Test practic la SO – varianta nr. 2

Realizați o aplicație formată din următoarele trei programe cooperante, plus un script de execuție a lor și un fișier cu date de intrare în format text, care vor fi plasate conform ierarhiei de directoare de mai jos:

```
.
├── starter.sh
├── input_data.txt
├── coordonator
│   └── supervisor.c
└── subordinates
    └── worker1.c
    └── worker2.c
```

1. Scriptul "starter.sh" va implementa funcționalitatea descrisă în specificația următoare:

i) Scriptul va primi în linia de comandă un parametru *p*, având valoarea 1 sau 2, iar în caz contrar va afișa un mesaj de eroare adecvat și se va termina.
ii) Dacă *p*=1, atunci scriptul va porni mai întâi execuția programului **worker1** (din subdirectorul corespunzător) în *background*, iar după o pauză de 2 secunde, va porni și execuția programului **worker2** (din subdirectorul corespunzător).
iii) Iar dacă *p*=2, atunci scriptul va porni mai întâi execuția programului **worker2** (din subdirectorul corespunzător) în *background*, iar după o pauză de 3 secunde, va porni și execuția programului **worker1** (din subdirectorul corespunzător).
iv) În ambele situații, programul **worker1** va fi pornit cu un argument în linia de comandă: calea (absolută sau relativă) către fișierul de intrare "input_data.txt" (pe care-l veți crea anterior, în orice editor de texte doriți, cu formatul specificat mai jos, în enunțul programului supervisor).
v) Apoi scriptul va aștepta terminarea execuției celor trei programe și va afișa conținutul mapării nepersistente cu nume, după care va face *curățenie*: va “distruge” maparea nepersistentă cu nume și canalul fifo.

2. Programul "supervisor.c" va implementa funcționalitatea descrisă în specificația următoare:

Programul va primi un argument în linia de comandă, ce reprezintă calea (absolută sau relativă) către un fișier existent pe disc, numit "input_data.txt". Acest fișier conține, pe fiecare linie, câte o tripletă de numere reale în format textual, separate prin spații: "a b c".
i) În funcția principală a programului, acesta va testa existența unui argument primit în linia de comandă și va face inițializările necesare pentru a putea trimite informații procesului **worker1** printr-un canal anonim (prin comunicații unu-la-unu) și, respectiv, pentru a putea primi rezultate de la procesul **worker1** printr-un canal fifo (prin comunicații unu-la-unu).
ii) Într-o funcție separată, apelată din funcția main, programul va citi pe rând, una câte una, fiecare linie din acel fișier input_data.txt și va converti fiecare tripletă citită la reprezentarea în format binar a numerelor reale, iar rezultatul conversiei (i.e., cele trei numere reale a, b, c în format binar) îl va transmite către procesul **worker1** prin acel canal anonim, folosind apeluri POSIX (așadar, folosind reprezentarea binară a numerelor reale, veți transmite mesaje de lungime constantă).
iii) Într-o altă funcție separată, apelată din funcția main, programul va citi, folosind apeluri POSIX, fiecare cvintet de 5 numere reale a,b,c,P,A transmis lui de către procesul **worker1** prin acel canal canal fifo și, dacă P!=0, afișează pe ecran mesajul: tripleta a,b,c reprezintă lungimile laturilor unui triunghi ce are perimetrul P și aria A, iar dacă P=0, afișează pe ecran mesajul: tripleta a,b,c nu poate reprezenta lungimile laturilor unui triunghi. De asemenea, va contoriza câte triplete a,b,c pot reprezenta lungimile laturilor unui triunghi și câte nu pot, afișând la final un mesaj adecvat cu cele două totaluri.

3. Programul "worker1.c" va implementa funcționalitatea descrisă în specificația următoare:

i) În funcția principală a programului, acesta va crea un proces fiu, iar în fiul creat va starta, printr-un apel exec adecvat, programul **supervisor**, transmițându-i ca parametru calea către fișierul cu date de intrare.
ii) De asemenea, tot în funcția main, va face inițializările necesare pentru a putea primi informații de la

procesul **supervisor** printr-un canal anonim (prin comunicații unu-la-unu) și, respectiv, pentru a-i putea trimite rezultate înapoi printr-un canal fifo (prin comunicații unu-la-unu).
iii) De asemenea, tot în funcția main, va face inițializările necesare pentru a putea schimba informații cu procesul **worker2**, în ambele sensuri, printr-un singur obiect de memorie partajată – o mapare nepersistentă cu nume, creată cu funcția *shm_open*.
iv) Într-o funcție separată, apelată din funcția main, programul va citi din acel canal anonim, folosind apeluri POSIX, fiecare tripletă a,b,c transmisă lui de către procesul **supervisor** (prin mesaje de lungime constantă, folosind reprezentarea binară a numerelor reale), și va verifica dacă cele trei numere reale a, b, c ar putea fi lungimile laturilor unui triunghi (i.e., condițiile: a,b,c > 0 și suma oricăror două să fie mai mică decât al treilea). Rezultatul acestei verificări va fi transmis mai departe către procesul **worker2,** prin intermediul acelei mapări nepersistente cu nume, sub forma a 4 numere a,b,c,r , unde a,b,c sunt cele trei numere reale, iar r este o valoare întreagă egală cu 0, dacă nu formează triunghi, respectiv 1, dacă formează triunghi (se va păstra reprezentarea în binar pentru toate cele 4 numere).
v) Într-o altă funcție separată, apelată din funcția main, programul va citi, rând pe rând, fiecare cvartet de forma a,b,c,p transmis lui de către procesul **worker2,** prin intermediul acelei mapări nepersistente cu nume**,** și va procesa informatia astfel: dacă p!=0, atunci calculează perimetrul P = 2*p , precum și aria triunghiului, după formula lui Heron: A = sqrt(p*(p-a)*(p-b)*(p-c)) , iar dacă p=0, atunci va asigna P=0 și A=0. Apoi va transmite rezultatul acestei procesări, sub forma a 5 numere reale a,b,c,P,A , către procesul **supervisor,** prin acel canal fifo, folosind apeluri POSIX (așadar, folosind reprezentarea binară a numerelor reale, veți transmite mesaje de lungime constantă).

4. Programul "worker2.c" va implementa funcționalitatea descrisă în specificația următoare:

i) În funcția principală a programului, va face inițializările necesare pentru a putea comunica cu procesul **worker1**, în ambele sensuri, prin acea mapare nepersistentă cu nume descrisă în specificația programului **worker1**.
ii) Într-o funcție separată, apelată din funcția main, programul va citi, rând pe rând, fiecare cvartet de forma a,b,c,r transmis lui de către procesul **worker1** prin intermediul acelei mapări nepersistente cu nume și, dacă r=1, atunci va calcula valoarea p = semi-perimetrul triunghiului cu laturile a,b,c, iar dacă r=0, atunci va seta p=0. Apoi va transmite rezultatul acestui calcul, sub forma a 4 numere a,b,c,p către procesul **worker1**, prin intermediul aceleiași mapări nepersistente cu nume (se va păstra reprezentarea în binar pentru toate cele 4 numere).

Recomandare:

Mai întâi, desenați de mână pe o foaie de hârtie o schemă cu **arhitectura aplicației**: ierarhia de procese, mijloacele de comunicație între procese și sensul de transmitere a informației prin aceste mijloace de comunicație (precum ați văzut în exemplele de diagrame realizate de mână, atașate la unele exerciții rezolvate în suporturile online de laborator).
Schema vă va ajuta să vă clarificați mai bine cerințele specificate în enunțul problemei și, de asemenea, vă va ajuta la partea de implementare!
De asemenea, citiți mai întâi și baremul asociat acestui subiect, pentru a vă clarifica mai bine cerințele specificate în enunțul problemei și modul în care trebuie să fie implementate.

Submitere:

La finalul testului, pentru a submite rezolvarea dvs. printr-un formular Google ce vă va fi indicat, veți face o arhivă (în format .zip) care să conțină cele trei programe sursă C (plus eventuale fișiere header .h proprii, în cazul în care veți defini și folosi astfel de fișiere), scriptul starter.sh și fișierul cu datele de intrare, **cu numele lor și poziția în ierarhia de directoare exact precum au fost specificate** în enunțul de mai sus. Iar arhiva o veți denumi în felul următor: TP2_NumePrenume.zip