Laborator #10 : exerciții de laborator

Sumar:

I) Exerciții de programare cu mai multe procese secvențiale (prima parte - creare diverse ierarhii de procese) :

a) *Exerciții propuse spre rezolvare* :

Intrați pe setul de exerciții propuse spre rezolvare, pe care vi-l va indica profesorul de laborator, în timpul laboratorului, și încercați să le rezolvați singuri:

Setul #1 Setul #2

Setul 1

1. [A particular tree of processes]
Să se scrie un program C care să creeze un arbore particular de procese, având 3 nivele, structurate astfel:
1) unicul proces $P_{1,1}$ de pe nivelul 1 al arborelui (i.e., rădăcina arborelui) va avea 2 procese fii, și anume procesele $P_{2,1}$ și $P_{2,2}$ de pe nivelul 2 al arborelui;
2) fiecare dintre cele două procese de pe nivelul 2 al arborelui, va avea la rândul său 3 procese fii pe nivelul 3 al arborelui, numerotate cu $P_{3,1}$, $P_{3,2}$, ... , $P_{3,6}$, care vor fi astfel "nepoți" ai procesului din rădăcina arborelui și, în plus, nu vor avea nici un fiu la rândul lor (i.e., vor fi "frunze" în arbore).
Fiecare proces își va tipări, printr-un singur mesaj, "numărul lui de ordine" în arbore (i.e., perechea (i,j) de forma: (1,1), (2,1),(2,2), (3,1),...,(3,6)), urmat de PID-ul lui, cel al părintelui său, precum și PID-urile fiilor acestuia și codurile acestora de terminare.

Show / Hide some suggestion for this problem

***Recomandare*:** pentru a testa corectitudinea programului pe care îl veți scrie, în sensul de a verifica dacă într-adevăr se creează în mod corect ierarhia de procese cerută în enunțul acestei probleme, puteți proceda în maniera descrisă aici.

Setul 2

1. [Another particular tree of processes]
Să se scrie un program C care să creeze un arbore particular de procese, având 3 nivele, structurate astfel:
1) unicul proces $P_{1,1}$ de pe nivelul 1 al arborelui (i.e., rădăcina arborelui) va avea 3 procese fii, și anume procesele $P_{2,1}$, $P_{2,2}$ și $P_{2,3}$ de pe nivelul 2 al arborelui;
2) fiecare dintre cele trei procese de pe nivelul 2 al arborelui, va avea la rândul său 2 procese fii pe nivelul 3 al arborelui, numerotate cu $P_{3,1}$, $P_{3,2}$, ... , $P_{3,6}$, care vor fi astfel "nepoți" ai procesului din rădăcina arborelui și, în plus, nu vor avea nici un fiu la rândul lor (i.e., vor fi "frunze" în arbore).
Fiecare proces își va tipări, printr-un singur mesaj, "numărul lui de ordine" în arbore (i.e., perechea (i,j) de forma: (1,1), (2,1),(2,2),(2,3), (3,1),...,(3,6)), urmat de PID-ul lui, cel al părintelui său, precum și PID-urile fiilor acestuia și codurile acestora de terminare.

Show / Hide some suggestion for this problem

***Recomandare*:** pentru a testa corectitudinea programului pe care îl veți scrie, în sensul de a verifica dacă într-adevăr se creează în mod corect ierarhia de procese cerută în enunțul acestei probleme, puteți proceda în maniera descrisă aici.

b) *Exerciții suplimentare, propuse spre rezolvare pentru acasă* :

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

1. [A perfect k-ary tree of processes #1 (v1 = the recursive version)]
   Să se scrie un program C care să creeze un arbore `K`-ar perfect cu `N` nivele, de procese (valorile `K` și `N` vor fi citite de la tastatură, sau primite din linia de comandă).
   Și anume: unicul proces $P_{1,1}$ de pe nivelul 1 al arborelui (i.e., rădăcina arborelui) va avea `K` procese fii, și anume procesele $P_{2,1},...,P_{2,K}$ de pe nivelul 2 al arborelui, fiecare dintre acestea va avea la rândul său `K` procese fii pe nivelul 3 al arborelui, ș.a.m.d. până la cele K^{N-1} procese de pe nivelul `N` al arborelui, care nu vor avea nici un fiu.
   Fiecare proces își va tipări, printr-un singur mesaj, numărul lui de ordine în arbore (i.e. perechea i,j cu $1 \leq i \leq N$ și $1 \leq j \leq K^{i-1}$), PID-ul lui, pe cel al părintelui său, precum și PID-urile celor `K` fii ai acestuia și codurile acestora de terminare.
   Cerință: proiectați o rezolvare recursivă a acestei probleme (i.e., folosiți o funcție recursivă pentru crearea proceselor), astfel:
   i) prin recursie doar după adâncimea în arbore, `N` (recursia este mai simplă);
   sau
   ii) prin recursie după `K` și `N` (recursia este mai complexă).

   Show / Hide some suggestion for this problem

   Recomandare: pentru a testa corectitudinea programului pe care îl veți scrie, în sensul de a verifica dacă într-adevăr se creează în mod corect ierarhia de procese cerută în enunțul acestei probleme, puteți proceda în maniera descrisă aici.

2. [A perfect k-ary tree of processes #2 (v2 = the iterative version)]
   Să se elaboreze o soluție iterativă a problemei **[A perfect k-ary tree of processes #1]** (i.e., folosiți bucle iterative pentru crearea proceselor).

   Show / Hide some suggestion for this problem

   Recomandare: a se reciti sugestia dată la problema suplimentară precedentă!

II) **Exerciții de programare cu mai multe procese secvențiale (a doua parte - diverse probleme de sincronizare a unor procese cooperante)** :

a) *Exerciții propuse spre rezolvare* :

Mai întâi, încercați să completați codul lipsă din cele două exemple parțial rezolvate din suportul de laborator #10.

Apoi, încercați să rezolvați singuri (măcar una dintre) cele două probleme de sincronizare din setul de exerciții propuse spre rezolvare, pe care vi-l va indica profesorul de laborator, în timpul laboratorului:

Setul #1 Setul #2

Setul 1

1. ['Supervisor-workers' pattern #2 (v2, using mmap-files for IPC)]
   Re-implementați programul care se cere a fi completat la exercițiul parțial rezolvat **['Supervisor-workers' pattern #2 (v1, using regular files for IPC)]** din suportul de laborator #10, astfel încât cele două fișiere folosite pentru comunicații să fie accesate prin maparea lor în memorie.

   Show / Hide some suggestion for this problem

   Recomandare: a se citi ideea de rezolvare prezentată în sugestia de rezolvare dată la exercițiul parțial rezolvat **['Supervisor-workers' pattern #2 (v1, using regular files for IPC)]**.

2. ['Ping-pong' pattern #1 (v3, using anon mmap for IPC)]
   Re-implementați programul care se cere a fi completat la exercițiul parțial rezolvat **['Ping-pong' pattern #1 (v1, using regular file for IPC)]** din suportul de laborator #10, astfel încât rolul fișierului `flag.bin` să fie îndeplinit de o mapare anonimă în memorie.

   Show / Hide some suggestion for this problem

   Recomandare: a se citi ideea de rezolvare prezentată în sugestia de rezolvare dată la exercițiul parțial rezolvat **['Ping-pong' pattern #1 (v1, using regular file for IPC)]**.

Setul 2

1. ['Supervisor-workers' pattern #2 (v3, using anon mmap for IPC)]
   Re-implementați programul care se cere a fi completat la exercițiul parțial rezolvat **['Supervisor-workers' pattern #2 (v1, using regular files for IPC)]** din suportul de laborator #10, astfel încât rolul celor două fișiere folosite pentru comunicații să fie îndeplinit de două mapări anonime în memorie (sau, mai eficient, se poate rezolva problema și folosind o singură mapare anonimă).

   Show / Hide some suggestion for this problem

   ***Recomandare*:** a se citi ideea de rezolvare prezentată în sugestia de rezolvare dată la exercițiul parțial rezolvat **['Supervisor-workers' pattern #2 (v1, using regular files for IPC)]**.

2. ['Ping-pong' pattern #1 (v2, using mmap-file for IPC)]
   Re-implementați programul care se cere a fi completat la exercițiul parțial rezolvat **['Ping-pong' pattern #1 (v1, using regular file for IPC)]** din suportul de laborator #10, astfel încât fișierul `flag.bin` să fie accesat prin maparea sa în memorie.

   Show / Hide some suggestion for this problem

   ***Recomandare*:** a se citi ideea de rezolvare prezentată în sugestia de rezolvare dată la exercițiul parțial rezolvat **['Ping-pong' pattern #1 (v1, using regular file for IPC)]**.

b) *Exerciții suplimentare, propuse spre rezolvare pentru acasă* :

Alte câteva exerciții de programare cu apelurile de sistem `fork` și `wait`, a unor probleme de sincronizare diversificate:

1. ['Supervisor-workers' pattern #1 (v2, using mmap-files for IPC)] Show / hide this problem

   Re-implementați programul prezentat în exercițiul rezolvat **['Supervisor-workers' pattern #1 : *A coordinated distributed sum #1* (v1, using regular files for IPC)]** din suportul de laborator #10, astfel încât cele 4 fișiere folosite pentru comunicații să fie accesate prin maparea lor în memorie.

   Show / Hide some suggestion for this problem

   ***Recomandare*:** revedeți rezolvarea prezentată în exercițiul **['Supervisor-workers' pattern #1 (v1, using regular files for IPC)]** indicat în enunț și modificați-o astfel încât cele 4 fișiere folosite pentru comunicații să fie accesate prin maparea lor în memorie.

2. ['Supervisor-workers' pattern #1 (v3, using anon mmap for IPC)] Show / hide this problem

   Re-implementați programul prezentat în exercițiul rezolvat **['Supervisor-workers' pattern #1 : *A coordinated distributed sum #1* (v1, using regular files for IPC)]** din suportul de laborator #10, astfel încât în locul celor 4 fișiere folosite pentru comunicații să se utilizeze 2 mapări anonime.

   Show / Hide some suggestion for this problem

   ***Recomandare*:** revedeți rezolvarea prezentată în exercițiul **['Supervisor-workers' pattern #1 (v1, using regular files for IPC)]** indicat în enunț și modificați-o astfel încât cele 4 fișiere folosite pentru comunicații să fie înlocuite cu 2 mapări anonime.
   O mapare anonimă se poate obține prin următorul apel, care trebuie plasat, în program, înaintea apelului fork():

```
map_addr = mmap( NULL,                      // Se va crea o mapare începând de la o adresă page-aligned aleasă de kernel (și returnată în map_addr)
                 length,                    // Lungimea dorită, preferabil multiplu de dimensiunea paginii (o alegeți în funcție de cantitatea de informație ce trebuie partajată)
                 PROT_READ | PROT_WRITE,    // Tipul de protecție a mapării: paginile mapării vor permite accese în citire și scriere
                 MAP_SHARED | MAP_ANONYMOUS, // Maparea este anonimă și partajată, pentru a avea memorie comună între procesele tată și fiu
                 -1,                        // La descriptorul de fișier se pune -1, conform documentației
                 0                          // Offset-ul (deplasamentul) nu este luat în seamă, conform documentației
               );
```

   Indicație: cele două fișiere folosite pentru comunicațiile dintre supervisor și un anumit worker pot fi "simulate" printr-o singură mapare anonimă, astfel: rezervați primii `sizeof(int)=4` octeți ai mapării pentru a stoca suma parțială calculată de acel worker, iar restul mapării, începând de la adresa `map_addr + sizeof(int)`, o folosiți pentru a stoca secvența de numere scrisă de supervisor pentru acel worker.

3. [#1: Yet another instance of the 'Supervisor-workers' pattern (using anon mmap for IPC)] Show / hide this problem

Să se scrie un program C care va crea două procese fii. Apoi, părintele va genera (pseudo-)aleator 1000 de numere naturale, mai mici decât 101, pe care le va transmite fiilor, astfel: numerele impare vor fi trimise primului fiu, iar numerele pare celui de-al doilea fiu (păstrându-le ordinea în care au fost generate). Primul fiu va determina care dintre numerele primite sunt numere prime, transmițându-le înapoi părintelui doar pe cele prime (cu păstrarea ordinii lor). Al doilea fiu va determina care dintre numerele primite sunt divizibile cu 4, transmițându-le înapoi părintelui doar pe acelea (cu păstrarea ordinii lor). Părintele va salva numerele primite de la ambii fii într-un singur vector, pe care-l va sorta crescător și va afișa la final rezultatul sortării.
Cerință: pentru comunicațiile dintre tată și fiecare fiu, se va folosi (câte) o mapare anonimă.

4. [#2: Yet another instance of the 'Supervisor-workers' pattern (using anon mmap for IPC)] Show / hide this problem

Să se scrie un program C care va crea 3 procese fii. Părintele va primi la linia de comandă un număr natural, pe care îl va transmite fiilor. Primul fiu va calcula suma cifrelor pare din reprezentarea 'textuală' a numărului primit de la părinte. Al doilea fiu va calcula produsul cifrelor impare din reprezentarea 'textuală' a numărului primit de la părinte. Al treilea fiu va număra câte cifre prime există în reprezentarea 'textuală' a numărului primit de la părinte. Fiii vor trimite părintelui rezultatele calculate, iar acesta va afișa pe ecran fiecare rezultat în parte, cât și suma lor.
Cerință: pentru comunicațiile dintre tată și fiecare fiu, se va folosi (câte) o mapare anonimă.
Exemplificare: dacă se va executa acest program cu linia de comandă: prompt> `./program.exe 123647` , atunci, în urma execuției sale, se va afișa pe ecran: `12 + 21 + 3 = 36`.

5. [#3: Yet another instance of the 'Supervisor-workers' pattern (using anon mmap for IPC)] Show / hide this problem

Să se scrie un program C care va crea două procese fii. Apoi, părintele va citi o secvență de numere întregi strict pozitive de la tastatură, secvență terminată cu Ctrl+D, și va transmite fiecărui proces fiu câte jumătate din secvența de numere citită, în felul următor: primul fiu va primi numerele aflate pe poziții impare în secvența citită, iar al doilea fiu va primi celelalte numere din secvență. Transmiterea numerelor către fii se va face sincron cu citirea lor, i.e. pe măsură ce părintele citește câte un număr, îl și transmite fiului corespunzător. Deci, nu se așteaptă până la finalul introducerii numerelor, pentru a începe să le transmită. Astfel, în timp ce se așteaptă utilizatorul ca să introducă următorul număr, în paralel are loc și procesarea numărului precedent introdus (!), procesare descrisă în cele de mai jos.
Fiecare fiu va calcula, pentru fiecare număr primit, lungimea reprezentării 'textuale' în baza 2 (!) a acelui număr, și va transmite înapoi părintelui lungimile astfel calculate.
Părintele va calcula valoarea maximă a numerelor reprezentând cele două secvențe de lungimi primite de la cei doi fii, și va afișa pe ecran această valoare maximă.
Definiție: lungimea reprezentării reprezentării 'textuale' în baza 2 a unui număr întreg strict pozitiv N este egală cu cea mai mică putere K a lui 2 astfel încât $2^{K-1} < N \leq 2^K$.
Exemplificare: lungimea reprezentării 'textuale' în baza 2 a numărului N=12 este K=4, iar pentru N=16 obținem K=5.
Cerință: pentru comunicațiile dintre tată și fiecare fiu, se va folosi (câte) o mapare anonimă.

6. ['Ping-pong' pattern #2 : "*Heigh-Ho, Heigh-Ho, ...*" (v1, using regular file for syncronization)] Hide / show this problem

7. ['Ping-pong' pattern #3 (v1, using anon mmap for syncronization)] Show / hide this problem

Se consideră două fișiere, `nume.txt` și `telefon.txt`, în care sunt scrise, pe câte o linie, numele și respectiv numerele de telefon ale unor persoane (se va trata și cazul de excepție când nu există o corespondență bijectivă la nivel de linie între cele două fișiere).
Să se scrie un program C care să creeze un fiu, 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 în fișierul `agenda_tel.txt`, iar fiul va citi, în mod repetat, câte o linie cu date din fișierul `telefon.txt` și o va scrie în fișierul `agenda_tel.txt`.
În plus, cele două procese trebuie să se sincronizeze conform șablonului 'Ping-pong', folosind o mapare anonimă, astfel încât informațiile să apară exact pe câte o linie, sub forma "NUME - TELEFON", în fișierul de ieșire `agenda_tel.txt`, și nu alte combinații posibile de "interclasare" a informațiilor parțiale scrise de cele două procese aflate în execuție simultană.
(Indicație: scopul acestui exercițiu este acela de a implementa corect un mecanism de sincronizare de forma "Acum e rândul meu --> acum e rândul tău --> acum e rândul meu --> acum e rândul tău --> ... ș.a.m.d.", folosind comunicații prin intermediul unei zone de memorie partajată (i.e., o mapare anonimă), în locul unui fișier obișnuit.
Notă: după cum spuneam (și) în preambulul acestui laborator, șablonul 'ping-pong' este de fapt o instanță cu `p=2` procese, pentru un șablon mai general de sincronizare între `p` procese, cunoscut în literatura de specialitate sub denumirea de șablonul de sincronizare *token ring*.)

Show / Hide some suggestion for this problem

***Recomandare*:** revedeți ultima observație de la pct.ii) din rezolvarea exercițiului **[Demo 'data race' _shmem #2 : ...]**, alias **['Ping-pong' pattern #0 (using anon mmap for IPC)]**, din suportul de laborator #10; luați de acolo mecanismul de sincronizare proiectat pentru cazul particular al șablonului de cooperare/sincronizare 'Token ring' cu "inelul" de lungime `p=2` procese, și utilizați-l pentru a realiza sincronizarea necesară în problema aceasta.

8. ['Token ring' pattern #1 (v1, using anon mmap for syncronization)] Show / hide this problem

Se consideră 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, folosind o mapare anonimă, 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.

(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.", folosind comunicații prin intermediul unei zone de memorie partajată (i.e., o mapare anonimă), în locul unui fișier obișnuit.)

Show / Hide some suggestion for this problem

***Recomandare*: revedeți ultima observație de la pct.ii) din rezolvarea exercițiului **[Demo 'data race'_shmem #2 : ...]**, alias **['Ping-pong' pattern #0 (using anon mmap for IPC)]**, din suportul de laborator #10; luați de acolo mecanismul de sincronizare proiectat pentru cazul general al șablonului de cooperare/sincronizare 'Token ring' cu "inelul" de lungime arbitrară, și adaptați-l la cazul particular al unui "inel" cu lungimea `p=3` procese, pentru a realiza sincronizarea necesară în problema aceasta.**

9. ['Token ring' pattern #2 : "*Eeny, meeny, miny, moe, ...*" (v1, using anon mmap for syncronization)] Hide / show this problem

Să se scrie un program C care să creeze trei fii, după care se vor realiza următoarele operații: tatăl va scrie pe ecran textul "ini-" în mod repetat, primul fiu va scrie pe ecran textul "mini-" în mod repetat, al doilea fiu va scrie pe ecran textul "maini-" în mod repetat, iar al treilea fiu 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.
(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 #4 --> acum e rândul jucătorului #1 --> acum e rândul jucătorului #2 --> acum e rândul jucătorului #3 --> acum e rândul jucătorului #4 ->... ș.a.m.d.", folosind comunicații prin intermediul unei zone de memorie partajată (i.e., o mapare anonimă), în locul unui fișier obișnuit.)

Show / Hide some suggestion for this problem

***Recomandare*: revedeți ultima observație de la pct.ii) din rezolvarea exercițiului **[Demo 'data race'_shmem #2 : ...]**, alias **['Ping-pong' pattern #0 (using anon mmap for IPC)]**, din suportul de laborator #10; luați de acolo mecanismul de sincronizare proiectat pentru cazul general al șablonului de cooperare/sincronizare 'Token ring' cu "inelul" de lungime arbitrară, și adaptați-l la cazul particular al unui "inel" cu lungimea `p=4` procese, pentru a realiza sincronizarea necesară în problema aceasta.**

10. [Another parallel sorting, based on merge-sort (v1, using regular file for IPC)] Show / hide this problem

Să se implementeze algoritmul de sortare merge sort lucrând pe un fișier ce conține o secvență de numere stocate în format binar, folosind lacăte pe porțiuni din fișier pentru secțiunile critice din program, într-o manieră similară ca la exercițiul rezolvat **[MyCritSec #2 : Parallel sorting]** prezentat în suportul de laborator #7.
Cerință: în loc de lansări simultane în execuție ale programului, printr-o comandă asemănătoare ca la exercițiul amintit mai sus:
`UNIX> ./mergesort 1 & ./mergesort 2 & ./mergesort 3 & ...` ,
acum veți crea prin program procesele necesare, cu apeluri `fork` (și astfel va fi suficientă o singură lansare în execuție a programului).
Practic, programul va crea un arbore binar de procese, "frunzele" făcând comparațiile și inversiunile propriu-zise, iar apoi procesele părinte realizează interclasarea secvențelor parțiale, ordonate de procesele copii.

Show / Hide some suggestion for this problem

***Recomandare*: a se reciti sugestia dată la problema suplimentară **[A perfect k-ary tree of processes #1]**, propusă spre rezolvare în prima parte a acestui laborator).**

11. [Another parallel sorting, based on merge-sort (v2, using mmap-file for IPC)] Show / hide this problem

Re-implementați programul care se cere a fi scris la exercițiul **[Another parallel sorting, based on merge-sort (v1, using regular file for IPC)]** propus spre rezolvare mai sus, astfel încât să utilizeze interfața de prelucrare a fișierelor prin mapare în memorie, plus mecanisme de sincronizare bazate pe memorie partajată, în locul interfeței clasice de acces I/O la disc (i.e., apelurile POSIX `read` și `write`) și a sincronizării bazate pe blocaje pe fișiere.

Show / Hide some suggestion for this problem

***Recomandare*: a se reciti sugestia dată la problema suplimentară **[A perfect k-ary tree of processes #1]**, propusă spre rezolvare în prima parte a acestui laborator).**