Laborator #9 : exerciții de laborator

Sumar:

I) Exerciții de programare cu fișiere mapate în memorie -- diverse procesări de fișiere, operate direct în memorie :

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. [#1: NoVocals_mmap]
   Să se rescrie **programul cerut în exercițiul [#1: The filter NoVocals], propus în laboratorul #6 în setul indicat de profesorul dvs. de laborator**, astfel încât să utilizeze interfața de prelucrare a fișierelor prin mapare în memorie, în locul interfeței clasice de acces I/O la disc (i.e., nu folosiți apelurile de sistem read() și write(), sau cele din biblioteci de genul stdio).
2. [#2: MyCut_mmap]
   Să se rescrie **programul cerut în exercițiul [#2: The command MyCut], propus în laboratorul #6 în setul indicat de profesorul dvs. de laborator**, astfel încât să utilizeze interfața de prelucrare a fișierelor prin mapare în memorie, în locul interfeței clasice de acces I/O la disc (i.e., nu folosiți apelurile de sistem read() și write(), sau cele din biblioteci de genul stdio).

Setul 2

1. [#1: ROT13_mmap]
   Să se rescrie **programul cerut în exercițiul [#1: The filter ROT13], propus în laboratorul #6 în setul indicat de profesorul dvs. de laborator**, astfel încât să utilizeze interfața de prelucrare a fișierelor prin mapare în memorie, în locul interfeței clasice de acces I/O la disc (i.e., nu folosiți apelurile de sistem read() și write(), sau cele din biblioteci de genul stdio).
2. [#2: MyGrep_mmap]
   Să se rescrie **programul cerut în exercițiul [#2: The command MyGrep], propus în laboratorul #6 în setul indicat de profesorul dvs. de laborator**, astfel încât să utilizeze interfața de prelucrare a fișierelor prin mapare în memorie, în locul interfeței clasice de acces I/O la disc (i.e., nu folosiți apelurile de sistem read() și write(), sau cele din biblioteci de genul stdio).

Atenție: cu alte cuvinte, trebuie să rescrieți **cele două programe propuse spre rezolvare în laboratorul #6** (în setul indicat de profesorul dvs. de laborator) astfel încât să utilizeze interfața de prelucrare a fișierelor prin mapare în memorie, în locul interfeței clasice de acces I/O la disc (i.e., nu folosiți apelurile de sistem read() și write(), sau cele din biblioteci de genul stdio).

Iată și câteva sugestii de rezolvare a acestor probleme propuse mai sus:

Show / Hide some suggestions for solving these problems

1) Pentru primul exercițiu (i.e., cel prin care se cere rescrierea unui program de tip filtru) din fiecare dintre seturile de probleme de mai sus, **ideea de rezolvare** constă în următoarele:
Se vor considera cele două cazuri prevăzute în enunțul inițial, din laboratorul #6, al problemei respective, și anume:

- i) cazul general, când fișierul de ieșire este diferit de fișierul de intrare.
  Acest caz se soluționează astfel: fișierul de intrare se mapează în memorie pentru acces *read-only*, similar ca la primul exemplu demonstrativ de mai sus, iar fișierul de ieșire se mapează în memorie pentru acces *write-only*, similar ca la al patrulea exemplu demonstrativ din suportul de laborator. Apoi, conținutul mapării fișierului de intrare se "copie" în maparea din memorie a fișierului de ieșire, aplicând în timpul copierii (în memorie) transformarea de filtrare specificată în problema respectivă. La final, nu uitați să faceți "sincronizarea" conținutului mapării fișierului de ieșire, pe disc.
- ii) cazul particular, când fișierul de ieșire coincide cu fișierul de intrare.
  Acest caz se soluționează astfel: unicul fișier (care are rolul și de intrare, și de ieșire, în acest caz) se mapează în memorie pentru acces *read-write*, similar ca la al doilea exemplu demonstrativ din suportul de laborator. Apoi, conținutul mapării fișierului este modificat "*in-place*" în memorie, aplicându-i transformarea de filtrare specificată în problema respectivă. La final, nu uitați să faceți "sincronizarea" conținutului mapării, pe disc.

2) Pentru al doilea exercițiu (i.e., cel prin care se cere rescrierea unui program ce simulează o comandă uzuală) din fiecare dintre seturile de probleme de mai sus, **ideea de rezolvare** constă în următoarele:
Fișierul de prelucrat se mapează în memorie pentru acces *read-only*, similar ca la primul exemplu demonstrativ din suportul de laborator. Apoi, conținutul mapării fișierului este "prelucrat", în conformitate cu comportamentul comenzii simulate, specificată în problema respectivă, iar rezultatul prelucrării va fi afișat pe ecran (i.e., pe fluxul de ieșire `stdout`, întocmai precum se comportă și comanda simulată).

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

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

1. [MyWc_mmap]
   Să se rescrie programul ilustrat în exercițiul rezolvat [MyWc] prezentat în suportul de laborator #6, astfel încât să utilizeze interfața de prelucrare a fișierelor prin mapare în memorie, în locul interfeței clasice de acces I/O la disc (i.e., nu folosiți apelurile de sistem read() și write(), sau cele din biblioteci de genul stdio).

2. [MyCp_mmap]
   Să se rescrie programul ilustrat în exercițiul rezolvat [MyCp] prezentat în suportul de laborator #6, astfel încât să utilizeze interfața de prelucrare a fișierelor prin mapare în memorie, în locul interfeței clasice de acces I/O la disc (i.e., nu folosiți apelurile de sistem read() și write(), sau cele din biblioteci de genul stdio).

3. [ArithmeticMean_mmap]
   Să se rescrie programul ilustrat în exercițiul rezolvat [ArithmeticMean] prezentat în suportul de laborator #6, astfel încât să utilizeze interfața de prelucrare a fișierelor prin mapare în memorie, în locul interfeței clasice de acces I/O la disc (i.e., nu folosiți apelurile fscanf() și fprintf() din biblioteca stdio). În schimb, pentru conversia datelor din memorie, între reprezentarea 'textuală' și cea 'binară', puteți apela funcțiile sscanf() și sprintf() din biblioteca stdio).

4. [MyExpr_mmap]
   Să se rescrie programul ilustrat în exercițiul rezolvat [MyExpr] prezentat în suportul de laborator #6, astfel încât să utilizeze interfața de prelucrare a fișierelor prin mapare în memorie, în locul interfeței clasice de acces I/O la disc (i.e., nu folosiți apelurile fscanf() și fprintf() din biblioteca stdio). În schimb, pentru conversia datelor din memorie, între reprezentarea 'textuală' și cea 'binară', puteți apela funcțiile sscanf() și sprintf() din biblioteca stdio).

II) **Exerciții de programare a unor probleme de sincronizare și cooperare între procese multiple, prin memorie partajată** :

a) *Exerciții propuse spre rezolvare* :

Încercați să rezolvați singuri măcar una dintre problemele de sincronizare din lista următoare:

1. [MyCS_shmem #1]
   Să se rescrie programul din exercițiul rezolvat [MyCritSec #1] din laboratorul #7, 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 read() și write(), etc.) și a sincronizării bazate pe blocaje pe fișiere.

2. [MyCS_shmem #2 : Parallel sorting]
   Să se rescrie programul de sortare paralelă din exercițiul rezolvat [MyCritSec #2 : Parallel sorting] din laboratorul #7, 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 read() și write(), etc.) și a sincronizării bazate pe blocaje pe fișiere.

3. [MyCREW_shmem #1]
   Să se rescrie programul din exercițiul [MyCREW #1], propus spre rezolvare în laboratorul #7, 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 read() și write(), etc.) și a sincronizării bazate pe blocaje pe fișiere.

4. [MyCS_shmem #2bis : Parallel sorting II]
   Să se rescrie programul de sortare paralelă din exercițiul suplimentar [MyCritSec #2bis : Parallel sorting II], propus spre rezolvare în laboratorul #7, 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 read() și write(), etc.) și a sincronizării bazate pe blocaje pe fișiere.

5. [MyCS_shmem #2game : Parallel sorting III]
   Să se rescrie programul de sortare paralelă din exercițiul suplimentar [MyCritSec #2game : Parallel sorting III], propus spre rezolvare în laboratorul #7, 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 read() și write(), etc.) și a sincronizării bazate pe blocaje pe fișiere.

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

Alte câteva exerciții de programare, a unor probleme de sincronizare studiate în cursurile teoretice #5 și #6:

1. [BinarySemaphoreFor2Processes_shmem]
   Să se scrie un program C care să implementeze soluția software pentru problema secțiunii critice pentru cazul a 2 procese (despre care ați învățat în cursul teoretic #5), și să se ilustreze execuția mutual-exclusivă a secțiunilor critice când se rulează două instanțe ale programului.
   Cerință: se va folosi interfața de prelucrare a fișierelor prin mapare în memorie, în locul interfeței clasice de acces I/O la disc (i.e., apelurile read() și write(), etc.), pentru a asigura memorie comună, partajată de cele două instanțe ale programului.

2. [BinarySemaphoreForNProcesses_shmem]
   Să se scrie un program C care să implementeze soluția software pentru problema secțiunii critice pentru cazul a N>2 procese (despre care ați învățat în cursul teoretic #5), și să se ilustreze execuția mutual-exclusivă a secțiunilor critice când se rulează N instanțe ale programului.
   Cerință: se va folosi interfața de prelucrare a fișierelor prin mapare în memorie, în locul interfeței clasice de acces I/O la disc (i.e., apelurile read() și write(), etc.), pentru a asigura memorie comună, partajată de cele N instanțe ale programului.

3. [Producer-Consumer_shmem]
   Să se scrie un program C care să implementeze soluția pentru problema producător-consumator, pentru cazul cu buffer limitat (despre care ați învățat în cursul teoretic #6), și să se ilustreze execuția mutual-exclusivă a secțiunilor critice când se rulează două sau mai multe instanțe ale programului.
   Cerință: se va folosi interfața de prelucrare a fișierelor prin mapare în memorie, în locul interfeței clasice de acces I/O la disc (i.e., apelurile read() și write(), etc.), pentru a asigura memorie comună, partajată de instanțele programului.

4. [CREW{1,2,3}_shmem]
   Să se scrie un program C care să implementeze soluția pentru problema CREW, pentru fiecare din cele 3 cazuri de prioritate (despre care ați învățat în cursul teoretic #6), și să se ilustreze execuția mutual-exclusivă a secțiunilor critice când se rulează două sau mai multe instanțe ale programului.
   Cerință: se va folosi interfața de prelucrare a fișierelor prin mapare în memorie, în locul interfeței clasice de acces I/O la disc (i.e., apelurile read() și write(), etc.), pentru a asigura memorie comună, partajată de instanțele programului.

5. ['Sleeping barber'_shmem]
   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".
   Pentru comunicarea stomatologului cu pacienții se va folosi un fișier mapat în memorie, pentru a asigura memorie comună, partajată de toate procesele.