Test practic la SO - varianta nr. 4

Realizați o aplicație formată din următoarele trei programe cooperante, plus un script de execuție a lor, care vor fi plasate conform ierarhiei de directoare de mai jos:

```
.
├── starter.sh
└── app
    ├── supervisor.c
    ├── worker1.c
    └── components
        └── worker2.c
```

1. Programul "worker1.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 oarecare existent pe disc.
i) În funcția principală a programului, acesta va testa existența unui argument primit în linia de comandă și apoi va crea un proces fiu. În fiul creat va starta, printr-un apel exec adecvat, programul **supervisor** căruia să-i fie transmis, ca argument, acel argument primit inițial de către **worker1**.
ii) Tot în funcția main, programul va face inițializările necesare pentru a putea primi informații de la procesul **supervisor** printr-un canal anonim și pentru a putea comunica cu procesul **worker2** printr-un canal fifo.
iii) Într-o funcție separată, apelată din funcția main, programul va citi din acel canal anonim, în mod repetat, câte un caracter (i.e., 1 octet), transmis lui de către procesul **supervisor** (vezi punctul 2.ii) pentru protocolul de comunicare) și va calcula frecvența fiecărui caracter ASCII (i.e., 1 octet), adică va calcula numărul de apariții ale acestuia întâlnite per total în secvența de caractere citită. După procesarea tuturor caracterelor din secvența citită, programul va trimite procesului **worker2**, prin canalul fifo, fiecare caracter ASCII întâlnit efectiv (i.e., cu frecvență nenulă) și numărul de apariții ale acestuia (vă recomand să folosiți reprezentarea binară a numerelor întregi pentru a avea mesaje de lungime constantă).

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 oarecare existent pe disc.
i) În funcția principală a programului, acesta va face inițializările necesare pentru a putea comunica cu procesul **worker2** printr-o mapare ne-persistentă cu nume (i.e., un obiect de memorie partajată).
ii) Într-o funcție separată, apelată din funcția main, programul va citi din fișierul primit ca argument, în mod repetat, câte un caracter (i.e., 1 octet) și le va transmite, pe rând, procesului **worker1** prin canalul anonim creat de acesta din urmă (vezi punctul 1.ii).
iii) Într-o altă funcție separată, apelată din funcția main, programul va citi cele două perechi formate din câte un caracter și numărul lui de apariții (i.e., rezultatele calculate de **worker2**), transmise lui de către procesul **worker2** prin intermediul mapării ne-persistente cu nume create la punctul 2.i, și le va afișa pe ecran.

3. Programul "worker2.c" va implementa funcționalitatea descrisă în specificația următoare:
i) În funcția principală a programului, acesta va face inițializările necesare pentru a putea primi informații de la procesul **worker1** printr-un canal fifo și, respectiv, pentru a putea comunica cu procesul **supervisor** printr-o mapare ne-persistentă cu nume.
ii) Într-o altă funcție separată, apelată din funcția main, programul va citi din acel canal fifo, pereche cu pereche, rezultatele transmise lui de către procesul **worker1** și va calcula, fără a stoca în memoria proprie întreaga secvență de rezultate primite, următoarele două perechi: caracterul vocală (considerat *case-insensitive*) cu cel mai mare număr de apariții și, respectiv, caracterul consoană (considerat *case-insensitive*) cu cel mai mic număr (nenul) de apariții, din secvența de rezultate primite de la **worker1**. În caz că sunt două vocale cu același număr maxim de apariții, se va lua în considerare vocala cea mai "mică" (în ordinea uzuală, alfabetică). Se va proceda similar și în caz că sunt două consoane cu același număr minim de apariții. Apoi programul va transmite cele două perechi calculate (i.e., prima formată din vocala însoțită de numărul ei de apariții și, respectiv, a doua formată din consoana însoțită de numărul ei de apariții) către procesul **supervisor**, prin acel obiect de memorie partajată inițializat la punctul 2.i).

4. Scriptul "starter.sh" va implementa funcționalitatea descrisă în specificația următoare:
i) Scriptul va porni mai întâi execuția programului **worker2** (din subdirectorul corespunzător) în *background*.
ii) După o pauză de 2 secunde, va porni și execuția programului **worker1***,* cu un argument în linia de comandă: calea (absolută sau relativă) către un fișier de intrare "input.txt" pe care-l veți crea anterior, în orice editor de texte doriți, cu un conținut oarecare.

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 de laborator).
Astfel vă veți clarifica mai bine cerințele specificate în enunțul problemei și, de asemenea, vă va ajuta la partea de implementare!

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 sau .tar / .tar.gz) care să conțină cele 3 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 de intrare “input.txt”, **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:

NumePrenume_TP2.zip (sau .tar sau .tar.gz)

Barem de corectare

Observații:
- programele și scriptul trebuie plasate într-o ierarhie de directoare conform celor descrise în enunțul problemei (și apelate în mod corespunzător poziției lor în ierarhia respectivă); de asemenea, conținutul arhivei finale va păstra ierarhia respectivă de directoare și fișiere sursă.
- pentru implementarea comunicațiilor între cele 3 procese cooperante se vor utiliza exact metodele de comunicare descrise în specificații.
- **dacă nu respectați toate condițiile precedente** (de exemplu, dacă folosiți, pentru vreuna dintre comunicații, un alt mijloc de comunicație decât cel specificat, sau dacă plasați vreunul dintre programele pornite din “starter.sh” într-un alt director decât cel specificat, atunci vom evalua corectitudinea rezolvării, dar **punctajul total acordat va fi înjumătățit**.
- fiecare punctaj din barem se acordă integral, sau deloc.

1. Baremul pentru programul "worker1.c":
a) 1p - implementarea corectă a operațiilor fork și exec descrise la punctul i) din specificația dată.
b) 1p - implementarea corectă a inițializărilor de comunicații descrise la punctul ii) din specificația dată.
c) 0.5p - implementarea corectă a comunicației prin *canal anonim* descrise la punctul iii) din specificația dată.
d) 1p - implementarea corectă a procesărilor de date descrise la punctul iii) din specificația dată.
e) 1p - implementarea corectă a comunicației prin *canal fifo* descrise la punctul iii) din specificația dată.
Notă: se va calcula fracția procentuală de îndeplinire a criteriilor de mai sus, iar cele trei criterii de mai jos vor fi notate prin ponderare cu acest procentaj: F = total puncte acordate pentru criteriile a)-e) / 4.5p
f) F * 1p - tratarea cazurilor de eroare la TOATE apelurile de sistem din program.
g) F * 1p - programul se compilează fără erori și fără warning-uri.
h) F * 0.5p - scrierea codului într-o formă indentată, plăcută ochiului, i.e. "readable code" (de exemplu, evitați copy-paste în mcedit).
Total: 7p

2. Baremul pentru programul "supervisor.c":
a) 0.5p - implementarea corectă a inițializărilor de comunicații descrise la punctul i) din specificația dată.
b) 1p - implementarea corectă a procesărilor de date descrise la punctul ii) din specificația dată.
c) 0.5p - implementarea corectă a comunicației prin *canal anonim* descrise la punctul ii) din specificația dată.
d) 0.5p - implementarea corectă a procesărilor de date descrise la punctul iii) din specificația dată.
e) 1p - implementarea corectă a comunicației prin *shared-memory* descrise la punctul iii) din specificația dată.
Notă: se va calcula fracția procentuală de îndeplinire a criteriilor de mai sus, iar cele trei criterii de mai jos vor fi notate prin ponderare cu acest procentaj: F = total puncte acordate pentru criteriile a)-e) / 3.5p
f) F * 1p - tratarea cazurilor de eroare la TOATE apelurile de sistem din program.
g) F * 1p - programul se compilează fără erori și fără warning-uri.
h) F * 0.5 - scrierea codului într-o formă indentată, plăcută ochiului, i.e. "readable code" (de exemplu, evitați copy-paste în mcedit).
Total: 6p

3. Baremul pentru programul "worker2.c":
a) 1p - implementarea corectă a inițializărilor de comunicații descrise la punctul i) din specificația dată.
b) 1p - implementarea corectă a comunicației prin *canal fifo* descrise la punctul ii) din specificația dată.
c) 1p - implementarea corectă a procesărilor de date descrise la punctul ii) din specificația dată.
d) 0.5p - implementarea corectă a comunicației prin *shared-memory* descrise la punctul ii) din specificația dată.
Notă: se va calcula fracția procentuală de îndeplinire a criteriilor de mai sus, iar cele trei criterii de mai jos vor fi notate prin ponderare cu acest procentaj: F = total puncte acordate pentru criteriile a)-d) / 3.5p
e) F * 1p - tratarea cazurilor de eroare la TOATE apelurile de sistem din program.
f) F * 1p - programul se compilează fără erori și fără warning-uri.
g) F * 0.5 - scrierea codului într-o formă indentată, plăcută ochiului, i.e. "readable code" (de exemplu, evitați copy-paste în mcedit).
Total: 6p

4. Baremul pentru scriptul "starter.sh":
a) 2 x 0.5p - implementarea corectă a apelării celor două programe în modul descris în specificația dată.
Total: 1p

Total general 1. - 4. : 20p