Programare concurentă în C (I) :

Gestiunea fişierelor, partea I-a: Primitivele I/O pentru lucrul cu fişiere

Cristian Vidraşcu

vidrascu@info.uaic.ro

Sumar

- Introducere
- Principalele primitive I/O
- Funcţiile I/O din biblioteca standard de C

Introducere

Sistemul de gestiune a fişierelor în `UNIX`/`Linux` furnizează următoarele categorii de apeluri sistem:

- primitive de creare de noi fişiere: `mknod`, `creat`, `mkfifo`, `mkdir`, ş.a.
- primitive de accesare a fişierelor existente: `open`, `read`, `write`, `lseek`, `close`
- primitive de manipulare a *i*-nodului: `chdir`, `chroot`, `chown`, `chmod`, `stat`, `fstat`
- primitiva de creare a canalelor de comunicaţie anonime: `pipe`

Introducere

Sistemul de gestiune a fişierelor în `UNIX`/`Linux` furnizează următoarele categorii de apeluri sistem:

- primitive de extindere a sistemului de fişiere: `mount`, `umount`
- primitive de schimbare a structurii sistemului de fişiere: `link`, `unlink`

Observaţie: în caz de eroare toate aceste primitive returnează valoarea `-1`, precum şi un număr de eroare ce este stocat în variabila globală `errno` (definită în fişierul *header* `errno.h`).

Primitive I/O pentru lucrul cu fişiere

- *Crearea de fişiere ordinare*: cu funcţia `creat`

  (*Notă*: se poate face şi cu funcţia `open`, folosind un anumit parametru.)

  Interfaţa funcţiei `creat`:

  `int creat(char*` *nume_cale*`, int` *perm_acces*`)`

  - *nume_cale* = numele fişierului ce se creează
  - *perm_acces* = drepturile de acces pentru noul fişier creat
  - valoarea returnată este descriptorul de fişier deschis, sau `-1` în caz de eroare.

  Efect: în urma execuţiei funcţiei `creat` se creează fişierul specificat şi este deschis în scriere.

  Observaţie: în cazul când acel fişier deja există, el este trunchiat la zero, păstrându-i-se drepturile de acces pe care le avea.

Primitive I/O pentru lucrul cu fişiere

- *Controlul dreptului de acces la un fişier*: cu `access`

  Interfaţa funcţiei `access`:

  `int access(char*` *nume_cale*`, int` *drept*`)`

  - *nume_cale* = numele fişierului
  - *drept* = dreptul de acces ce se verifică, ce poate fi o combinaţie (*i.e.*, disjuncţie logică pe biţi) a următoarelor constante simbolice:
    - `X_OK` (=1) : procesul apelant are drept de execuţie a fişierului?
    - `W_OK` (=2) : procesul apelant are drept de scriere a fişierului?
    - `R_OK` (=4) : procesul apelant are drept de citire a fişierului?

    Notă: pentru *drept*=`F_OK` (=0) se verifică doar existenţa fişierului.
  - valoarea returnată este `0`, dacă accesul/accesele verificat(e) este/sunt permis(e), respectiv `-1` în caz de eroare.

Primitive I/O pentru lucrul cu fişiere

- *Deschiderea unui fişier*: cu funcţia `open`. Interfaţa funcţiei:
  `int open(char* nume_cale, int tip_desch, int perm_acces)`
  - *nume_cale* = numele fişierului ce se deschide
  - *perm_acces* = drepturile de acces pentru fişier (utilizat numai în cazul creării acelui fişier)
  - *tip_desch* = specifică tipul deschiderii, fiind exact unul dintre `O_RDONLY` ori `O_WRONLY` ori `O_RDWR`, putând fi urmat apoi de o combinaţie (*i.e.*, disjuncţie logică pe biţi) a următoarelor constante simbolice: `O_APPEND`, `O_CREAT`, `O_TRUNC`, `O_EXCL`, `O_CLOEXEC`, `O_NONBLOCK`, ş.a.
  - valoarea returnată este descriptorul de fişier deschis (*i.e.*, indexul în tabela locală de fişiere deschise), sau `-1` în caz de eroare.

Primitive I/O pentru lucrul cu fişiere

- *Citirea dintr-un fişier*: cu funcţia `read`

  Interfaţa funcţiei `read`:

  `int read(int` *df*`, char*` *buffer*`, unsigned` *nr_oct*`)`

  - *df* = descriptorul fişierului din care se citeşte
  - *buffer* = adresa de memorie la care se depun octeţii citiţi
  - *nr_oct* = numărul de octeţi de citit din fişier
  - valoarea returnată este numărul de octeţi efectiv citiţi, dacă citirea a reuşit (chiar şi parţial), sau `-1` în caz de eroare.

  Observaţii: i) numărul de octeţi efectiv citiţi poate fi inclusiv 0, dacă la începutul citirii cursorul în fişier este pe poziţia EOF (*i.e. end-of-file*);

  ii) la sfârşitul citirii cursorul va fi poziţionat pe următorul octet după ultimul octet efectiv citit.

Primitive I/O pentru lucrul cu fişiere

- *Scrierea într-un fişier*: cu funcţia `write`

  Interfaţa funcţiei `write`:

  `int write(int` *df*`, char*` *buffer*`, unsigned` *nr_oct*`)`

  - *df* = descriptorul fişierului în care se scrie
  - *buffer* = adresa de memorie al cărei conţinut se scrie în fişier
  - *nr_oct* = numărul de octeţi de scris în fişier
  - valoarea returnată este numărul de octeţi efectiv scrişi, dacă scrierea a reuşit (chiar şi parţial), sau `-1` în caz de eroare.

  Observaţie: la sfârşitul scrierii cursorul va fi poziţionat pe următorul octet după ultimul octet efectiv scris.

Primitive I/O pentru lucrul cu fişiere

- *Poziţionarea cursorului într-un fişier* (*i.e.* ajustarea deplasamentului curent în fişier): cu funcţia `lseek`.
  Interfaţa funcţiei `lseek`:
  `long lseek(int` *df*`, long` *val_ajust*`, int` *mod_ajust*`)`
  - *df* = descriptorul fişierului ce se poziţionează
  - *val_ajust* = valoarea de ajustare a deplasamentului
  - *mod_ajust* = modul de ajustare, indicat după cum urmează:
    - `SEEK_SET` (=0) : ajustare în raport cu începutul fişierului
    - `SEEK_CUR` (=1) : ajustare în raport cu deplasamentul curent
    - `SEEK_END` (=2) : ajustare în raport cu sfârşitul fişierului
  - valoarea returnată este noul deplasament în fişier (în raport cu începutul fişierului), sau `-1` în caz de eroare.

Primitive I/O pentru lucrul cu fişiere

- *Închiderea unui fişier*: cu funcţia `close`

  Interfaţa funcţiei `close`:

  `int close(int df)`

  - *df* = descriptorul de fişier deschis
  - valoarea returnată este `0`, dacă închiderea a reuşit, respectiv `-1` în caz de eroare.

Primitive I/O pentru lucrul cu fişiere

- *Închiderea unui fişier*: cu funcţia `close`
  Interfaţa funcţiei `close`:
  `int close(int df)`
  - *df* = descriptorul de fişier deschis
  - valoarea returnată este `0`, dacă închiderea a reuşit, respectiv `-1` în caz de eroare.

Observaţie: Maniera uzuală de prelucrare a unui fişier constă în deschiderea fişierului, urmată de o buclă de parcurgere a acestuia cu operaţii de citire şi/sau de scriere, şi eventual cu schimbări ale poziţiei curente în fişier, iar în final închiderea acestuia.

Exemplu: a se vedea programele filtru `dos2unix.c` şi `unix2dos.c`

Alte primitive I/O pentru fişiere (cont.)

- *Duplicarea unui descriptor de fişier*: cu funcţia `dup`
  (o altă funcţie asemănătoare este funcţia `dup2`)
- *Controlul operaţiilor I/O*: cu funcţia `fcntl`
- *Obţinerea de informaţii conţinute de i-nodul unui fişier*: cu funcţiile `stat` sau `fstat`
- *Stabilirea/eliberarea unei legături pentru un fişier*: cu funcţia `link`, respectiv cu `unlink`
- *Schimbarea drepturilor de acces la un fişier*: cu `chmod`
- *Schimbarea proprietarului unui fişier*: cu `chown` şi `chgrp`
- *Transmiterea măştii drepturilor de acces la crearea unui fişier*: cu funcţia `umask`

Alte primitive I/O pentru fişiere (cont.)

- *Crearea pipe-urilor* (*i.e.* canale de comunicaţie anonime): cu `pipe`
- *Crearea fifo-urilor* (*i.e.* canale de comunicaţie cu nume): cu `mkfifo`
  Interfaţa funcţiei `mkfifo`:
  `int mkfifo(char*` *nume_cale*`, int` *perm_acces*`)`
  - *nume_cale* = numele fişierului *fifo* ce se creează
  - *perm_acces* = drepturile de acces pentru acesta
  - valoarea returnată este `0` în caz de succes, sau `-1` în caz de eroare.
- *Montarea/demontarea unui sistem de fişiere*: cu funcţia `mount`, respectiv cu funcţia `umount`

Alte primitive I/O pentru fişiere (cont.)

- *Crearea/ştergerea unui director*: cu funcţia `mkdir`, respectiv cu funcţia `rmdir`
- *Aflarea directorului curent de lucru*: cu funcţia `getcwd`
- *Schimbarea directorului curent*: cu funcţia `chdir`
- *Prelucrarea fişierelor dintr-un director*: cu funcţiile `opendir`, `readdir` şi `closedir`

Lucrul cu directoare decurge asemănător ca cel cu fişiere, tot o buclă de forma: deschidere, operaţii de citire, închidere.
Se folosesc structurile de date `DIR` şi `struct dirent` şi funcţiile enumerate mai sus, în felul următor . . .

Şablonul de lucru cu directoare

```
DIR           *dd;     // descriptor de director deschis
struct dirent *de;     // intrare în director

/* deschiderea directorului */
if( (dd = opendir(nume_director)) == NULL) {
   ...    // tratează eroarea
}

/* prelucrarea secvenţială a tuturor intrărilor din director */
while( (de = readdir(dd)) != NULL) {
   ...    // prelucrează intrarea curentă, ce are numele : de->d_name
}

/* închiderea directorului */
closedir(dd);
```

Funcţiile I/O din biblioteca standard de C

Biblioteca standard de C conţine un set de funcţii I/O (cele din *header*-ul `stdio.h`), care permit şi ele prelucrarea unui fişier în maniera uzuală:

- `fopen` = pentru deschidere
- `fread`, `fwrite` = pentru citire, respectiv scriere binară
- `fscanf`, `fprintf` = pentru citire, respectiv scriere formatată
- `fclose` = pentru închidere

Observaţie: acestea sunt funcţii de bibliotecă (nu sunt apeluri sistem) şi lucrează *buffer*-izat, cu *stream*-uri I/O, iar descriptorii de fişiere utilizaţi de ele nu sunt de tip `int`, ci de tip `FILE*`.

Notă: implementările acestor funcţii de bibliotecă utilizează totuşi apelurile de sistem corespunzătoare.

Bibliografie obligatorie

Cap.3, §3.1 din manualul, în format PDF, accesibil din pagina disciplinei “Sisteme de operare”:

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

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

- `http://profs.info.uaic.ro/~vidrascu/SO/cursuri/C-programs/file/`