Test practic la SO – varianta nr. 1

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
├── manager
│   └── supervisor.c
└── workers
    └── 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 **worker2** (din subdirectorul corespunzător) în *background*, iar după o pauză de 2 secunde, va porni și execuția programului **supervisor** (din subdirectorul corespunzător).
iii) Iar dacă *p*=2, atunci scriptul va porni mai întâi execuția programului **supervisor** (din subdirectorul corespunzător) în *background*, iar după o pauză de 3 secunde, va porni și execuția programului **worker2** (din subdirectorul corespunzător).
iv) În ambele situații, programul supervisor 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 o secvență de numere întregi, în format textual.
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) De asemenea, tot în funcția principală a programului, acesta va crea un proces fiu, iar în fiul creat va starta, printr-un apel exec adecvat, programul **worker1**.
iii) Într-o funcție separată, apelată din funcția main, programul va citi pe rând, unul câte unul, fiecare număr din acel fișier input_data.txt și-l va converti la reprezentarea în format binar a numerelor întregi, iar rezultatul conversiei îl va transmite către procesul **worker1** prin acel canal anonim, folosind apeluri POSIX (așadar, folosind reprezentarea binară a numerelor întregi, le veți transmite prin mesaje de lungime constantă).
iv) Într-o altă funcție separată, apelată din funcția main, programul va citi, folosind apeluri POSIX, numerele transmise lui de către procesul **worker1** prin acel canal fifo și le va aduna. Rezultatul acestei adunări va fi convertit la reprezentarea textuală în baza 10 a numerele întregi și apoi se va calcula suma cifrelor acestei reprezentări, afișând-o la final pe ecran.

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

i) 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).

ii) 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*.
iii) Într-o funcție separată, apelată din funcția main, programul va citi din acel canal anonim, folosind apeluri POSIX, fiecare număr transmis lui de către procesul **supervisor** (prin mesaje de lungime constantă, folosind reprezentarea binară a numerelor întregi) și va verifica dacă acesta este număr impar, caz în care îl înmulțește cu 2, sau este număr par, caz în care îl păstrează neschimbat; în ambele cazuri, valoarea astfel obținută va fi transmisă mai departe către procesul **worker2,** prin intermediul acelei mapări nepersistente cu nume (se va păstra reprezentarea în binar pentru numărul respectiv). De asemenea, programul va contoriza într-o variabilă *contor_impare* câte operații de înmulțire cu 2 a efectuat pe parcursul procesării tuturor numerelor primite de la procesul **supervisor**.
iv) Într-o altă funcție separată, apelată din funcția main, programul va citi fiecare număr transmis lui de către procesul **worker2,** prin intermediul acelei mapări nepersistente cu nume**,** și va calcula suma tuturor numerelor primite de la **worker2**. La final va transmite această sumă, precum și valoarea calculată în contorul *contor_impare*, către procesul **supervisor,** prin acel canal fifo, folosind apeluri POSIX (așadar, folosind reprezentarea binară a numerelor întregi, le veți transmite prin 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 număr transmis lui de către procesul **worker1** prin intermediul acelei mapări nepersistente cu nume și va calcula pătratul acelui număr. Apoi va transmite rezultatul acestui calcul de ridicare la pătrat către procesul **worker1**, prin intermediul aceleiași mapări nepersistente cu nume (se va păstra reprezentarea în binar pentru numere întregi).

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