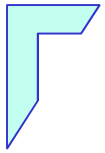
Esercitazione di Lab. di Sistemi Operativi 1 a.a. 2011/2012

- Thread -

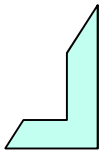
Sommario

- Thread:
  - Cosa è?
  - Creazione:
    - funzione: pthread_create
  - Terminazione esplicita
    - funzione: pthread_exit
  - Attesa terminazione
    - funzione: pthread_join

- Esercizi:
  - Thread



- Cosa è un Thread -



- Cosa è un Thread -

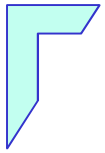
- Un "thread" è una parte di un processo, che quando viene creato in analogia ai processi, viene associato ad un pezzo di codice

- Se il processo ospite (e quindi il suo programma) è mono-thread il thread è associato all'intero programma.

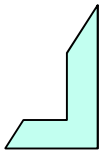
- Se il processo ospite è **multi-thread** ciascun thread di tale processo, è associato ad una funzione da eseguire chiamata funzione di avvio.

- Quando un programma viene mandato in esecuzione tramite una chiamata exec, viene creato un singolo thread detto thread principale

- Ulteriori thread vanno creati esplicitamente



- Thread: creazione -



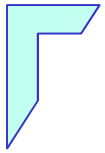
- Creazione Thread -

- Funzione "pthread_create"

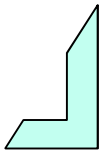
Per creare thread addizionali relativi ad uno stesso processo, Posix prevede la funzione:

```
#include <pthread.h>

int  pthread_create ( pthread_t *tid, const pthread_attr_t *attr,
                      void * ( *start_func) (void *), void *arg );
```

- se la chiamata ha successo, *tid* punta al thread ID;
- *attr* permette di specificare gli attributi del thread (se *attr* = NULL, gli attributi sono quelli di default);
- *start_func* è l'indirizzo della funzione di avvio;
- *arg* è l'indirizzo dell'argomento accettato dalla funzione di avvio;
- restituisce 0 in caso di successo, un intero positivo – secondo le convenzioni di <sys/errno.h> – in caso di errore.



- Thread: Terminazione esplicita -



- Terminare un Thread -

Un thread può richiedere esplicitamente la propria terminazione grazie alla chiamata seguente, lasciando traccia del proprio stato di terminazione per quei thread che attendono per lui:

```
#include <pthread.h>

void  pthread_exit ( void *status);
```

•*status* punta all'oggetto che definisce lo stato di terminazione del thread. Quest'ultimo non deve essere una variabile locale al thread chiamante, pena la sua scomparsa alla terminazione del thread stesso.

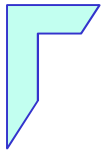
- Esempio 1 Creazione e TerminazioneThread -

```
#include <pthread.h>
void *thread(void *vargp);
int main() {
pthread_t tid;

   pthread_create(&tid, NULL, thread, NULL);
   exit(0);
}
/* thread routine */
void *thread(void *vargp) {
printf("Hello, world!\n");
pthread_exit((void*)status);
}
```

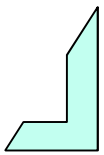
attributi Thread

argomenti Thread

Fa terminare il Thread corrente



- Thread: Attesa terminazione -



- Aspettare la terminazione di un Thread -

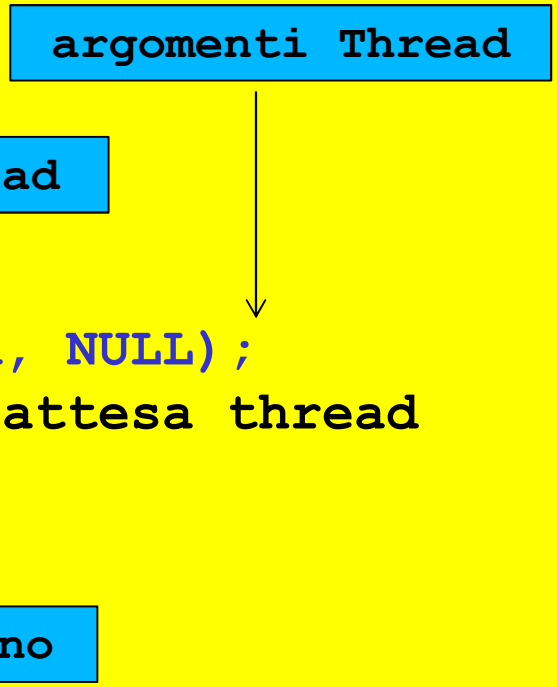
Un thread può attendere per la terminazione di un altro thread relativo allo stesso processo:

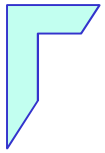
```
#include <pthread.h>

int  pthread_join ( pthread_t *tid, void **status );
```

- *tid* è l'ID del thread del quale si vuole attendere la terminazione;
- *status* punta al valore restituito dal thread per cui si è atteso, indicante il suo stato di terminazione (se *status* = NULL, tale stato non viene restituito);
- restituisce 0 in caso di successo, un intero positivo – secondo le convenzioni di <sys/errno.h> – in caso di errore.

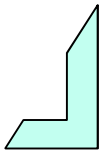
- Esempio 2 Creazione e attesa Thread -

```
#include<pthread.h>
#include<stdlib.h>
#include <unistd.h>
#include <stdio.h>
void *thread(void *vargp);
int main() {
pthread_t tid;
int i;
   pthread_create(&tid, NULL, thread, NULL);
   pthread_join(tid, (void **)&i);//attesa thread

   printf("%d\n",i);
   exit(0);
}
/* thread routine */
void *thread(void *vargp) {
printf("Hello, world!\n");
pthread_exit((void*)status);//termina il thread
}
```





- Esercizi: Thread
Creazione e Terminazione -



Esercizio n° 1 – Creazione e Terminazione Thread –

Scrivere un programma C che crei un "**Thread**" ed attende la terminazione dello stesso. Il thread scriverà a video in 20 secondi i numeri da 0 a 19.

Compilazione:

$ gcc nomeProgramma.c -lpthread

Opzione di compilazione che serve a linkare la libreria `pthread.h` nel programma

Esecuzione

$ ./a.out

Output

Scrivo il numero: 0 1 2,……..10,……19

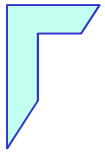
Soluzione Esercizio n° 1 - Thread -

Programma con due thread

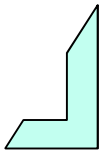
```
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
void *thread_function(void *arg)
{  int i;
   void *val;
  for ( i=0; i<20; i++ )
    { printf("Scrivo il Numero: %d\n",i);
      sleep(1);
    }
pthread_exit(val);}
int main(void)  //thread main o principale
{  pthread_t mythread;
    void *status;
if (pthread_create(&mythread, NULL, thread_function, NULL) )
{  printf("error creating thread.");
    exit(1);  }
printf("Sto aspettando la terminazione del thread\n");
if ( pthread_join ( mythread, (void*)&status ) ) {
    printf("error joining thread.");
exit(1);  }
exit(0);}
```

Esercizio n° 1 – Creazione e Terminazione Thread –

- Il main
  - dichiara una variabile di tipo **`pthread_t`** definito nella libreria **`pthread.h`** usata per tid (id thread)
  - chiama la funzione **`pthread_create`** per creare un nuovo processo thread associato alla funzione **`thread_function`**
- Quindi abbiamo due thread:
  - quello principale (main thread)
  - ed il nuovo thread (**`thread_function`**) che lavorano in parallelo
- Il main thread, continua nella sua esecuzione eseguendo l'istruzione successiva **`pthread_join`**, mentre il nuovo thread impiega 20 secondi prima di terminare
- Il main si addormenta in attesa che il nuovo thread termini



- Esercizi: multi-thread -

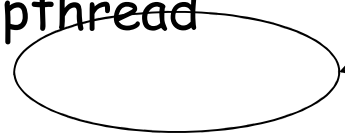


Esercizio n° 2 – Creazione Thread –

- Scrivere un programma C che crei 2 **Thread** indipendenti ciascuno dei quali esegue una fuznione che stampa a video il messaggio "Thread 1", "Thread 2" ed attenda la terminazione degli stessi.

Compilazione:

```
$ gcc nomeProgramma.c  -lpthread
```



Esecuzione

```
$ ./a.out
```

Output:

```
Thread 1
Thread 2
Thread 1 return:  0
Thread 2 return: 0
```

Soluzione Esercizio N° 2

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *print_message_function( void *ptr );
main(){
pthread_t thread1, thread2;
char *message1 = "Thread 1";
char *message2 = "Thread 2";
void *status;
pthread_create(&thread1, NULL, print_message_function, (void*) message1);
pthread_create(&thread2, NULL, print_message_function, (void*) message2);
pthread_join(thread1, (void*)&status);
pthread_join(thread2, (void*)&status);
printf("Thread 1 return: %d\n", (int)status);
printf("Thread 2 return: %d\n", (int)status);
exit(0);}
void *print_message_function( void *ptr ){
char *message;
void *val;
message = (char *) ptr;
printf("%s \n", message);
pthread_exit(val);
}
```

Esercizio n° 3 – Creazione Thread –

- Scrivere un programma C che crei 5 "**Thread**" ciascuno dei quali stampi a video il messaggio "**Hello Word sono il Thread #**" ed attenda la terminazione.

Compilazione:

$ gcc nomeProgramma.c -lpthread

Opzione di compilazione

Output:

Alternanza tra threads

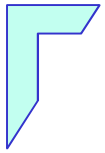
```
nain() : creazione thread #0
nain() : creazione thread #1
nain() : creazione thread #2
Hello Word sono il thread #1!
nain() : creazione thread #3
Hello Word sono il thread #0!
nain() : creazione thread #4
Hello Word sono il thread #2!
Hello Word sono il thread #3!
nain() : attesa per thread 0
Hello Word sono il thread #4!
nain() : attesa per thread 1
nain() : attesa per thread 2
nain() : attesa per thread 3
nain() : attesa per thread 4
```

Soluzione Esercizio N° 3 1/2

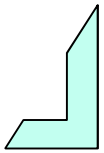
```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 5
void *PrintHello(void *threadid)
{   int tid;
    void *val;
    tid = (int)threadid;
    printf("Hello Word sono il thread #%d!\n", tid);
    pthread_exit(val);
}
int main (void){
pthread_t threads[NUM_THREADS];
int t;
void *status;
for(t=0; t<NUM_THREADS; t++){
printf("main(): creazione thread #%d\n", t);
if (pthread_create(&threads[t], NULL, PrintHello, (void *)t))
{
    printf("ERROR; from pthread_create()");
    exit(-1);}
}
```

Soluzione Esercizio N° 3 2/2

```
//Attesa terminazione threads
for(t=0; t<NUM_THREADS; t++){

    printf("main() : attesa per thread %d\n", t);

    if (pthread_join(threads[t],(void*)&status))
    {
        printf("ERROR; from pthread_join()");
        exit(-1);
    }

    printf("main(): terminato con status: %d\n", (int)status);
}

exit(0);

}/*Fine programma*/
```



- Esercizio multi-thread: con passaggio di più argomenti -



Esercizio n° 4 – Creazione Thread –

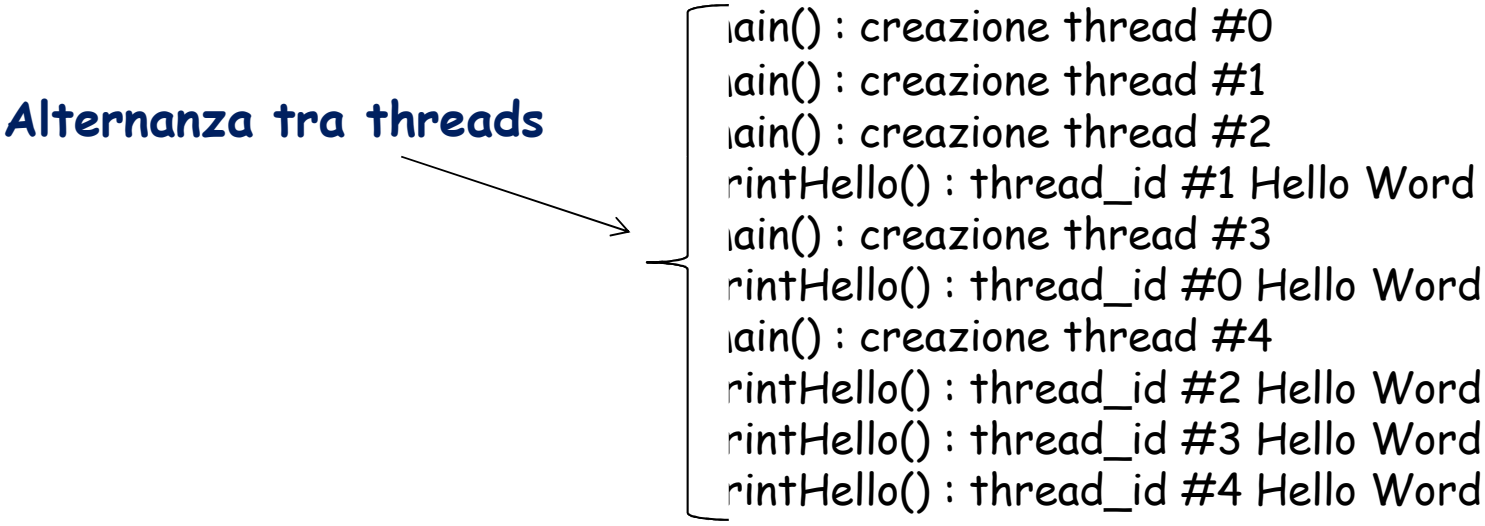
- Scrivere un programma C che crei 5 "**Thread**" a cui vengono passati tramite una struttura due parametri: il numero del thread ed un messaggio

Compilazione:

$ gcc nomeProgramma.c -lpthread

Output:

Alternanza tra threads



Soluzione Esercizio N° 4 1/2

```
#include <unistd.h>
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>
#define NUM_THREADS  5

struct thread_data{
int  thread_id;
char *message;};

void *PrintHello(void *threadarg){

struct thread_data *my_data;

my_data = (struct thread_data *) threadarg;

printf("PrintHello() : thread_id #%d Messagge %s\n",
       my_data->thread_id,my_data->message);

pthread_exit(NULL);

}
```

Soluzione Esercizio N° 4 2/2

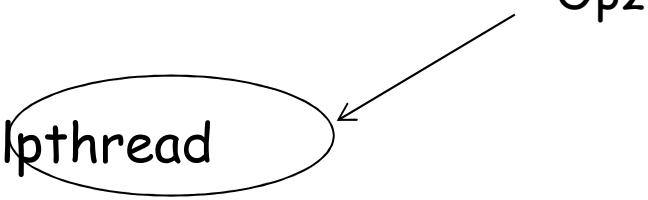
```
int main ()
{
pthread_t threads[NUM_THREADS];
struct thread_data td[NUM_THREADS];
int i;
for( i=0; i < NUM_THREADS; i++ ){
printf("main() : creazione thread #%d\n", i);
td[i].thread_id = i;
td[i].message = "Hello Word";
if (pthread_create(&threads[i], NULL,PrintHello,
   (void*)&td[i]))
{
   printf("ERROR; from pthread_create()");
   exit(-1);
}
}
pthread_exit(NULL);
}
```

Esercizio n° 5 – Thread –

- Scrivere un programma C che accetta un numero intero <**n**> da riga di comando, crea un Thread che sommerà ad una variabile globale intera di valore **m** il valore di **n** ed infine ne stampa il risultato a video.

Compilazione:

Opzione di compilazione



```
$ gcc nomeProgramma.c  -lpthread
```

Esecuzione

```
$ ./a.out <n>
```

Output

La somma di m + <n> è: xxx

Soluzione Esercizio n° 5 - Thread -

```
#include <unistd.h>
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>
int m=10; /* Variabile globale modificata dal thread */

void *somma(void* arg) {//Funzione eseguita dal thread

    int n = (int)arg;
    void *val;
    printf ("La somma di m + n è: %d\n", m + n);
    sleep(2); /*Aspetta due secondi prima di terminare*/
    pthread_exit(val);

}
```

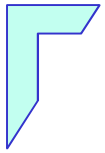
Permette di passare alla funzione **somma** strutture di diverso tipo

cast ad int del tipo void passato

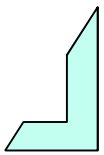
Soluzione Esercizio n° 5 - Thread -

```
main(int argc, char **argv) {
    pthread_t t;
    void *status;
    int arg;
    if (argc<2) {
        printf ("uso: nome_programma <valore> \n");
        exit(1);
    }
    arg=atoi(argv[1]);/*valore da sommare*/
if (pthread_create(&t, NULL, somma, (void*)arg) != 0) {
        printf ("Errore nella creazione del nuovo
         thread\n");
        exit(1);
    }
    // Attendo che il thread venga terminato
    pthread_join(t, (void*)&status);
    printf ("Il thread è terminato con status %d\n",
    (int)status);
}
```

Passaggio di parametri alla funzione **somma**



- Esercizio Thread: condivisione di variabili -



Esercizio n° 6 – Thread – (condivisione di var.)

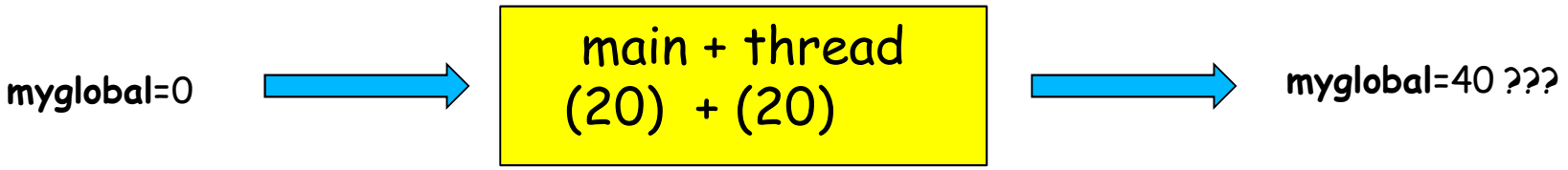
Scrivere un programma C che crei un **Thread** il quale insieme al programma main incrementa di venti volte una variabile globale (myglobal). Il thread chiama una funzione "**incrementa**" Supporremo che l' accesso alla variabile globale non venga regolamentato da alcun meccanismo di gestione della concorrenza.

Compilazione:

```
$ gcc nomeProgramma.c  -lpthread
```

Esecuzione

```
$ ./a.out
```



Soluzione Esercizio n° 6 - Thread -

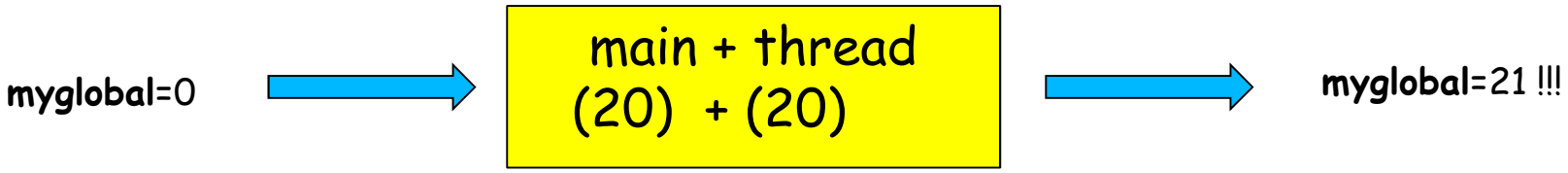
```
#include <pthread.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
int myglobal;//Variabile globale

void *thread_incrementa(void *arg)
{  int i,j;
   void *val;
for ( i=0; i<20; i++ )
{
   j=myglobal; //variabile locale temporanea
   j=j+1;
   printf("thread_incrementa\n");
   fflush(stdout);
   sleep(1); //Si addormenta per 1 secondo
   myglobal=j;
}
pthread_exit(val);
}
```

Soluzione Esercizio n° 6 - Thread -

```
int main(void)
{
pthread_t mythread;
int i;
void *status;
if(pthread_create(&mythread,NULL,thread_incrementa,NULL) )
{ printf("error creating thread.");
    exit(1);
}
for ( i=0; i<20; i++) {
myglobal=myglobal+1;
printf("main\n");
fflush(stdout);
sleep(1); //Si addormenta per 1 secondo
}
if(pthread_join(mythread,(void*)&status)){
printf("error joining thread.");
exit(1);  }
printf("nmyglobal uguale %d\n",myglobal);
exit(0);
}
```

Esercizio n° 6 - Thread -

- Il risultato è:



Perché???

```
void *thread_incrementa(void *arg)
{  int i,j;
   void *val;
for ( i=0; i<20; i++ )
{
   j=myglobal; //variabile locale temporanea
   j=j+1;
   printf("thread_incrementa\n");
   fflush(stdout);
   sleep(1); //Si addormenta per 1 secondo
   myglobal=j;
}
pthread_exit(val);
}
```

Esercizio n° 6 - Thread -

- La funzione **thread_incrementa**:
  1. copia myglobal in una variabile locale j
  2. incrementa per venti volte tale variabile
  3. si addormenta per un secondo (sleep(1))
  4. copia il valore della variabile j in myglobal

- Il **main**:
  1. Incrementa per venti volte la variabile myglobal in maniera parallela al thread

- Il punto (4) della funzione **thread_incrementa** provoca una sovrascrittura del valore presente in myglobal relativo al main (annulla ripetutamente l'incremento di myglobal da parte del main)

Esercizio n° 6 – Thread –

- Soluzioni al problema:

1. Evitare di modificare il valore di una variabile globale passando per una variabile locale e quindi incrementare direttamente la stessa (soluzione non corretta anche se nel nostro caso funziona)

2. Sincronizzazione di thread: i **mutex** (nella prossima lezione)

Esercizio n° 7 - Thread - (condivisione di var.)

- Scrivere un programma C che crea due Thread "**somma1**" e "**somma2**", entrambi accedono alle variabili **test.a** e **test.b** di una struttura dati **test** condivisa incrementandole di 1 per 10 volte, aspettano 2 secondi prima di stampare a video i valori delle due variabile. Supporremo che tale accesso non venga regolamentato da alcun meccanismo di gestione della concorrenza.

Compilazione:

$ gcc nomeProgramma.c -lpthread

Esecuzione

$ ./a.out

Soluzione Esercizio n° 7 - Thread -

```
#include <stdio.h>
#include <pthread.h>
#include <unistd.h>
#include <stdlib.h>
#define CICLI 10 /*Costante usata per incrementare le
variabili*/
struct test {/* Memoria Condivisa fra i thread */
  int a;
  int b;
} mytest;
/* dichiarazione delle due funzioni dei due thread*/
void *somma1(void *);
void *somma2(void *);
int main(void)
{ pthread_t som1TID, som2TID;
  void *status;
  /* Inizializzo la memoria condivisa */
  mytest.a = 0;
  mytest.b = 0;
```

Soluzione Esercizio n° 7 - Thread -

```
/* A questo punto posso creare i thread  */
  if (pthread_create(&som1TID, NULL, somma1, NULL) != 0) {
        printf ("Errore nella creazione del thread somma1\n");
        exit(1);
    }
  /* A questo punto posso creare i thread .... */
  if (pthread_create(&som2TID, NULL, somma2, NULL) != 0) {
        printf ("Errore nella creazione del thread somma2\n");
        exit(1);
    }
  /* A questo punto aspetto che i due thread finiscano */
  pthread_join(som1TID, (void*)&status);
  pthread_join(som2TID, (void*)&status);

  printf("E' finito il programma ....\n");
  exit (0);
}
```

Soluzione Esercizio n° 7 - Thread -

```
void *somma1(void *in) //Funzione che verrà eseguita dal
thread 1
{
  int i;
  void *val
  for(i=0; i<CICLI; i++) {
    mytest.a++;
    mytest.b++;
    /* sleep di 2 secondi */
    sleep(2);
    printf("somma1 -- a = %d \n", mytest.a);
    printf("somma1 -- b = %d \n", mytest.a);
  }
  pthread_exit(val);
}
```

Soluzione Esercizio n° 7 - Thread -

```
void *somma2(void *in) //Funzione che verrà eseguita dal
thread 2
{
  int i;
  void *val
  for(i=0; i<CICLI; i++) {
    mytest.a++;
    mytest.b++;
    /* sleep di 2 secondi */
    sleep(2);
    printf("somma2 -- a = %d \n", mytest.a);
    printf("somma2 -- b = %d \n", mytest.b);
  }
  pthread_exit(val);
}
```

- Fine Esercitazione -