Fişiere. Sisteme de fişiere

Vladuca Adrian

455E

Fişiere. Sisteme de fişiere

Fişierul este una din abstracţiile fundamentale în lumea sistemelor de operare; cealaltă abstracţie este procesul. Dacă procesul abstractizează execuţia unei anumite sarcini pe procesor, fişierul abstractizează informaţia persistentă a unui sistem de operare. Un fişier este folosit pentru a stoca informaţiile necesare funcţionării sistemului de operare şi interacţiunii cu utilizatorul.

Un sistem de fişiere este un mod de organizare a fişierelor şi prezentare a acestora utilizatorului. Din punct de vedere al utilizatorului un sistem de fişiere are o structură ierarhică de fişiere şi directoare, începând cu un director rădăcină. Localizarea unei intrări în sistemul de fişiere (fişier sau director) se realizează cu ajutorul unei căi în care sunt prezentate toate intrările de până atunci. Astfel, calea

/usr/local/app/file.txt

înseamnă că directorul rădăcină / are un subdirector usr urmat de subdirectorul local. Acesta are la rândul său are un subdirector app care conţine un fişier file.txt.

Fiecare fişier are asociat, aşadar, un nume cu ajutorul căruia se face identificarea, un set de drepturi de acces şi zone conţinând informaţia utilă.

Sistemele de fişiere suportate de sistemele de operare de tip Unix şi Windows sunt ierarhice. Caracterele interzise în nume sunt / în Unix şi ?, ", /, \, < , > , *, |, : în Windows. De altfel, este recomandat ca denumirile de fişiere să nu se termine cu punct sau spaţiu în Windows.

O diferenţă majoră între Linux/Unix şi Windows este senzitivitatea la litere mari sau mici. Linux/Unix sunt case-sensitive (Data este diferit de data), iar Windows nu este case-sensitive.

Ierarhia sistemului de fişiere Unix are un singur director cunoscut sub numele de root şi notat /, prin care se localizează orice fişier. Notaţia Unix pentru căile fişierelor este un şir de nume de directoare despărţite prin /, urmat de numele fişierului. Există şi căi relative la directorul curent . sau la directorul părinte ...

În Unix nu se face nici o deosebire între fişierele aflate pe partiţiile discului local, pe CD sau pe o maşină din reţea. Toate aceste fişiere vor face parte din ierarhia unică a directorului root. Acest lucru se realizează prin **montare**: sistemele de fişiere vor fi montate într-unul din directoarele sistemului de fişiere rădăcină.

În Windows există mai multe ierarhii, câte una pentru fiecare partiţie şi pentru fiecare locaţie din reţea. Spre deosebire de Unix, delimitatorul între numele directoarelor dintr-o cale este \, şi

pentru căile absolute trebuie specificat numele ierarhiei în forma C:\, E:\ sau \\FILESERVER\myFile (pentru locaţiile din reţea). Ca şi Unix, Windows foloseşte . şi ...

Operaţii pe fişiere

Un **descriptor de fişier** în Unix este un întreg care indexează o tabelă cu pointeri spre structuri care descriu fişierele deschise de un proces. Un program care rulează într-un shell Unix îşi deschide 3 fişiere standard cu file descriptori cu valori speciale:

- **standard input** (cu file descriptorul 0) (implicit, citit de la tastatură)
- **standard output** (cu file descriptorul 1) (implicit, afişat pe display)
- **standard error** (cu file descriptorul 2) (implicit, afişat pe display)

Pentru a asocia alţi descriptori cu obiecte deschise, se foloseşte apelul de sistem open.

Pe Windows noţiunea de bază pentru managementul fişierelor este **handle**-ul, o valoare din care se obţine un pointer spre o structură descriptivă a fişierului. Aceleaşi 3 fişiere standard sunt deschise de fiecare program, şi orice fişier suplimentar se deschide cu OpenFile sau CreateFile.

În continuare, pentru descrierea comportării operaţiilor de intrare-ieşire pe Windows, s-a ales ca toate apelurile să facă parte din API-ul Win32, care este cel mai aproape de kernelul Windows. Sistemul oferă ca alternativă apeluri standard (POSIX, de exemplu, compatibile între Windows şi Linux), dar acestea se implementează în Windows prin apelurile Win32 şi formează un nivel mai îndepărtat de kernel.

Un fişier are asociat **cursorul de fişier** (file pointer) care indică poziţia curentă în cadrul fişierului. Cursorul de fişier este un întreg care reprezintă deplasamentul (offset-ul) faţă de începutul fişierului.

Operaţiile tipice de executat pe un fişier sunt prezentate în continuare. Exceptând deschiderea unui fişier, toate operaţiile primesc ca argument descriptorul sau handle-ul acelui fişier. Pentru deschiderea fişierului se foloseşte ca argument numele acestuia.

- **deschiderea unui fişier**: înseamnă asocierea unui descriptor de fişier sau un handle cu un fişier; acest descriptor sau handle este folosit în cadrul celorlalte operaţii; fişierul este identificat prin nume; apeluri pentru deschiderea unui fişier sunt fopen (ISO C), open (POSIX), OpenFile (Win32 API); aceste apeluri pot fi folosite şi pentru crearea unui fişier; alternativ există apelurile creat (POSIX) şi CreateFile (Win32 API)
- **închiderea unui fişier**: înseamnă eliberarea structurilor de fişier asociate procesului şi a descriptorului (handle-ului) acelui fişier; apelurile sunt fclose (ISO C), close (POSIX), CloseFile (Win32 API)
- **citirea dintr-un fişier**: înseamnă copierea unui bloc de date într-un buffer; după ce se realizează citirea se actualizează cursorul de fişier; apelurile sunt fread (ISO C), read (POSIX), ReadFile (Win32 API)
- **scrierea într-un fişier**: înseamnă copierea unui bloc de date dintr-un buffer în fişier; efectuarea scrierii înseamnă şi actualizarea cursorului de fişier; apelurile sunt fwrite (ISO C), write (POSIX), WriteFile (Win32 API)
- **poziţionarea într-un fişier**: înseamnă schimbarea valorii cursorului de fişier; citiri sau scrieri ulterioare vor porni din locaţia indicată de acest cursor de fişier; apelurile sunt fseek (ISO C), lseek (POSIX), SetFilePointer (Win32 API)
- **schimbarea atributelor unui fişier**: înseamnă stabilirea unor parametri pentru fişier; apelurile sunt fcntl (POSIX), SetFileAttributes (Win32 API)

Operaţii pe fişiere în Linux

În continuare sunt descrise operaţiile de fişiere într-un sistem Linux. Apelurile descrise mai jos sunt valabile în orice sistem compatibil POSIX.

Crearea, deschiderea şi închiderea fişierelor

open

Funcţia este folosită pentru deschiderea unui fişier; funcţia este declarată în fcntl.h iar sintaxa apelului este următoarea:

int open(const char *FILENAME, int FLAGS[, mode_t MODE]);

unde

- FILENAME este numele fişierului care se doreşte deschis
- FLAGS controlează modul în care se realizează deschiderea. Se pot specifica mai multe flag-uri făcând **SAU** pe biţi. Cele mai uzuale flag-uri sunt O_RDONLY, O_WRONLY, O_RDRW, O_CREAT, O_EXCL, O_TRUNC, O_APPEND, O_NONBLOCK/O_NDELAY
- MODE este folosit pentru a stabili drepturile în cazul în care se creează un nou fişier (adică se precizează flag-ul O_CREAT

Dacă operaţia are succes funcţia întoarce file descriptorul alocat (o valoare întreagă pozitivă). În caz de eroare se întoarce -1, iar variabila errno primeşte o valoare care indică motivul eşecului.

Dacă, spre exemplu, dorim să deschidem fişierul alina.txt pentru scriere, cu trunchiere, şi fişierul dan.txt pentru citire şi scriere, cu eventuala creare a acestuia, vom folosi următoarea secvenţă de cod:

Exemplu:

```
[...]
#include <sys/types.h>
#include <sys/stat.h>
```

```
#include <fcntl.h>

[...]
   int fd1, fd2;

   fd1 = open ("alina.txt", O_WRONLY | O_TRUNC);
   if (fd1 < 0) {
      perror ("open");
      exit (EXIT_FAILURE);
   }

   fd2 = open ("dan.txt", O_RDWR | O_CREAT, 0644);
   if (fd1 < 0) {
      perror ("open");
      exit (EXIT_FAILURE);
   }
```

creat

Declarată tot în fcntl.h este şi funcţia creat, care creează un fişier şi are următoarea sintaxă:

int creat(const char *FILENAME, mode_t MODE)

Funcţia este echivalentă cu:

open(FILENAME, O_WRONLY|O_CREAT|O_TRUNC, MODE)

close

Funcţia este declarată în unistd.h iar sintaxa apelului este următoarea:

int close(int FILEDES)

unde FILEDES este filedescriptorul care se doreşte închis.

Dacă dealocarea are loc fără probleme atunci valoarea întoarsă este 0, altfel -1. Ca şi la open, mai multe informaţii despre cum s-a terminat operaţia se pot obţine inspectând variabila errno.

O greşeală frecventă de programare este neverificarea codului de eroare întors la close, pentru că se poate întâmpla ca o eroare la scriere (EIO) să fie întoarsă utilizatorului abia la close.

Scrierea şi citirea

Scrierea într-un fişier şi citirea dintr-un fişier se realizează cu ajutorul apelurilor read şi write. Aceste apeluri primesc trei argumente:

1. descriptorul fişierului folosit
2. buffer-ul ce conţine datele pentru scriere sau unde se stochează datele citite din fişier
3. numărul de octeţi care vor fi citiţi/scrişi

read

Funcţia read este declarată în unistd.h şi are sintaxa de apel:

ssize_t read(int FILEDES, void *BUFFER, size_t SIZE);

unde

- BUFFER e un pointer spre zona în care vor fi depuse datele citite din fişierul indicat de FILEDES
- SIZE e cantitatea maxima de date care poate fi primită.

Funcţia întoarce numărul de octeţi citiţi. Valoarea minimă este de un octet, iar când se ajunge la sfârşit se va întoarce 0. Dacă se face read după ce s-a ajuns la sfârşit se va întoarce în continuare 0. Dacă apare o eroare se întoarce -1 şi variabila errno este setată corespunzător cauzei care a determinat eroarea.

write

Funcţia write este declarată, de asemenea, în unistd.h iar sintaxa este următoarea:

ssize_t write(int FILEDES, const void *BUFFER, size_t SIZE);

unde parametrii au semnificaţii similare cu cei ai apelului read. Valoarea întoarsă este numărul de octeti ce au fost efectiv scrişi. În mod implicit nu se garanteaza că la revenirea din write scrierea în fişier s-a terminat. Pentru a forţa actualizarea se poate folosi fsync sau fişierul se poate deschide folosind flagul O_FSYNC, caz în care se garanteaza că după fiecare write fişierul a fost actualizat.

Observaţie: pentru read şi write există o versiune pread, respectiv pwrite, care permite specificarea unui offset în fişier de la care să se efectueaze operaţia de citire / scriere. Mai există şi o versiune pread64 şi pwrite64 la care offset-urile sunt pe 64 de biţi.

Operaţii pe fişiere în Windows

Sistemele de fişiere folosite pe Windows sunt FAT32 şi NTFS.

Sistemul de fişiere NTFS consideră un fişier ca fiind o înşiruire de octeţi, care formează datele, împreună cu o colecţie de atribute ale fişierului.

Pentru managementul datelor propriu-zise Windows foloseşte noţiunea de **clustere**, care sunt în fapt grupări de sectoare fizice de pe disc; numărul de sectoare dintr-un cluster poate fi adaptat de sistem în funcţie de mărimea unui sector, astfel încât operaţiile pe date să fie optimizate.

În plus, NTFS foloseşte un concept relativ exotic numit **streaming**. Datele dintr-un fişier sunt memorate implicit într-un stream fără nume, dar stream-uri alternative de date pot fi adăugate fişierului; fiecare din ele conţine date şi are atribute de acces proprii care fac streamurile obiecte relativ independente.

Atributele care descriu un obiect de tip fişier în NTFS includ numele fişierului, poziţia curentă a cursorului de fişier, modul de partajare a fişierului între procese, tipul operaţiilor de citire/scriere (de exemplu, sincrone sau asincrone) şi pointeri către locul fizic de pe disc unde sunt stocate datele.

Fişierele şi directoarele sunt şi în Windows obiecte securizate; securizarea unui astfel de obiect se face prin două moduri de securitate:

- **modul de acces** specifică setul de operaţii pe care un proces îl poate face pe un obiect
- **modul de partajare** specifică felul în care mai multe procese pot partaja un acelaşi obiect

Crearea, deschiderea şi închiderea fişierelor

CreateFile

Pentru a crea un handle al unui fişier, director sau al unei alte resurse cu care se comunică (în genul unui port COM, al unui pipe sau modem) se foloseşte funcţia CreateFile, pentru care trebuie inclus headerul windows.h. Funcţia se ocupă atât de crearea, cât şi de deschiderea unui fişier:

```
HANDLE CreateFile(
      LPCTSTR lpFileName,
      DWORD dwDesiredAccess,
      DWORD dwShareMode,
      LPSECURITY_ATTRIBUTES lpSecurityAttributes,
      DWORD dwCreationDisposition,
      DWORD dwFlagsAndAttributes,
      HANDLE hTemplateFile
);
```

Numele fişierului trebuie precizat în parametrul lpFileName [in] în forma unui string cu terminator nul. Convenţiile Windows specifică backslash-ul ca delimitator al componentelor dintr-o cale şi nu permite denumirea fişierelor ca device-uri speciale de sistem (cum ar fi LPT2, COM1); numele se consideră a fi case insensitive şi lungimea maximă a unui nume este de MAXPATH (260) de caractere (incluzand prefix-ul de drive si caracterul '\0' de sfârşit).

Drepturile de acces cerute la deschiderea fişierului sunt specificate in dwDesiredAccess [in]. Pentru simplitatea lucrului cu drepturile de acces, există variabile generice care încapsulează unele drepturi de acces, cum ar fi GENERIC_EXECUTE, GENERIC_READ, GENERIC_WRITE etc.

Un obiect poate fi deschis de mai multe procese, sau poate fi deschis de mai multe ori în acelaşi proces; în aceste cazuri, la prima deschidere parametrul dwShareMode [in] trebuie specificat ca având cel puţin una din valorile:

- FILE_SHARE_DELETE permite unor operaţii de deschidere ulterioare să capete acces de tip *delete*.
- FILE_SHARE_READ permite unor operaţii de deschidere ulterioare să capete acces de tip *read*.
- FILE_SHARE_WRITE permite unor operaţii de deschidere ulterioare să capete acces de tip *write*.

La deschiderea unui fişier se poate preciza prin parametrul lpSecurityAttributes [in] modul în care handle-ul returnat de apel poate fi moştenit de procesele fii ale procesului apelant. Daca lpSecurityAttributes este NULL, handle-ul nu poate fi moştenit.

Parametrul dwCreationDisposition [in] precizează modul în care apelul acţionează în cazul în care fişierul există sau nu; poate avea valori de forma:

- CREATE_ALWAYS creează un fişier nou; dacă fişierul există, apelul îl suprascrie, ştergând atributele existente;
- CREATE_NEW creează un fişier nou; apelul eşuează dacă fişierul există deja;
- OPEN_ALWAYS deschide fişierul, daca acesta există; altfel, se comportă ca şi CREATE_NEW;
- OPEN_EXISTING deschide fişierul; dacă nu există, apelul eşuează;
- TRUNCATE_EXISTING deschide fişierul (cu drept de acces GENERIC_WRITE) şi îl trunchiază la dimensiunea zero; dacă fişierul nu există, apelul eşuează.

Un set de flaguri şi atribute suplimentare (valabile numai în cazul fişierelor) pot fi precizate în dwFlagsAndAttributes [in]. Valori uzuale sunt:

- FILE_ATTRIBUTE_NORMAL fişierul nu are alte atribute setate (folosit numai singur)
- FILE_ATTRIBUTE_READONLY fişierul va fi read only pentru toate procesele

hTemplateFile [in] este un handle la un fişier template cu drepturi de acces pentru citire, template ale cărui atribute şi flaguri vor fi folosite şi pentru fişierul nou creat.

Apelul returnează în caz de succes un handle al fişierului. Dacă dwCreationDisposition este CREATE_ALWAYS sau OPEN_ALWAYS, apelul nu eşuează, dar GetLastError returnează ERROR_ALREADY_EXISTS. În caz de eroare, apelul returnează INVALID_HANDLE_VALUE, iar informaţii suplimentare despre eroare pot fi returnate de GetLastError.

Pentru copierea şi mutarea fişierelor există apelurile CopyFile, MoveFile şi ReplaceFile.

CloseHandle

Când handle-ul nu mai este folosit, fişierul este închis cu apelul generic pentru orice tip de handle-uri:

```
CloseHandle(myFileHandle);
```

unde CloseHandle are signatura

```
BOOL CloseHandle(
      HANDLE hObject
);
```

Ştergerea (după ce fişierul este închis) se face cu

```
CloseHandle(hFile);
DeleteFile("myfile.txt");
```

unde DeleteFile are signatura

```
BOOL DeleteFile(
      LPCTSTR lpFileName
);
```

Citirea şi scrierea unui fişier

Un proces poate face operaţii de citire şi de scriere asupra unui fişier prin apelurile ReadFile, WriteFile şi cele extinse ReadFileEx, WriteFileEx. Apelurile primesc un handle de fişier creat cu drepturi corespunzătoare, scriu sau citesc un număr specificat de octeţi din poziţia curentă a cursorului (sau dintr-o altă poziţie specificată).

ReadFile

ReadFile operează asupra unui fişier care are drepturi de acces cel puţin pentru citire, copiind un număr de octeţi (începând din poziţia curentă a cursorului de fişier sau dintr-o poziţie specificată printr-o structură de tip OVERLAPPED) într-un buffer şi întoarce într-o variabilă numărul de octeţi citiţi. Daca handle-ul de fişier nu este creat pentru operaţii de intrare-ieşire de tip overlapped, atunci cursorul de fişier este updatat la încheierea apelului de citire. Altfel, procesul trebuie să facă asta manual.

Apelul ReadFile poate opera atât sincron, cât şi asincron; ReadFileEx este exclusiv asincronă.

```
BOOL ReadFile(
      HANDLE hFile,
      LPVOID lpBuffer,
      DWORD nNumberOfBytesToRead,
      LPDWORD lpNumberOfBytesRead,
      LPOVERLAPPED lpOverlapped
);
```

ReadFile primeşte un handle de fişier hFile [in], care trebuie să fi fost creat cu un drept de acces cel puţin egal cu GENERIC_READ. Pentru ca operaţia să fie asincronă, hFile trebuie să fi fost deschis cu FILE_FLAG_OVERLAPPED precizat în CreateFile. Bufferul în care se copiază rezultatul citirii este lpBuffer [out].

Numărul de octeţi care se doreşte a fi citit se precizează în nNumberOfBytesToRead [in], iar numărul efectiv citit este returnat în variabila pointată de lpNumberOfBytesRead [out].

ReadFile returnează o valoare diferită de zero în caz de succes, şi zero altfel. Dacă se returnează o valoare diferită de zero, dar numărul de octeţi citiţi este zero, atunci s-a ajuns la sfârşitul de fişier. Dar dacă fişierul a fost deschis cu FILE_FLAG_OVERLAPPED şi există o

structură în lpOverlapped, la sfârşit de fişier se returnează zero şi GetLastError returnează ERROR_HANDLE_EOF.

O operaţie de citire simplă dintr-un fişier poate arăta astfel:

WriteFile

Apelul WriteFile copiază în mod sincron sau asincron un număr specificat de octeţi dintr-un buffer în conţinutul unui fişier şi returnează într-o variabilă numărul efectiv de octeţi copiaţi. Scrierea în fişier se face în general începând din poziţia curentă a cursorului şi după terminarea operaţiei poziţia cursorului fişierului este updatată; dacă handle-ul de fişier a fost creat pentru operaţii suprapuse (overlapped), atunci procesul trebuie să actualizeze singur poziţia cursorului.

```
BOOL WriteFile(
    HANDLE hFile,
    LPCVOID lpBuffer,
    DWORD nNumberOfBytesToWrite,
    LPDWORD lpNumberOfBytesWritten,
    LPOVERLAPPED lpOverlapped
);
```

Handle-ul de fişier în care se scrie hFile [in] trebuie să fi fost creat cu drepturi de acces GENERIC_WRITE, iar pentru ca operaţiile să fie asincrone hFile trebuie să fie deschis cu FILE_FLAG_OVERLAPPED. Parametrii WriteFile au aceleaşi semnificaţii cu parametrii ReadFile, adaptate pentru operaţii de scriere.

Bibliografie:

www.wikipedia.com
www.wikipedia.ro
http://www.stern.nyu.edu
Cursuri electronice de Windows si Linux