Sistemi Operativi – a.a. 2023/2024
prova di laboratorio
– 4 ottobre 2024 –

Creare un programma **`calc-verifier-3.c`** in linguaggio C che accetti invocazioni sulla riga di comando del tipo:

`calc-verifier-3 <calc-file-1> <calc-file-2> ... <calc-file-`*`n`*`>`

Il programma dovrà essere in grado di verificare la correttezza di una sequenza di semplici operazioni matematiche descritte nei file testuali forniti in input.

Ogni file contiene:
- nella prima riga il valore iniziale del calcolo;
- una serie di righe del tipo `<operazione valore>`, dove operazione può essere una tra (`+` , `-` , `x`) e valore un valore intero da usare come secondo operando insieme al valore ottenuto dall'operazione precedente;
- una ultima riga contenente il valore finale atteso.

Alcuni esempi sono forniti a corredo: `calc1.txt`, `calc2.txt`, `calc3.txt`.

Al suo avvio il programma creerà *n+1* thread che dovranno lavorare in parallelo:
- *n* thread `FILE-i` che si occuperanno di leggere il rispettivo file in input e di coordinare il calcolo/verifica;
- un unico thread `CALC` che avrà il compito di supportare gli altri thread per applicare i singoli calcoli quando necessario.

I thread si coordineranno tramite generici semafori e/o mutex: da usare in numero minimo e modalità opportune da determinare da parte dello studente. L'unica struttura dati condivisa dai thread avrà il seguente contenuto di massima:
- gli strumenti di coordinamento (semafori e/o mutex) previsti;
- un buffer di richieste con una capienza esatta pari a *`MAX_REQUESTS=5`* record;
- un vettore di *n* risposte numeriche di tipo `long long` (una per ogni thread `FILE-i`);
- il numero di thread che hanno completato il proprio compito;
- il numero di verifiche andate a buon fine.

Il record che rappresenta una richiesta conterrà le seguenti informazioni:
- i due operandi numerici di tipo `long long`;
- il tipo di operazione da applicare agli operandi;
- l'identificativo *`i`* del thread a cui restituire il risultato.

Un thread di tipo `FILE-i` dovrà leggere il valore iniziale dalla prima riga e, per ogni operazione incontrata, creare una richiesta corrispondente da inserire nel buffer condiviso e attendere la risposta. Come ultima riga del file (quindi senza segno d'operazione) troverà il risultato atteso: il thread riporterà l'esito del controllo sul proprio standard output. Il thread `CALC` dovrà estrarre richieste dal buffer condiviso e, una volta computato il calcolo richiesto, inserire il risultato nel vettore delle risposte nella posizione corrispondente al richiedente che dovrà anche essere risvegliato.

Il programma dovrà funzionare con un qualunque numero di file in input (anche superiore a `MAX_REQUESTS`). Tutti i thread secondari dovranno terminare spontaneamente alla fine dei lavori e alla fine il thread principale dovrà visualizzare un riepilogo sul numero di verifiche andate a buon fine. Non si dovranno usare strutture dati con visibilità globale nel codice e si dovrà cercare di rispettare la struttura dell'output riportato nell'esempio a seguire.

Tempo: 2 ore e 15 minuti

L'output atteso sui file di esempio è il seguente:

```
$ ./calc-verifier-3 calc1.txt calc2.txt

[FILE-1] file da verificare: 'calc1.txt'
[FILE-2] file da verificare: 'calc2.txt'
[FILE-1] valore iniziale della computazione: 7
[FILE-1] prossima operazione: '+ 30'
[CALC] calcolo effettuato: 7 + 30 = 37
[FILE-1] risultato ricevuto: 37
[FILE-2] valore iniziale della computazione: 3
[FILE-2] prossima operazione: '+ 6'
[CALC] calcolo effettuato: 3 + 6 = 9
[FILE-2] risultato ricevuto: 9
[FILE-1] prossima operazione: '+ 4'
[CALC] calcolo effettuato: 37 + 4 = 41
[FILE-1] risultato ricevuto: 41
[FILE-2] prossima operazione: '- 5'
[CALC] calcolo effettuato: 9 - 5 = 4
[FILE-2] risultato ricevuto: 4
[FILE-1] prossima operazione: '+ 3'
[CALC] calcolo effettuato: 41 + 3 = 44
[FILE-1] risultato ricevuto: 44
[FILE-1] prossima operazione: 'x 6'
[CALC] calcolo effettuato: 44 x 6 = 264
[FILE-1] risultato ricevuto: 264

   ...

[FILE-2] prossima operazione: 'x 3'
[CALC] calcolo effettuato: -1704708437975 x 3 = -5114125313925
[FILE-2] risultato ricevuto: -5114125313925
[FILE-2] computazione terminata in modo corretto: -5114125313925
[FILE-1] prossima operazione: '- 2'
[CALC] calcolo effettuato: 3512341814982522 - 6 = 3512341814982520
[FILE-1] risultato ricevuto: 3512341814982520
[FILE-1] computazione terminata in modo corretto: 3512341814982520
[MAIN] verifiche completate con successo: 2/2
```