



Laborator 03

Exerciții

1. Compilați și rulați codul din **raceCondition.c**.
 - o Parametrii sunt: (la fel pentru restul programelor)
 - **N** numărul de elemente (argument 1).
 - **printLevel** controlează câte elemente sunt printate (argument 2).
 - **P** numărul de thread-uri (argument 3). **Este suprascris cu 2.**
 - o **Atenție:** Compilați fără **-O3** (optimizările vor șterge mai tot)
 - o Ce rezultat obțineți?
 - o Dar cu **N** mare? (1000-10.000)
 - o Dar rulat de câteva ori cu **N = 1**?
2. Rulați programul folosind scriptul **testCorrectnessIntensive.sh**.
 - o Parametrii:
 - Program secvențial pentru comparație (poate fi și cel paralel)
 - Program paralel ce va fi rulat de multe ori
 - **N** va fi dat ca parametru programelor secvențial și paralel
 - Numărul de rulări ale programului paralel
 - Opțional, numărul de thread-uri (default 1 2 4 8)
 - o Vom rula cu **N** mic (chiar 1) și Număr de rulări mare (10.000)
 - o Care este rezultatul?
3. Rezolvați **race condition**-ul folosind mutex.
- o Testați iarăși cu scriptul.
4. Modificați programul **barrier.c**.
 - o Se dorește ca mereu afișarea să se facă într-o anumită ordine.
 - o Se vor folosi doar bariere.
 - o Programul este gata doar după ce este testat cu scriptul oferit (10.000).
5. Modificați programul **semaphoreSignal.c**.
 - o Se dorește ca mereu afișarea să se facă într-o anumită ordine.
 - o Se vor folosi doar semafoare.
 - o Programul este gata doar după ce este testat cu scriptul oferit (10.000).
6. Rezolvați problema din **deadLock1.c** fără a scoate lock de pe mutex.
- o De ce avem dead lock?
7. Rezolvați problema din **deadLock2.c** fără a scoate lock de pe mutexA și mutexB.
 - o De ce avem dead lock?
 - o Mai avem și dacă scoatem acele sleep-uri?
 - o Rulați de foarte multe ori cu scriptul pentru a fi siguri (100.000)
8. Rezolvați problema din **deadLock3.c** fără a scoate lock de pe mutex.
 - o De ce avem dead lock?
 - o **ATENȚIE:** În unele limbaje se poate face lock pe un mutex de pe același thread care a făcut o dată lock.
9. Paralelizați programul **sumVectorValues.c**.
 - o Măsurăți timpii de execuției ai variantei secvențiale și variantei voastre.

- la **N** mic nu intampinam probleme,
 - la **N** mare, sunt probleme
 Obs: scriptul nu mai șterge fisierele de out.x.x, ar trebui sa le stearga si sa le lase doar pe cele diferite

-> pe la testul 200 se blocheaza (fara sleepuri)



- Timpul de execuție se măsoară cu time scris înainte de linia de rulare a programului.
- Avem nevoie de timpi mari deci vom rula cu 1.000.000.000 .
- Nu este obligatoriu să obțineți scalabilitate.
- Dacă nu obțineți scalabilitate, explicați de ce nu?
- Exercițiul este gata doar după ce ați testat cu scriptul (N și număr de rulări mari aproximativ 10.000)

Exercițiile de la 1 la 9 sunt obligatorii. Conceptele explorate sunt esențiale pentru obținerea notei **minime** de promovare.

Vă recomandăm, pentru a crește șansele de a obține o notă cât mai mare să explorați și următoarele exerciții:

10. Implementați programul precedent în așa fel încât să scaleze.
 - Hint: Mai multe sum.
11. Paralelizați programul **prepStrassen.c**.
 - Veți folosi mai multe funcții diferite pentru thread-uri.
 - Fiind funcții diferite, numărul de thread-uri poate fi fix.
 - Se vor folosi doar bariere pentru sincronizare.
 - Extrem de mare atenție la dependențe read-read, read-write, write-write.
 - Afișarea trebuie să fie identică la oricât de multe rulări.