Part II – test programare Seria: 3

a) Specificație (cerințe):

i) Scrieți un program C care să poată fi rulat în mod **cooperativ**. Acesta va primi ca și argumente de intrare calea către un fișier BINAR (posibil inexistent) și calea către un fișier text (cu date de intrare).
Alternativ, programul va suporta invocarea cu argumentul "--words", caz în care va afișa textual conținutul fișierului binar primit ca parametru.
Fișierul text va conține, pe fiecare linie, câte un string ce reprezintă un "cuvânt".

Structura fișierului binar este următoarea: o secvență de (maxim) 10 de înregistrări consecutive (inițial este secvența vidă), unde fiecare înregistrare este formată dintr-un *char* (adică un singur caracter), urmat de o singură valoare de tip *unsigned int* (stocată în formatul de reprezentare internă/binară a întregilor) ce reprezintă un număr, a cărui semnificație este descrisă mai jos.

Programul va citi, pe rând, câte un cuvânt din fișierul text și va căuta **doar** în directorul curent de lucru (NU recursiv) toate fișierele de tip normal, al căror nume de fișier conține acel cuvânt și pentru care utilizatorul curent are drepturi de citire și scriere.
Pentru fiecare fișier de tip normal găsit în acest mod, programul va calcula numărul de apariții ale fiecărui caracter cifră din intervalul '0',...,'9' în conținutul fișierului. Apoi va actualiza, în fișierul binar specificat, numărul întreg din înregistrarea corespunzătoare acelui caracter, prin adunarea sa cu numărul de apariții calculat anterior. Dacă fișierul binar specificat nu conține o înregistrare corespunzătoare acelui caracter cifră, atunci programul va adăuga o nouă înregistrare la finalul acestuia, ce va conține caracterul cifră și numărul de apariții calculat.

Accesul la fișierul binar va fi realizat în manieră **sincronizată**. Cooperarea mai multor instanțe ale programului rulate în paralel trebuie să fie eficientă, ceea ce implică punerea de blocaje în mod eficient, adică pe o durată minimală și pe porțiunea minimală din fișierul binar.

Un posibil exemplu de rulare:

```
./sum freq.bin words1.txt & ./sum freq.bin words2.txt & ./sum freq.bin words3.txt &
```

Se vor folosi doar apeluri POSIX pentru toate operațiile cu fișiere, cu excepția doar a operațiilor de citire a cuvintelor din fișierele text de intrare și, respectiv, de afișare pe ecran a rezultatului final obținut în fișierul binar, pentru care puteți folosi, la alegere, fie apeluri POSIX, fie apeluri de funcții din biblioteca libc/stdio.
Se vor trata în mod corespunzător toate erorile survenite la apelurile de funcții POSIX.

ii) Scrieți un script bash de automatizare, care va face următoarele:
a) Mai întâi scriptul va citi un număr *ni* (=numărul de instanțe), apoi va inițializa fișierul binar (resetare la zero), și eventual va compila sursa C (dacă nu găsește executabilul).
b) Apoi va lansa în execuție un job SPMD, prin care să se execute în paralel *ni* instanțe ale programului de la pct.i), având ca parametri același fișier binar și câte un fișier text pentru fiecare dintre cele *ni* instanțe.
c) Iar după terminarea execuției celor *ni* instanțe scriptul va apela programul de la pct.i) cu parametrul "--words" și numele fișierului binar, pentru a afișa rezultatele obținute în fișierul binar.

b) Baremul pentru program:

– implementarea corectă a parcurgerii directorului curent de lucru (nerecursivă): 1p
– implementarea corectă a cerințelor de procesare, creare și actualizare a fișierului binar: 6p
– implementarea mecanismelor de sincronizare pentru corectitudinea cooperării jobului SPMD: 3p
– implementarea în manieră eficientă a sincronizării: 2p
– implementarea corectă a procesării opțiunii "--words": 1p
– tratarea tuturor erorilor la apelurile de funcții POSIX: 1p

Baremul pentru script:

– implementarea corectă a cerinței a) formulate pentru script: 1.5p
– implementarea corectă a cerinței b) formulate pentru script: 1p
– implementarea corectă a cerinței c) formulate pentru script: 1p