, si scriva NON ESISTE; e se può essere esistente o inesistente, si scriva PUÒ ESISTERE. Ogni casella della tabella va riempita in uno dei tre modi (non va lasciata vuota).

Si badi bene alla colonna condizione: con subito dopo statement X si chiede lo stato che la variabile assume tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**variabile locale**	
	gone in **TR1**	*gone* in **TR2**
subito dopo stat. **A**		
subito dopo stat. **C** in **TR1**		
subito dopo stat. **D**		

```
/* variabili globali                                                    */
pthread_mutex_t gate = PTHREAD_MUTEX_INITIALIZER;
sem_t pass;
int posted = 1;

void * header (void * arg) {                          /* funzione di thread  */

    printf ("Started.\n");
    pthread_mutex_lock (&gate);
    sem_post (&pass);
    posted = posted + 1;                              /* statement A */
    pthread_mutex_unlock (&gate);
    printf ("Finished.\n");
    return NULL;

} /* header */

void * trailer (void * id) {                          /* funzione di thread  */

    int gone = 0;                                     /* variabile locale    */

    pthread_mutex_lock (&gate);                       /* statement B */
    sem_wait (&pass);
    pthread_mutex_unlock (&gate);
    gone = 1;                                         /* statement C */
    sem_post (&pass);
    posted = posted + (int) id;
    return NULL;

} /* trailer */

int main (int argc, char * argv [ ]) {                /* thread principale   */

    pthread_t hr, tr1, tr2;
    sem_init (&pass, 0, 0);
    pthread_create (&hr, NULL, &header, NULL);
    pthread_create (&tr1, NULL, &trailer, (void *) 1);
    pthread_create (&tr2, NULL, &trailer, (void *) 2);
    pthread_join (hr);
    pthread_join (tr2);                               /* statement D */
    pthread_join (tr1);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di lunedì 4 luglio 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto indicando lo stato di esistenza della variabile locale nell'istante di tempo specificato da ciascuna condizione, così: se la variabile certamente esiste, si scriva ESISTE; se certamente non esiste, si scriva NON ESISTE; e se può essere esistente o inesistente, si scriva PUÒ ESISTERE. Ogni casella della tabella va riempita in uno dei tre modi (non va lasciata vuota).

Si badi bene alla colonna condizione: con subito dopo statement X si chiede lo stato che la variabile assume tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**variabile locale**	
	gone in **TR1**	*gone* in **TR2**
subito dopo stat. **A**	*PUÒ ESISTERE*	
subito dopo stat. **C** in **TR1**		
subito dopo stat. **D**		

```
/* variabili globali                                                    */
pthread_mutex_t gate = PTHREAD_MUTEX_INITIALIZER;
sem_t pass;
int posted = 1;

void * header (void * arg) {                          /* funzione di thread  */

    printf ("Started.\n");
    pthread_mutex_lock (&gate);
    sem_post (&pass);
    posted = posted + 1;                              /* statement A */
    pthread_mutex_unlock (&gate);
    printf ("Finished.\n");
    return NULL;

} /* header */

void * trailer (void * id) {                          /* funzione di thread  */

    int gone = 0;                                     /* variabile locale    */

    pthread_mutex_lock (&gate);                       /* statement B */
    sem_wait (&pass);
    pthread_mutex_unlock (&gate);
    gone = 1;                                         /* statement C */
    sem_post (&pass);
    posted = posted + (int) id;
    return NULL;

} /* trailer */

int main (int argc, char * argv [ ]) {                /* thread principale   */

    pthread_t hr, tr1, tr2;
    sem_init (&pass, 0, 0);
    pthread_create (&hr, NULL, &header, NULL);
    pthread_create (&tr1, NULL, &trailer, (void *) 1);
    pthread_create (&tr2, NULL, &trailer, (void *) 2);
    pthread_join (hr);
    pthread_join (tr2);                               /* statement D */
    pthread_join (tr1);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di lunedì 4 luglio 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto indicando lo stato di esistenza della variabile locale nell'istante di tempo specificato da ciascuna condizione, così: se la variabile certamente esiste, si scriva ESISTE; se certamente non esiste, si scriva NON ESISTE; e se può essere esistente o inesistente, si scriva PUÒ ESISTERE. Ogni casella della tabella va riempita in uno dei tre modi (non va lasciata vuota).

Si badi bene alla colonna condizione: con subito dopo statement X si chiede lo stato che la variabile assume tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**variabile locale**	
	gone in **TR1**	*gone* in **TR2**
subito dopo stat. **A**	*PUÒ ESISTERE*	*PUÒ ESISTERE*
subito dopo stat. **C** in **TR1**		
subito dopo stat. **D**		

```
/* variabili globali                                                    */
pthread_mutex_t gate = PTHREAD_MUTEX_INITIALIZER;
sem_t pass;
int posted = 1;

void * header (void * arg) {                          /* funzione di thread  */

    printf ("Started.\n");
    pthread_mutex_lock (&gate);
    sem_post (&pass);
    posted = posted + 1;                              /* statement A */
    pthread_mutex_unlock (&gate);
    printf ("Finished.\n");
    return NULL;

} /* header */

void * trailer (void * id) {                          /* funzione di thread  */

    int gone = 0;                                     /* variabile locale    */

    pthread_mutex_lock (&gate);                       /* statement B */
    sem_wait (&pass);
    pthread_mutex_unlock (&gate);
    gone = 1;                                         /* statement C */
    sem_post (&pass);
    posted = posted + (int) id;
    return NULL;

} /* trailer */

int main (int argc, char * argv [ ]) {                /* thread principale   */

    pthread_t hr, tr1, tr2;
    sem_init (&pass, 0, 0);
    pthread_create (&hr, NULL, &header, NULL);
    pthread_create (&tr1, NULL, &trailer, (void *) 1);
    pthread_create (&tr2, NULL, &trailer, (void *) 2);
    pthread_join (hr);
    pthread_join (tr2);                               /* statement D */
    pthread_join (tr1);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di lunedì 4 luglio 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto indicando lo stato di esistenza della variabile locale nell'istante di tempo specificato da ciascuna condizione, così: se la variabile certamente esiste, si scriva ESISTE; se certamente non esiste, si scriva NON ESISTE; e se può essere esistente o inesistente, si scriva PUÒ ESISTERE. Ogni casella della tabella va riempita in uno dei tre modi (non va lasciata vuota).

Si badi bene alla colonna condizione: con subito dopo statement X si chiede lo stato che la variabile assume tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**variabile locale**	
	gone in **TR1**	*gone* in **TR2**
subito dopo stat. **A**	*PUÒ ESISTERE*	*PUÒ ESISTERE*
subito dopo stat. **C** in **TR1**	*ESISTE*	
subito dopo stat. **D**		

```
/* variabili globali                                                    */
pthread_mutex_t gate = PTHREAD_MUTEX_INITIALIZER;
sem_t pass;
int posted = 1;

void * header (void * arg) {                          /* funzione di thread  */

    printf ("Started.\n");
    pthread_mutex_lock (&gate);
    sem_post (&pass);
    posted = posted + 1;                              /* statement A */
    pthread_mutex_unlock (&gate);
    printf ("Finished.\n");
    return NULL;

} /* header */

void * trailer (void * id) {                          /* funzione di thread  */

    int gone = 0;                                     /* variabile locale    */

    pthread_mutex_lock (&gate);                       /* statement B */
    sem_wait (&pass);
    pthread_mutex_unlock (&gate);
    gone = 1;                                         /* statement C */
    sem_post (&pass);
    posted = posted + (int) id;
    return NULL;

} /* trailer */

int main (int argc, char * argv [ ]) {                /* thread principale   */

    pthread_t hr, tr1, tr2;
    sem_init (&pass, 0, 0);
    pthread_create (&hr, NULL, &header, NULL);
    pthread_create (&tr1, NULL, &trailer, (void *) 1);
    pthread_create (&tr2, NULL, &trailer, (void *) 2);
    pthread_join (hr);
    pthread_join (tr2);                               /* statement D */
    pthread_join (tr1);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di lunedì 4 luglio 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto indicando lo stato di esistenza della variabile locale nell'istante di tempo specificato da ciascuna condizione, così: se la variabile certamente esiste, si scriva ESISTE; se certamente non esiste, si scriva NON ESISTE; e se può essere esistente o inesistente, si scriva PUÒ ESISTERE. Ogni casella della tabella va riempita in uno dei tre modi (non va lasciata vuota).

Si badi bene alla colonna condizione: con subito dopo statement X si chiede lo stato che la variabile assume tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	variabile locale	
	gone in **TR1**	*gone* in **TR2**
subito dopo stat. **A**	*PUÒ ESISTERE*	*PUÒ ESISTERE*
subito dopo stat. **C** in **TR1**	*ESISTE*	*PUÒ ESISTERE*
subito dopo stat. **D**		

```
/* variabili globali                                                    */
pthread_mutex_t gate = PTHREAD_MUTEX_INITIALIZER;
sem_t pass;
int posted = 1;

void * header (void * arg) {                          /* funzione di thread  */

    printf ("Started.\n");
    pthread_mutex_lock (&gate);
    sem_post (&pass);
    posted = posted + 1;                              /* statement A */
    pthread_mutex_unlock (&gate);
    printf ("Finished.\n");
    return NULL;

} /* header */

void * trailer (void * id) {                          /* funzione di thread  */

    int gone = 0;                                     /* variabile locale    */

    pthread_mutex_lock (&gate);                       /* statement B */
    sem_wait (&pass);
    pthread_mutex_unlock (&gate);
    gone = 1;                                         /* statement C */
    sem_post (&pass);
    posted = posted + (int) id;
    return NULL;

} /* trailer */

int main (int argc, char * argv [ ]) {                /* thread principale   */

    pthread_t hr, tr1, tr2;
    sem_init (&pass, 0, 0);
    pthread_create (&hr, NULL, &header, NULL);
    pthread_create (&tr1, NULL, &trailer, (void *) 1);
    pthread_create (&tr2, NULL, &trailer, (void *) 2);
    pthread_join (hr);
    pthread_join (tr2);                               /* statement D */
    pthread_join (tr1);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di lunedì 4 luglio 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto indicando lo stato di esistenza della variabile locale nell'istante di tempo specificato da ciascuna condizione, così: se la variabile certamente esiste, si scriva ESISTE; se certamente non esiste, si scriva NON ESISTE; e se può essere esistente o inesistente, si scriva PUÒ ESISTERE. Ogni casella della tabella va riempita in uno dei tre modi (non va lasciata vuota).

Si badi bene alla colonna condizione: con subito dopo statement X si chiede lo stato che la variabile assume tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**variabile locale**	
	gone in **TR1**	*gone* in **TR2**
subito dopo stat. **A**	*PUÒ ESISTERE*	*PUÒ ESISTERE*
subito dopo stat. **C** in **TR1**	*ESISTE*	*PUÒ ESISTERE*
subito dopo stat. **D**	*PUÒ ESISTERE*	

```
/* variabili globali                                                    */
pthread_mutex_t gate = PTHREAD_MUTEX_INITIALIZER;
sem_t pass;
int posted = 1;

void * header (void * arg) {                          /* funzione di thread  */

    printf ("Started.\n");
    pthread_mutex_lock (&gate);
    sem_post (&pass);
    posted = posted + 1;                              /* statement A */
    pthread_mutex_unlock (&gate);
    printf ("Finished.\n");
    return NULL;

} /* header */

void * trailer (void * id) {                          /* funzione di thread  */

    int gone = 0;                                     /* variabile locale    */

    pthread_mutex_lock (&gate);                       /* statement B */
    sem_wait (&pass);
    pthread_mutex_unlock (&gate);
    gone = 1;                                         /* statement C */
    sem_post (&pass);
    posted = posted + (int) id;
    return NULL;

} /* trailer */

int main (int argc, char * argv [ ]) {                /* thread principale   */

    pthread_t hr, tr1, tr2;
    sem_init (&pass, 0, 0);
    pthread_create (&hr, NULL, &header, NULL);
    pthread_create (&tr1, NULL, &trailer, (void *) 1);
    pthread_create (&tr2, NULL, &trailer, (void *) 2);
    pthread_join (hr);
    pthread_join (tr2);                               /* statement D */
    pthread_join (tr1);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di lunedì 4 luglio 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto indicando lo stato di esistenza della variabile locale nell'istante di tempo specificato da ciascuna condizione, così: se la variabile certamente esiste, si scriva ESISTE; se certamente non esiste, si scriva NON ESISTE; e se può essere esistente o inesistente, si scriva PUÒ ESISTERE. Ogni casella della tabella va riempita in uno dei tre modi (non va lasciata vuota).

Si badi bene alla colonna condizione: con subito dopo statement X si chiede lo stato che la variabile assume tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**variabile locale**	
	gone in **TR1**	*gone* in **TR2**
subito dopo stat. **A**	*PUÒ ESISTERE*	*PUÒ ESISTERE*
subito dopo stat. **C** in **TR1**	*ESISTE*	*PUÒ ESISTERE*
subito dopo stat. **D**	*PUÒ ESISTERE*	*NON ESISTE*

```
/* variabili globali                                                    */
pthread_mutex_t gate = PTHREAD_MUTEX_INITIALIZER;
sem_t pass;
int posted = 1;

void * header (void * arg) {                          /* funzione di thread  */

    printf ("Started.\n");
    pthread_mutex_lock (&gate);
    sem_post (&pass);
    posted = posted + 1;                              /* statement A */
    pthread_mutex_unlock (&gate);
    printf ("Finished.\n");
    return NULL;

} /* header */

void * trailer (void * id) {                          /* funzione di thread  */

    int gone = 0;                                     /* variabile locale    */

    pthread_mutex_lock (&gate);                       /* statement B */
    sem_wait (&pass);
    pthread_mutex_unlock (&gate);
    gone = 1;                                         /* statement C */
    sem_post (&pass);
    posted = posted + (int) id;
    return NULL;

} /* trailer */

int main (int argc, char * argv [ ]) {                /* thread principale   */

    pthread_t hr, tr1, tr2;
    sem_init (&pass, 0, 0);
    pthread_create (&hr, NULL, &header, NULL);
    pthread_create (&tr1, NULL, &trailer, (void *) 1);
    pthread_create (&tr2, NULL, &trailer, (void *) 2);
    pthread_join (hr);
    pthread_join (tr2);                               /* statement D */
    pthread_join (tr1);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di lunedì 4 luglio 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto, indicando i valori delle variabili globali (sempre esistenti)
Inserire in tabella la struttura in pagine della memoria virtuale dei due programmi X e Y (notazione: CX0, CX1, DX0, PX0,... CY0, ..., COND0, ...).
nell'istante di tempo specificato da ciascuna condizione. Il valore della variabile va indicato così:

- intero, carattere, stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quanti
- il semaforo può avere valore positivo o nullo (non valore negativo)

Si badi bene alla colonna condizione: con subito dopo statement X si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**variabili globali**	
	pass	*posted*
subito dopo stat. **A**		
subito dopo stat. **B** in **TR1**		
subito dopo stat. **C** in **TR2**		

```
/* variabili globali                                                    */
pthread_mutex_t gate = PTHREAD_MUTEX_INITIALIZER;
sem_t pass;
int posted = 1;

void * header (void * arg) {                          /* funzione di thread  */

    printf ("Started.\n");
    pthread_mutex_lock (&gate);
    sem_post (&pass);
    posted = posted + 1;                              /* statement A */
    pthread_mutex_unlock (&gate);
    printf ("Finished.\n");
    return NULL;

} /* header */

void * trailer (void * id) {                          /* funzione di thread  */

    int gone = 0;                                     /* variabile locale    */

    pthread_mutex_lock (&gate);                       /* statement B */
    sem_wait (&pass);
    pthread_mutex_unlock (&gate);
    gone = 1;                                         /* statement C */
    sem_post (&pass);
    posted = posted + (int) id;
    return NULL;

} /* trailer */

int main (int argc, char * argv [ ]) {                /* thread principale   */

    pthread_t hr, tr1, tr2;
    sem_init (&pass, 0, 0);
    pthread_create (&hr, NULL, &header, NULL);
    pthread_create (&tr1, NULL, &trailer, (void *) 1);
    pthread_create (&tr2, NULL, &trailer, (void *) 2);
    pthread_join (hr);
    pthread_join (tr2);                               /* statement D */
    pthread_join (tr1);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di lunedì 4 luglio 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto, indicando i valori delle variabili globali (sempre esistenti)
Inserire in tabella la struttura in pagine della memoria virtuale dei due programmi X e Y (notazione: CX0, CX1, DX0, PX0,... CY0, ..., COND0, ...).
nell'istante di tempo specificato da ciascuna condizione. Il valore della variabile va indicato così:

- intero, carattere, stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quanti
- il semaforo può avere valore positivo o nullo (non valore negativo)

Si badi bene alla colonna condizione: con subito dopo statement X si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**variabili globali**	
	pass	*posted*
subito dopo stat. **A**	*1*	
subito dopo stat. **B** in **TR1**		
subito dopo stat. **C** in **TR2**		

```
/* variabili globali                                                    */
pthread_mutex_t gate = PTHREAD_MUTEX_INITIALIZER;
sem_t pass;
int posted = 1;

void * header (void * arg) {                          /* funzione di thread  */

    printf ("Started.\n");
    pthread_mutex_lock (&gate);
    sem_post (&pass);
    posted = posted + 1;                              /* statement A */
    pthread_mutex_unlock (&gate);
    printf ("Finished.\n");
    return NULL;

} /* header */

void * trailer (void * id) {                          /* funzione di thread  */

    int gone = 0;                                     /* variabile locale    */

    pthread_mutex_lock (&gate);                       /* statement B */
    sem_wait (&pass);
    pthread_mutex_unlock (&gate);
    gone = 1;                                         /* statement C */
    sem_post (&pass);
    posted = posted + (int) id;
    return NULL;

} /* trailer */

int main (int argc, char * argv [ ]) {                /* thread principale   */

    pthread_t hr, tr1, tr2;
    sem_init (&pass, 0, 0);
    pthread_create (&hr, NULL, &header, NULL);
    pthread_create (&tr1, NULL, &trailer, (void *) 1);
    pthread_create (&tr2, NULL, &trailer, (void *) 2);
    pthread_join (hr);
    pthread_join (tr2);                               /* statement D */
    pthread_join (tr1);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di lunedì 4 luglio 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto, indicando i valori delle variabili globali (sempre esistenti)
Inserire in tabella la struttura in pagine della memoria virtuale dei due programmi X e Y (notazione: CX0, CX1, DX0, PX0,... CY0, ..., COND0, ...).
nell'istante di tempo specificato da ciascuna condizione. Il valore della variabile va indicato così:

- intero, carattere, stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quanti
- il semaforo può avere valore positivo o nullo (non valore negativo)

Si badi bene alla colonna condizione: con subito dopo statement X si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	variabili globali	
	pass	*posted*
subito dopo stat. **A**	*1*	*2*
subito dopo stat. **B** in **TR1**		
subito dopo stat. **C** in **TR2**		

```
/* variabili globali                                                    */
pthread_mutex_t gate = PTHREAD_MUTEX_INITIALIZER;
sem_t pass;
int posted = 1;

void * header (void * arg) {                          /* funzione di thread  */

    printf ("Started.\n");
    pthread_mutex_lock (&gate);
    sem_post (&pass);
    posted = posted + 1;                              /* statement A */
    pthread_mutex_unlock (&gate);
    printf ("Finished.\n");
    return NULL;

} /* header */

void * trailer (void * id) {                          /* funzione di thread  */

    int gone = 0;                                     /* variabile locale    */

    pthread_mutex_lock (&gate);                       /* statement B */
    sem_wait (&pass);
    pthread_mutex_unlock (&gate);
    gone = 1;                                         /* statement C */
    sem_post (&pass);
    posted = posted + (int) id;
    return NULL;

} /* trailer */

int main (int argc, char * argv [ ]) {                /* thread principale   */

    pthread_t hr, tr1, tr2;
    sem_init (&pass, 0, 0);
    pthread_create (&hr, NULL, &header, NULL);
    pthread_create (&tr1, NULL, &trailer, (void *) 1);
    pthread_create (&tr2, NULL, &trailer, (void *) 2);
    pthread_join (hr);
    pthread_join (tr2);                               /* statement D */
    pthread_join (tr1);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di lunedì 4 luglio 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto, indicando i valori delle variabili globali (sempre esistenti)
Inserire in tabella la struttura in pagine della memoria virtuale dei due programmi X e Y (notazione: CX0, CX1, DX0, PX0,... CY0, ..., COND0, ...).
nell'istante di tempo specificato da ciascuna condizione. Il valore della variabile va indicato così:

- intero, carattere, stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quanti
- il semaforo può avere valore positivo o nullo (non valore negativo)

Si badi bene alla colonna condizione: con subito dopo statement X si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	variabili globali	
	pass	*posted*
subito dopo stat. **A**	*1*	*2*
subito dopo stat. **B** in **TR1**	*0 / 1*	
subito dopo stat. **C** in **TR2**		

```
/* variabili globali                                                    */
pthread_mutex_t gate = PTHREAD_MUTEX_INITIALIZER;
sem_t pass;
int posted = 1;

void * header (void * arg) {                          /* funzione di thread  */

    printf ("Started.\n");
    pthread_mutex_lock (&gate);
    sem_post (&pass);
    posted = posted + 1;                              /* statement A */
    pthread_mutex_unlock (&gate);
    printf ("Finished.\n");
    return NULL;

} /* header */

void * trailer (void * id) {                          /* funzione di thread  */

    int gone = 0;                                     /* variabile locale    */

    pthread_mutex_lock (&gate);                       /* statement B */
    sem_wait (&pass);
    pthread_mutex_unlock (&gate);
    gone = 1;                                         /* statement C */
    sem_post (&pass);
    posted = posted + (int) id;
    return NULL;

} /* trailer */

int main (int argc, char * argv [ ]) {                /* thread principale   */

    pthread_t hr, tr1, tr2;
    sem_init (&pass, 0, 0);
    pthread_create (&hr, NULL, &header, NULL);
    pthread_create (&tr1, NULL, &trailer, (void *) 1);
    pthread_create (&tr2, NULL, &trailer, (void *) 2);
    pthread_join (hr);
    pthread_join (tr2);                               /* statement D */
    pthread_join (tr1);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di lunedì 4 luglio 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto, indicando i valori delle variabili globali (sempre esistenti)
Inserire in tabella la struttura in pagine della memoria virtuale dei due programmi X e Y (notazione: CX0, CX1, DX0, PX0,... CY0, ..., COND0, ...).
nell'istante di tempo specificato da ciascuna condizione. Il valore della variabile va indicato così:

- intero, carattere, stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quanti
- il semaforo può avere valore positivo o nullo (non valore negativo)

Si badi bene alla colonna condizione: con subito dopo statement X si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	variabili globali	
	pass	*posted*
subito dopo stat. **A**	*1*	*2*
subito dopo stat. **B** in **TR1**	*0 / 1*	*1 / 2 / 4*
subito dopo stat. **C** in **TR2**		

```
/* variabili globali                                                    */
pthread_mutex_t gate = PTHREAD_MUTEX_INITIALIZER;
sem_t pass;
int posted = 1;

void * header (void * arg) {                          /* funzione di thread  */

    printf ("Started.\n");
    pthread_mutex_lock (&gate);
    sem_post (&pass);
    posted = posted + 1;                              /* statement A */
    pthread_mutex_unlock (&gate);
    printf ("Finished.\n");
    return NULL;

} /* header */

void * trailer (void * id) {                          /* funzione di thread  */

    int gone = 0;                                     /* variabile locale    */

    pthread_mutex_lock (&gate);                       /* statement B */
    sem_wait (&pass);
    pthread_mutex_unlock (&gate);
    gone = 1;                                         /* statement C */
    sem_post (&pass);
    posted = posted + (int) id;
    return NULL;

} /* trailer */

int main (int argc, char * argv [ ]) {                /* thread principale   */

    pthread_t hr, tr1, tr2;
    sem_init (&pass, 0, 0);
    pthread_create (&hr, NULL, &header, NULL);
    pthread_create (&tr1, NULL, &trailer, (void *) 1);
    pthread_create (&tr2, NULL, &trailer, (void *) 2);
    pthread_join (hr);
    pthread_join (tr2);                               /* statement D */
    pthread_join (tr1);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di lunedì 4 luglio 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto, indicando i valori delle variabili globali (sempre esistenti)
Inserire in tabella la struttura in pagine della memoria virtuale dei due programmi X e Y (notazione: CX0, CX1, DX0, PX0,... CY0, ..., COND0, ...).
nell'istante di tempo specificato da ciascuna condizione. Il valore della variabile va indicato così:

- intero, carattere, stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quanti
- il semaforo può avere valore positivo o nullo (non valore negativo)

Si badi bene alla colonna condizione: con subito dopo statement X si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	**variabili globali**	
	pass	*posted*
subito dopo stat. **A**	*1*	*2*
subito dopo stat. **B** in **TR1**	*0 / 1*	*1 / 2 / 4*
subito dopo stat. **C** in **TR2**	*0*	

```
/* variabili globali                                                    */
pthread_mutex_t gate = PTHREAD_MUTEX_INITIALIZER;
sem_t pass;
int posted = 1;

void * header (void * arg) {                          /* funzione di thread  */

    printf ("Started.\n");
    pthread_mutex_lock (&gate);
    sem_post (&pass);
    posted = posted + 1;                              /* statement A */
    pthread_mutex_unlock (&gate);
    printf ("Finished.\n");
    return NULL;

} /* header */

void * trailer (void * id) {                          /* funzione di thread  */

    int gone = 0;                                     /* variabile locale    */

    pthread_mutex_lock (&gate);                       /* statement B */
    sem_wait (&pass);
    pthread_mutex_unlock (&gate);
    gone = 1;                                         /* statement C */
    sem_post (&pass);
    posted = posted + (int) id;
    return NULL;

} /* trailer */

int main (int argc, char * argv [ ]) {                /* thread principale   */

    pthread_t hr, tr1, tr2;
    sem_init (&pass, 0, 0);
    pthread_create (&hr, NULL, &header, NULL);
    pthread_create (&tr1, NULL, &trailer, (void *) 1);
    pthread_create (&tr2, NULL, &trailer, (void *) 2);
    pthread_join (hr);
    pthread_join (tr2);                               /* statement D */
    pthread_join (tr1);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di lunedì 4 luglio 2011

esercizio n. 1 – thread e parallelismo

Si completi la tabella predisposta qui sotto, indicando i valori delle variabili globali (sempre esistenti)
Inserire in tabella la struttura in pagine della memoria virtuale dei due programmi X e Y (notazione: CX0, CX1, DX0, PX0,... CY0, ..., COND0, ...).
nell'istante di tempo specificato da ciascuna condizione. Il valore della variabile va indicato così:

- intero, carattere, stringa, quando la variabile ha un valore definito; oppure X quando è indefinita
- se la variabile può avere due o più valori, li si riporti tutti quanti
- il semaforo può avere valore positivo o nullo (non valore negativo)

Si badi bene alla colonna condizione: con subito dopo statement X si chiede il valore (o i valori) che la variabile ha tra lo statement X e lo statement immediatamente successivo del thread indicato.

condizione	variabili globali	
	pass	*posted*
subito dopo stat. **A**	*1*	*2*
subito dopo stat. **B** in **TR1**	*0 / 1*	*1 / 2 / 4*
subito dopo stat. **C** in **TR2**	*0*	*2 / 3*

```
/* variabili globali                                                    */
pthread_mutex_t gate = PTHREAD_MUTEX_INITIALIZER;
sem_t pass;
int posted = 1;

void * header (void * arg) {                          /* funzione di thread  */

    printf ("Started.\n");
    pthread_mutex_lock (&gate);
    sem_post (&pass);
    posted = posted + 1;                              /* statement A */
    pthread_mutex_unlock (&gate);
    printf ("Finished.\n");
    return NULL;

} /* header */

void * trailer (void * id) {                          /* funzione di thread  */

    int gone = 0;                                     /* variabile locale    */

    pthread_mutex_lock (&gate);                       /* statement B */
    sem_wait (&pass);
    pthread_mutex_unlock (&gate);
    gone = 1;                                         /* statement C */
    sem_post (&pass);
    posted = posted + (int) id;
    return NULL;

} /* trailer */

int main (int argc, char * argv [ ]) {                /* thread principale   */

    pthread_t hr, tr1, tr2;
    sem_init (&pass, 0, 0);
    pthread_create (&hr, NULL, &header, NULL);
    pthread_create (&tr1, NULL, &trailer, (void *) 1);
    pthread_create (&tr2, NULL, &trailer, (void *) 2);
    pthread_join (hr);
    pthread_join (tr2);                               /* statement D */
    pthread_join (tr1);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di lunedì 4 luglio 2011

esercizio n. 1 – thread e parallelismo

Il sistema può andare in stallo (deadlock). Qui si indichino gli statement dove si bloccano i thread incorsi in stallo (non necessariamente tutti):

HR	TR1	TR2

```
/* variabili globali                                                    */
pthread_mutex_t gate = PTHREAD_MUTEX_INITIALIZER;
sem_t pass;
int posted = 1;

void * header (void * arg) {                          /* funzione di thread  */

    printf ("Started.\n");
    pthread_mutex_lock (&gate);
    sem_post (&pass);
    posted = posted + 1;                              /* statement A */
    pthread_mutex_unlock (&gate);
    printf ("Finished.\n");
    return NULL;

} /* header */

void * trailer (void * id) {                          /* funzione di thread  */

    int gone = 0;                                     /* variabile locale    */

    pthread_mutex_lock (&gate);                       /* statement B */
    sem_wait (&pass);
    pthread_mutex_unlock (&gate);
    gone = 1;                                         /* statement C */
    sem_post (&pass);
    posted = posted + (int) id;
    return NULL;

} /* trailer */

int main (int argc, char * argv [ ]) {                /* thread principale   */

    pthread_t hr, tr1, tr2;
    sem_init (&pass, 0, 0);
    pthread_create (&hr, NULL, &header, NULL);
    pthread_create (&tr1, NULL, &trailer, (void *) 1);
    pthread_create (&tr2, NULL, &trailer, (void *) 2);
    pthread_join (hr);
    pthread_join (tr2);                               /* statement D */
    pthread_join (tr1);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di lunedì 4 luglio 2011

esercizio n. 1 – thread e parallelismo

Il sistema può andare in stallo (deadlock). Qui si indichino gli statement dove si bloccano i thread incorsi in stallo (non necessariamente tutti):

HR	TR1	TR2
pthread_mutex_lock *(&gate)*		

```
/* variabili globali                                                    */
pthread_mutex_t gate = PTHREAD_MUTEX_INITIALIZER;
sem_t pass;
int posted = 1;

void * header (void * arg) {                          /* funzione di thread  */

    printf ("Started.\n");
    pthread_mutex_lock (&gate);
    sem_post (&pass);
    posted = posted + 1;                              /* statement A */
    pthread_mutex_unlock (&gate);
    printf ("Finished.\n");
    return NULL;

} /* header */

void * trailer (void * id) {                          /* funzione di thread  */

    int gone = 0;                                     /* variabile locale    */

    pthread_mutex_lock (&gate);                       /* statement B */
    sem_wait (&pass);
    pthread_mutex_unlock (&gate);
    gone = 1;                                         /* statement C */
    sem_post (&pass);
    posted = posted + (int) id;
    return NULL;

} /* trailer */

int main (int argc, char * argv [ ]) {                /* thread principale   */

    pthread_t hr, tr1, tr2;
    sem_init (&pass, 0, 0);
    pthread_create (&hr, NULL, &header, NULL);
    pthread_create (&tr1, NULL, &trailer, (void *) 1);
    pthread_create (&tr2, NULL, &trailer, (void *) 2);
    pthread_join (hr);
    pthread_join (tr2);                               /* statement D */
    pthread_join (tr1);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di lunedì 4 luglio 2011

esercizio n. 1 – thread e parallelismo

Il sistema può andare in stallo (deadlock). Qui si indichino gli statement dove si bloccano i thread incorsi in stallo (non necessariamente tutti):

HR	TR1	TR2
pthread_mutex_lock *(&gate)*	***sem_wait*** *(&pass)*	

```
/* variabili globali                                                    */
pthread_mutex_t gate = PTHREAD_MUTEX_INITIALIZER;
sem_t pass;
int posted = 1;

void * header (void * arg) {                          /* funzione di thread  */

    printf ("Started.\n");
    pthread_mutex_lock (&gate);
    sem_post (&pass);
    posted = posted + 1;                              /* statement A */
    pthread_mutex_unlock (&gate);
    printf ("Finished.\n");
    return NULL;

} /* header */

void * trailer (void * id) {                          /* funzione di thread  */

    int gone = 0;                                     /* variabile locale    */

    pthread_mutex_lock (&gate);                       /* statement B */
    sem_wait (&pass);
    pthread_mutex_unlock (&gate);
    gone = 1;                                         /* statement C */
    sem_post (&pass);
    posted = posted + (int) id;
    return NULL;

} /* trailer */

int main (int argc, char * argv [ ]) {                /* thread principale   */

    pthread_t hr, tr1, tr2;
    sem_init (&pass, 0, 0);
    pthread_create (&hr, NULL, &header, NULL);
    pthread_create (&tr1, NULL, &trailer, (void *) 1);
    pthread_create (&tr2, NULL, &trailer, (void *) 2);
    pthread_join (hr);
    pthread_join (tr2);                               /* statement D */
    pthread_join (tr1);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di lunedì 4 luglio 2011

esercizio n. 1 – thread e parallelismo

Il sistema può andare in stallo (deadlock). Qui si indichino gli statement dove si bloccano i thread incorsi in stallo (non necessariamente tutti):

HR	TR1	TR2
pthread_mutex_lock *(&gate)*	***sem_wait*** *(&pass)*	*si trova a monte di* ***pthread_mutex_lock*** *o è anch'esso fermo su* ***pthread_mutex_lock***

```
/* variabili globali                                                    */
pthread_mutex_t gate = PTHREAD_MUTEX_INITIALIZER;
sem_t pass;
int posted = 1;

void * header (void * arg) {                          /* funzione di thread  */

    printf ("Started.\n");
    pthread_mutex_lock (&gate);
    sem_post (&pass);
    posted = posted + 1;                              /* statement A */
    pthread_mutex_unlock (&gate);
    printf ("Finished.\n");
    return NULL;

} /* header */

void * trailer (void * id) {                          /* funzione di thread  */

    int gone = 0;                                     /* variabile locale    */

    pthread_mutex_lock (&gate);                       /* statement B */
    sem_wait (&pass);
    pthread_mutex_unlock (&gate);
    gone = 1;                                         /* statement C */
    sem_post (&pass);
    posted = posted + (int) id;
    return NULL;

} /* trailer */

int main (int argc, char * argv [ ]) {                /* thread principale   */

    pthread_t hr, tr1, tr2;
    sem_init (&pass, 0, 0);
    pthread_create (&hr, NULL, &header, NULL);
    pthread_create (&tr1, NULL, &trailer, (void *) 1);
    pthread_create (&tr2, NULL, &trailer, (void *) 2);
    pthread_join (hr);
    pthread_join (tr2);                               /* statement D */
    pthread_join (tr1);

} /* main */
```

AXO – Architettura dei Calcolatori e Sistemi Operativi

esame di lunedì 4 luglio 2011

esercizio n. 1 – thread e parallelismo

Il sistema può andare in stallo (deadlock). Qui si indichino gli statement dove si bloccano i thread incorsi in stallo (non necessariamente tutti):

HR	TR1	TR2
pthread_mutex_lock *(&gate)*	***sem_wait*** *(&pass)*	*si trova a monte di* ***pthread_mutex_lock*** *o è anch'esso fermo su* ***pthread_mutex_lock***

Si possono scambiare i ruoli di TR1 e TR2.

```
/* variabili globali                                                    */
pthread_mutex_t gate = PTHREAD_MUTEX_INITIALIZER;
sem_t pass;
int posted = 1;

void * header (void * arg) {                          /* funzione di thread  */

    printf ("Started.\n");
    pthread_mutex_lock (&gate);
    sem_post (&pass);
    posted = posted + 1;                              /* statement A */
    pthread_mutex_unlock (&gate);
    printf ("Finished.\n");
    return NULL;

} /* header */

void * trailer (void * id) {                          /* funzione di thread  */

    int gone = 0;                                     /* variabile locale    */

    pthread_mutex_lock (&gate);                       /* statement B */
    sem_wait (&pass);
    pthread_mutex_unlock (&gate);
    gone = 1;                                         /* statement C */
    sem_post (&pass);
    posted = posted + (int) id;
    return NULL;

} /* trailer */

int main (int argc, char * argv [ ]) {                /* thread principale   */

    pthread_t hr, tr1, tr2;
    sem_init (&pass, 0, 0);
    pthread_create (&hr, NULL, &header, NULL);
    pthread_create (&tr1, NULL, &trailer, (void *) 1);
    pthread_create (&tr2, NULL, &trailer, (void *) 2);
    pthread_join (hr);
    pthread_join (tr2);                               /* statement D */
    pthread_join (tr1);

} /* main */
```

Temi d’Esame

17 novembre 2009

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella A

condizione	variabili locali al thread th_A		variabili globali	
	Aa	**Ab**	**Ga**	**sem_A**
subito dopo STATEMENT S1 in th_A				
subito dopo STATEMENT S2 in th_A				
subito dopo STATEMENT S3 in th_B				
subito dopo STATEMENT S4 in main				

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella A

condizione	variabili locali al thread th_A		variabili globali	
	Aa	**Ab**	**Ga**	**sem_A**
subito dopo STATEMENT S1 in th_A				
subito dopo STATEMENT S2 in th_A				
subito dopo STATEMENT S3 in th_B				
subito dopo STATEMENT S4 in main				

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella A

condizione	variabili locali al thread th_A		variabili globali	
	Aa	**Ab**	**Ga**	**sem_A**
subito dopo STATEMENT S1 in th_A				
subito dopo STATEMENT S2 in th_A				
subito dopo STATEMENT S3 in th_B				
subito dopo STATEMENT S4 in main				

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella A

condizione	variabili locali al thread th_A		variabili globali	
	Aa	**Ab**	**Ga**	**sem_A**
subito dopo STATEMENT S1 in th_A	*2*			
subito dopo STATEMENT S2 in th_A				
subito dopo STATEMENT S3 in th_B				
subito dopo STATEMENT S4 in main				

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella A

condizione	variabili locali al thread th_A		variabili globali	
	Aa	**Ab**	**Ga**	**sem_A**
subito dopo STATEMENT S1 in th_A	*2*	*X*		
subito dopo STATEMENT S2 in th_A				
subito dopo STATEMENT S3 in th_B				
subito dopo STATEMENT S4 in main				

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella A

condizione	variabili locali al thread th_A		variabili globali	
	Aa	**Ab**	**Ga**	**sem_A**
subito dopo STATEMENT S1 in th_A	*2*	*X*	*4*	
subito dopo STATEMENT S2 in th_A				
subito dopo STATEMENT S3 in th_B				
subito dopo STATEMENT S4 in main				

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella A

condizione	variabili locali al thread th_A		variabili globali	
	Aa	**Ab**	**Ga**	**sem_A**
subito dopo STATEMENT S1 in th_A	*2*	*X*	*4*	*0*
subito dopo STATEMENT S2 in th_A				
subito dopo STATEMENT S3 in th_B				
subito dopo STATEMENT S4 in main				

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella A

condizione	variabili locali al thread th_A		variabili globali	
	Aa	**Ab**	**Ga**	**sem_A**
subito dopo STATEMENT S1 in th_A	*2*	*X*	*4*	*0*
subito dopo STATEMENT S2 in th_A	*2*			
subito dopo STATEMENT S3 in th_B				
subito dopo STATEMENT S4 in main				

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella A

condizione	variabili locali al thread th_A		variabili globali	
	Aa	**Ab**	**Ga**	**sem_A**
subito dopo STATEMENT S1 in th_A	*2*	*X*	*4*	*0*
subito dopo STATEMENT S2 in th_A	*2*	*2*		
subito dopo STATEMENT S3 in th_B				
subito dopo STATEMENT S4 in main				

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella A

condizione	variabili locali al thread th_A		variabili globali	
	Aa	**Ab**	**Ga**	**sem_A**
subito dopo STATEMENT S1 in th_A	*2*	*X*	*4*	*0*
subito dopo STATEMENT S2 in th_A	*2*	*2*	*4 / 6*	
subito dopo STATEMENT S3 in th_B				
subito dopo STATEMENT S4 in main				

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella A

condizione	variabili locali al thread th_A		variabili globali	
	Aa	**Ab**	**Ga**	**sem_A**
subito dopo STATEMENT S1 in th_A	*2*	*X*	*4*	*0*
subito dopo STATEMENT S2 in th_A	*2*	*2*	*4 / 6*	*0*
subito dopo STATEMENT S3 in th_B				
subito dopo STATEMENT S4 in main				

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella A

condizione	variabili locali al thread th_A		variabili globali	
	Aa	**Ab**	**Ga**	**sem_A**
subito dopo STATEMENT S1 in th_A	*2*	*X*	*4*	*0*
subito dopo STATEMENT S2 in th_A	*2*	*2*	*4 / 6*	*0*
subito dopo STATEMENT S3 in th_B	*U (NE / X / 2)*			
subito dopo STATEMENT S4 in main				

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella A

condizione	variabili locali al thread th_A		variabili globali	
	Aa	**Ab**	**Ga**	**sem_A**
subito dopo STATEMENT S1 in th_A	*2*	*X*	*4*	*0*
subito dopo STATEMENT S2 in th_A	*2*	*2*	*4 / 6*	*0*
subito dopo STATEMENT S3 in th_B	*U (NE / X / 2)*	*U (NE / X / 2)*		
subito dopo STATEMENT S4 in main				

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella A

condizione	variabili locali al thread th_A		variabili globali	
	Aa	**Ab**	**Ga**	**sem_A**
subito dopo STATEMENT S1 in th_A	*2*	*X*	*4*	*0*
subito dopo STATEMENT S2 in th_A	*2*	*2*	*4 / 6*	*0*
subito dopo STATEMENT S3 in th_B	*U (NE / X / 2)*	*U (NE / X / 2)*	*3 / 6*	
subito dopo STATEMENT S4 in main				

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella A

condizione	variabili locali al thread th_A		variabili globali	
	Aa	**Ab**	**Ga**	**sem_A**
subito dopo STATEMENT S1 in th_A	*2*	*X*	*4*	*0*
subito dopo STATEMENT S2 in th_A	*2*	*2*	*4 / 6*	*0*
subito dopo STATEMENT S3 in th_B	*U (NE / X / 2)*	*U (NE / X / 2)*	*3 / 6*	*0 / 1*
subito dopo STATEMENT S4 in main				

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella A

condizione	variabili locali al thread th_A		variabili globali	
	Aa	**Ab**	**Ga**	**sem_A**
subito dopo STATEMENT S1 in th_A	*2*	*X*	*4*	*0*
subito dopo STATEMENT S2 in th_A	*2*	*2*	*4 / 6*	*0*
subito dopo STATEMENT S3 in th_B	*U (NE / X / 2)*	*U (NE / X / 2)*	*3 / 6*	*0 / 1*
subito dopo STATEMENT S4 in main	*NE*			

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella A

condizione	variabili locali al thread th_A		variabili globali	
	Aa	**Ab**	**Ga**	**sem_A**
subito dopo STATEMENT S1 in th_A	*2*	*X*	*4*	*0*
subito dopo STATEMENT S2 in th_A	*2*	*2*	*4 / 6*	*0*
subito dopo STATEMENT S3 in th_B	*U (NE / X / 2)*	*U (NE / X / 2)*	*3 / 6*	*0 / 1*
subito dopo STATEMENT S4 in main	*NE*	*NE*		

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella B

condizione	variabili locali al thread th_B
	Ba
subito dopo STATEMENT S1 in th_A	
subito dopo STATEMENT S2 in th_A	
subito dopo STATEMENT S3 in th_B	
subito dopo STATEMENT S4 in main	

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella B

condizione	variabili locali al thread th_B
	Ba
subito dopo STATEMENT S1 in th_A	
subito dopo STATEMENT S2 in th_A	
subito dopo STATEMENT S3 in th_B	
subito dopo STATEMENT S4 in main	

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella B

condizione	variabili locali al thread th_B
	Ba
subito dopo STATEMENT S1 in th_A	
subito dopo STATEMENT S2 in th_A	
subito dopo STATEMENT S3 in th_B	
subito dopo STATEMENT S4 in main	

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella B

condizione	variabili locali al thread th_B
	Ba
subito dopo STATEMENT S1 in th_A	*U (NE / X / 3 / 9)*
subito dopo STATEMENT S2 in th_A	
subito dopo STATEMENT S3 in th_B	
subito dopo STATEMENT S4 in main	

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella B

condizione	variabili locali al thread th_B
	Ba
subito dopo STATEMENT S1 in th_A	*U (NE / X / 3 / 9)*
subito dopo STATEMENT S2 in th_A	*U (NE / X / 3 / 9 / 18)*
subito dopo STATEMENT S3 in th_B	
subito dopo STATEMENT S4 in main	

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella B

condizione	variabili locali al thread th_B
	Ba
subito dopo STATEMENT S1 in th_A	*U (NE / X / 3 / 9)*
subito dopo STATEMENT S2 in th_A	*U (NE / X / 3 / 9 / 18)*
subito dopo STATEMENT S3 in th_B	*9 / 18*
subito dopo STATEMENT S4 in main	

AXO – Architettura dei Calcolatori e Sistemi Operativi

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella B

condizione	variabili locali al thread th_B
	Ba
subito dopo STATEMENT S1 in th_A	*U (NE / X / 3 / 9)*
subito dopo STATEMENT S2 in th_A	*U (NE / X / 3 / 9 / 18)*
subito dopo STATEMENT S3 in th_B	*9 / 18*
subito dopo STATEMENT S4 in main	*U (NE / X / 3 / 9 / 18)*

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella C

condizione	variabili locali al thread main	variabili globali	
	Ma	Ga	sem_A
subito dopo STATEMENT S4 in main			

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella C

condizione	variabili locali al thread main	variabili globali	
	Ma	Ga	sem_A
subito dopo STATEMENT S4 in main			

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella C

condizione	variabili locali al thread main	variabili globali	
	Ma	Ga	sem_A
subito dopo STATEMENT S4 in main			

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella C

condizione	variabili locali al thread main	variabili globali	
	Ma	Ga	sem_A
subito dopo STATEMENT S4 in main			

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella C

condizione	variabili locali al thread main	variabili globali	
	Ma	Ga	sem_A
subito dopo STATEMENT S4 in main	*2*		

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella C

condizione	variabili locali al thread main	variabili globali	
	Ma	Ga	sem_A
subito dopo STATEMENT S4 in main	*2*	*U (3 / 4 / 6)*	

Prova di martedì 17 novembre 2009

esercizio n. 1 – thread e parallelismo

Si completino le tabelle, indicando gli state delle variabili globali e locali. Lo stato di una variabile può essere:

- intero, carattere, stringa (quando il valore è definito)
- X, quando la variabile non è ancora stata inizializzata
- NE, se la variabile può non esistere
- se la variabile si può trovare in due o più stati, li si riporti tutti quanti
- U quando la variable può trovarsi in tre o più stati o in qualsiasi combinazione

Si presti attenzione alla colonna "condizione". In particolare, con "subito dopo statement X" si intendono richiedere i valori che le variabili possono assumere tra lo statement X e lo statement immediatamente successivo del thread indicato nella condizione stessa.

```
pthread_mutex_t mutex_A = PTHREAD_MUTEX_INITIALIZER;
/* PTHREAD_MUTEX_INITIALIZER è equivalente a pthread_mutex_init */
sem_t sem_A;
int Ga = 1;
void * A (void * arg) {
     int Aa = (int) arg;
     int Ab;
     pthread_mutex_lock (&mutex_A);
     Ga = Aa * 2;                          /* STATEMENT S1 */
     Ab = Ga - 2;
     pthread_mutex_unlock (&mutex_A);      /* STATEMENT S2 */
     return (void *) Ab;
} /* thread A */
```

```
void * B (void * arg) {
     int Ba = (int) arg;
     pthread_mutex_lock (&mutex_A);
     Ga = Ga + 2;
     Ba = Ba * Ga;                         /* STATEMENT S3 */
     pthread_mutex_unlock (&mutex_A);
     sem_wait (&sem_A);
     Ga = 3;
     return NULL;
} /* thread B */
```

```
int main ( ) {
     pthread_t th_A, th_B;
     int Ma;
     sem_init (&sem_A, 0, 0);
     pthread_create (&th_A, NULL, A, (void *) 2);
     pthread_create (&th_B, NULL, B, (void *) 3);
     pthread_join (th_A, (void *) &Ma);
     sem_post (&sem_A);                    /* STATEMENT S4 */
     pthread_join (th_B, NULL);
     return Ma;
} /* thread principale */
```

Tabella C

condizione	variabili locali al thread main	variabili globali	
	Ma	Ga	sem_A
subito dopo STATEMENT S4 in main	*2*	*U (3 / 4 / 6)*	*0 / 1*

Alla prossima lezione

alessandro.nacci@polimi.it