Laborator 02 - Operații I/O simple

Materiale ajutătoare

* [lab02-slides.pdf](http://elf.cs.pub.ro/so/res/laboratoare/lab02-slides.pdf)
* [lab02-refcard.pdf](http://elf.cs.pub.ro/so/res/laboratoare/lab02-refcard.pdf)
* [Video Operații IO](http://elf.cs.pub.ro/so/res/tutorial/lab-02-operatii-io/)

Nice to read

* TLPI - Chapter 4, File I/O: The Universal I/O model
* WSP4 - Chapter 2, Using the Windows File System

Fișiere. Sisteme de fișiere

**Fișierul** este una dintre abstractizările fundamentale în domeniul sistemelor de operare; cealaltă abstractizare este procesul. Dacă procesul abstractizează execuția unei anumite sarcini pe procesor, fișierul abstractizează informația persistentă a unui sistem de operare. Un fișier este folosit pentru a stoca informațiile necesare funcționării sistemului de operare și interacțiunii cu utilizatorul.

Un **sistem de fișiere** este un mod de organizare a fișierelor și prezentare a acestora utilizatorului. Din punctul de vedere al utilizatorului, un sistem de fișiere are o structură ierarhică de fișiere și directoare, începând cu un director rădăcină. Localizarea unei intrări (fișier sau director) se realizează cu ajutorul unei căi în care sunt prezentate toate intrările de până atunci. Astfel, pentru calea /usr/local/file.txt directorul rădăcină '/' are un subdirector usr care include subdirectorul local ce conține un fișier file.txt.

Fiecare fișier are asociat, așadar, un nume cu ajutorul căruia se face identificarea, un set de drepturi de acces și zone conținând informația utilă.

Sistemele de fișiere suportate de sistemele de operare de tip Unix și Windows sunt ierarhice. Sistemele Linux/Unix sunt case-sensitive (Data este diferit de data), iar sistemele Windows sunt case-insensitive.

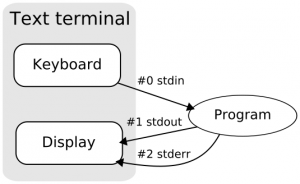
Ierarhia sistemului de fișiere Unix are un singur director cunoscut sub numele de root și notat '/', prin care se localizează orice fișier (a nu se confunda cu directorul /root, care este home-ul utilizatorului privilegiat, root). Notația Unix pentru căile fișierelor este un șir de nume de directoare despărțite prin '/', urmat de numele fișierului. Există și căi relative la directorul curent '.' sau la directorul părinte '..'.

În Unix nu se face nicio deosebire între fișierele aflate pe partițiile discului local, pe CD sau pe o mașină din rețea. Toate aceste fișiere vor face parte din ierarhia unică a directorului root. Acest lucru se realizează prinmontare: sistemele de fișiere vor fi montate într-unul dintre directoarele sistemului de fișiere rădăcină.

În Windows există mai multe ierarhii, câte una pentru fiecare partiție și pentru fiecare loc din rețea. Spre deosebire de Unix, delimitatorul între numele directoarelor dintr-o cale este '\', și pentru căile absolute trebuie specificat numele ierarhiei în forma C:\, E:\ sau \\FILESERVER\myFile (pentru rețea). Ca și Unix, Windows folosește '.' pentru directorul curent și '..' pentru directorul părinte.

Operații pe fișiere

În Unix, un **descriptor de fișier** este un întreg care indexează o tabelă cu pointeri spre structuri care descriu fișierele deschise de un proces. În cazul în care un program rulează într-un shell Unix, procesul părinte (shell-ul) deschide pentru procesul copil (programul respectiv) 3 fișiere standard având descriptori de fișiere cu valori speciale:

[](http://ocw.cs.pub.ro/courses/_detail/so/laboratoare/standard_filedes.png?id=so%3Alaboratoare%3Alaborator-02)

* **standard input**(0) - citirea de la intrarea standard (tastatură)
* **standard output** (1) - afișarea la ieşirea standard (consolă)
* **standard error**(2) - afișarea la ieşirea standard de eroare (consolă)

În Windows, noțiunea de bază pentru managementul fișierelor este **handle**-ul, o valoare din care se obține un pointer spre o structură descriptivă a fișierului. Aceleași 3 fișiere standard sunt deschise de fiecare [proces](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-03#prezentare-concepte).

În continuare, pentru descrierea comportamentului operațiilor de intrare-ieșire 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 și Linux), dar acestea se implementează în Windows prin apelurile Win32 și formează un nivel de abstractizare aflat mai departe de kernel.

Un fișier are asociat cursorul de fișier (file pointer) care indică poziția curentă în cadrul fișierului. Cursorul de fișier este un întreg care reprezintă deplasamentul (offset-ul) față de începutul fișierului.

Operațiile specifice pentru lucrul cu fișiere:

* **deschiderea/crearea unui fișier** - înseamnă asocierea unui descriptor de fișier sau a unui handle cu un fișier identificat prin numele său [1)](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#fn__1). ( [Linux](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#crearea_deschiderea_si_inchiderea_fisierelor), [Windows](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#crearea_deschiderea_si_inchiderea) )
* **închiderea unui fișier** - înseamnă eliberarea structurilor de fișier asociate procesului și a descriptorului (handle-ului) acelui fișier - doar dacă nu mai există nici o intrare în tabela file descriptorilor care să puncteze spre acea structură [2)](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#fn__2). ( [Linux](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#crearea_deschiderea_si_inchiderea_fisierelor), [Windows](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#crearea_deschiderea_si_inchiderea) )
* **citirea dintr-un fișier** - înseamnă copierea unui bloc de date într-un buffer; după ce se realizează citirea se actualizează cursorul de fișier [3)](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#fn__3). ( [Linux](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#scrierea_si_citirea), [Windows](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#citirea_si_scrierea) )
* **scrierea într-un fișier** - înseamnă copierea unui bloc de date dintr-un buffer în fișier; efectuarea scrierii înseamnă și actualizarea cursorului de fișier [4)](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#fn__4). ( [Linux](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#scrierea_si_citirea), [Windows](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#citirea_si_scrierea) )
* **poziționarea într-un fișier** - înseamnă schimbarea valorii cursorului de fișier; citirile sau scrierile ulterioare vor porni din locul indicat de acest cursor de fișier [5)](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#fn__5). ( [Linux](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#pozitionarea_in_fisier_lseek), [Windows](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#pozitionarea_in_fisier) )
* **schimbarea atributelor unui fișier** - înseamnă stabilirea unor parametri pentru fișier [6)](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#fn__6). ( [Linux](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#operatii_speciale))

Operații pe fișiere în Linux

Crearea, deschiderea și închiderea fișierelor

open

Pentru deschiderea/crearea unui fișier se folosește funcția [open](http://linux.die.net/man/2/open).

int open(const char \*pathname, int flags); */\* deschidere \*/*

int open(const char \*pathname, int flags, mode\_t mode); */\* creare \*/*

creat

Pentru crearea de fișiere se poate utiliza și [creat](http://linux.die.net/man/2/creat):

int creat(const char \*pathname, mode\_t mode);

Funcția este echivalentă cu apelul open unde flag-ul O\_CREAT e setat și fișierul nu există deja:

open(pathname, O\_WRONLY | O\_CREAT | O\_TRUNC, mode);

close

Închiderea de fișiere se realizează cu [close](http://linux.die.net/man/2/close):

int close(int fd)

O greșeală frecventă de programare este neverificarea codului de eroare întors la [close](http://linux.die.net/man/2/close), pentru că se poate întâmpla ca o eroare la scriere (EIO) să fie întoarsă utilizatorului abia la close.

unlink

Ștergerea efectivă a unui fișier de pe disk se realizează cu funcția [unlink](http://linux.die.net/man/2/unlink):

int unlink(const char \*pathname);

Exemplu

Dacă, spre exemplu, dorim să deschidem fișierul in.txt pentru citire și scriere, cu eventuala creare a acestuia, iar fișierul out.txt pentru scriere, cu trunchiere putem folosi următoarea secvență de cod:

[io-01.c](http://ocw.cs.pub.ro/courses/_export/code/so/laboratoare/laborator-02?codeblock=5)

*#include <sys/types.h> /\* open \*/*

*#include <sys/stat.h> /\* open \*/*

*#include <fcntl.h> /\* O\_RDWR, O\_CREAT, O\_TRUNC, O\_WRONLY \*/*

*#include <unistd.h> /\* close \*/*

*#include "utils.h"*

int main(void)

{

int rc;

int fd1, fd2;

fd1 = open("in.txt", O\_RDWR | O\_CREAT, 0644);

DIE(fd1 < 0, "open in.txt");

*/\* will fail if out.txt does not exist \*/*

fd2 = open("out.txt", O\_WRONLY | O\_TRUNC);

DIE(fd2 < 0, "open out.txt");

rc = close(fd1);

DIE(rc < 0, "close fd1");

rc = close(fd2);

DIE(rc < 0, "close fd2");

return 0;

}

**Atenție!** O greșeală frecventă este omiterea drepturilor de creare a fișierului (0644 în exemplul de mai sus) când se apelează open cu flag-ul O\_CREAT setat.

Scrierea și citirea

read

Funcția [read](http://linux.die.net/man/2/read) e folosită pentru citirea din fișier a maxim count octeți:

ssize\_t read(int fd, void \*buf, size\_t count);

Funcția [read](http://linux.die.net/man/2/read) întoarce numărul de octeți efectiv citiți, cel mult count. Valoarea minimă este de 1 octet, iar când se ajunge la sfârșitul de fișier se va întoarce 0.

write

Funcția [write](http://linux.die.net/man/2/write) e folosită pentru scrierea în fișier a maxim count octeți:

ssize\_t write(int fd, const void \*buf, size\_t count);

Valoarea întoarsă este numărul de octeți ce au fost efectiv scriși, cel mult count. În mod implicit nu se garantează că la revenirea din [write](http://linux.die.net/man/2/write) scrierea în fișier s-a terminat. Pentru a forța actualizarea se poate folosi [fsync](http://linux.die.net/man/2/fsync) sau fișierul se poate deschide folosind flagul O\_FSYNC, caz în care se garantează că după fiecare write fișierul a fost actualizat.

**Observație**: Pentru [read](http://linux.die.net/man/2/read)/[write](http://linux.die.net/man/2/write) există versiunile [pread](http://linux.die.net/man/2/pread)/[pwrite](http://linux.die.net/man/2/pwrite), care permit specificarea unui offset în fișier de la care să se efectueaze operația de citire/scriere. (De asemenea, există și versiunilepread64/pwrite64 care folosesc offset-uri de 64 de biți - pentru a putea specifica offset-uri mai mari decât 4GB).

Poziționarea în fișier (lseek)

lseek

Funcția [lseek](http://linux.die.net/man/2/lseek) permite mutarea cursorului unui fișier la o poziție absolută sau relativă.

off\_t lseek(int fd, off\_t offset, int whence)

Parametrul whence reprezintă poziția relativă de la care se face deplasarea:

* SEEK\_SET - față de poziția de început
* SEEK\_CUR - față de poziția curentă
* SEEK\_END - față de poziția de sfârșit

**Observație** [lseek](http://linux.die.net/man/2/lseek) permite și poziționări după sfârșitul fișierului. Scrierile care se fac în astfel de zone nu se pierd, ceea ce se obține fiind un fișier cu *goluri*, o zonă care este *sărită* - nu este alocată pe disc.

Pentru această funcție există și o versiune [lseek64](http://linux.die.net/man/3/lseek64) la care offset-ul este pe 64 de biți.

Trunchierea fișierelor

Pe lângă trunchierea la 0 care se poate face prin apelul open cu flag-ul O\_TRUNC, se poate specifica trunchierea unui fișier la o dimensiune specificată, prin apelurile de sistem [ftruncate](http://linux.die.net/man/2/ftruncate) și [truncate](http://linux.die.net/man/2/ftruncate):

int ftruncate(int fd, off\_t length);

int truncate(const char \*path, off\_t length);

În cazul [ftruncate](http://linux.die.net/man/2/ftruncate), parametrul fd este file descriptorul obținut cu un apel open care a asigurat drept de scriere. În cazul [truncate](http://linux.die.net/man/2/ftruncate), fișierul reprezentat prin path trebuie să aibă drept de scriere.

Exemplu utilizare operații I/O

[io-2.c](http://ocw.cs.pub.ro/courses/_export/code/so/laboratoare/laborator-02?codeblock=10)

*#include <stdio.h>*

*#include <string.h>*

*#include <stdlib.h>*

*#include <sys/types.h> /\* open \*/*

*#include <sys/stat.h> /\* open \*/*

*#include <fcntl.h> /\* O\_CREAT, O\_RDONLY \*/*

*#include <unistd.h> /\* close, lseek, read, write \*/*

*#include "utils.h"*

*/\* Print the last 100 bytes from a file \*/*

int main (void)

{

int fd, rc;

char \*buf;

ssize\_t bytes\_read;

*/\* alocate space for the read buffer \*/*

buf = [malloc](http://www.opengroup.org/onlinepubs/009695399/functions/malloc.html)(101);

DIE(buf == NULL, "malloc");

*/\* open file \*/*

fd = open("file.txt", O\_RDONLY);

DIE(fd < 0, "open");

*/\* set file pointer at 100 characters*

*\_before\_ the end of the file \*/*

rc = lseek(fd, -100, SEEK\_END);

DIE(rc < 0, "lseek");

*/\* read the last 100 characthers \*/*

bytes\_read = read(fd, buf, 100);

DIE(bytes\_read < 0, "read");

*/\* set '\0' at end of buffer for printing purposes\*/*

buf[bytes\_read] = '\0';

[printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("the last %ld bytes: \n%s\n", bytes\_read, buf);

*/\* close file \*/*

rc = close(fd);

DIE(rc < 0, "close");

*/\* cleanup \*/*

[free](http://www.opengroup.org/onlinepubs/009695399/functions/free.html)(buf);

return 0;

}

Redirectări

În Linux redirectările se realizează cu ajutorul funcțiilor de duplicare a descriptorilor de fișiere [dup](http://linux.die.net/man/2/dup) și[dup2](http://linux.die.net/man/2/dup2) (observați diferența dintre cele 2 în link-urile anterioare):

int dup(int oldfd);

int dup2(int oldfd, int newfd);

De exemplu, pentru redirectarea ieșirii în fișierul output.txt, sunt necesare două linii de cod:

fd = open("output.txt", O\_RDWR|O\_CREAT|O\_TRUNC, 0600);

dup2(fd, STDOUT\_FILENO);

Operații speciale

Funcția [fcntl](http://linux.die.net/man/2/fcntl) permite efectuarea unor operații speciale asupra descriptorilor de fișier.

Click to hide ⇱

int fcntl(int fd, int cmd);

int fcntl(int fd, int cmd, long arg);

int fcntl(int fd, int cmd, struct flock \*lock);

|  |  |
| --- | --- |
| **cmd** | **efect** |
| F\_DUPFD | duplicarea unui file descriptor |
| F\_GETFD | citește flag-urile pentru fd |
| F\_SETFD | setează flag-urile pentru fd la valoarea specificată de arg |
| F\_GETFL | citește flag-urile de stare pentru fd |
| F\_SETFL | setează flag-urile de stare pentru fd la valoarea specificată de arg |
| F\_GETLK | obținerea informațiilor despre un lock pe fișier |
| F\_SETLK | obținerea / eliberarea unui lock pe fișier |
| F\_SETLKW | similar cu F\_SETLK dar se așteaptă terminarea operației |
| F\_GETOWN | obținerea PID-ului procesului care primește semnalul SIGIO |
| F\_SETOWN | stabilirea procesului care va primi semnalul SIGIO |

Operații pe fișiere în Windows

Crearea, deschiderea și închiderea

CreateFile

Pentru a crea un handle asociat cu un fișier, director sau altă resursă abstractizată sub forma unui fișier (port COM, pipe, modem etc.) se folosește funcția [CreateFile](http://msdn.microsoft.com/en-us/library/aa363858%28VS.85%29.aspx). Funcția se ocupă atât de crearea, cât și de deschiderea unui fișier (și întoarce în ambele cazuri un handle asociat cu fișierul):

|  |  |
| --- | --- |
| HANDLE CreateFile(  LPCTSTR lpFileName,  DWORD dwDesiredAccess,  DWORD dwShareMode,  LPSECURITY\_ATTRIBUTES lpSecAttributes,  DWORD dwCreationDisposition,  DWORD dwFlagsAndAttributes,  HANDLE hTemplateFile  ); | handle1 = CreateFile(  "out.txt",  GENERIC\_READ, */\* access mode \*/*  FILE\_SHARE\_READ, */\* sharing option \*/*  NULL, */\* security attributes \*/*  OPEN\_EXISTING, */\* open only if it exists \*/*  FILE\_ATTRIBUTE\_NORMAL,*/\* file attributes \*/*  NULL  ); |

**Atenție!** Explicațiile complete se găsesc pe pagina de manual pentru [CreateFile](http://msdn.microsoft.com/en-us/library/aa363858%28VS.85%29.aspx). În continuare vom prezenta cele mai importante proprietăți.

Drepturile de acces cerute la deschiderea fișierului sunt specificate în dwDesiredAccess:

* GENERIC\_WRITE
* GENERIC\_READ

Lista completă [aici](http://msdn.microsoft.com/en-us/library/aa363874%28v=vs.85%29.aspx)

Parametrul dwCreationDisposition precizează modul în care apelul acționează în cazul în care fișierul există sau nu; poate avea valori de forma:

* CREATE\_ALWAYS - creează un fișier nou; dacă fișierul există, apelul îl suprascrie, ștergând atributele existente;
* CREATE\_NEW - creează un fișier nou; apelul eșuează dacă fișierul există deja;
* OPEN\_ALWAYS - deschide fișierul, dacă acesta există; altfel, se comportă ca și CREATE\_NEW;
* OPEN\_EXISTING - deschide fișierul; dacă nu există, apelul eșuează;
* TRUNCATE\_EXISTING - deschide fișierul (cu drept de acces GENERIC\_WRITE) și îl trunchiază la dimensiunea zero; dacă fișierul nu există, apelul eșuează.

Dacă fișierul există deja și dwCreationDisposition este CREATE\_ALWAYS sau OPEN\_ALWAYS, apelul NU eșuează, dar GetLastError returnează ERROR\_ALREADY\_EXISTS.

Mai multe aici

La deschiderea unui fișier se poate preciza prin parametrul lpSecurityAttributes [in] modul în care handle-ul returnat de apel poate fi moștenit de procesele fii ale procesului apelant. Mai multe detalii în[laboratorul de procese](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-03).

Un fișier poate fi deschis de mai multe ori (de procese diferite, sau de același proces). În acest caz, la prima deschidere, parametrul dwShareMode [in] va avea una dintre valorile:

* FILE\_SHARE\_DELETE permite unor operații de deschidere ulterioare să capete acces de tip delete.
* FILE\_SHARE\_READ permite unor operații de deschidere ulterioare să capete acces de tip read.
* FILE\_SHARE\_WRITE permite unor operații de deschidere ulterioare să capete acces de tip write.

Un set de flaguri și atribute suplimentare (valabile numai în cazul fișierelor) pot fi precizate îndwFlagsAndAttributes [in]. Valori uzuale sunt:

* FILE\_ATTRIBUTE\_NORMAL fișierul nu are alte atribute setate (folosit numai singur)
* FILE\_ATTRIBUTE\_READONLY fișierul va fi read only pentru toate procesele

Pentru copierea și mutarea fișierelor există apelurile [CopyFile](http://msdn.microsoft.com/en-us/library/aa363851(VS.85).aspx), [MoveFile](http://msdn.microsoft.com/en-us/library/aa365239(VS.85).aspx) și [ReplaceFile](http://msdn.microsoft.com/en-us/library/aa365512(VS.85).aspx). Un exemplu de schimbare a atributelor găsiți [aici](http://msdn.microsoft.com/en-us/library/aa365522(v=VS.85).aspx).

CloseHandle

Când fișierul nu mai este folosit, fișierul este închis cu apelul generic pentru orice tip de handle-uri[CloseHandle](http://msdn.microsoft.com/en-us/library/ms724211%28VS.85%29.aspx)

BOOL CloseHandle(HANDLE hObject);

DeleteFile

Ștergerea se face prin închiderea fișierului și folosirea apelului de sistem [DeleteFile](http://msdn.microsoft.com/en-us/library/aa363915%28VS.85%29.aspx)

CloseHandle(hFile);

DeleteFile("myfile.txt");

unde [DeleteFile](http://msdn.microsoft.com/en-us/library/aa363915%28VS.85%29.aspx) are signatura

BOOL DeleteFile(LPCTSTR lpFileName);

Citirea și scrierea

ReadFile

[ReadFile](http://msdn.microsoft.com/en-us/library/aa365467%28VS.85%29.aspx) operează asupra unui fișier care are drepturi de acces cel puțin pentru citire, copiind un număr de octeți (începând cu poziția curentă a cursorului de fișier) într-un buffer și întoarce într-o variabilă numărul de octeți citiți.

|  |  |
| --- | --- |
| BOOL ReadFile(  HANDLE hFile,  LPVOID lpBuffer,  DWORD nNumberOfBytesToRead,  LPDWORD lpNumberOfBytesRead,  LPOVERLAPPED lpOverlapped  ); | bRet = ReadFile(  hFile, */\* open file handle \*/*  lpBuffer, */\* where to put data \*/*  dwBytesToRead,*/\* number of bytes to read \*/*  &dwBytesRead, */\* number of bytes that were read \*/*  NULL */\* no overlapped structure \*/*  ); |

[ReadFile](http://msdn.microsoft.com/en-us/library/aa365467%28VS.85%29.aspx) primește un handle de fișier hFile, creat anterior cu drepturi cel puțin de citire. Rezultatul citirii este copiat în lpBuffer, iar numărul de octeți efectiv citiți este întors în variabila pointată delpNumberOfBytesRead. Numărul de octeți efectiv citiți poate fi mai mic decât numărul de octeți care se doresc a fi citiți - nNumberOfBytesToRead.

În mod normal, după acest apel, cursorul de fișier este actualizat cu numărul de octeți citiți. Singura excepție este cazul în care fișierul este deschis pentru operații de I/O de tip OVERLAPPED - asincrone, caz în care conceptul de cursor de fișier nu mai este folositor (și deci nu mai este actualizat). Mai multe detalii despre operațiile asincrone în [Laborator 10 - Operatii IO avansate - Windows](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-10).

[ReadFile](http://msdn.microsoft.com/en-us/library/aa365467%28VS.85%29.aspx) returnează o valoare diferită de zero în caz de succes, și zero altfel. Dacă se returnează o valoare diferită de zero, dar numărul de octeți citiți este zero, atunci s-a ajuns la sfârșitul de fișier.

WriteFile

Apelul [WriteFile](http://msdn.microsoft.com/en-us/library/aa365747%28VS.85%29.aspx) copiază în mod sincron sau asincron un număr specificat de octeți dintr-un buffer în conținutul unui fișier și returnează într-o variabilă numărul efectiv de octeți copiați. Scrierea în fișier se face în general începând din poziția curentă a cursorului și după terminarea operației, poziția cursorului fișierului este actualizată (rămân valabile observațiile anterioare despre operații OVERLAPPED).

|  |  |
| --- | --- |
| BOOL WriteFile(  HANDLE hFile,  LPCVOID lpBuffer,  DWORD nNumberOfBytesToWrite,  LPDWORD lpNumberOfBytesWritten,  LPOVERLAPPED lpOverlapped  ); | bRet = WriteFile(  hFile, */\* open file handle \*/*  lpBuffer, */\* start of data to write \*/*  dwBytesToWrite, */\* number of bytes to write \*/*  &dwBytesWritten,*/\* number of bytes that were written \*/*  NULL */\* no overlapped structure \*/*  ); |

Handle-ul de fișier în care se scrie hFile [in] trebuie să fi fost creat cu drepturi de acces GENERIC\_WRITE. Parametrii [WriteFile](http://msdn.microsoft.com/en-us/library/aa365747%28VS.85%29.aspx) au aceleași semnificații cu parametrii [ReadFile](http://msdn.microsoft.com/en-us/library/aa365467%28VS.85%29.aspx), adaptate pentru operații de scriere.

Poziționarea în fișier

SetFilePointer

Fiecare fișier deschis are asociat un cursor (memorat pe 64 de biți) care reprezintă poziția curentă de citire/scriere. Un proces poziționează cursorul la un offset specificat cu [SetFilePointer](http://msdn.microsoft.com/en-us/library/aa365541(VS.85).aspx):

|  |  |
| --- | --- |
| DWORD SetFilePointer(  HANDLE hFile,  LONG lDistanceToMove,  PLONG lpDistanceToMoveHigh,  DWORD dwMoveMethod  ); | */\* Example: How to get current position \*/*  currentPos = SetFilePointer(  myFileHandle,  0, */\* offset 0 \*/*  NULL, */\* no 64bytes offset \*/*  FILE\_CURRENT  ); |

Deplasarea se face asupra unui fișier reprezentat prin handle-ul hFile deschis în prealabil, creat cu unul din drepturile de acces GENERIC\_READ sau GENERIC\_WRITE. O valoare pozitivă înseamnă o deplasare înainte, iar una negativă, înapoi.

Numărul de octeți cu care se mută cursorul este specificat de lDistanceToMove [in] și lpDistanceToMoveHigh; cele două câmpuri de 32 de biți formează o valoare de 64 de biți. Uzual cel de-al doilea câmp este NULL.

Parametrul dwMoveMethod specifică punctul de start pentru mutarea cursorului, și poate avea una dintre valorile:

* FILE\_BEGIN - punctul de start este începutul fișierului; lDistanceToMove este considerat unsigned
* FILE\_CURRENT - punctul de start este valoarea curentă a cursorului
* FILE\_END - punctul de start este valoarea curentă a sfârșitului de fișier

Apelul returnează noua valoare a cursorului, dacă lpDistanceToMoveHigh este NULL; altfel, se returnează jumătatea low a valorii, jumătatea high luând locul lpDistanceToMoveHigh.

Varianta extinsă [SetFilePointerEx](http://msdn.microsoft.com/en-us/library/aa365542(VS.85).aspx) a apelului [SetFilePointer](http://msdn.microsoft.com/en-us/library/aa365541(VS.85).aspx) memorează valoarea cursorului într-un singur câmp, în loc de două câmpuri separate, apelul extins făcând lucrul cu valorile cursorului mai ușor.

Trunchierea fișierelor

SetEndOfFile

Un fișier poate fi trunchiat sau extins folosind apelul [SetEndOfFile](http://msdn.microsoft.com/en-us/library/aa365531(VS.85).aspx), care face poziția sfârșitului de fișierEOF egală cu poziția curentă a cursorului fișierului. În cazul extinderii fișierului peste limita sa, conținutul adăugat este nedefinit.

BOOL SetEndOfFile(HANDLE hFile);

Exemplu

[win\_io.c](http://ocw.cs.pub.ro/courses/_export/code/so/laboratoare/laborator-02?codeblock=26)

*#include <stdio.h>*

*#include <string.h>*

*#include <stdlib.h>*

*#include <windows.h>*

*#include "utils.h"*

*#define BUF\_SIZE 100*

int main (void)

{

HANDLE hFile;

DWORD dwBytesRead, dwPos, dwBytesToRead = BUF\_SIZE, dwRet;

BOOL bRet;

CHAR outBuffer[BUF\_SIZE+1];

*/\* deschidem fisierul \*/*

hFile = CreateFile(

"file.txt",

GENERIC\_READ,

FILE\_SHARE\_READ,

NULL, */\* no security attributes \*/*

OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL,

NULL */\* no pattern \*/*

);

DIE(hFile == INVALID\_HANDLE\_VALUE, "CreateFile");

*/\* set file pointer at 100 bytes*

*\_before\_ the end of file \*/*

dwPos = SetFilePointer(

hFile,

-100,

NULL, */\* used only for offsets on 64bytes \*/*

FILE\_END

);

DIE(dwPos == INVALID\_SET\_FILE\_POINTER, "SetFilePointer");

*/\* read last 100 bytes into buffer \*/*

dwRet = ReadFile(

hFile,

outBuffer,

dwBytesToRead,

&dwBytesRead,

NULL); */\* do nothing asynchronous \*/*

DIE(dwRet == FALSE, "ReadFile");

*/\* print buffer \*/*

outBuffer[dwBytesRead] = '\0';

[printf](http://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("last %ld bytes: \n%s\n", dwBytesRead, outBuffer);

[fflush](http://www.opengroup.org/onlinepubs/009695399/functions/fflush.html)(stdout);

*/\* close file \*/*

bRet = CloseHandle (hFile);

DIE(bRet == FALSE, "CloseHandle");

return 0;

}

Wrapper-e

Click to hide ⇱

În domeniul sistemelor de operare, prin wrapper înțelegem un layer software subțire (care nu aduce un overhead prea mare) peste sistemul de operare, cu scopul de a abstractiza serviciile oferite de acesta, adaptându-le la o interfață comună. Interfața comună este definită astfel încât să se potrivească cu mai multe sisteme de operare. Programele pe care le scriem ulterior nu vor folosi direct apelurile de sistem specifice fiecărui sistem de operare, ci interfața comună.

Un wrapper este folositor atunci când dorim să scriem software portabil pe mai multe platforme (spre exemplu, temele de la Sisteme de Operare) cu un “overhead” minim de portare și fără a plăti un cost de performanță prea scump (există și alte soluții pentru această problemă, de exemplu, mașina virtuală Java - JVM).

Una din metodele posibile pentru realizarea unui wrapper este folosirea preprocesorului. Să presupunem că încercăm să abstractizăm conceptul de fișier și operațiile disponibile cu el. Vom exemplifica doar operațiile de read/write.

[io-wrapper.h](http://ocw.cs.pub.ro/courses/_export/code/so/laboratoare/laborator-02?codeblock=27)

*#ifdef \_\_linux\_\_*

*#include <sys/types.h>*

*#include <sys/stat.h>*

*#include <unistd.h>*

*#include <fcntl.h>*

typedef int os\_handle;

typedef size\_t os\_size;

typedef ssize\_t os\_ssize;

*#elif defined(\_WIN32)*

*#include <windows.h>*

typedef HANDLE os\_handle;

typedef DWORD os\_size;

typedef DWORD os\_ssize;

*#else*

*#error "Unknown OS!"*

*#endif*

os\_ssize os\_read(os\_handle fd, void \*buffer, os\_size count);

os\_ssize os\_write(os\_handle fd, const void \*buffer, os\_size count);

Se observă că în funcție de sistemul de operare definit, diferă:

* fișierele header incluse
* definițiile tipurilor cu care lucrează wrapper-ul

De asemenea, se observă că semnăturile funcțiilor definite sunt identice pentru ambele sisteme de operare. Iată un exemplu de implementare a lor:

[io-wrapper.c](http://ocw.cs.pub.ro/courses/_export/code/so/laboratoare/laborator-02?codeblock=28)

*#include "io-wrapper.h"*

*#ifdef \_\_linux\_\_*

os\_ssize os\_read(os\_handle fd, void \*buffer, os\_size count)

{

return read(fd, buffer, count);

}

os\_ssize os\_write(os\_handle fd, const void \*buffer, os\_size count)

{

return write(fd, buffer, count);

}

*#elif defined(\_WIN32)*

os\_ssize os\_read(os\_handle fd, void \*buffer, os\_size count)

{

os\_ssize result = -1;

ReadFile(fd, buffer, count, &result, NULL);

return result;

}

os\_ssize os\_write(os\_handle fd, void \*buffer, os\_size count)

{

os\_ssize result = -1;

WriteFile(fd, buffer, count, &result, NULL);

return result;

}

*#endif*

Acum putem genera fișiere executabile compatibile cu o platformă Linux sau Windows, în funcție de un singur macro, definit automat de către compilator.

Se observă că folosind această tehnică putem să convertim inclusiv între procedură și funcție (funcțiile de pe Windows primesc ca parametru transmis prin referință numărul de octeți citiți/scriși, iar cele de pe Linux îl întorc direct). Desigur, abordarea de mai sus este incompletă, pentru că ar fi trebuit convertite și codurile de eroare într-un format comun.

Odată scris acest wrapper, putem folosi în continuare funcțiile os\_read și os\_write pentru a citi / scrie din fișiere, fară a ne preocupa de sistemul de operare pe care rulează programul nostru. Acesta este însă un caz fericit, pentru că așa după cum veți observa la laboratorul de procese, nu toate serviciile oferite de sisteme de operare diferite se pot “unifica” atât de ușor (este vorba de fork() + exec() vs. CreateProcess).

Exerciții

În rezolvarea laboratorului folosiți arhiva de sarcini [lab02-tasks.zip](http://elf.cs.pub.ro/so/res/laboratoare/lab02-tasks.zip)

**Observații**: Pentru a vă ajuta la implementarea exercițiilor din laborator, în directorul utils din arhivă există un fișier utils.h cu funcții utile.

Folosiți man/MSDN pentru informații despre apelurile de sistem

**Verificați valorile de retur a apelurilor de sistem**  
Puteți folosi macro-ul [DIE](http://elf.cs.pub.ro/so/wiki/laboratoare/resurse/die)(valoare\_retur == eroare, “mesaj eroare”);

Exercițiul -1 - GSOC (0p)

Google Summer of Code este un program de vară în care studenții (indiferent de anul de studiu) sunt implicați în proiecte Open Source pentru a își dezvolta skill-urile de programare, fiind răsplătiți cu o bursă în valoare totală de 5500$ [gsoc](https://developers.google.com/open-source/gsoc).

UPB se află în top ca număr de studenți acceptați; în fiecare an fiind undeva la aprox. 30-40 de studenți acceptați. Vă încurajăm să aplicați! Există și un grup de fb cu foști participanți unde puteti să îi contactați pentru sfaturi [facebook page](https://www.facebook.com/groups/240794072931431/)

Exercițiul 0 - Joc interactiv (2p)

* Detalii desfășurare [joc](http://ocw.cs.pub.ro/courses/so/meta/notare#joc_interactiv).

Linux (5p)

Exercițiul 1 - redirect (1p)

Intrați în directorul 1-redirect și urmăriți conținutul fișierului redirect.c.

Compilați fișierul (folosiți make). Rulați programul obținut folosind comanda ./redirect.

Deschideți alt terminal și rulați comanda:

**watch** -d lsof -p $(**pidof** redirect)

[lsof](http://linux.die.net/man/8/lsof) este un utilitar care afișează informații despre fișierele deschise (ce fișiere sunt deschise în sistem, ce fișiere a deschis un anumit user etc). Căutați în manual (man 8 lsof) pentru a identifica semnificația coloanei FD și a coloanei TYPE.

Folosiți comanda ENTER pentru a continua programul. În paralel urmăriți cum se modifică tabela de file-descriptori.

În cod, observați parametrii cu care s-a realizat redirectarea cu ajutorul funcției[dup2](http://linux.die.net/man/2/dup2) (dup2(fd2, STDERR\_FILENO)). Ce se întampla dacă erau în ordine inversă?

* revedeți secțiunea de [redirectări](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#redirectari)

Exercițiul 2 - lseek (1p)

Intrați în directorul 2-lseek și urmăriți codul sursă din lseek.c. Ce valoare va întoarce al doilea apel al funcțieilseek? De ce?

Decomentați linia de afișare, compilați și rulați pentru verificare.

Sursa închide doar file descriptorul fd1. Este nevoie să se închidă și file descriptorul fd2? De ce?

Exercițiul 3 - mcat (3p)

Intrați în directorul 3-mcat.

3a. Similitudine cat (1p)

Completați fișierul astfel încât programul rezultat mcat să aibă funcționalitate similară cu a utilitarului cat(urmăriți comentariile cu TODO 1)

Programul mcat va primi ca argument în linia de comandă numele unui fișier al cărui conținut îl va afișa la ieșirea standard. Nu aveți voie să citiți tot fișierul în memorie. Puteți citi doar bucăți de dimensiune maximum BUFSIZE.

**Verificați codul de eroare** întors de apelurile de sistem. Puteți folosi macro-ul [DIE](http://elf.cs.pub.ro/so/wiki/laboratoare/resurse/die). Revedeți secțiunile[Crearea, deschiderea și închiderea fișierelor](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#crearea_deschiderea_si_inchiderea_fisierelor) și [Scrierea și citirea fișierelor](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#scrierea_si_citirea).

Testați cu o comandă de genul:

./mcat Makefile

3b. Similitudine cp (1p)

Extindeți funcționalitatea astfel încât output-ul să fie redirectat într-un fișier primit ca al doilea argument - funcționalitate similară cu a utilitarului cp. (urmăriți comentariile cu TODO 2)

Revedeți secțiunea de [redirectări](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#redirectari).

Testați funcționalitatea:

./mcat Makefile out ; ./mcat out

3c. /dev/nasty (1p)

Inițializați fișierul /dev/nasty:

./set\_nasty.sh

Încercați funcționalitatea de copiere pe fișierul /dev/nasty:

./mcat /dev/nasty

./mcat /dev/nasty out ; ./mcat out

Dacă apar diferențe, fiți atenți la ce întorc funcțiile read și write (eventual afișați aceste valori) și reparați problema.

Testați **scrierea** cu:

./mcat Makefile /dev/nasty ; **cat** /dev/nasty

În cazul în care ultima comandă nu produce rezultatul așteptat, cel mai probabil nu ați tratat corect cazurile în care read/write întorc o valoare mai mică decât al treilea parametru.

Windows (4p)

Executabilele sunt generate în directorul win/Debug (în directorul Debug al soluției, nu al fiecărui proiect în parte).

Exercițiul 1 - cat (0.5p)

Deschideți folderul win din arhiva laboratorului 2 și intrați în proiectul 1-cat, iar apoi urmăriți sursa cat.c

Compilați și testați executabilul cat.exe folosind command prompt-ul de Visual Studio: Tools → Visual Studio Command Prompt

Exercițiul 2 - CRC (3.5p)

Exercițiul are ca scop realizarea unui utilitar care, pentru un fișier dat, calculeaza CRC-ul pentru fiecare bucată de 512 bytes din fișier și o salvează într-un fișier de output.

2a. Generare (1.5p)

Deschideți fișierul crc.c din proiectul 2-crc și completați funcția GenerateCrc.

Funcția primește ca prim argument fișierul pentru care trebuie calculat CRC-ul, iar ca al doilea argument fișierul în care se salvează CRC-urile pentru fiecare bucată de câte 512 bytes. La ultima bucată se va face padding.

Revedeți secțiunile [Crearea, deschiderea și închiderea fișierelor](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#crearea_deschiderea_si_inchiderea), cât și [Citirea și scrierea fișierelor](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#citirea_si_scrierea).

Urmăriți comentariile cu TODO 1.

2b. Comparare (2p)

Odată calculat fișierul cu CRC, vrem să vedem dacă două fișiere de CRC sunt egale. Extindeți funcționalitatea programului anterior astfel încât să compare 2 fișiere. Vom lucra în funcția CompareFiles.

Inițial comparați dimensiunile fișierelor astfel:

* Completați funcția GetSize pentru calcularea dimensiunii unui fișier, urmărind comentariile din TODO 2
* Folosiți doar funcția [SetFilePointer](http://msdn.microsoft.com/en-us/library/aa365541%28VS.85%29.aspx)

Dacă dimensiunile sunt egale, comparați cele 2 fișiere bucată cu bucată (nu citiți tot fișierul în memorie), urmărind comentariile cu TODO 3.

BONUS - Linux

**1 so karma - Troubleshooting**

Intrați în directorul 4-trouble. Compilați și rulați programul trouble.

Programul ar trebui să afișeze în fișierul tmp1.txt mesajul din msg. Afișați fișierul tmp1.txt.

Ce observați? Identificați și remediați problema. Revedeți secțiunea: [Crearea, deschiderea și închiderea fișierelor](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#crearea_deschiderea_si_inchiderea_fisierelor).

**1 so karma - File lock**

Vrem să ne asigurăm că doar **o instanță** a unui program rulează la un moment dat. Pentru asta se creează un fișier temporar pe care se încearcă obținerea unui lock folosind apelul [flock](http://linux.die.net/man/2/flock).

Intrați în directorul 5-singular și completați sursa singular.c (urmăriți comentariile cu TODO ).

Hint: man 2 flock, *nonblocking*

Testați rulând executabilul din două terminale diferite, sau cu comanda:

./singular & **sleep** 3 ; ./singular

Cum ne mai putem asigura că programul nostru are doar o singură instanță, folosind mai **puține** apeluri de sistem?

BONUS - Windows

Utilitar echivalent cu ls -a -R.

**1 so karma - Creare utilitar ls**

Deschideți din arhiva laboratorului 2 proiectul 3-ls. Completați fișierul ls.c pentru ca programul 3-ls.exe să se comporte ca utilitarul ls.

Afișarea fișierelor dintr-un director se face în doi pași:

* se obține un handle la o primă intrare din lista de fișiere a directorului cu funcția: [FindFirstFile](http://msdn.microsoft.com/en-us/library/aa364418%28VS.85%29.aspx)
* se iterează această listă folosind funcția: [FindNextFile](http://msdn.microsoft.com/en-us/library/aa364428%28VS.85%29.aspx)

Pentru rezolvare, urmăriți comentariile marcate cu TODO 1. Pentru testare folosiți dintr-un prompt Visual Studio:

ls.exe ..

**1 so karma - Afișare detalii pentru parametrul -a**

Pentru fișiere afișați numele, dimensiunea și data la care au fost modificate ultima oară. Pentru directoare afișați numele și un indicator de director (ex: <DIR> nume ).

Atributele unui fișier sunt definite într-o structură de forma: [WIN32\_FIND\_DATA](http://msdn.microsoft.com/en-us/library/aa365740%28VS.85%29.aspx). Pentru a verifica dacă un fișier e director, trebuie să aibă bitul “FILE\_ATTRIBUTE\_DIRECTORY” din câmpul “dwFileAttributes” ( vezi [File Attributes](http://msdn.microsoft.com/en-us/library/ee332330%28v=VS.85%29.aspx)).

* Urmăriți comentariile marcate cu TODO 2

**1 so karma - Afișare detalii pentru parametrul -R**

Realizați parcurgerea recursivă a directoarelor prin apelarea recursivă a funcției ListFile.

Pentru rezolvare, urmăriți comentariile marcate cu TODO 3. Aveți grijă să concatenați numele noului director la calea deja existentă.

**1 so karma - Troubleshooting**

Deschideți din arhiva laboratorului 2 proiectul 4-trouble. Programul ar trebui să creeze un fișier cu mesajul “Testing 123”.

Compilați și rulați programul trouble. Identificați și remediați problema.

Revedeți secțiunea: [Crearea, deschiderea și închiderea fișierelor](http://ocw.cs.pub.ro/courses/so/laboratoare/laborator-02#crearea_deschiderea_si_inchiderea).

EXTRA

* Operații cu fișiere în Python
* Studiați exemplele din [arhivă](http://ocw.cs.pub.ro/courses/_media/so/laboratoare/pyfileoperations.zip), citiți documentația și observați diferențele între API-uri