Laborator #13 : exerciții de laborator

Sumar:

I) Exerciții de programare cu semnale, pentru programare asincronă și tratarea excepțiilor:

Exerciții propuse spre rezolvare:

1. [Signal processing #2]
   Să se scrie un program C care realizează urmatoarele: mai întâi creează două procese fii, după care generează aleator într-un fișier linii de text. După închiderea fișierului în care scrie acele linii de text aleatoare, procesul inițial va trimite fiilor creați semnalul `SIGUSR2` (primului fiu) și respectiv `SIGUSR1` (celui de-al doilea fiu). La recepționarea semnalului, procesul fiu care a primit `SIGUSR1` va calcula numărul de linii de text din fișier (i.e., numărul de caractere '\n') și îl va afișa pe ecran, iar procesul fiu care a primit `SIGUSR2` va sorta liniile de text din fișier în ordine lexicografică crescătoare și va afișa pe ecran rezultatul sortării.
   ***Cerință*:** sortarea să se facă în paralel cu calculul numărului de linii, dar rezultatele să fie afișate pe ecran de cele două procese fii în așa fel încât numărul total de linii de text să apară obligatoriu înaintea liniilor de text sortate.
   (Indicație: pentru a realiza acest tip de sincronizare la afișarea pe ecran, procesul care face sortarea va aștepta un semnal (e.g. `SIGUSR1`) de la celălalt proces fiu, înainte de a afișa rezultatul sortării.)

2. [Signal processing #3]
   Să se scrie un program C care la execuție va genera un proces fiu și va avea comportamentul descris în continuare.
   Procesul părinte numără încontinuu, la intervale de 1 secundă (i.e., într-o buclă repetitivă, incrementează o variabilă și așteaptă cu un apel `sleep(1);` la fiecare iterație), începând de la 0, până în momentul în care este întrerupt de către utilizator prin trimiterea unui semnal `SIGINT` (sau până la terminarea procesului fiu, vezi mai jos). De fiecare dată când procesul părinte ajunge la un număr divizibil cu 10, trimite procesului fiu semnalul `SIGUSR1`. Iar când primește semnalul `SIGINT`, procesul părinte trimite procesului fiu semnalul `SIGUSR2`. La terminarea procesului fiu, procesul părinte își va încheia și el execuția.
   Procesul fiu, în momentul primirii semnalului `SIGUSR1`, afișează pe ecran un mesaj de genul "Fiul: am primit USR1", iar la primirea semnalului `SIGUSR2`, își încheie execuția, afișând pe ecran un mesaj corespunzător (e.g. "Fiul: sfarsit executie").

Exerciții suplimentare, propuse spre rezolvare pentru acasă:

Iată alte câteva exerciții de programare cu procesări de semnale, pe care să încercați să le rezolvați singuri în timpul liber, pentru a vă auto-evalua cunoștințele dobândite în urma acestui laborator:

1. [Signal processing #4]
   Să se scrie un program C utilizat pentru copierea intrării standard într-un fișier, al cărui nume este specificat în linia de comandă a programului. Dacă în intervalul de `3*N` secunde de la începerea execuției programului, nu este introdus nici un caracter de la tastatură, atunci programul va fi terminat. Valoarea constantei `N` este specificată, de asemenea, ca și argument al programului în linia de comandă. În plus, la fiecare interval de `N` secunde neutilizate (i.e., fără input din partea utilizatorului), utilizatorul va fi atenționat printr-un mesaj corespunzător.
   (Indicație: pentru măsurarea scurgerii intervalelor de `N` secunde, tratați semnalul `SIGALRM`, generat cu apelul `alarm`.)

2. [Sum&Sort, using signals for syncronization]

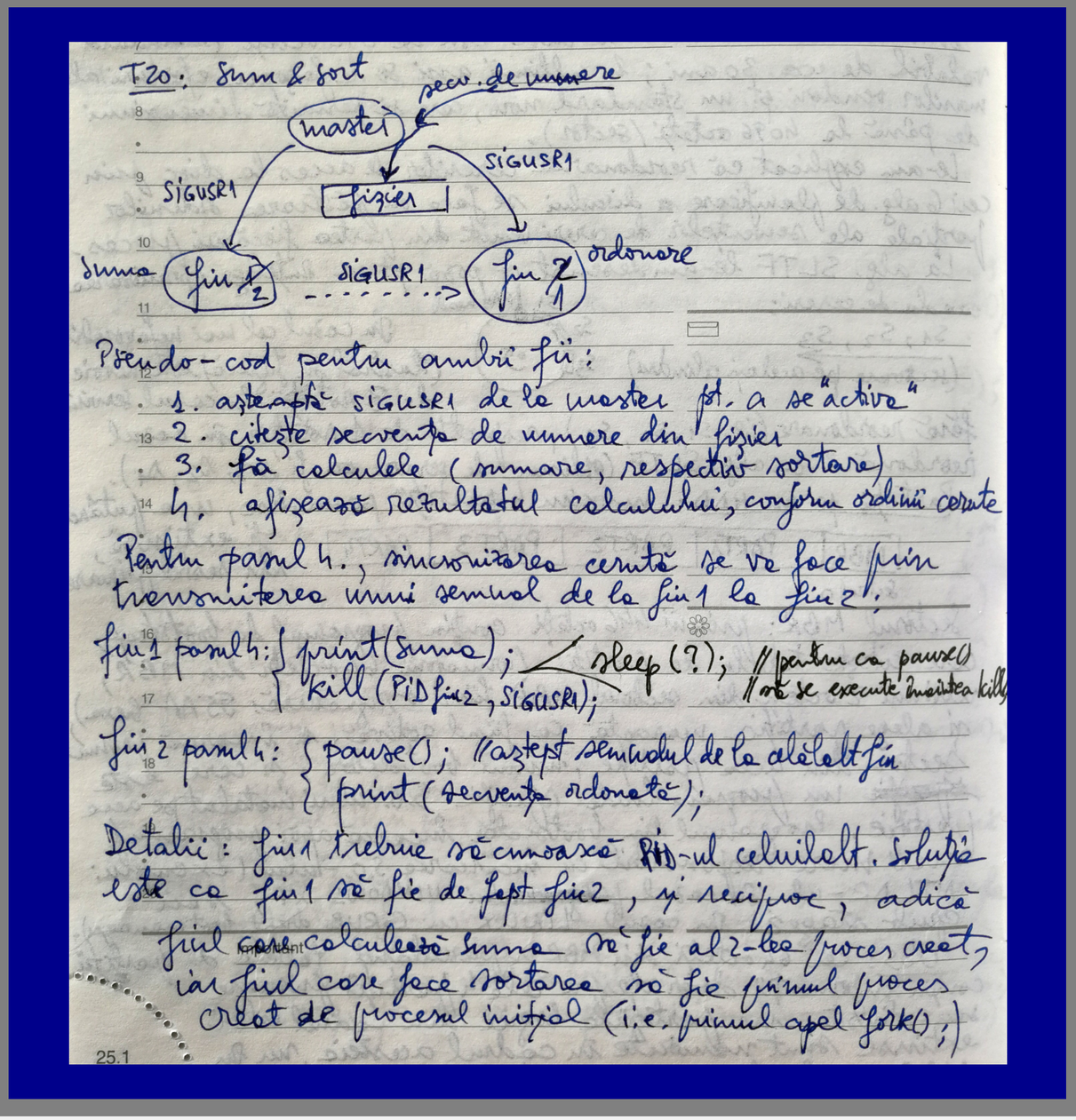
Să se scrie un program C care realizează urmatoarele: mai întâi creează două procese fii, după care generează aleator într-un fișier numere întregi. După închiderea fișierului în care scrie acele numere aleatoare, procesul inițial va trimite fiilor creați semnalul `SIGUSR1`. La recepționarea semnalului, unul dintre procesele fii va calcula suma numerelor și o va afișa pe ecran, iar celălalt fiu va sorta numerele din fișier în ordine crescătoare și va afișa pe ecran secvența sortată.

***Cerință*:** sortarea să se facă în paralel cu calculul sumei, dar rezultatele să fie afișate pe ecran de către cele două procese fii în așa manieră încât suma numerelor să apară obligatoriu înaintea secvenței ordonate.

(Indicație: pentru a realiza acest tip de sincronizare la afișarea pe ecran, procesul fiu care face sortarea va aștepta un semnal (e.g. `SIGUSR2`) de la celălalt proces fiu, înainte de a afișa rezultatul sortării.)

Show / Hide some suggestions for solving the problem

Ideea de rezolvare se poate desprinde din următoarea diagramă (schemă logică), scrisă de mână:



3. ['Ping-pong' pattern #5, using signals for syncronization]
   Două procese, ce nu sunt înrudite (i.e. nu sunt în relația părinte-fiu), trebuie să citească alternativ câte un caracter dintr-un fișier text `fis.txt`. Sincronizarea execuției celor două procese, pentru citirea alternativă din fișier, va fi realizată exclusiv prin semnale. Comunicarea reciprocă a PID-urilor (PID-urile sunt necesare pentru a putea trimite semnale unul altuia) se va face la începutul execuției lor, prin canale de comunicație fifo. Caracterele citite de fiecare proces vor fi scrise în fișierele `fis_poz-impare.txt` și, respectiv, `fis_poz-pare.txt`.

4. ['Token-ring' pattern #1 (v2, using signals for syncronization)] Show / hide this problem

   Se dau trei fișiere, `nume.txt`, `prenume.txt` și `nota.txt`, în care sunt înregistrate, câte unul pe linie, numele, prenumele și respectiv nota obținută de mai mulți studenți la o anumită disciplină (se va trata și cazul de excepție când nu există o corespondență bijectivă la nivel de linie între cele trei fișiere).
   Să se scrie un program C care să creeze doi fii, după care se vor realiza următoarele operații: tatăl va citi, în mod repetat, câte o linie cu date din fișierul `nume.txt` și o va scrie într-un fișier de ieșire, numit `tabel.txt`, primul fiu va citi, în mod repetat, câte o linie cu date din fișierul `prenume.txt` și o va scrie în fișierul de ieșire `tabel.txt`, iar al doilea fiu va citi, în mod repetat, câte o linie cu date din fișierul `nota.txt` și o va scrie în fișierul de ieșire `tabel.txt`.
   În plus, cele trei procese trebuie să se sincronizeze conform șablonului general 'Token-ring' particularizat pentru `p=3` procese, astfel încât informațiile să apară exact pe câte o linie, sub forma "NUME - PRENUME - NOTA", în fișierul de ieșire `tabel.txt`, și nu alte combinații posibile de "interclasare" a informațiilor parțiale scrise de cele trei procese aflate în execuție simultană. Sincronizarea proceselor va fi realizată exclusiv prin semnale.
   (Indicație: scopul acestui exercițiu este acela de a implementa corect un mecanism de sincronizare de forma "Acum e rândul jucătorului #1 --> acum e rândul jucătorului #2 --> acum e rândul jucătorului #3 --> acum e rândul jucătorului #1 --> acum e rândul jucătorului #2 --> acum e rândul jucătorului #3 -->... ș.a.m.d.". Practic, acest exercițiu este inspirat după exercițiul ['Token ring' pattern #1 (v1, using anon mmap for syncronization)] propus spre rezolvare în partea a doua a laboratorului #10, doar că acum trebuie să folosiți un alt mecanism pentru sincronizarea celor trei procese (și anume, folosiți semnale pentru sincronizare, în loc de comunicații prin intermediul unei zone de memorie partajată). Revedeți mai întâi acel exercițiu, în caz că l-ați rezolvat la vremea respectivă.)

5. ['Token ring' pattern #2 : "*Eeny, meeny, miny, moe,* ..." (v2, using signals for syncronization)] Show / hide this problem

   Să se scrie patru programe C, iar fiecare dintre ele va realiza următoarele operații: un proces va scrie pe ecran textul "ini-" în mod repetat, al doilea proces va scrie pe ecran textul "mini-" în mod repetat, al treilea proces va scrie pe ecran textul "maini-" în mod repetat, iar al patrulea proces va scrie pe ecran textul "mo," în mod repetat.
   În plus, cele patru procese trebuie să se sincronizeze conform șablonului 'Token-ring' particularizat pentru `p=4` procese, astfel încât pe ecran să apară exact succesiunea de mesaje:
   ini-mini-maini-mo, ini-mini-maini-mo, ini-mini-maini-mo, ini-mini-maini-mo, ...
   și nu alte combinații posibile de "interclasare" a mesajelor afișate de cele patru procese aflate în execuție simultană.
   Executabilele rezultate în urma compilării celor 4 programe vor fi lansate simultan în execuție paralelă neînlănțuită (e.g., cu ajutorul unui script), iar sincronizarea necesară va fi realizată exclusiv prin semnale.
   (Indicație: acest exercițiu este inspirat după exercițiul ['Token ring' pattern #2 : "*Eeny, meeny, miny, moe,* ..." (v1, using anon mmap for syncronization)] propus spre rezolvare în laboratorul #10, doar că acum trebuie să folosiți un alt mecanism pentru sincronizarea celor patru procese. Revedeți mai întâi acel exercițiu, în caz că l-ați rezolvat la vremea respectivă.)

6. ['Sleeping barber' problem] Show / hide this problem

   Să se scrie un program C care simulează desfășurarea activității într-un cabinet de stomatologie. Procesul inițial va genera un prim proces fiu, cu rol de asistent: acesta se va ocupa de intrarea pacienților în sala de așteptare, generându-i la intervale de timp fixe sau aleatoare. Pentru fiecare pacient intrat în sala de așteptare, procesul asistent va genera un proces fiu separat, care va reprezenta pacientul respectiv. Pacientul își va anunța cumva PID-ul procesului principal.
   Procesul principal (ce reprezintă doctorul stomatolog) se va ocupa cu tratarea pacienților, într-o buclă repetitivă, semnalându-le pacienților intrarea în cabinet (din sala de așteptare) și apoi ieșirea din cabinet după terminarea tratamentului. Durata tratamentului poate fi aleatoare sau fixă.
   Procesele create pentru fiecare pacient în parte vor afișa schimbarea stării pacientului, conform tranzițiilor: "intrat în sala de așteptare --> în cabinetul doctorului --> ieșire".
   Cerință: pentru comunicarea (sincronizarea) stomatologului cu pacienții se vor folosi semnale.
   (Indicație: acest exercițiu este o instanțiere a problemei de sincronizare 'Sleeping barber' ce a fost studiată în cursul teoretic #6. Ca atare, vă recomand să recitiți mai întâi cursul respectiv.)