Sisteme de Operare 1 - Curs 6

Curs tinut in 2012-2013 *de catre* lector dr. Sanda-Maria Dragos

Apeluri sistem pentru lucru cu fisiere

Orice sistem de operare pune la dispozitia programatorilor o serie de servicii prin intermediul carora acestora li se ofera acces la resursele hardware si software gestionate de sistem: lucrul cu tastatura, cu discurile, cu dispozitivul de afisare, gestionarea fisierelor si directoarelor etc. Aceste servicii se numesc *apeluri sistem*.

slide 2

De cele mai multe ori, operatiile pe care ele le pot face asupra resurselor gestionate sunt operatii simple, cu destul de putine facilitati. De aceea, frecvent, se pot intalni in bibliotecile specifice limbajelor de programare colectii de functii mai complicate care gestioneaza resursele respective, dar oferind programatorului niveluri suplimentare de abstractizare a operatiilor efectuate, precum si importante facilitati in plus. Acestea sunt *functiile de biblioteca*.

Pentru a putea actiona asupra unui fisier, este nevoie inainte de toate de o metoda de a identifica in mod unic fisierul. In cazul functiilor discutate, identificarea fisierului se face printr-un asa-numit *descriptor de fisier* (*file descriptor*). Acesta este un numar intreg care este asociat fisierului in momentul deschiderii acestuia.

Functiile open si close

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open(const char *pathname, int flags, [, mode_t mode]);
```

Deschiderea unui fisier este operatia prin care fisierul este pregatit pentru a putea fi prelucrat in continuare. Aceasta operatie se realizeaza prin intermediul functiei open:

Functia returneaza -1 in caz de eroare. In caz contrar, ea returneaza descriptorul de fisier asociat fisierului deschis.

slide 3

Parametri:

» **pathname** - contine numele fisierului
» **flags** - optiunile de deschidere a fisierului. Este, in realitate un sir de biti, in care fiecare bit sau grupa de biti are o anumita semnificatie. Pentru fiecare astfel de semnificatie exista definite in fisierul *header* C **fcntl.h** cate o constanta. Constantele se pot combina folosind operatorul '|' (*sau logic pe biti*) din C, pentru a seta mai multi biti (deci a alege mai multe optiuni) in parametrul intreg **oflag**. Iata cateva din aceste constante:
» O_RDONLY - deschidere numai pentru citire
» O_WRONLY - deschidere numai pentru scriere
» O_RDWR - deschidere pentru citire si scriere
» O_APPEND - deschidere pentru adaugare la sfarsitul fisierului
» O_CREAT - crearea fisierului, daca el nu exista deja; daca e folosita cu aceasta optiune, functia **open** trebuie sa primeasca si parametrul *mode*.
» O_EXCL - creare "exclusiva" a fisierului: daca s-a folosit O_CREAT si fisierul exista deja, functia **open** va returna eroare
» O_TRUNC - daca fisierul exista, continutul lui este sters
» **mode** - se foloseste numai in cazul in care fisierul este creat si specifica drepturile de acces asociate fisierului. Acestea se obtin prin combinarea unor constante folosind operatorul *sau* ('|'), la fel ca si la optiunea precedenta. Constantele pot fi:
» S_IRUSR - drept de citire pentru proprietarul fisierului (*user*)
» S_IWUSR - drept de scriere pentru proprietarul fisierului (*user*)
» S_IXUSR - drept de executie pentru proprietarul fisierului (*user*)
» S_IRGRP - drept de citire pentru grupul proprietar al fisierului
» S_IWGRP - drept de scriere pentru grupul proprietar al fisierului
» S_IXGRP - drept de executie pentru grupul proprietar al fisierului
» S_IROTH - drept de citire pentru ceilalti utilizatori
» S_IWOTH - drept de scriere pentru ceilalti utilizatori
» S_IXOTH - drept de executie pentru ceilalti utilizatori

Pentru crearea fisierelor poate fi folosita si functia

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

creat (const char *pathname, mode_t mode);
```

slide 4

echivalenta cu specificarea optiunilor O_WRONLY | O_CREAT | O_TRUNC la functia open.

Dupa utilizarea fisierului, acesta trebuie inchis, folosind functia

```
#include <unistd.h>

int close (int fd);
```

slide 5

in care *fd* este descriptorul de fisier obtinut la open.

Functiile read si write

Citirea datelor dintr-un fisier deschis se face cu functia

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);
```

Functia citeste un numar de exact *count* octeti de la pozitia curenta in fisierul al carui descriptor este *fd* si ii pune in zona de memorie indicata de pointerul *buf*.

Este posibil ca in fisier sa fie de citit la un moment dat mai putin de *count* octeti (de exemplu daca s-a ajuns spre sfarsitul fisierului), astfel ca functia read va pune in buffer doar atatia octeti cati poate citi. In orice caz, *functia returneaza numarul de octeti cititi din fisier*, deci acest lucru poate fi usor observat.

Daca s-a ajuns exact la sfarsitul fisierului, functia returneaza zero, iar in caz de eroare, -1.

Scrierea datelor se face cu

slide 6

```
#include <unistd.h>


ssize_t write(int fd, void *buf, size_t count);
```

Functia scrie in fisier primii *count* octeti din bufferul indicat de *buf*. Returneaza -1 in caz de eroare.

Functia lseek

Operatiile de scriere si citire in si din fisier se fac la o anumita pozitie in fisier, considerata pozitia curenta. Fiecare operatie de citire, de exemplu, va actualiza indicatorul pozitiei curente incrementand-o cu numarul de octeti cititi. Indicatorul pozitiei curente poate fi setat si in mod explicit, cu ajutorul functiei lseek:

slide 7

```
#include <sys/types.h>
#include <unistd.h>


off_t lseek(int fd, off_t offset, int whence);
```

Functia pozitioneaza indicatorul la deplasamentul *offset* in fisier, astfel:

» daca parametrul *whence* == SEEK_SET, pozitionarea se face relativ la inceputul fisierului
» daca parametrul *whence* == SEEK_CUR, pozitionarea se face relativ la pozitia curenta
» daca parametrul *whence* == SEEK_END, pozitionarea se face relativ la sfarsitul fisierului

Parametrul *offset* poate lua si valori negative si reprezinta deplasamentul, calculat in octeti.

In caz de eroare, functia returneaza -1.

slide 8

slide 9

slide 10

Functii de biblioteca

Deschiderea/Inchiderea unui fisier

```
#include <stdio.h>

FILE *fopen(const char *path, const char *mode);
```

Functia deschide fisierul indicat prin *path*, creeaza o structura FILE continand informatii despre fisier si returneaza un pointer catre aceasta. Acest pointer va fi elementul care va putea fi folosit in continuare pentru accesarea fisierului. Parametrul *mode* este un sir de caractere care indica modul de deschidere a fisierului. "r" semnifica deschidere pentru citire, "w" deschidere pentru scriere. Poate fi specificat si tipul fisierului: "t" pentru fisier text, "b" pentru fisier binar. Optiunile pot fi combinate, de exemplu sub forma "r+t".

Inchiderea fisierului se face cu

```
#include <stdio.h>

int fclose(FILE *fp);
```

unde *stream* este pointerul spre structura FILE obtinut la deschiderea fisierului.

Operatii asupra fisierelor

```
#include <stdio.h>

int fprintf(FILE *stream, const char *format, ...);
/* scriere in fisier cu formatare; sirul de caractere care specifica formatul
 este similar celui de la instructiunea printf. */

int fscanf(FILE *stream, const char *format, ...);
/* citire din fisier, asemanator cu functia scanf. */

size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
/* citeste din fisierul indicat de stream un numar de nmemb elemente,
 fiecare de dimensiunea size, si le pune in zona de memorie indicata de ptr. */

size_t fwrite( void *ptr, size_t size, size_t nmemb,  FILE *stream);
/* scrie in fisierul indicat de stream un numar de nmemb elemente,
 fiecare de dimensiunea size, pe care le ia din zona de memorie indicata de ptr. */
```

Apelurile sistem dup si dup2

slide 11

```
#include <unistd.h>

int dup(int oldfd);
int dup2(int oldfd, int newfd);
```

Aceste doua functii permit ca un acelasi fisier sa fie accesibil prin doi descriptori diferiti.

dup - face o copie a oldfd

dup2 - face o copie a oldfd in newfd, inchizand, daca este cazul, fisierul care puncta inainte catre oldfd.

slide 12

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdlib.h>

main(){
  int oldfd, newfd;

  oldfd = open("ERORI", O_CREAT|O_WRONLY, 0755);
  if(oldfd < 0){
    fprintf(stderr, "pe stderr: open ERORI imposibil\n");
    fprintf(stdout, "pe stdout: open ERORI imposibil\n");
  exit(1);
  }

  newfd = dup2(oldfd, 2); /*inchide automat fisierul standard stderr*/
  /*noul fisier standard de erori va fi ERORI*/

  if(newfd != 2){
    fprintf(stderr, "pe stderr: dup2 imposibil\n");
    fprintf(stdout, "pe stdout: dup2 imposibil\n");
    exit(1);
  }

  fprintf(stderr, "Mesaj in ERORI prin stderr: dup2 REUSIT\n");
  fprintf(stdout, "Mesaj pe terminal prin stdout: dup2 REUSIT\n");
}
```

Apelul sistem fcntl

```
#include <unistd.h>
#include <fcntl.h>

int fcntl(int fd, int cmd, ... /* arg */ );
```

Furnizeaza sau schimba proprietati ale unui fisier deschis, in fucntie de valorile parametrului *cmd*, astfel:

» F_DUPFD - Duplicarea unui descriptor
» F_GETFD sau F_SETFD – flagurile descriptorului de fisiere
» F_GETFL sau F_SETFL – flagurile de stare a fisierului
» F_GETOWN sau F_SETOWN – manipularea semnalelor I/O
» F_GETLK sau F_SETLK – blocarea fisierelor

slide 13

slide 14

Blocarea fisierelor

Este uneori necesar ca un proces sa poata obtine acces exclusiv asupra unui fisier (cel putin pe perioada efectuarii unei actualizari asupra fisierului).

Sistemul de operare Unix implementeaza ambele tipuri de blocare de fisiere: concilianta si obligatorie. In mod normal însa optiunea obligatorie nu este activata ea trebuind specificata la montarea sistemului de fisiere de catre administratorul sistemului. Mecanismul de blocare concilianta nu impune restrictii fizice asupra accesului la fisiere oferind doar un sistem de atentionare la accesul concurent.

» **blocare concilianta** (advisory) - cand sistemul stie care fisiere au fost blocate si de catre cine, iar procesele coopereaza prin apeluri sistem corespunzatoare la accesarea fisierului.
» **blocare obligatorie** (mandatory) - cand sistemul verifica la fiecare scriere si citire daca fisierul este blocat sau nu.
» *blocare exclusiva* - cand un singur proces are access la fisier sau la portiunea din fisier
» *blocare partajata* - cand portiunea partajata poate fi citita simultan de catre mai multe procese, dar nici un proces nu scrie

Blocare prin fcntl

fcntl - functie sistem care furnizeaza sau schimba proprietati ale unui fisier indicat prin descriptor.

Pentru a bloca un fisier trebuie sa utilizati urmatoarea structura de date:

slide 15

```
struct flock  /* este definita in fisierul header fcntl.h */
{
short   l_type;
                /* indica tipul blocarii, avind ca valoare una dintre constantele:

                   F_RDLCK = blocaj in citire
                   F_WRLCK = blocaj in scriere
                   F_UNLCK = deblocaj (se inlatura lacatul)
                */

short   l_whence;
                /* indica pozitia relativa (originea) in raport cu care este
                   interpretat cimpul l_start:

                   SEEK_SET = 0 = originea este BOF=begin of file
                   SEEK_CUR = 1 = originea este CURR=current position in file
                   SEEK_END = 2 = originea este EOF=end of file
                */

off_t   l_start;
                /* indica pozitia (i.e. offset-ul in raport cu originea l_whence)
                   de la care incepe zona blocata.
                   Observatie: trebuie sa fie negativa pentru l_whence = 2.
                */

off_t   l_len;
                /* indica lungimea in octeti a portiunii blocate */

pid_t   l_pid;
                /* acest cimp este gestionat de functia fcntl care pune blocajul, fiind
                   folosit pentru a memora PID-ul procesului proprietar al acelui lacat

                   Are sens consultarea lui atunci cind functia fcntl se apeleaza cu F_
                */
}
```

apoi se foloseste apelul sistem fcntl:

```
#include <unistd.h>
#include <fcntl.h>

int fcntl(int fd, int cmd, struct flock* lock)
```

slide 16

unde:

» *fd* - descriptorul de fisier deschis care se blocheaza;
» *cmd* - indica modul de blocare, putind lua una dintre valorile:
» F_SETLK - permite blocarea unui fisier (in read sau in write, in functie de tipul specificat in structura flock).
In caz de esec se seteaza variabila errno la EACCES sau la EAGAIN;
» F_GETLK - permite extragerea informatiilor despre blocarea unui fisier;
» F_SETLKW - permite punerea/scoaterea blocajelor in mod "blocant", adica se asteapta (i.e. functia nu returneaza) pina cind se poate realiza blocajul. Posibile motive de asteptare: se incearca blocarea unei zone deja blocate de un alt proces, s.a.
»
» *lock* - adresa structurii flock ce defineste acea blocare;

Blocare prin lockf

slide 17

```
#include <unistd.h>

int lockf(int fd, int cmd, off_t len);
```

unde:

» *fd* - descriptorul de fisier;
» *cmd* - poate lua una dintre valorile:
» F_ULOCK - deblocarea unei regiuni blocate
» F_LOCK - blocarea unei regiuni
» F_TLOCK - testare si blocare daca nu este deja blocata
» F_TEST - testarea unei regiuni daca e blocata sau nu
»
» *len* - spune cati octeti vor fi blocati in fisier incepand cu pozitia curenta; 0=tot fisierul;