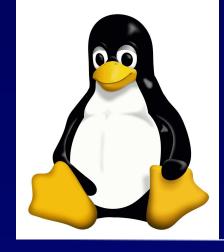
Programare de sistem în C pentru platforma Linux (I)

Gestiunea fișierelor, partea l-a: Primitivele I/O pentru lucrul cu fișiere

Cristian Vidrașcu

cristian.vidrascu@info.uaic.ro

Aprilie, 2025



Sumar

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere

Biblioteca standard de C: funcții pentru operații I/O cu fișiere

Referințe bibliografice

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere

Principalele categorii de primitive I/O

Primitiva access

Primitiva creat

Primitiva open

Primitiva read

Primitiva write

Primitiva Iseek

Primitiva close

Demo: exemple de sesiuni de lucru cu fișiere

Alte primitive I/O pentru fișiere

Primitive I/O pentru directoare

Sablonul de lucru cu directoare

Despre file-system cache-ul gestionat de nucleul Linux

Biblioteca standard de C: funcții pentru operații I/O cu fișiere

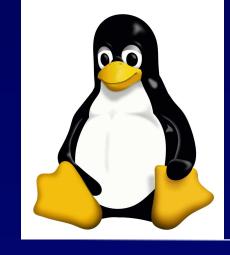
Despre biblioteca standard de C

Functiile I/O din biblioteca standard de C

Funcțiile de bibliotecă pentru I/O formatat

Demo: un exemplu de sesiune de lucru cu fisiere

Referințe bibliografice



Introducere

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere

Biblioteca standard de C: funcții pentru operații I/O cu fisiere

Referinte bibliografice

Funcțiile pe care le puteți apela în programele C pe care le scrieți, pentru a accesa și prelucra fișiere (atât fișiere obișnuite, cât și directoare sau alte tipuri de fișiere), se împart în două categorii:

- API-ul POSIX, ce oferă funcții *wrapper* pentru apelurile de sistem furnizate de nucleul Linux; aceste funcții pot fi apelate din programe C ce vor fi compilate pentru platforma Linux și, mai general, pentru orice sistem de operare din familia UNIX ce implementează standardul POSIX.
 - Avantaj: funcțiile din acest API oferă, practic, acces la toate funcționalitățile din nucleul Linux "exportate" către user-mode.
 - Dezavantaj: programele care folosesc aceste funcții nu sunt portabile, *e.g.* nu pot fi compilate pentru platforma Windows (cel puțin nu direct, ci doar în mediul WINDOWS SUBSYTEM FOR LINUX, introdus în Windows 10).
- C STANDARD LIBRARY (biblioteca standard de C), ce oferă o serie de funcții de nivel mai înalt, inclusiv pentru lucrul cu fișiere; aceste funcții pot fi apelate din programe C ce vor fi compilate pentru orice platformă ce oferă un compilator de C, plus o implementare a bibliotecii standard de C. Spre exemplu, pentru platforma Linux cel mai folosit este compilatorul GCC (the GNU Compiler Collection) și implementarea GLIBC (the GNU libc) a bibliotecii standard de C.
 - Avantaj: permite scrierea de programe portabile, între diverse platforme (e.g., Windows, UNIX/Linux, etc.).
 - Dezavantaj: conține funcții cu capacitate limitată de a gestiona resursele sistemului de operare (e.g., fișiere), fiind din acest motiv adecvată pentru scrierea unor programe simple.



Agenda

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere

Principalele categorii de primitive I/O

Primitiva access

Primitiva creat

Primitiva open

Primitiva read

Primitiva write

Primitiva Iseek

Primitiva close

Demo: exemple de sesiuni de

lucru cu fișiere

Alte primitive I/O pentru fisiere

Primitive I/O pentru directoare

Sablonul de lucru cu directoare

Despre *file-system cache*-ul gestionat de nucleul Linux

Biblioteca standard de C: funcții pentru operații I/O cu fișiere

Referinte bibliografice

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere

Principalele categorii de primitive I/O

Primitiva access

Primitiva creat

Primitiva open

Primitiva read

Primitiva write

Primitiva Iseek

Primitiva close

Demo: exemple de sesiuni de lucru cu fisiere

Alte primitive I/O pentru fișiere

Primitive I/O pentru directoare

Sablonul de lucru cu directoare

Despre file-system cache-ul gestionat de nucleul Linux

Biblioteca standard de C: funcții pentru operații I/O cu fișiere

Despre biblioteca standard de C

Functiile I/O din biblioteca standard de C

Funcțiile de bibliotecă pentru I/O formatat

Demo: un exemplu de sesiune de lucru cu fisiere

Referințe bibliografice



Principalele categorii de primitive I/O

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere

Principalele categorii de primitive I/O

Primitiva access

Primitiva creat

Primitiva open

Primitiva read

Primitiva write

Primitiva Iseek

Primitiva close

Demo: exemple de sesiuni de

lucru cu fișiere

Alte primitive I/O pentru fisiere

Primitive I/O pentru directoare

Şablonul de lucru cu directoare

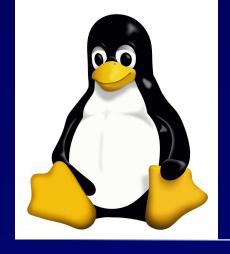
Despre *file-system cache*-ul gestionat de nucleul Linux

Biblioteca standard de C: funcții pentru operații I/O cu fișiere

Referințe bibliografice

Sistemul de gestiune a fișierelor în UNIX/Linux furnizează următoarele categorii de apeluri sistem, în conformitate cu standardul POSIX:

- primitive de creare de noi fişiere, de diverse tipuri: mknod, mkfifo, mkdir, link, symlink, creat, socket
- primitive de ștergere a unor fișiere: rmdir (pentru directoare), unlink (pentru toate celelalte tipuri de fișiere)
- primitiva de redenumire a unui fişier, de orice tip: rename
- primitive de consultare a *i*-nodului unui fișier: stat/fstat/lstat, access
- primitive de manipulare a i-nodului unui fişier: chmod/fchmod, chown/fchown/lchown
- primitive de extindere a sistemului de fișiere: mount, umount
- primitive de accesare și manipulare a conținutului unui fișier, printr-o *sesiune de lucru*: open/creat, read, write, lseek, close, fcntl, ș.a.
- primitive de duplicare, într-un proces, a unei sesiuni de lucru cu un fișier: dup, dup2



Principalele categorii de primitive I/O (cont.)

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere

Principalele categorii de primitive I/O

Primitiva access

Primitiva creat

Primitiva open

Primitiva read

Primitiva write

Primitiva Iseek

Primitiva close

Demo: exemple de sesiuni de

lucru cu fișiere

Alte primitive I/O pentru fișiere Primitive I/O pentru directoare Șablonul de lucru cu directoare Despre *file-system cache*-ul

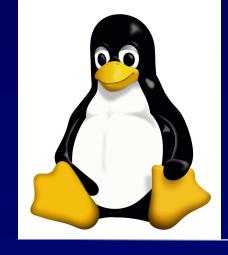
Biblioteca standard de C: funcții pentru operații I/O cu fisiere

gestionat de nucleul Linux

Referinte bibliografice

- primitive pentru consultarea "stării" unor *sesiuni de lucru* cu fișiere (operații I/O sincrone multiplexate): select, poll
- primitiva de "trunchiere" a conținutului unui fișier: truncate/ftruncate
- primitive de modificare a unor atribute dintr-un proces:
 - chdir: modifică directorul curent de lucru
 - umask: modifică "masca" permisiunilor implicite la crearea unui fișier
 - chroot: modifică rădăcina sistemului de fisiere accesibil procesului
- primitive pentru acces exclusiv la fișiere: flock, fcntl
- primitiva de "mapare" a unui fişier în memoria unui proces: mmap
- primitiva de creare, într-un proces, a unui canal de comunicație anonim: pipe
- ş.a.

Observație: în caz de eroare, toate aceste primitive returnează valoarea –1, precum și un număr de eroare ce este stocat în variabila globală errno (definită în fișierul header <errno.h>), eroare ce poate fi diagnosticată cu funcția perror().



Primitiva access

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere
Principalele categorii de primitive I/O

Primitiva access

Primitiva creat

Primitiva open

Primitiva read

Primitiva write

Primitiva Iseek

Primitiva close

Demo: exemple de sesiuni de

lucru cu fișiere

Alte primitive I/O pentru fișiere

Primitive I/O pentru directoare

Sablonul de lucru cu directoare

Despre *file-system cache*-ul gestionat de nucleul Linux

Biblioteca standard de C: funcții pentru operații I/O cu fișiere

Referinte bibliografice

- Verificarea permisiunilor de acces la un fișier: primitiva access.
 Interfața funcției access ([5]):
 int access (char* nume_cale, int perm_acces)
 - nume_cale = numele fișierului ce se verifică
 - perm_acces = dreptul de acces ce se verifică, ce poate fi o combinație (i.e., disjuncție logică pe biți) a următoarelor constante simbolice:
 - X_OK (=1): procesul apelant are drept de execuție a fișierului?
 - $_{\text{W}}$ $_{\text{U}}$ $_{\text{$
 - R_OK (=4): procesul apelant are drept de citire a fișierului?

Observații: i) pentru $perm_acces = F_OK$ (=0) se verifică doar existența fișierului; ii) celelalte drepturi, dacă sunt verificate, implică existența fișierului; iii) aici, prin procesul apelant se înțelege proprietarul real al acestuia, nu proprietarul efectiv.

valoarea returnată este 0, dacă accesul(ele) verificat(e) este/sunt permis(e),
 respectiv -1 în caz că cel puțin unul dintre drepturi este interzis sau alte erori.



Primitiva creat

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere
Principalele categorii de primitive I/O

Primitiva access

Primitiva creat

Primitiva open

Primitiva read

Primitiva write

Primitiva Iseek

Primitiva close Demo: exemple de sesiuni de lucru cu fisiere

Alte primitive I/O pentru fișiere Primitive I/O pentru directoare Șablonul de lucru cu directoare Despre *file-system cache*-ul gestionat de nucleul Linux

Biblioteca standard de C: funcții pentru operații I/O cu fișiere

Referinte bibliografice

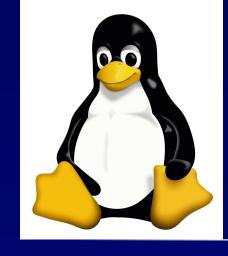
Crearea de fișiere de tip obișnuit: primitiva creat.
Interfața funcției creat ([5]):
int creat(char* nume_cale, mode_t perm_acces)

- nume_cale = numele fișierului ce se creează
- perm_acces = permisiunile de acces pentru noul fişier creat
- valoarea returnată este descriptorul de fișier deschis, sau -1 în caz de eroare.

Efect: în urma execuției funcției creat se creează fișierul specificat și, în plus, acesta este "deschis" în scriere (!), valoarea returnată având aceeași semnificație ca la funcția open.

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

```
Notă: practic, un apel creat(nume_cale, perm_acces); este echivalent cu apelul următor: open(nume_cale, O_WRONLY | O_CREAT | O_TRUNC, perm_acces);
```



Primitiva open

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere
Principalele categorii de

primitive I/O

Primitiva access

Primitiva creat

Primitiva open

Primitiva read

Primitiva write

Primitiva Iseek

Primitiva close

Demo: exemple de sesiuni de

lucru cu fișiere

Alte primitive I/O pentru fișiere Primitive I/O pentru directoare Șablonul de lucru cu directoare Despre *file-system cache-*ul gestionat de nucleul Linux

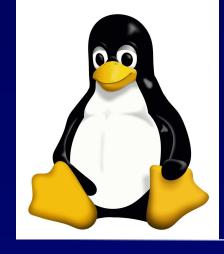
Biblioteca standard de C: funcții pentru operații I/O cu fișiere

Referințe bibliografice

"Deschiderea" unui fișier, i.e. inițializarea unei sesiuni de lucru: primitiva open. Interfața funcției open ([5]):

```
int open(char* nume_cale, int tip_desch, mode_t perm_acces)
```

- nume_cale = numele fișierului ce se deschide
- perm_acces = permisiunile de acces pentru fişier (utilizat numai în cazul în care apelul va avea ca efect crearea acelui fişier)
- tip_desch = specifică tipul deschiderii, putând fi exact una singură dintre valorile O_RDONLY ori O_WRONLY ori O_RDWR, şi, eventual, combinată cu o combinație (i.e., disjuncție logică pe biți) a unora dintre următoarele constante simbolice: O_APPEND, O_CREAT, O_TRUNC, O_CLOEXEC, O_NONBLOCK, O_EXCL, O_DIRECT, O_SYNC, O_ASYNC, ş.a.
- valoarea returnată este descriptorul de fișier deschis, sau -1 în caz de eroare.
 Observație: descriptorul de fișier este indexul unei noi intrări create în tabela locală procesului de fișiere deschise, care referențiază o nouă intrare creată în tabela globală de fișiere deschise la nivel de sistem. Pentru mai multe detalii a se vedea man 2 open.



Primitiva read

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere
Principalele categorii de

principalele categorii de primitive I/O

Primitiva access

Primitiva creat

Primitiva open

Primitiva read

Primitiva write

Primitiva Iseek

Primitiva close

Demo: exemple de sesiuni de

lucru cu fișiere

Alte primitive I/O pentru fișiere Primitive I/O pentru directoare Șablonul de lucru cu directoare Despre *file-system cache*-ul gestionat de nucleul Linux

Biblioteca standard de C: funcții pentru operații I/O cu fișiere

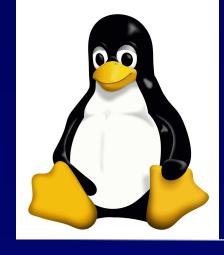
Referinte bibliografice

Citirea dintr-un fișier: primitiva read.
Interfața funcției read ([5]):
int read(int df, char* buffer, size_t nr_oct)

- -df = descriptorul fișierului din care se citește
- buffer = adresa de memorie la care se depun octeții citiți
- nr_oct = numărul de octeți de citit din fișier
- valoarea returnată este numărul de octeți efectiv citiți, dacă citirea a reușit
 (chiar și parțial), sau -1 în caz de eroare.

Observații:

- 1. La sfârșitul citirii cursorul va fi poziționat pe următorul octet după ultimul octet efectiv citit.
- 2. Numărul de octeți efectiv citiți poate fi mai mic decât s-a specificat (*e.g.*, dacă la începutul citirii cursorul în fișier este prea apropiat de sfârșitul fișierului); în particular, acesta poate fi chiar 0, dacă la începutul citirii cursorul în fișier este chiar pe poziția E0F (*i.e.*, *end-of-file*).



Primitiva write

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere Principalele categorii de

primitive I/O

Primitiva access

Primitiva creat

Primitiva open

Primitiva read

Primitiva write

Primitiva Iseek

Primitiva close Demo: exemple de sesiuni de lucru cu fisiere

Alte primitive I/O pentru fisiere Primitive I/O pentru directoare Sablonul de lucru cu directoare Despre file-system cache-ul gestionat de nucleul Linux

Biblioteca standard de C: funcții pentru operații I/O cu fișiere

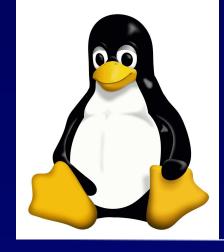
Referinte bibliografice

Scrierea într-un fișier: primitiva write.
Interfața funcției write ([5]):
int write(int df, char* buffer, size_t nr_oct)

- df = descriptorul fișierului în care se scrie
- buffer = adresa de memorie al cărei conținut se scrie în fișier
- nr_oct = numărul de octeți de scris în fișier
- valoarea returnată este numărul de octeți efectiv scriși, dacă scrierea a reușit (chiar și parțial), sau -1 în caz de eroare.

Observații:

- 1. La sfârșitul scrierii cursorul va fi poziționat pe următorul octet după ultimul octet efectiv scris.
- 2. Numărul de octeți efectiv scriși poate fi mai mic decât s-a specificat (*e.g.*, dacă acea scriere ar provoca mărirea spațiului alocat fișierului, iar aceasta nu se poate face din diverse motive lipsă de spațiu liber sau depășire *quota*).



Primitiva Iseek

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere Principalele categorii de

principalele categorii de primitive I/O

Primitiva access

Primitiva creat

Primitiva open

Primitiva read

Primitiva write

Primitiva Iseek

Primitiva close

Demo: exemple de sesiuni de lucru cu fisiere

Alte primitive I/O pentru fișiere Primitive I/O pentru directoare

Şablonul de lucru cu directoare

Despre *file-system cache*-ul gestionat de nucleul Linux

Biblioteca standard de C: funcții pentru operații I/O cu fișiere

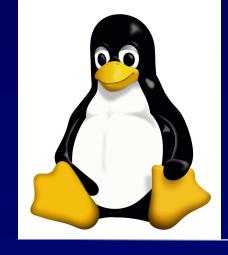
Referințe bibliografice

■ *Poziționarea cursorului într-un fișier* (*i.e.* ajustarea deplasamentului curent în fișier): primitiva lseek.

Interfața funcției lseek ([5]):

long lseek(int df, off_t val_ajust, int mod_ajust)

- df = descriptorul fișierului ce se (re)poziționează
- val_ajust = valoarea de ajustare a deplasamentului
- mod_ajust = modul de ajustare, indicat după cum urmează:
 - SEEK_SET (=0) : ajustare în raport cu începutul fișierului
 - SEEK_CUR (=1): ajustare în raport cu deplasamentul curent
 - ▲ SEEK_END (=2): ajustare în raport cu sfârșitul fișierului
- valoarea returnată este noul deplasament în fișier (întotdeauna, în raport cu începutul fișierului), sau -1 în caz de eroare.



Primitiva close

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere Principalele categorii de

primitive I/O
Primitiva access

Primitiva creat

Primitiva open

Primitiva read

Primitiva write

Primitiva Iseek

Primitiva close

Demo: exemple de *sesiuni de lucru* cu fisiere

Alte primitive I/O pentru fișiere Primitive I/O pentru directoare Șablonul de lucru cu directoare Despre *file-system cache*-ul gestionat de nucleul Linux

Biblioteca standard de C: funcții pentru operații I/O cu fișiere

Referinte bibliografice

"Închiderea" unui fișier, i.e. finalizarea unei sesiuni de lucru: primitiva close. Interfața funcției close ([5]):

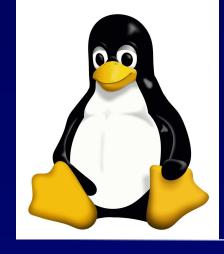
```
int close(int df)
```

- df = descriptorul de fisier deschis
- valoarea returnată este 0, dacă închiderea a reușit, respectiv -1 în caz de eroare.

Observație: maniera uzuală de prelucrare a unui fișier, i.e. o sesiune de lucru, constă în următoarele: "deschiderea fișierului", urmată de o buclă de parcurgere a acestuia cu operații de citire și/sau de scriere, și eventual cu schimbări ale poziției curente în fișier, iar în final "închiderea" acestuia.

Exemplu: a se vedea cele două programe filtru dos2unix.c și unix2dos.c ([2]).

Demo: exercițiile rezolvate [AsciiStatistics] și [MyCp] prezentate în suportul de laborator #6 ([3]) exemplifică alte programe care apelează funcții I/O din API-ul POSIX pentru procesarea unor fișiere.



Demo: exemple de sesiuni de lucru cu fișiere

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere

Principalele categorii de primitive I/O

Primitiva access

Primitiva creat

Primitiva open

Primitiva read

Primitiva write

Primitiva Iseek

Primitiva close

Demo: exemple de sesiuni de

lucru cu fișiere

Alte primitive I/O pentru fișiere Primitive I/O pentru directoare

Sablonul de lucru cu directoare

Despre file-system cache-ul gestionat de nucleul Linux

Biblioteca standard de C: funcții pentru operații I/O cu fișiere

Referinte bibliografice

lată un prim exemplu de program ce efectuează două sesiuni de lucru cu fișiere, mai exact realizează o copiere secvențială a unui fișier dat:

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
\#define BUF_SIZE 4096 // This is exactly the page size, for disk I/O efficiency!
   main (int argc, char *argv []) {
       input_fd, output_fd;
   ssize_t bytes_in, bytes_out;
   char buffer[BUF_SIZE];
      (argc != 3) { printf("Usage: cp file-src file-dest\n"); return 1; }
   input_fd = open(argv[1], O_RDONLY);
      (input_fd == -1) { perror(argv[1]); return 2;
   output_fd = open(argv[2], O_WRONLY|O_CREAT|O_TRUNC, 0600);
      (output_fd == -1) { perror(argv[2]); return 3; }
   while ((bytes_in = read(input_fd, buffer, BUF_SIZE)) > 0) {
       bytes_out = write(output_fd, buffer, bytes_in);
       if (bytes_out != bytes_in) {
           fprintf(stderr, "Fatal write error!\n"); return 4;
   close(input_fd);
                     close(output_fd);
                                        return 0;
```



Demo: exemple de sesiuni de lucru cu fișiere (cont.)

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere

Principalele categorii de primitive I/O

Primitiva access

Primitiva creat

Primitiva open

Primitiva read

Primitiva write

Primitiva Iseek

Primitiva close

Demo: exemple de sesiuni de lucru cu fisiere

Alte primitive I/O pentru fișiere

Primitive I/O pentru directoare

Sablonul de lucru cu directoare

Despre *file-system cache*-ul gestionat de nucleul Linux

Biblioteca standard de C: funcții pentru operații I/O cu fișiere

Referinte bibliografice

lată un al doilea exemplu ce ilustrează o sesiune de lucru cu un fișier, cu folosirea primitivei lseek pentru a citi de la un anumit offset din fișierul dat:

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
int main () {
   int input_fd; long offset; ssize_t bytes_in; char buffer[6];
   input_fd = open("datafile.txt", O_RDONLY);
   if (input_fd == -1) { perror("open"); return 1; }
   offset = lseek(input_fd, 10, SEEK_SET);
   if (offset == -1) { perror("1st lseek"); return 2; }
   bytes_in = read(input_fd, buffer, 5);
      (bytes_in == -1) { perror("1st read"); return 3; }
      (bytes_in != 5) { fprintf(stderr, "1st read warning: insufficient info in file!");}
   buffer[bytes_in]=0; printf("First read from file: %s\n", buffer);
   lseek(input_fd, -10, SEEK_END); /* test for lseek error ... */
   bytes_in = read(input_fd, buffer, 5); /* test for read errors ... */
   buffer[bytes_in]=0; printf("Second read from file: %s\n", buffer);
   close(input_fd); return 0;
```

UNIX> gcc -Wall 2nd_program.c; echo -n "0123456789ABCDEabcde01234" > datafile.txt; ./a.out

```
First read from file: ABCDE Second read from file: abcde
```

Meniu de navigare 15 / 26



Alte primitive I/O pentru fișiere

Introducere

API-ul POSIX: functii pentru operatii I/O cu fisiere Principalele categorii de

primitive I/O

Primitiva access

Primitiva creat

Primitiva open

Primitiva read

Primitiva write

Primitiva Iseek

Primitiva close

Demo: exemple de sesiuni de *lucru* cu fisiere

Alte primitive I/O pentru fisiere

Primitive I/O pentru directoare

Sablonul de lucru cu directoare

Despre file-system cache-ul gestionat de nucleul Linux

Biblioteca standard de C: funcții pentru operații I/O cu fișiere

Meniu de navigare

Referinte bibliografice

- Obtinerea de informații conținute de i-nodul unui fișier: primitivele stat, 1stat sau fstat
- Schimbarea permisiunilor de acces la un fișier: primitiva chmod
- Schimbarea proprietarului unui fișier: primitivele chown și chgrp
- Configurarea măștii permisiunilor de acces la crearea unui fișier: primitiva umask
- Crearea/stergerea unei legături pentru un fisier: primitiva link, respectiv unlink
- "Duplicarea" unui descriptor de fisier: primitivele dup și dup2
- Controlul operațiilor I/O: primitivele fcntl și ioctl
- Montarea / demontarea unui sistem de fișiere: primitiva mount, respectiv umount
- Crearea pipe-urilor (i.e., canale de comunicație anonime): primitiva pipe
- Crearea fișierelor de tip fifo (i.e., canale de comunicație cu nume): primitiva mkfifo Interfata functiei mkfifo ([5]):

```
int mkfifo(char* nume_cale, mode_t perm_acces);
```

- nume cale = numele fisierului 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.



Primitive I/O pentru directoare

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere

Principalele categorii de primitive I/O

Primitiva access

Primitiva creat

Primitiva open

Primitiva read

Primitiva write

Primitiva Iseek

Primitiva close

Demo: exemple de *sesiuni de lucru* cu fisiere

Alte primitive I/O pentru fisiere

Primitive I/O pentru directoare

Şablonul de lucru cu directoare Despre *file-system cache*-ul gestionat de nucleul Linux

Biblioteca standard de C: funcții pentru operații I/O cu fișiere

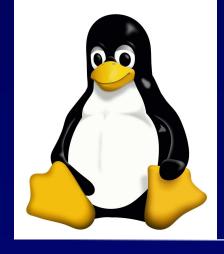
Referinte bibliografice

Crearea/ștergerea unui director: primitiva mkdir, respectiv rmdir Interfața funcției mkdir ([5]):

```
int mkdir(char* nume_cale, mode_t perm_acces);
```

- nume cale = numele directorului ce se creează
- perm_acces = drepturile de acces pentru acesta
- valoarea returnată este 0 în caz de succes, sau -1 în caz de eroare.
- Aflarea directorului curent de lucru al unui proces: primitiva getcwd
- Schimbarea directorului curent de lucru al unui proces: primitiva chdir Interfața funcției chdir ([5]): int chdir(char* nume_cale);
 - nume_cale = numele noului director curent de lucru, al procesului apelant
 - valoarea returnată este 0 în caz de succes, sau -1 în caz de eroare.
- "Prelucrarea" fișierelor dintr-un director: primitivele opendir, readdir și closedir. Alte funcții utile: rewinddir, seekdir, telldir și scandir.

O sesiune de lucru cu directoare se implementează asemănător ca una cu fișiere, *i.e.* este o secvență de forma: "deschidere director", o buclă cu operații de citire, "închidere director".



Sablonul de lucru cu directoare

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere

Principalele categorii de primitive I/O

Primitiva access

Primitiva creat

Primitiva open

Primitiva read

Primitiva write

Primitiva Iseek

Primitiva close

Demo: exemple de sesiuni de

lucru cu fișiere

Alte primitive I/O pentru fișiere

Primitive I/O pentru directoare

Sablonul de lucru cu directoare

Despre *file-system cache-*ul gestionat de nucleul Linux

Biblioteca standard de C: funcții pentru operații I/O cu fișiere

Referinte bibliografice

Se folosesc tipurile de date DIR și struct dirent, împreună cu funcțiile enumerate mai sus, astfel:

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

/* deschiderea directorului */
if( (dd = opendir(nume_director)) == NULL)
{
    ... // TODO: trateaza eroarea la deschidere
}

/* prelucrarea secventiala a tuturor intrarilor din director */
while( (de = readdir(dd)) != NULL)
{
    ... // TODO: prelucreaza intrarea curenta, ce are numele: de->d_name
}

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

Demo: un exemplu de program ce utilizează acest șablon – a se vedea exercițiul rezolvat [MyFind #1] prezentat în suportul de laborator #6 ([3]). De asemenea, acest exemplu ilustrează și folosirea apelului de sistem stat, pentru aflarea proprietăților unui fișier.



Despre file-system cache-ul gestionat de nucleul Linux

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere

Principalele categorii de primitive I/O

Primitiva access

Primitiva creat

Primitiva open

Primitiva read

Primitiva write

Primitiva Iseek

Primitiva close

Demo: exemple de sesiuni de lucru cu fisiere

Alte primitive I/O pentru fișiere Primitive I/O pentru directoare

Şablonul de lucru cu directoare

Despre *file-system cache*-ul gestionat de nucleul Linux

Biblioteca standard de C: funcții pentru operații I/O cu fișiere

Referinte bibliografice

La nivelul componentei de gestiune a sistemelor de fișiere din cadrul nucleului unui SO, se folosește o zonă de memorie internă din *kernel-space* ce implementează un *cache* pentru operațiile cu discul (*i.e.*, se păstrează în memoria RAM conținutul celor mai recent accesate blocuri de disc).

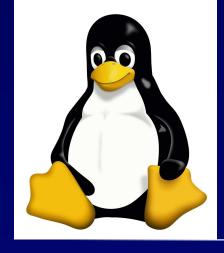
Acest *cache* este denumit *file-system cache* (sau *disk cache*) în literatura de specialitate ([4]), iar el funcționează după aceleași reguli generale ale *cache*-urilor de orice fel:

i) citiri repetate ale aceluiași bloc de disc, la intervale de timp foarte scurte, vor regăsi informația direct din *cache*-ul din memorie; ii) scrieri repetate ale aceluiași bloc de disc, la intervale de timp foarte scurte, vor actualiza informația direct în *cache*-ul din memorie, iar informația stocată pe disc va fi actualizată o singură dată, la momentul operației de *cache-flushing*; iii) operațiile de invalidare/actualizare a informației din *cache*: ...; ș.a.

Granularitatea acestui *cache* (*i.e.*, **unitatea de alocare** în *cache*) este pagina fizică, care are o dimensiune dependentă de arhitectura hardware (*e.g.*, pentru arhitectura x86/x64 dimensiunea paginii este de 4096 octeți). Cu alte cuvinte, operațiile efective de I/O prin DMA între memorie și disc transferă blocuri de informatie cu această dimensiune!

Acest *file-system cache* este unic per sistem, *i.e.* există o singură instanță a sa, gestionată de SO și utilizată simultan (ca și "resursă partajată") de toate procesele ce se execută în sistem.

Observație: puteți citi aici mai multe detalii despre implicațiile existenței acestui *file-system cache* pentru programarea aplicațiilor folosind funcțiile read și write din API-ul POSIX, inclusiv despre utilizarea flagurilor O_SYNC și O_DIRECT pentru a controla folosirea acestui *cache*.



Agenda

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere

Biblioteca standard de C: funcții pentru operații I/O cu fisiere

Despre biblioteca standard de

Funcțiile I/O din biblioteca standard de C Funcțiile de bibliotecă pentru I/O formatat Demo: un exemplu de sesiune

Referințe bibliografice

de lucru cu fisiere

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere

Principalele categorii de primitive I/O

Primitiva access

Primitiva creat

Primitiva open

Primitiva read

Primitiva write

Primitiva Iseek

Primitiva close

Demo: exemple de sesiuni de lucru cu fisiere

Alte primitive I/O pentru fișiere

Primitive I/O pentru directoare

Sablonul de lucru cu directoare

Despre file-system cache-ul gestionat de nucleul Linux

Biblioteca standard de C: funcții pentru operații I/O cu fișiere

Despre biblioteca standard de C

Functiile I/O din biblioteca standard de C

Funcțiile de bibliotecă pentru I/O formatat

Demo: un exemplu de sesiune de lucru cu fisiere

Referințe bibliografice



Despre biblioteca standard de C

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere

Biblioteca standard de C: funcții pentru operații I/O cu fisiere

Despre biblioteca standard de

Funcțiile I/O din biblioteca standard de C Funcțiile de bibliotecă pentru I/O formatat *Demo*: un exemplu de *sesiune de lucru* cu fisiere

Referinte bibliografice

- Biblioteca standard de C conține funcții cu capacitate limitată de a gestiona resursele sistemului de operare (*e.g.*, fișiere)
- Este adeseori adecvată pentru scrierea unor programe simple
- Permite scrierea de programe portabile, între diverse platforme (e.g., Windows, UNIX/Linux, etc.)
- Include fișierele: <stdlib.h>, <stdio.h>,i <string.h>([6])
- Performanță competitivă
- Este restricționată doar la operații I/O sincrone
- Nu avem control al securității fișierelor prin biblioteca standard de C
- Apelul fopen() specifică dacă fișierul este text sau binar
- Sesiunile de lucru cu fișiere sunt identificate prin pointeri către structuri FILE
 - NULL semnifică valoare invalidă
 - Pointerii sunt "handles" pentru obiecte de tipul sesiune de lucru cu un fișier
- Erorile sunt diagnosticate cu funcțiile perror() sau ferror()



Funcțiile I/O din biblioteca standard de C

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere

Biblioteca standard de C: funcții pentru operații I/O cu fisiere

Despre biblioteca standard de C

Funcțiile I/O din biblioteca standard de C

Funcțiile de bibliotecă pentru I/O formatat

Demo: un exemplu de sesiune de lucru cu fisiere

Referinte bibliografice

Biblioteca standard de C conține un set de funcții I/O (cele din *header*-ul <stdio.h> ([6])), care permit și ele prelucrarea unui fișier în maniera uzuală:

- **fopen** = pentru "deschiderea" fișierului
- fread, fwrite = pentru citire, respectiv scriere binară
- fscanf, fprintf = pentru citire, respectiv scriere formatată
- fclose = pentru "închiderea" fișierului

Observație: acestea sunt funcții de bibliotecă (nu sunt apeluri sistem) și lucrează buffer-izat, cu stream-uri I/O, iar descriptorii de fișiere utilizați de ele nu sunt de tip int, ci de tip FILE*.

Notă: implementările acestor funcții de bibliotecă utilizează totuși apelurile de sistem corespunzătoare fiecărei platforme în parte (*i.e.*, Windows vs. Linux/UNIX).

Observație: sunt mult mai multe funcții I/O în biblioteca <stdio.h>; pentru a vedea lista lor și descrierea bibliotecii standard de I/O, inclusiv detalii despre cele trei fluxuri I/O standard (i.e., stdin, stdout și stderr), vă recomand consultarea paginii de manual man 3 stdio ([6]).



Funcțiile I/O din biblioteca standard de C (cont.)

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere

Biblioteca standard de C: funcții pentru operații I/O cu fisiere

Despre biblioteca standard de C

Funcțiile I/O din biblioteca standard de C

Funcțiile de bibliotecă pentru I/O formatat

Demo: un exemplu de sesiune de lucru cu fisiere

Referinte bibliografice

Ce înseamnă că aceste funcții de bibliotecă lucrează buffer-izat?

Răspuns: înseamnă că folosesc un *cache* pentru disc implementat la nivelul bibliotecii standard de C, adică "deasupra" *file-system cache*-ului gestionat la nivelul nucleului SO-ului, despre care vă voi vorbi la cursurile teoretice.

Cu alte cuvinte, acesta este un *cache* al informațiilor din *file-system cache*, care la rândul său este un *cache* al informațiilor de pe disc.

În plus, acest *cache* gestionat de biblioteca stdio este implementat în *user-space* (la fel ca și toate funcțiile bibliotecii), ceea ce înseamnă că este *unic per proces* și nu per sistem, adică nu există un singur *cache* al bibliotecii care să fie partajat de toate procesele ce utilizează apeluri ale bibliotecii stdio.

Concluzie: rețineți faptul că acest cache gestionat de biblioteca stdio nu este unic per sistem, ca în cazul file-system cache-ului gestionat de SO, ci este "local" procesului.



Funcțiile de bibliotecă pentru I/O formatat

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere

Biblioteca standard de C: funcții pentru operații I/O cu fisiere

Despre biblioteca standard de

Funcțiile I/O din biblioteca standard de C Funcțiile de bibliotecă pentru I/O formatat

Demo: un exemplu de sesiune de lucru cu fisiere

Referinte bibliografice

Biblioteca conține o serie de funcții care efectuează citiri/scrieri "formatate", adică efectuează conversia între cele două reprezentări, *binară* vs. *textuală*, ale fiecărui tip de dată, pe baza unui argument *format* ce descrie conversiile de făcut prin niște "specificatori de format". Funcțiile respective sunt:

- perechea scanf/printf : citire de la stdin/scriere pe stdout;
- perechea fscanf/fprintf: citire dintr-un fișier de pe disc/scriere într-un fișier de pe disc;
- perechea sscanf/sprintf: citire dintr-un *string* în memorie/scriere într-un *string* în memorie.

Argumentul *format* folosește "specificatori de format", de forma '%literă', pentru a descrie diferite tipuri de date și, astfel, determină ce fel de conversie se va face între cele două reprezentări, *binară* vs. *textuală*, ale tipului respectiv de dată. Spre exemplu, iată câtiva specificatori de format si tipul de dată asociat fiecăruia:

- %c : un caracter
- %s : un string (*null-terminated*)
- %d: un int (un întreg cu semn), reprezentarea textuală fiind cea corespunzătoare scrierii numărului în baza 10
- %u : un unsigned int (un întreg fără semn), reprezentarea textuală fiind cea corespunzătoare scrierii numărului în baza 10
- %o : un unsigned int, reprezentarea *textuală* fiind cea corespunzătoare scrierii numărului în baza 8
- %x sau %X : un unsigned int, reprezentarea textuală fiind cea corespunzătoare scrierii numărului în baza 16
- %f: un double (un număr real, cu semn), reprezentarea textuală fiind cea corespunzătoare scrierii numărului în notația cu punct zecimal
- %e : un double, reprezentarea *textuală* fiind cea corespunzătoare scrierii numărului în notatia cu mantisă E
- ş.a.

Pentru detalii suplimentare despre aceste perechi de funcții și despre argumentul *format* utilizat de ele, consultați documentația: man 3 scanf și man 3 printf ([6]).

Suplimentar, puteti consulta si materialul disponibil aici.

Meniu de navigare 24 / 26



Demo: un exemplu de sesiune de lucru cu fișiere

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere

Biblioteca standard de C: funcții pentru operații I/O cu fisiere

Despre biblioteca standard de C

Funcțiile I/O din biblioteca standard de C Funcțiile de bibliotecă pentru I/O formatat Demo: un exemplu de sesiune

Referințe bibliografice

de lucru cu fisiere

lată un exemplu de program ce efectuează două *sesiuni de lucru* cu fișiere, mai exact realizează o copiere secvențială a unui fișier dat:

```
#include <stdio.h>
define BUF_SIZE 4096 // This is exactly the page size, for disk I/O efficiency!
int main (int argc, char *argv []) {
   FILE *input_file, *output_file;
   ssize_t bytes_in, bytes_out;
   char buffer[BUF_SIZE];
      (argc != 3) { printf("Usage: cp file-src file-dest\n"); return 1; }
   input_file = fopen(argv[1], "rb");
      (input_file == NULL) { perror(argv[1]); return 2; }
   output_file = fopen(argv[2], "wb");
   if (output_file == NULL) { perror(argv[2]); return 3; }
   while ((bytes_in = fread(buffer, 1, BUF_SIZE, input_file)) > 0) {
       bytes_out = fwrite(buffer, 1, bytes_in, output_file);
       if (bytes_out != bytes_in) {
           fprintf(stderr, "Fatal write error!\n"); return 4;
   fclose(input_file); fclose(output_file);
   return 0;
```

Notă: acest exemplu este disponibil pentru descărcare de aici: cp_stdio.c ([2]).

Demo: exercițiile rezolvate [ArithmeticMean], [MyExpr] și [MyWc] prezentate în suportul de laborator #6 ilustrează alte exemple de programe care apelează funcții I/O din biblioteca standard de C.



Bibliografie obligatorie

Introducere

API-ul POSIX: funcții pentru operații I/O cu fișiere

Biblioteca standard de C: funcții pentru operații I/O cu fisiere

Referinte bibliografice

- [1] Cap. 3, §3.1 din cartea "Sisteme de operare manual pentru ID", autor C. Vidrașcu, editura UAIC, 2006. Notă: este accesibilă, în format PDF, din pagina disciplinei "Sisteme de operare":
 - https://edu.info.uaic.ro/sisteme-de-operare/SO/books/ManualID-SO.pdf
- [2] Programele demonstrative amintite pe parcursul acestei prezentări pot fi descărcate de la:
 - https://edu.info.uaic.ro/sisteme-de-operare/SO/lectures/Linux/demo/files/
- [3] Suportul de laborator online asociat acestei prezentări:
 - https://edu.info.uaic.ro/sisteme-de-operare/SO/support-lessons/C/suport_lab6.html

Bibliografie suplimentară:

- [4] Cap. 4, 5, 13, 15 și 18 din cartea "The Linux Programming Interface : A Linux and UNIX System Programming Handbook", autor M. Kerrisk, editura No Starch Press, 2010.
 - https://edu.info.uaic.ro/sisteme-de-operare/SO/books/TLPI1.pdf
- [5] POSIX API: man 2 access, man 2 open, man 2 read, man 2 write, s.a.
- [6] C STANDARD LIBRARY: man 3 stdio, man 3 string, man 0p stdlib.h.