Laboratorio di Sistemi Operativi, A.A. 2022–2023

Simulazione di esercitazione

Lunedì 29 maggio 2023

Esercizio 1

Scrivere un programma C che, date alcune costanti definite a inizio file

```
#define NOME_FILE "/dev/random"
#define DIMENSIONE_VETTORE 10000
#define NUMERO_THREAD 10
#define ELEMENTI_PER_THREAD (DIMENSIONE_VETTORE/NUMERO_THREAD)
```

legge un array di `DIMENSIONE_VETTORE` interi senza segno dal file `NOME_FILE`. Si supponga che il file contenga già l'immagine dell'array come rappresentata in memoria (come prova utilizzeremo per l'appunto `/dev/random` che fornisce valori casuali).
Lo scopo del programma è calcolare la somma degli interi letti dal file suddividendola in 10 somme parziali, ciascuna delle quali calcolata da un diverso thread e accumulata in una variabile condivisa contenente la somma totale.
Il programma deve avviare `NUMERO_THREAD` thread, ciascuno dei quali calcolerà la somma di una sottosequenza dell'array:

- Il thread 0 calcola la somma degli elementi con indici $0 \ldots$ `ELEMENTI_PER_THREAD` $- 1$;
- il thread 1 calcola la somma degli elementi con indici `ELEMENTI_PER_THREAD` $\ldots 2 \cdot$ `ELEMENTI_PER_THREAD` $- 1$;
- il thread i calcola la somma degli elementi con indici $i \cdot$ `ELEMENTI_PER_THREAD` $\ldots (i+1) \cdot$ `ELEMENTI_PER_THREAD` $- 1$;

Ogni thread dovrà, una volta calcolata la somma parziale di sua competenza, sommarla a un totale condiviso.
Quando tutti i thread saranno terminati, il programma principale dovrà stampare il totale calcolato.
Infine, per verificare il risultato dovrà ripetere il calcolo in modo sequenziale:

- se i due risultati coincidono, stamperà la frase "`I due totali sono uguali`" e restituirà il valore 0 (successo) alla shell;
- altrimenti stamperà "`I due totali sono diversi`" e restituirà un valore non nullo.

Per avere punteggio pieno, il programma non dovrà definire variabili globali: tutti gli oggetti condivisi fra i thread e il programma principale dovranno essere passati in una struttura di parametri. Inoltre, si dovrà utilizzare un mutex per regolare l'eventuale modifica di variabili condivise, che dovrà essere comunque ridotta allo stretto necessario.

Esercizio 2

Realizzare lo script bash `test.sh` che riceva come parametro il numero di test da effettuare ed esegua il programma dell'esercizio 1 un numero corrispondente di volte, verificando il codice di uscita del processo per determinarne il successo o il fallimento.
In caso di successo, l'output prodotto dallo script dovrà essere simile al seguente:

```
# ./test.sh 3
Test 1
Riuscito.
Test 2
Riuscito.
Test 3
Riuscito.
Tutti i test sono riusciti.
```

Il test dovrà fermarsi al primo fallimento:

```
# ./test.sh 3
Test 1
Riuscito.
Test 2
Fallito.
```

Per avere punteggio pieno, lo script dovrà stampare esclusivamente le linee indicate e dovrà essere invocato come `./test.sh 3` senza menzionare esplicitamente la shell nella riga di comando.

Esercizio 3

Realizzare un Makefile con i seguenti tre obiettivi:

- `somma` (obiettivo di default) crea l'eseguibile dell'esercizio 1;
- `test` partendo dai soli sorgenti arriva a eseguire lo script di test con 10 iterazioni;
- `clean` cancella tutti i file eseguibili, lasciando solo i file sorgenti scritti dallo studente.

Indicazioni utili

Le seguenti informazioni servono solo a facilitare l stesura del codice, ma non sono in alcun modo obbligatorie.

- Comandi bash utili (manuale accessibile con `man 1` *nomecomando*):
  `echo`.
- Comandi bash built-in utili (manuale accessibile con `help` *nomecomando*):
  `exit`, `for`, `if`.
- Chiamate di sistema utili (manuale accessibile con `man 2` *nomefunzione*):
  `open`, `read`, `close`, `pthread_mutex_init`, `pthread_mutex_lock`, `pthread_mutex_unlock`, `pthread_start`, `pthread_join`.
- Altre funzioni utili (manuale accessibile con `man 3` *nomefunzione*):
  `printf`, `puts`.
- I seguenti file header dovrebbero essere sufficienti per tutte le funzioni utilizzate nell'esercitazione:

```
#include <stdio.h>
#include <pthread.h>
#include <fcntl.h>
#include <unistd.h>
```