Programare concurentă în C (II) :

Gestiunea fișierelor, partea a II-a:

Accesul concurent/exclusiv la fișiere în UNIX *– blocaje pe fișiere*

Cristian Vidrașcu

vidrascu@info.uaic.ro

Sumar

- Introducere
- Exemplu (1): Acces concurent la un fișier
- Blocaje pe fișiere. Primitivele folosite
- Exemplu (2): Acces exclusiv la un fișier
- Exemplu (3): Acces exclusiv optimizat

Introducere

UNIX-ul fiind un sistem multi-tasking, în mod uzual este permis *accesul concurent* la fișiere, adică mai multe procese pot accesa “simultan” în citire și/sau în scriere un același fișier, sau chiar o aceeași înregistrare dintr-un fișier.

Acest acces concurent (“simultan”) la un fișier de către procese diferite poate avea uneori efecte nedorite (ca de exemplu, distrugerea integrității datelor din fișier), și din acest motiv s-au implementat în UNIX mecanisme care să permită *accesul exclusiv* (*i.e.*, un singur proces are permisiunea de acces la un moment dat) la un fișier, sau chiar la o anumită înregistrare dintr-un fișier.

Exemplu (1): Acces concurent la un fișier

Un program ce exemplifică accesul concurent la fișiere:

A se vedea programul `access_v1.c`

Exemplu (1): Acces concurent la un fișier

Un program ce exemplifică accesul concurent la fișiere:

A se vedea programul `access_v1.c`

Mai întâi, un demo cu *accesul secvențial la fișier*:

Creăm un fișier `fis.dat` ce conține următoarea linie de text:

```
aaa#bbb#ccc#ddd#eee
```

Apoi lansăm în execuție secvențială mai multe instanțe ale acestui program, e.g. prin comanda:

```
UNIX> access_v1 1 ; access_v1 2 ; access_v1 3
```

Care va fi conținutul fișierului după terminarea execuției ?

?

Exemplu (1): Acces concurent la un fișier

Un program ce exemplifică accesul concurent la fișiere:

A se vedea programul `access_v1.c`

După execuția primei instanțe, fișierul va arăta astfel:

```
aaa1bbb#ccc#ddd#eee
```

După execuția instanței a doua, fișierul va arăta astfel:

```
aaa1bbb2ccc#ddd#eee
```

Iar la final, după execuția instanței a treia, fișierul va arăta astfel:

```
aaa1bbb2ccc3ddd#eee
```

Exemplu (1): Acces concurent la un fișier

Un program ce exemplifică accesul concurent la fișiere:

A se vedea programul `access_v1.c`

Iar acum, un demo cu *accesul concurent la fișier*:

"Reinițializăm" fișierul `fis.dat` cu următoarea linie de text:

```
aaa#bbb#ccc#ddd
```

Apoi lansăm în execuție simultană două instanțe ale acestui program, prin comanda:

```
UNIX> access_v1 1 & access_v1 2 &
```

Care va fi conținutul fișierului după terminarea execuției ?

?

Exemplu (1): Acces concurent la un fișier

Un program ce exemplifică accesul concurent la fișiere:

A se vedea programul `access_v1.c`

Probabil vă așteptați ca după execuție fișierul să arate astfel:

`aaa1bbb2ccc#ddd` sau `aaa2bbb1ccc#ddd`

(în funcție de care dintre cele două procese a reușit mai întâi să suprascrie primul caracter '#' din acest fișier, celuilalt proces rămânându-i al doilea caracter '#' pentru a-l suprascrie.)

Exemplu (1): Acces concurent la un fișier

Un program ce exemplifică accesul concurent la fișiere:

A se vedea programul `access_v1.c`

Probabil vă așteptați ca după execuție fișierul să arate astfel:

`aaa1bbb2ccc#ddd` sau `aaa2bbb1ccc#ddd`

(în funcție de care dintre cele două procese a reușit mai întâi să suprascrie primul caracter '#' din acest fișier, celuilalt proces rămânându-i al doilea caracter '#' pentru a-l suprascrie.)

De fapt, oricâte execuții s-ar face, întotdeauna se va obține:

`aaa1bbb#ccc#ddd` sau `aaa2bbb#ccc#ddd`

Motivul: datorită apelului `sleep(5)` care provoacă o așteptare de 5 secunde între momentul depistării primei înregistrări din fișier care este '#' și momentul suprascrierii acestei înregistrări cu alt caracter.

Blocaje pe fișiere. Primitivele folosite

Sistemul UNIX pune la dispoziție un mecanism de *blocare* (*i.e.* de punere de "*lacăte*") pe porțiuni de fișier pentru acces exclusiv.
Prin acest mecanism se definește o zonă de *acces exclusiv* la fișier.
O asemenea porțiune nu va putea fi accesată în mod concurent de mai multe procese pe toată durata de existență a blocajului.

Pentru a pune un blocaj (*lacăt*) pe fișier se utilizează structura de date:

```
struct flock        /* este definită în fișierul header fcntl.h */
{
  short l_type;    /* indică tipul blocării */
  short l_whence;  /* indică poziția relativă (originea) */
  long  l_start;   /* indică poziția în raport cu originea */
  long  l_len;     /* indică lungimea porțiunii blocate */
  int   l_pid;
}
```

Blocaje pe fișiere. Primitivele folosite

- câmpul `l_type` indică tipul blocării, putând avea ca valoare una dintre constantele:
  - `F_RDLCK` : blocaj în citire
  - `F_WRLCK` : blocaj în scriere
  - `F_UNLCK` : deblocaj (*i.e.* se înlătură lacătul)
- câmpul `l_whence` indică poziția relativă (*i.e.* originea) în raport cu care este interpretat câmpul `l_start`, putând avea ca valoare una dintre următoarele constante simbolice:
  - `SEEK_SET` (=0) : originea este BOF (*i.e. begin of file*)
  - `SEEK_CUR` (=1) : originea este CURR (*i.e. current position in file*)
  - `SEEK_END` (=2) : originea este EOF (*i.e. end of file*)

Blocaje pe fișiere. Primitivele folosite

- câmpul `l_start` indică poziția (*i.e.* *offset*-ul în raport cu originea `l_whence`) de la care începe zona blocată.
  Observație: `l_start` trebuie sa fie negativ pentru `l_whence=SEEK_END`.
- câmpul `l_len` indică lungimea în octeți a porțiunii blocate.
- câmpul `l_pid` este gestionat de funcția `fcntl` care pune blocajul, fiind utilizat pentru a memora `PID`-ul procesului proprietar al acelui lacăt.
  Observație: are sens consultarea acestui câmp doar atunci când funcția `fcntl` se apelează cu parametrul `F_GETLK`.

Blocaje pe fișiere. Primitivele folosite

- câmpul `l_start` indică poziția (*i.e.* *offset*-ul în raport cu originea `l_whence`) de la care începe zona blocată.
  Observație: `l_start` trebuie sa fie negativ pentru `l_whence=SEEK_END`.
- câmpul `l_len` indică lungimea în octeți a porțiunii blocate.
- câmpul `l_pid` este gestionat de funcția `fcntl` care pune blocajul, fiind utilizat pentru a memora `PID`-ul procesului proprietar al acelui lacăt.
  Observație: are sens consultarea acestui câmp doar atunci când funcția `fcntl` se apelează cu parametrul `F_GETLK`.

Pentru a pune lacătul pe fișier, după ce s-au completat câmpurile structurii de mai sus, trebuie apelată funcția `fcntl`.

Blocaje pe fișiere. Primitivele folosite

Interfața funcției `fcntl` (una dintre ele, cea pentru blocaje):

```
int fcntl(int fd, int mod, struct flock* sfl)
```

- *fd* = descriptorul de fișier deschis pe care se pune lacătul
- *sfl* = adresa structurii `flock` ce definește acel lacăt
- *mod* = indică modul de punere, putând lua una dintre valorile:
  - `F_SETLK` : permite punerea unui lacăt pe fișier (în citire sau în scriere, funcție de tipul specificat în structura `flock`). În caz de eșec datorită conflictului cu alt lacăt, se setează variabila `errno` la valoarea `EACCES` sau la `EAGAIN`.
  - `F_GETLK` : permite extragerea informațiilor despre un lacăt pus pe fișier.
  - `F_SETLKW` : permite punerea/scoaterea lacătelor în mod “blocant”, adică se așteaptă (*i.e.* funcția nu returnează) pînă când se poate pune lacătul. Motive posibile: se încearcă blocarea unei zone deja blocate de un alt proces, ș.a.
- valoarea returnată este `0`, sau `-1` în caz de eroare.

Blocaje pe fișiere. Primitivele folosite

Observații:

- Câmpul `l_pid` din structura `flock` este actualizat de funcția `fcntl`.
- Blocajul este scos automat atunci când procesul care l-a pus închide acel fișier, sau își termină execuția.
- Scoaterea (deblocarea) unui segment dintr-o porțiune mai mare anterior blocată poate produce două segmente blocate.
- Blocajele (lacătele) nu se transmit proceselor fii în momentul creării acestora cu funcția `fork`.
  Motivul: fiecare lacăt are în structura `flock` asociată `PID`-ul procesului care l-a creat (și care este deci proprietarul lui), iar procesele fii au, bineînțeles, `PID`-uri diferite de cel al părintelui.

Blocaje pe fișiere. Primitivele folosite

Observații (cont.):

- Important: lacătele în scriere (*i.e.* cele cu tipul `F_WRLCK`) sunt exclusive, iar cele în citire (*i.e.* cele cu tipul `F_RDLCK`) sunt partajate, în sensul CREW (“Concurrent Read or Exclusive Write”).
  Cu alte cuvinte:
  În orice moment, pentru orice porțiune dintr-un fișier, cel mult un proces poate deține un lacăt în scriere pe acea porțiune (și atunci nici un proces nu poate deține concomitent vreun lacăt în citire), sau este posibil ca mai multe procese să dețină lacăte în citire pe acea porțiune (și atunci nici un proces nu poate deține concomitent vreun lacăt în scriere).
- Pentru a putea pune un lacăt în citire, respectiv în scriere, pe un descriptor de fișier, acesta trebuie să fi fost anterior deschis în citire, respectiv în scriere.

Blocaje pe fișiere. Primitivele folosite

Observații (cont.):

- Important: funcționarea corectă a lacătelor se bazează pe *cooperarea* proceselor pentru asigurarea accesului exclusiv la fișiere, *i.e.* toate procesele care vor să acceseze mutual exclusiv un fișier (sau o porțiune dintr-un fișier) vor trebui să folosească lacăte pentru accesul respectiv.

  Altfel, spre exemplu, dacă un proces scrie direct un fișier (sau o porțiune dintr-un fișier), apelul său de scriere nu va fi împiedicat de un eventual lacăt în scriere (sau citire) pus pe acel fișier (sau acea porțiune de fișier) de către un alt proces.

  Cu alte cuvinte: blocajele puse pe fișiere sunt *advisory*, nu sunt *mandatory*!

Exemplu (2): Acces exclusiv la un fișier

Putem rescrie programul inițial folosind lacăte în scriere pentru a inhiba accesul concurent la fișier:

A se vedea programul `access_v2.c`

Exemplu (2): Acces exclusiv la un fișier

Putem rescrie programul inițial folosind lacăte în scriere pentru a inhiba accesul concurent la fișier:
A se vedea programul `access_v2.c`

Refacem fișierul `fis.dat` conținând linia de text:

```
aaa#bbb#ccc#ddd
```

și apoi lansăm în execuție simultană două instanțe ale acestui program, prin comanda:

```
UNIX> access_v2 1 & access_v2 2 &
```

Care va fi conținutul fișierului după terminarea execuției ?

?

Exemplu (2): Acces exclusiv la un fișier

Putem rescrie programul inițial folosind lacăte în scriere pentru a inhiba accesul concurent la fișier:
A se vedea programul `access_v2.c`

De data aceasta, oricâte execuții s-ar face, întotdeauna se va obține rezultatul urmărit:

`aaa1bbb2ccc#ddd` sau `aaa2bbb1ccc#ddd`

Exemplu (2): Acces exclusiv la un fișier

Putem rescrie programul inițial folosind lacăte în scriere pentru a inhiba accesul concurent la fișier:
A se vedea programul `access_v2.c`

Observație: în programul anterior apelul de punere a lacătului era neblocant (*i.e.*, cu parametrul `F_SETLK`). Se poate face și un apel blocant, *i.e.* funcția `fcntl` nu va returna imediat, ci va sta în așteptare până când reușește să pună lacătul.

A se vedea programul `access_v2w.c`

Lansând simultan în execuție două instanțe ale acestui program, se va constata că obținem același rezultat ca și în cazul variantei neblocante.

Exemplu (2): Acces exclusiv la un fișier

Putem rescrie programul inițial folosind lacăte în scriere pentru a inhiba accesul concurent la fișier:
A se vedea programul `access_v2.c`

Justificarea observației despre caracterul *advisory* al blocajelor:
Refacem fișierul `fis.dat` conținând linia de text: `aaa#bbb#ccc#ddd` ,
și apoi rulăm următoarea comandă:

```
UNIX> access_v2 1 & sleep 2 ; echo "AAAAA" > fis.dat
```

La finalul execuției, fișierul `fis.dat` va conține `AAA1A` , ceea ce ne demonstrează că suprascrierea făcută de comanda `echo` în fișier s-a executat în intervalul de timp al celor 5 secunde în care instanța `access_v2` deținea blocajul pe fișier!

Exemplu (3): Acces exclusiv optimizat la un fișier

Observație importantă: versiunea a 2-a a programului nostru (ambele variante, și cea neblocantă, și cea blocantă) nu este optimă:
practic, cele două procese își fac treaba *secvențial*, unul după altul, și nu concurent, deoarece de abia după ce se termină acel proces care a reușit primul să pună lacăt pe fișier, va putea începe și celălalt proces să-și facă treaba (*i.e.* parcurgerea fișierului și înlocuirea primului caracter '#' întâlnit).

Această observație ne sugerează că putem îmbunătăți timpul total de execuție, permițând celor două procese să se execute într-adevăr concurent, pentru aceasta fiind nevoie să punem lacăt doar pe un singur caracter (și anume pe primul caracter '#' întâlnit), în loc să blocăm tot fișierul de la început.

Exemplu (3): Acces exclusiv optimizat la un fișier (cont.)

Versiunea a 3-a a acestui program, cu blocaj la nivel de caracter.

Ideea de rezolvare: programul va trebui sa facă următorul lucru: când întâlnește primul caracter '#' în fișier, pune lacăt pe el (*i.e.* pe exact un caracter) și apoi îl rescrie.

A se vedea programul `access_v3.c`

Care va fi conținutul fișierului după terminarea execuției?

?

Exemplu (3): Acces exclusiv optimizat la un fișier (cont.)

Observație: ideea de rezolvare aplicată în programul `access_v3.c` nu este întrutotul corectă, în sensul că nu se va obține întotdeauna rezultatul scontat, deoarece între momentul primei depistări a '#'-ului și momentul reușitei blocajului există posibilitatea ca acel '#' să fie suprascris de celălalt proces!

(*Notă*: tocmai pentru a forța apariția unei situații care cauzează producerea unui rezultat nedorit, am introdus în program acel apel `sleep(5)` între punerea blocajului pe caracterul '#' și rescrierea lui.)

Cum se poate remedia acest neajuns al programului `access_v3.c`?

?

Exemplu (3): Acces exclusiv optimizat la un fișier (cont.)

Observație: ideea de rezolvare aplicată în programul `access_v3.c` nu este întrutotul corectă, în sensul că nu se va obține întotdeauna rezultatul scontat, deoarece între momentul primei depistări a '#'-ului și momentul reușitei blocajului există posibilitatea ca acel '#' să fie suprascris de celălalt proces!

Această idee de rezolvare se poate corecta astfel: după punerea blocajului, se verifică din nou dacă acel caracter este într-adevăr '#' (pentru că între timp s-ar putea să fi fost rescris de celălalt proces), și dacă nu mai este '#', atunci trebuie scos blocajul și reluată bucla de căutare a primului caracter '#' întâlnit în fișier.

v4 $\rightarrow$ *Temă*: adăugați această corecție la programul `access_v3.c` .

Bibliografie obligatorie

Cap.3, §3.2 din manualul, în format PDF, accesibil din pagina disciplinei “Sisteme de operare”:

- `https://profs.info.uaic.ro/~vidrascu/SO/books/ManualID-SO.pdf`

Programele demonstrative amintite pe parcursul acestei prezentări pot fi descărcate de la adresa următoare:

- `https://profs.info.uaic.ro/~vidrascu/SO/cursuri/C-programs/lock/`

Topici avansate

Mai există două interfețe ce oferă lacăte pe fișiere:

- funcția flock → pentru detalii consultați documentația: `man 2 flock`
- funcția lockf → pentru detalii consultați documentația: `man 3 lockf`