

# Gestiunea fișierelor în WIN32 API

---

CURS NR. 2

# Sistemul de fișiere Windows

---

Fișier – unitatea de stocare persistentă a informațiilor – pe sisteme de memorare secundare (HDD, SSD, etc.)

SF - Realizează gestiunea fișierelor pe dispozitivele de memorare nonvolatile atașate la sistemul de calcul

- Controlează modul de alocare a spațiului, stocarea, regăsirea, controlul accesului, etc.

Oferă suport pentru

- Sistemul de fișiere NTFS (New Technology File System)
  - Suport pentru fișiere de dimensiuni mari (peste 4GB), nume lungi
  - Suport pentru protecție și securitate: controlul accesului prin ACL (Access Control List), criptare, compresie, etc.
- Sistemul de fișiere FAT și FAT32 (File Allocation Table)
  - În FAT32 limita maximă pentru dimensiunea unui fișier este 4GB, numele de fișier de 8 caractere, cu extensia de 3 caractere
  - Nu oferă suport pentru mecanisme de protecție și securitate
- Sistemul de fișiere UDF (Universal Disk Format) numit și Live File System
  - Suport pentru CD-ROM și DVD

Win32 API accesează sistemul de fișiere în mod uniform – cu limitările impuse de SF

# Sistemul de fișiere

---

## Structură ierarhică – fiecare partiție este un arbore

- Rădăcina este numele drive-ului
  - A: și B: sunt rezervate dischetelor,
  - C: și D: șamd sunt de regulă hdd/ssd-uri, DVD-uri și alte dispozitive atașate direct, iar
  - drive-urile de rețea au asociate litere mai de la finalul alfabetului
- Separatorul dintre directoare și subdirectoare sau fișiere este simbolul \ (backslash)
  - În unele funcții ale API-ului de nivel coborât prametrul cale acceptă și separatorul / (slash)

## Acces la fișiere: căi de acces

- Căi absolute
  - Încep cu numele drive-ului
  - Sau încep cu \\ (dublu backslash) indicând o rădăcină globală și este urmată de numele unui server și numele de partajare (share name)
    - Prima parte a căii absolute arată astfel: \\server\sharename
- Căi relative
  - Relative la directorul curent
- . și .. sunt nume de directoare
  - . referă drectorului curent
  - .. referă directorul părinte

# Sistemul de fișiere

---

## Reguli de numire pentru fișiere și directoare

- Numele de directoare și fișiere NU pot conține caractere speciale din intervalul codrilor ASCII 1-31 sau caractere cum ar fi < > : " | ? \* / \
  - aceste caractere au semnificații speciale când sunt folosite în linia de comandă
- Numele poate conține spațiu
  - Numele care conține spațiu se pune între ghilimele - dacă se specifică din linia de comandă
    - Pentru a nu fi parsate ca două fișiere separate
- Numele este case-insensitive dar case-retaining
  - Un fișier cu numele MyFile va fi listat ca atare, dar poate fi accesat și sub nume de genul myfile sau myFILE
- Lungimea numelui: maxim 255 de caractere
- Lungimea unei căi de acces: MAX\_PATH caractere (de regulă 260 de caractere)
- Extensia (de regulă între 2-4 caractere) care indică tipul fișierului
  - Un punct separă numele fișierului de extensie (punctul cel mai din dreapta – numele poate conține mai multe puncte)
    - Ex. Sort.exe desemnează un executabil, fis.c este un fișier sursă C

# Gestiunea fișierelor

Correspondențele dintre **funcțiile CRT** și **apelurile sistem Win32 API** – pentru fluxuri (stream-uri)

Stream Routines

CRT	Win32 API	CRT	Win32 API
clearerr	none	getc	none
fclose	CloseHandle	getchar	none
_fcloseall	none	gets	none
_fdopen	none	_getw	none
feof	none	printf	none
ferror	none	putc	none
fflush	FlushFileBuffers	putchar	none
fgetc	none	puts	none
_fgetchar	none	_putw	none
fgetpos	none	rewind	SetFilePointer
fgets	none	_rmtmp	none
_fileno	none	scanf	none
_flushall	none	setbuf	none
fopen	CreateFile	setvbuf	none
fprintf	none	_snprintf	none
fputc	none	sprintf	wsprintf
_fputchar	none	sscanf	none
fputs	none	_tempnam	GetTempFileName
fread	ReadFile	tmpfile	none
freopen (std handles)	SetStdHandle	tmpnam	GetTempFileName
fscanf	none	ungetc	none
fseek	SetFilePointer	vfprintf	none
fsetpos	SetFilePointer	vprintf	none
_fsopen	CreateFile	_vsnprintf	none
ftell	SetFilePointer (check return value)	vsprintf	wvsprintf
fwrite	WriteFile		

# Gestiunea fișierelor

Correspondențele dintre **funcțiile CRT** și **apelurile sistem Win32 API** – pentru fișiere

## File Handling

### CRT

```
_access  
_chmod  
_chsize  
_filelength  
_fstat  
_fullpath  
_get_osfhandle  
_isatty  
_locking  
_makepath  
_mktemp  
_open_osfhandle  
remove  
rename  
_setmode  
_splitpath  
_stat  
_umask  
_unlink
```

### Win32 API

```
none  
SetFileAttributes  
SetEndOfFile  
GetFileSize  
See Note 5  
GetFullPathName  
none  
GetFileType  
LockFileEx  
none  
GetTempFileName  
none  
DeleteFile  
MoveFile  
none  
none  
none  
none  
DeleteFile
```

## Low-Level I/O

### CRT

```
_close  
_commit  
_creat  
_dup  
_dup2  
_eof  
_lseek  
_open  
_read  
_sopen  
_tell  
_write
```

### Win32 API

```
_lclose, CloseHandle  
FlushFileBuffers  
_lcreat, CreateFile  
DuplicateHandle  
none  
none  
_llseek, SetFilePointer  
_lopen, CreateFile  
_lread, ReadFile  
CreateFile  
SetFilePointer (check return value)  
_lread
```

# Exemplu comparativ: copierea unui fișier

---

## Abordări

- Folosind biblioteca C standard
  - Folosind apelurile Windows
  - Folosind o funcție utilitară (CopyFile)
- 
- Exemplele includ parcurgerea și copierea secvențială simplificată
    - Fișierele sursă și destinație sunt specificate ca argumente ale programului (din linia de comandă)
    - Testarea și validarea este minimală
    - Fișierele existente sunt suprascrise

# Copiere folosind C standard

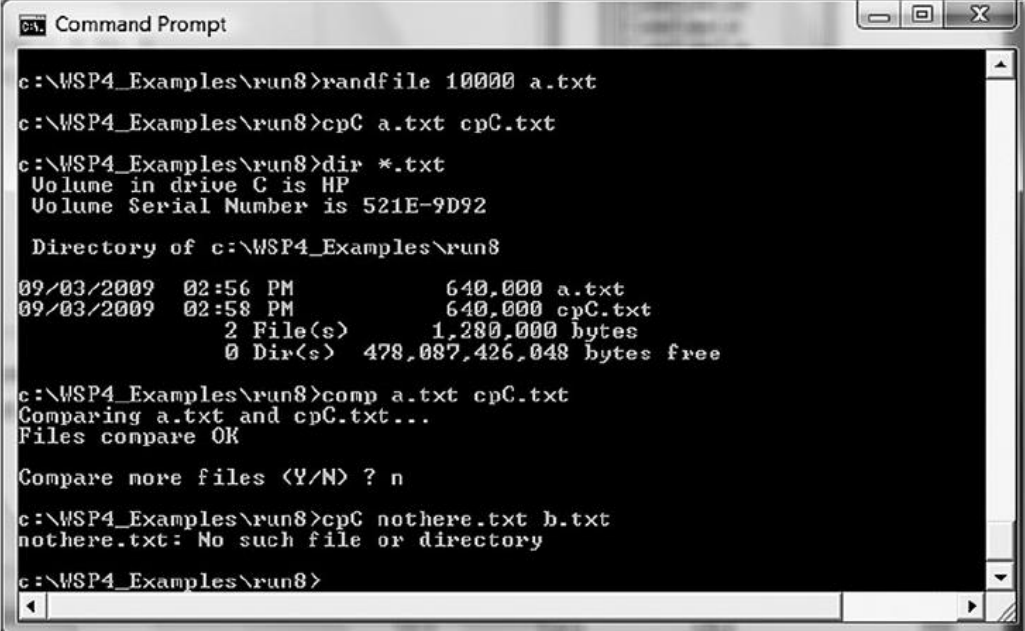
```
/* Chapter 1. Basic cp file copy program. C library Implementation. */
/* cpC file1 file2: Copy file1 to file2. */
#include <stdio.h>
#include <errno.h>
#define BUF_SIZE 256

int main (int argc, char *argv []) {
    FILE *inFile, *outFile;
    char rec[BUF_SIZE];
    size_t bytesIn, bytesOut;
    if (argc != 3) {
        fprintf (stderr, "Usage: cp file1 file2\n");
        return 1;
    }

    /* In later chapters, we'll use the more secure functions, such as fopen_s
     * Note that this project defines the macro _CRT_SECURE_NO_WARNINGS to avoid a warning */
    inFile = fopen (argv[1], "rb");
    if (inFile == NULL) {
        perror (argv[1]);
        return 2;
    }
    outFile = fopen (argv[2], "wb");
    if (outFile == NULL) {
        perror (argv[2]);
        fclose(inFile);
        return 3;
    }
    /* Process the input file a record at a time. */
    while ((bytesIn = fread (rec, 1, BUF_SIZE, inFile)) > 0) {
        bytesOut = fwrite (rec, 1, bytesIn, outFile);
        if (bytesOut != bytesIn) {
            perror ("Fatal write error.");
            fclose(inFile); fclose(outFile);
            return 4;
        }
    }
    fclose (inFile);
    fclose (outFile);
    return 0;
}
```

## Observații

- **Pointer la fișier**
- Modul de deschidere
- Captarea erorilor cu **perror**
- Funcțiile de **citire/scriere** și valorile returnate (nr de elemente citite/scrise)
- Închiderea fișierului
- Funcții cu **număr redus de argumente**, exprimate **succint**



```
Command Prompt

c:\WSP4_Examples\run8>randfile 10000 a.txt
c:\WSP4_Examples\run8>cpC a.txt cpC.txt
c:\WSP4_Examples\run8>dir *.txt
Volume in drive C is HP
Volume Serial Number is 521E-9D92

Directory of c:\WSP4_Examples\run8

09/03/2009  02:56 PM             640,000 a.txt
09/03/2009  02:58 PM             640,000 cpC.txt
               2 File(s)          1,280,000 bytes
               0 Dir(s)  478,087,426,048 bytes free

c:\WSP4_Examples\run8>comp a.txt cpC.txt
Comparing a.txt and cpC.txt...
Files compare OK

Compare more files (Y/N) ? n

c:\WSP4_Examples\run8>cpC nowhere.txt b.txt
nowhere.txt: No such file or directory

c:\WSP4_Examples\run8>
```



# Copiere folosind Windows

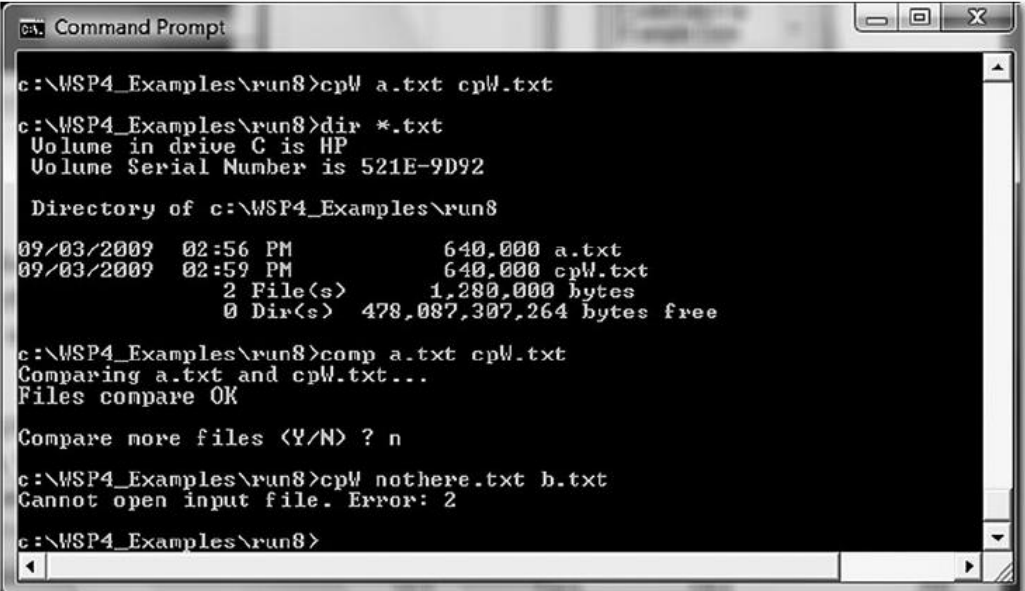
```
/* Chapter 1. Basic cp file copy program. Win32 Implementation. */
/* cpW file1 file2: Copy file1 to file2. */
#include <windows.h>
#include <stdio.h>
#define BUF_SIZE 16384 /* Optimal in several experiments.
Small values such as 256 give very bad performance */

int main (int argc, LPTSTR argv []) {
    HANDLE hIn, hOut;
    DWORD nIn, nOut;
    CHAR buffer [BUF_SIZE];
    if (argc != 3) {
        fprintf (stderr, "Usage: cp file1 file2\n");
        return 1;
    }
    hIn = CreateFile (argv[1], GENERIC_READ, FILE_SHARE_READ, NULL,
                     OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hIn == INVALID_HANDLE_VALUE) {
        fprintf (stderr, "Cannot open input file. Error: %x\n", GetLastError ());
        return 2;
    }

    hOut = CreateFile (argv[2], GENERIC_WRITE, 0, NULL,
                      CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hOut == INVALID_HANDLE_VALUE) {
        fprintf (stderr, "Cannot open output file. Error: %x\n", GetLastError ());
        CloseHandle(hIn);
        return 3;
    }
    while (ReadFile (hIn, buffer, BUF_SIZE, &nIn, NULL) && nIn > 0) {
        WriteFile (hOut, buffer, nIn, &nOut, NULL);
        if (nIn != nOut) {
            fprintf (stderr, "Fatal write error: %x\n", GetLastError ());
            CloseHandle(hIn); CloseHandle(hOut);
            return 4;
        }
    }
    CloseHandle (hIn);
    CloseHandle (hOut);
    return 0;
}
```

## Observații

- Fișierul header **windows.h**
- **Handle** al fișierului
- Constante simbolice și flaguri
- Captarea erorilor cu **GetLastError**
- Funcțiile de **citire/scriere** și valorile returnate (BOOL)
- Închiderea handle-ului
- Funcțiile Windows au **multe argumente** și **multe opțiuni** – adesea se folosesc valori implicite



```
Command Prompt

c:\WSP4_Examples\run8>cpW a.txt cpW.txt
c:\WSP4_Examples\run8>dir *.txt
Volume in drive C is HP
Volume Serial Number is 521E-9D92

Directory of c:\WSP4_Examples\run8

09/03/2009  02:56 PM                640,000 a.txt
09/03/2009  02:59 PM                640,000 cpW.txt
               2 File(s)                1,280,000 bytes
               0 Dir(s)  478,087,307,264 bytes free

c:\WSP4_Examples\run8>comp a.txt cpW.txt
Comparing a.txt and cpW.txt...
Files compare OK

Compare more files (Y/N) ? n

c:\WSP4_Examples\run8>cpW nowhere.txt b.txt
Cannot open input file. Error: 2

c:\WSP4_Examples\run8>
```

# Copiere folosind o funcție utilitară (CopyFile)

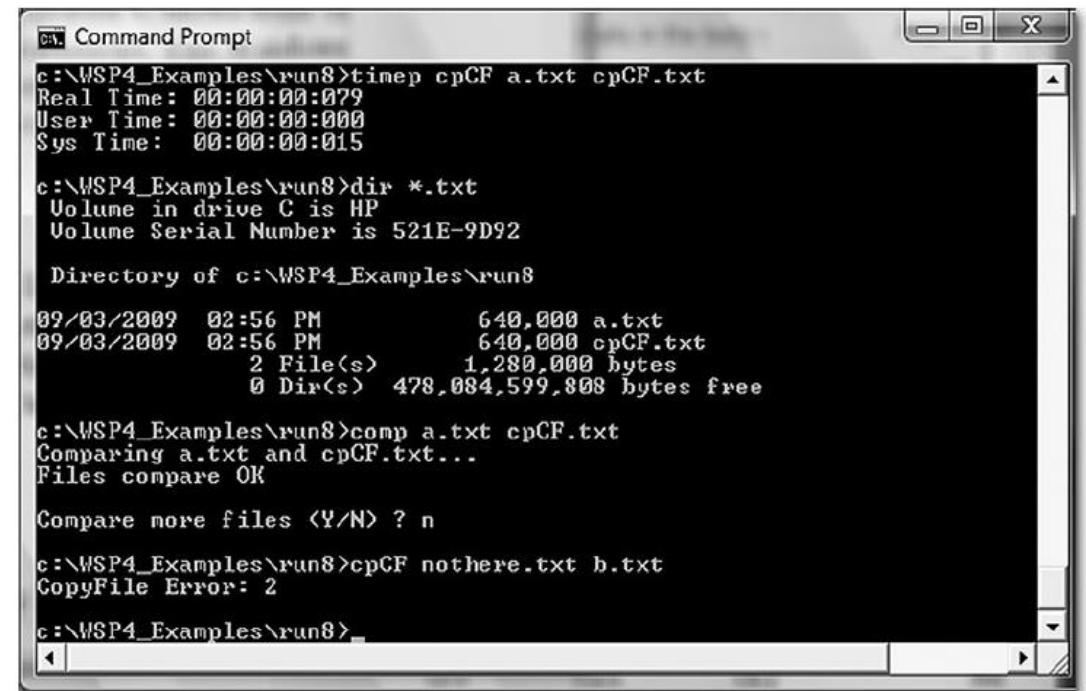
```
/* Chapter 1. Basic cp file copy program.
Windows Implementation using CopyFile for convenience and possible performance. */
/* cp file1 file2: Copy file1 to file2. */

#include <windows.h>
#include <stdio.h>
#define BUF_SIZE 256

int main (int argc, LPCTSTR argv []) {
    if (argc != 3) {
        fprintf (stderr, "Usage: cp file1 file2\n");
        return 1;
    }
    if (!CopyFile (argv[1], argv[2], FALSE)) {
        fprintf (stderr, "CopyFile Error: %x\n", GetLastError ());
        return 2;
    }
    return 0;
}
```

## Observații

- Funcțiile utilitare grupează mai multe funcții pentru a realiza o sarcină frecvent întâlnită
- Simplifică codul, mărește performanța (frecvent)
- CopyFile copiază și metadatele fișierului copiat



```
Command Prompt
c:\WSP4_Examples\run8>tinep cpCF a.txt cpCF.txt
Real Time: 00:00:00:079
User Time: 00:00:00:000
Sys Time: 00:00:00:015

c:\WSP4_Examples\run8>dir *.txt
Volume in drive C is HP
Volume Serial Number is 521E-9D92

Directory of c:\WSP4_Examples\run8

09/03/2009  02:56 PM                640,000 a.txt
09/03/2009  02:56 PM                640,000 cpCF.txt
               2 File(s)            1,280,000 bytes
               0 Dir(s)  478,084,599,808 bytes free

c:\WSP4_Examples\run8>comp a.txt cpCF.txt
Comparing a.txt and cpCF.txt...
Files compare OK

Compare more files (Y/N) ? n

c:\WSP4_Examples\run8>cpCF nothere.txt b.txt
CopyFile Error: 2

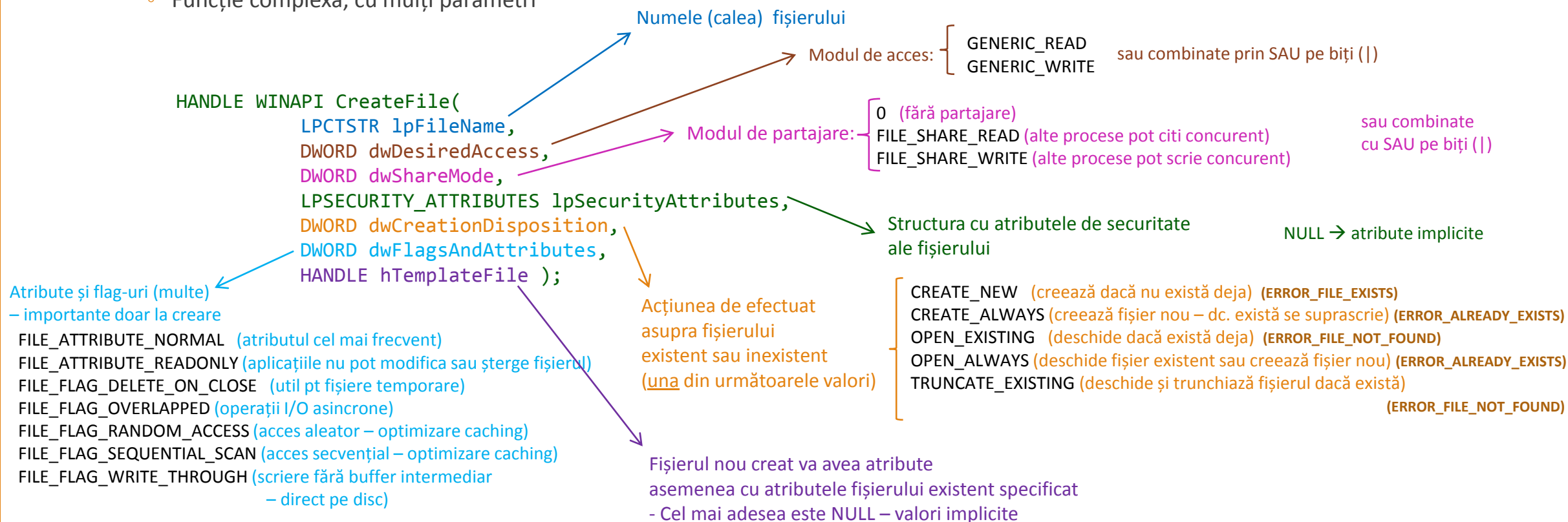
c:\WSP4_Examples\run8>
```

# Crearea și deschiderea unui fișier

Fișierul este un obiect kernel

Funcția de creare a unui nou fișier sau deschidere a unui fișier existent: CreateFile

- Funcție complexă, cu mulți parametri



- Returnează handle către fișierul deschis sau `INVALID_HANDLE_VALUE` în caz de eroare

# Redeschiderea și închiderea fișierului

Redeschiderea fișierului cu alte moduri de acces, moduri de partajare sau flag-uri

```
HANDLE WINAPI ReOpenFile(  
    HANDLE hOriginalFile,  
    DWORD dwDesiredAccess,  
    DWORD dwShareMode,  
    DWORD dwFlags );
```

Handle-ul fișierului care se redeschide (handle obținut prin CreateFile)

Modul de acces

Modul de partajare

Flag-uri

- Permite utilizarea diferitelor handle-uri pentru același obiect dar situații distincte
  - Protecție împotriva utilizării nepotrivite

## Închiderea handle-ului la fișier

- Nu implică ștergerea fișierului

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

Handle-ul fișierului care se închide

- Returnează TRUE în caz de succes și FALSE în caz de eșec
- Este o funcție generală pentru închiderea și invalidarea handle-ului asociat unui obiect kernel
- Ca efect se decrementează contorul de referiri al obiectului kernel

# Citire din / scriere în fișier

## Citirea din fișier

- Începând de la poziția curentă a pointerului din fișier se citește numărul dorit de octeți și avansează pointerul după ultimul octet citit

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

Diagram illustrating the parameters of `ReadFile`:

- `HANDLE hFile`: Handle-ul fișierului din care se citește. Trebuie să fie deschis cu drept de citire
- `LPVOID lpBuffer`: Pointer către bufferul care reține datele citite
- `DWORD nNumberOfBytesToRead`: Numărul de octeți de citit
- `LPDWORD lpNumberOfBytesRead`: Numărul de octeți citați
- `LPOVERLAPPED lpOverlapped`: Pointer la structura OVERLAPPED. Folosim NULL deocamdată

- Returnează
  - TRUE în caz de succes
    - Dacă pointerul din fișier este la finalul fișierului, funcția returnează TRUE și `*lpNumberOfBytesRead` este setat la 0
  - FALSE dacă oricare parametru este invalid sau citirea eșuează

## Scrierea în fișier

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

Interpretarea parametrilor este analogică cu cele de mai sus

- Revenirea cu succes din funcție nu garantează și scrierea efectivă pe disc
  - Excepție: dacă se specifică flagul `FILE_FLAG_WRITE_THROUGH` în funcția `CreateFile` (caz în care se revine din funcție doar după scrierea efectivă a datelor pe disc)

# Procesarea erorilor

ReportError - program de procesare a erorilor cu scop general

```
#include "Everything.h"

VOID ReportError (LPCTSTR userMessage, DWORD exitCode, BOOL printErrorMessage)

/* General-purpose function for reporting system errors.
Obtain the error number and convert it to the system error message.
Display this information and the user-specified message to the standard error device.
userMessage: Message to be displayed to standard error device.
exitCode: 0 - Return.
          > 0 - ExitProcess with this code.
printErrorMessage: Display the last system error message if this flag is set. */
{
    DWORD eMsgLen, errNum = GetLastError ();
    LPTSTR lpvSysMsg;
    _ftprintf (stderr, _T("%s\n"), userMessage);
    if (printErrorMessage) {
        eMsgLen = FormatMessage ( FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM,
                                NULL, errNum, MAKELANGID (LANG_NEUTRAL, SUBLANG_DEFAULT),
                                (LPTSTR) &lpvSysMsg, 0, NULL);

        if (eMsgLen > 0) {
            _ftprintf (stderr, _T("%s\n"), lpvSysMsg);
        }
        else {
            _ftprintf (stderr, _T("Last Error Number; %d.\n"), errNum);
        }

        if (lpvSysMsg != NULL) LocalFree (lpvSysMsg); /* Explained in Chapter 5. */
    }

    if (exitCode > 0)
        ExitProcess (exitCode);

    return;
}
```

Convertește un număr de mesaj într-un mesaj inteligibil – în engleză sau altă limbă

- Folosește numărul de eroare returnat de **GetLastError**
- Mesajul va fi **generat de sistem**
- Mesajul este stocat într-un buffer **alocat de funcție**
- Adresa bufferului este **returnat** într-un parametru

# Dispozitive standard

De regulă procesele Windows au implicit asociate trei dispozitive standard

- De intrare
- De ieșire
- De eroare
- În mod normal dispozitivele standard asociate sunt consola și tastatura

Obținerea handle-ului de la dispozitivele standard

```
HANDLE WINAPI GetStdHandle(  
    DWORD nStdHandle );
```

Poate avea una din valorile: STD\_INPUT\_HANDLE  
STD\_OUTPUT\_HANDLE  
STD\_ERROR\_HANDLE

- Returnează un handle valid în caz de succes, sau INVALID\_HANDLE\_VALUE în caz de eroare, sau NULL dacă aplicația nu are handle standard asociat
- Închiderea handle-ului asociat unui dispozitiv standard face ca dispozitivul să nu fie disponibil în continuare pentru proces

Setarea handle-ului pentru dispozitive standard: redirectare

```
BOOL WINAPI SetStdHandle(  
    DWORD nStdHandle,  
    HANDLE hHandle );
```

Poate avea una din valorile: STD\_INPUT\_HANDLE  
STD\_OUTPUT\_HANDLE  
STD\_ERROR\_HANDLE

Handle-ul unui fișier care va fi considerat ca dispozitiv standard

Căi rezervate pentru consola:  
“CONIN\$” și “CONOUT\$”

# Exemplu: afișarea fișierelor la ieșirea standard

Programul **cat.c** copiază fișierele primite ca și argument din linia de comandă (sau intrarea standard) la ieșirea standard

```
/* cat [options] [files]
Only the -s option is used. Others are ignored.
-s suppresses error report when a file does not exist */

#include "Everything.h"

#define BUF_SIZE 0x200

static VOID CatFile (HANDLE hInFile, HANDLE hOutFile) {
    DWORD nIn, nOut;
    BYTE buffer [BUF_SIZE];

    while (ReadFile (hInFile, buffer, BUF_SIZE, &nIn, NULL) &&
        (nIn != 0) && WriteFile (hOutFile, buffer, nIn, &nOut, NULL));

    return;
}

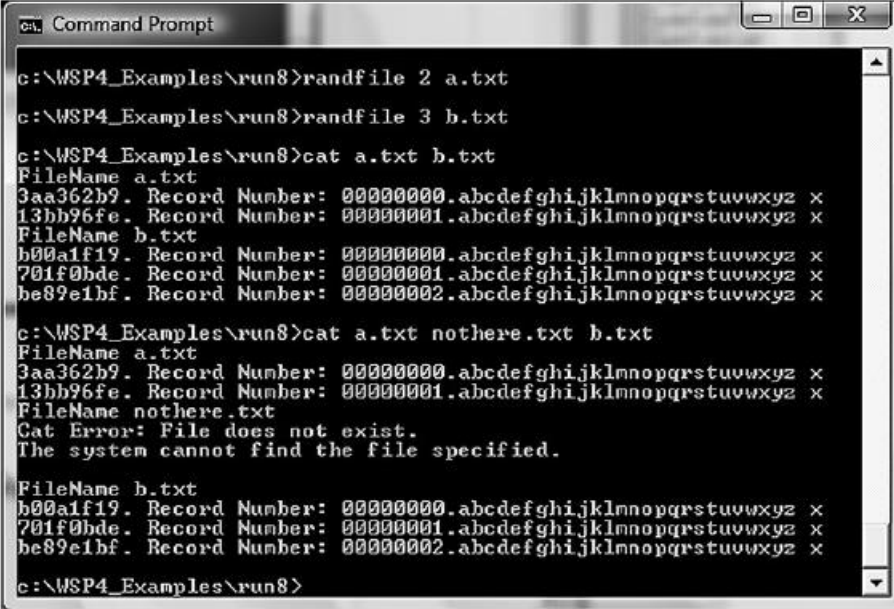
int _tmain (int argc, LPTSTR argv []) {
    HANDLE hInFile, hStdIn = GetStdHandle (STD_INPUT_HANDLE);
    HANDLE hStdOut = GetStdHandle (STD_OUTPUT_HANDLE);
    BOOL dashS;
    int iArg, iFirstFile;

    /*dashS will be set only if "-s" is on the command line. */
    /*iFirstFile is the argv [] index of the first input file. */
    iFirstFile = Options (argc, argv, _T("-s"), &dashS, NULL);

    if (iFirstFile == argc) { /* No files in arg list. */
        CatFile (hStdIn, hStdOut);
        return 0;
    }
}
```

```
/* Process the input files. */
for (iArg = iFirstFile; iArg < argc; iArg++) {
    hInFile = CreateFile (argv [iArg], GENERIC_READ,
        0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

    if (hInFile == INVALID_HANDLE_VALUE) {
        if (!dashS) ReportError (_T ("Cat Error: File does not exist."), 0, TRUE);
    }
    else {
        CatFile (hInFile, hStdOut);
        if (GetLastError() != 0 && !dashS) {
            ReportError (_T ("Cat Error: Could not process file completely."),
                0, TRUE);
        }
        CloseHandle (hInFile);
    }
}
return 0;
}
```



```
cat. Command Prompt
c:\WSP4_Examples\run8>randfile 2 a.txt
c:\WSP4_Examples\run8>randfile 3 b.txt
c:\WSP4_Examples\run8>cat a.txt b.txt
FileName a.txt
3aa362b9. Record Number: 00000000.abcdefghijklmnopqrstuvwxyz x
13bb96fe. Record Number: 00000001.abcdefghijklmnopqrstuvwxyz x
FileName b.txt
b00a1f19. Record Number: 00000000.abcdefghijklmnopqrstuvwxyz x
701f0bde. Record Number: 00000001.abcdefghijklmnopqrstuvwxyz x
be89e1bf. Record Number: 00000002.abcdefghijklmnopqrstuvwxyz x
c:\WSP4_Examples\run8>cat a.txt nowhere.txt b.txt
FileName a.txt
3aa362b9. Record Number: 00000000.abcdefghijklmnopqrstuvwxyz x
13bb96fe. Record Number: 00000001.abcdefghijklmnopqrstuvwxyz x
FileName nowhere.txt
Cat Error: File does not exist.
The system cannot find the file specified.
FileName b.txt
b00a1f19. Record Number: 00000000.abcdefghijklmnopqrstuvwxyz x
701f0bde. Record Number: 00000001.abcdefghijklmnopqrstuvwxyz x
be89e1bf. Record Number: 00000002.abcdefghijklmnopqrstuvwxyz x
c:\WSP4_Examples\run8>
```



# Program utilitar: preluarea opțiunilor

**Options.c** este un program utilitar cu număr variabil de argumente care permite **evaluarea opțiunilor** specificate în linia de comandă și **returnează indexul primului argument din argv de după opțiuni**

```
/* Utility function to extract option flags from the command line. */

#include "Everything.h"
#include <stdarg.h>

DWORD Options (int argc, LPCTSTR argv [], LPCTSTR OptStr, ...)

/* argv is the command line.
   The options, if any, start with a '-' in argv[1], argv[2], ...
   OptStr is a text string containing all possible options,
   in one-to-one correspondence with the addresses of Boolean variables in the variable argument list (...).
   These flags are set if and only if the corresponding option character occurs in argv [1], argv [2], ...
   The return value is the argv index of the first argument beyond the options. */
{
    va_list pFlagList;
    LPBOOL pFlag;
    int iFlag = 0, iArg;

    va_start (pFlagList, OptStr);

    while ((pFlag = va_arg (pFlagList, LPBOOL)) != NULL && iFlag < (int)_tcslen (OptStr)) {
        *pFlag = FALSE;
        for (iArg = 1; !(*pFlag) && iArg < argc && argv [iArg] [0] == _T('-'); iArg++)
            *pFlag = _memtchr (argv [iArg], OptStr [iFlag], _tcslen (argv [iArg])) != NULL;
        iFlag++;
    }

    va_end (pFlagList);

    for (iArg = 1; iArg < argc && argv [iArg] [0] == _T('-'); iArg++);

    return iArg;
}
```

# Exemplu: criptarea fișierelor

Programul cci.c este similar cu programul de copiere a unui fișier în alt fișier, dar se introduce un pas de proesare a fiecărui octet înainte de copiere – pe baza cifrului Caesar (un cifru de substituție cu deplasare)

```
/* Chapter 2. cci Version 1. Modified Caesar cipher
(http://en.wikipedia.org/wiki/Caesar_cipher). */
/* Main program, which can be linked to different implementations */
/* of the cci_f function. */

/* cci shift file1 file2
 *shift is the integer added mod 256 to each byte.
 *Otherwise, this program is like cp and cpCF but there is no direct UNIX
equivalent. */

/* This program illustrates:
 *1. File processing with conversion.
 *2. Boilerplate code to process the command line.
 */

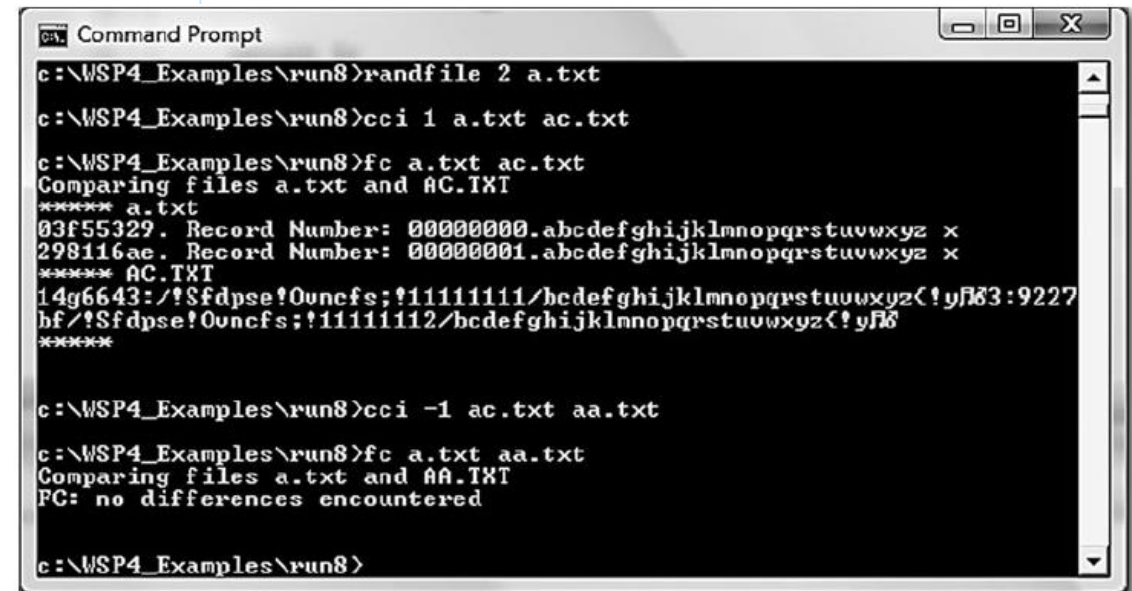
#include "Everything.h"
#include <io.h>

BOOL cci_f (LPCTSTR, LPCTSTR, DWORD);

int _tmain (int argc, LPTSTR argv []) {
    if (argc != 4)
        ReportError (_T ("Usage: cci shift file1 file2"), 1, FALSE);

    if (!cci_f (argv [2], argv [3], _ttoi(argv[1])))
        ReportError (_T ("Encryption failed."), 4, TRUE);

    return 0;
}
```



```
Command Prompt
c:\WSP4_Examples\run8>randfile 2 a.txt
c:\WSP4_Examples\run8>cci 1 a.txt ac.txt
c:\WSP4_Examples\run8>fc a.txt ac.txt
Comparing files a.txt and AC.TXT
***** a.txt
03f55329. Record Number: 00000000.abcdefghijklmnopqrstuvmxyz x
298116ae. Record Number: 00000001.abcdefghijklmnopqrstuvmxyz x
***** AC.TXT
14g6643:/!$fdpse!0uncfs;!1111111/bcdefghijklmnopqrstuvmxyz<!yfl3:9227
bf/!$fdpse!0uncfs;!1111112/bcdefghijklmnopqrstuvmxyz<!yfl3
*****
c:\WSP4_Examples\run8>cci -1 ac.txt aa.txt
c:\WSP4_Examples\run8>fc a.txt aa.txt
Comparing files a.txt and AA.TXT
FC: no differences encountered
c:\WSP4_Examples\run8>
```

# Exemplu: criptarea fișierelor (cont)

Funcția `cci_f` este apelată de programul `cci.c` pentru criptarea octeților

- implementează o variantă a cifrului Caesar (un cifru de substituție cu deplasare)

```
/* Chapter 2. Simple cci_f (modified Caesar cipher) implementation */
#include "Everything.h"

#define BUF_SIZE 65536 /* Generally, you will get better performance with larger buffers (use powers of 2).
/* 65536 worked well; larger numbers did not help in some simple tests. */

BOOL cci_f(LPCTSTR fIn, LPCTSTR fOut, DWORD shift)
/*fIn:Source file pathname
*fOut:Destination file pathname
*shift:Numerical shift */
{
    HANDLE hIn, hOut;
    DWORD nIn, nOut, iCopy;
    BYTE buffer[BUF_SIZE], bShift = (BYTE)shift;
    BOOL writeOK = TRUE;

    hIn = CreateFile(fIn, GENERIC_READ, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hIn == INVALID_HANDLE_VALUE) return FALSE;

    hOut = CreateFile(fOut, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hOut == INVALID_HANDLE_VALUE) {
        CloseHandle(hIn);
        return FALSE;
    }

    while (writeOK && ReadFile(hIn, buffer, BUF_SIZE, &nIn, NULL) && nIn > 0) {
        for (iCopy = 0; iCopy < nIn; iCopy++)
            buffer[iCopy] = buffer[iCopy] + bShift;
        writeOK = WriteFile(hOut, buffer, nIn, &nOut, NULL);
    }

    CloseHandle(hIn);
    CloseHandle(hOut);

    return writeOK;
}
```

# Gestiunea fișierelor și directoarelor

---

## Ștergerea fișierului

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

## Copierea fișierului

```
BOOL WINAPI CopyFile( LPCTSTR lpExistingFileName, LPCTSTR lpNewFileName, BOOL bFailIfExists );
```

## Legături fizice și simbolice

```
BOOL WINAPI CreateHardLink( LPCTSTR lpFileName, LPCTSTR lpExistingFileName, LPSECURITY_ATTRIBUTES lpSecurityAttributes );
```

```
BOOLEAN WINAPI CreateSymbolicLink( LPTSTR lpSymLinkFileName, LPTSTR lpTargetFileName, DWORD dwFlags );
```

## Redenumire și mutare

```
BOOL WINAPI MoveFile( LPCTSTR lpExistingFileName, LPCTSTR lpNewFileName );
```

```
BOOL WINAPI MoveFileEx( LPCTSTR lpExistingFileName, LPCTSTR lpNewFileName, DWORD dwFlags );
```

# Gestiunea fișierelor și directoarelor

## Ștergerea fișierului

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

Numele (calea) fișierului

## Copiarea fișierului

```
BOOL WINAPI CopyFile( LPCTSTR lpExistingFileName,  
LPCTSTR lpNewFileName,  
BOOL bFailIfExists );
```

Numele (calea) fișierului  
existent și

nou creat (copie)

Dacă FALSE: fișierul existent va fi suprascris

Se copiază inclusiv metadatele fișierului

## Legături fizice

```
BOOL WINAPI CreateHardLink(  
LPCTSTR lpFileName,  
LPCTSTR lpExistingFileName,  
LPSECURITY_ATTRIBUTES lpSecurityAttributes );
```

Fișierul legat și legătură  
trbuie să fie pe același  
sistem de fișiere

Numele (calea) fișierului legătură

Numele (calea) fișierului legat

Atribute de securitate

## Legături simbolice

```
BOOLEAN WINAPI CreateSymbolicLink(  
LPTSTR lpSymLinkFileName,  
LPTSTR lpTargetFileName,  
DWORD dwFlags );
```

0 pt fișier și  
SYMBOLIC\_LINK\_FLAG\_DIRECTORY pt director

Fișierul legat și legătură  
pot fi pe sisteme de  
fișiere distincte

## Redenumire și mutare

```
BOOL WINAPI MoveFile(  
LPCTSTR lpExistingFileName,  
LPCTSTR lpNewFileName );
```

Eșuează dacă noul nume  
există deja

Numele (calea) fișierului / directorului  
existent respectiv

noul nume – trebuie să nu existe deja

```
BOOL WINAPI MoveFileEx(  
LPCTSTR lpExistingFileName,  
LPCTSTR lpNewFileName,  
DWORD dwFlags );
```

Poate suprascrie fișierul  
dacă există deja

MOVEFILE\_REPLACE\_EXISTING (suprascrie fișierul – doar în cazul fișierelor)  
MOVEFILE\_WRITE\_THROUGH (revine doar după scrierea efectivă pe disc)  
MOVEFILE\_COPY\_ALLOWED (folosește CopyFile și DeleteFile – pentru mutarea pe alt drive)

Directoarele trebuie să fie pe același drive – fișierele nu neaparat

# Gestiunea directoarelor

## Crearea și ștergerea directorului

```
BOOL WINAPI CreateDirectory(  
    LPCTSTR lpPathName,  
    LPSECURITY_ATTRIBUTES lpSecurityAttributes );
```

Numele (calea) directorului

Atribute de securitate

```
BOOL WINAPI RemoveDirectory(  
    LPCTSTR lpPathName );
```

Numele (calea) directorului

## Setarea și obținerea directorului curent

```
BOOL WINAPI SetCurrentDirectory(  
    LPCTSTR lpPathName );
```

Calea relativă sau absolută către noul director curent

Calea absolută începe cu literă și : (D:) sau nume UNC sub forma \\servername\sharename

```
DWORD WINAPI GetCurrentDirectory(  
    DWORD nBufferLength,  
    LPTSTR lpBuffer );
```

Dimensiunea în număr de caractere a bufferului

Pointer la bufferul care va reține calea către directorul curent

- Returnează
  - lungimea căii sau
  - dimensiunea bufferului necesar dacă bufferul este prea mic
    - Trebuie testat dacă valoarea returnată este mai mare decât `nBufferLength`
  - 0 în caz de eroare

# Exemplu: afișarea directorului curent

```
/* pwd: Print the current directory. */
/* This program illustrates:
1. Windows GetCurrentDirectory
2. Testing the length of a returned string */

#include "Everything.h"

#define DIRNAME_LEN (MAX_PATH + 2)

int _tmain(int argc, LPTSTR argv[]) {
    /* Buffer to receive current directory allows for the CR,
    LF at the end of the longest possible path. */

    TCHAR pwdBuffer[DIRNAME_LEN];
    DWORD lenCurDir;
    lenCurDir = GetCurrentDirectory(DIRNAME_LEN, pwdBuffer);

    if (lenCurDir == 0)
        ReportError(_T("Failure getting pathname."), 1, TRUE);
    if (lenCurDir > DIRNAME_LEN)
        ReportError(_T("Pathname is too long."), 2, FALSE);

    PrintMsg(GetStdHandle(STD_OUTPUT_HANDLE), pwdBuffer);
    return 0;
}
```



# Operații de I/O la consolă

Operațiile de citire și scriere la consolă pot fi efectuate cu funcțiile ReadFile și WriteFile sau mai simplu cu ReadConsole și WriteConsole

- Avantaj:
  - lucrează cu caractere text generice de tipul TCHAR – în loc de octeți
  - Procesează caracterele în funcție de modul consolei

## Setarea modului de procesare

```
BOOL WINAPI SetConsoleMode(  
    HANDLE hConsoleHandle,  
    DWORD dwMode );
```

Handle-ul consolei de intrare sau bufferul  
ecranului – cu drept de scriere  
(GENERIC\_WRITE)

ENABLE\_ECHO\_INPUT (scriere cu ecou – caracterele introduse sunt afișate pe ecran  
– doar împreună cu ENABLE\_LINE\_INPUT)  
ENABLE\_LINE\_INPUT (revine doar la final de linie)  
ENABLE\_PROCESSED\_INPUT (procesarea caracterelor speciale – ex. sfârșit de linie, backspace, etc.)  
ENABLE\_PROCESSED\_OUTPUT (procesarea caracterelor speciale – ex. sfârșit de linie, backspace, etc.)  
ENABLE\_WRAP\_AT\_EOL\_OUTPUT (permite trecerea la linie nouă dacă se umple o linie)

- Returnează
  - TRUE în caz de succes și
  - FALSE în caz de eroare - și modul nu se schimbă



# Operații de I/O la consolă

## Citire de la consolă – similar cu ReadFile

```
BOOL WINAPI ReadConsole(  
    HANDLE hConsoleInput,   
    LPVOID lpBuffer,   
    DWORD nNumberOfCharsToRead,   
    LPDWORD lpNumberOfCharsRead,   
    LPVOID pInputControl );
```

Diagram illustrating the parameters of `ReadConsole`:

- `HANDLE hConsoleInput`: Handle-ul consolei de intrare cu drept de citire (`GENERIC_READ`)
- `LPVOID lpBuffer`: Pointer către bufferul care reține datele citite
- `DWORD nNumberOfCharsToRead`: Numărul de caractere TCHAR de citit
- `LPDWORD lpNumberOfCharsRead`: Numărul de caractere TCHAR citite (se returnează în acest parametru)
- `LPVOID pInputControl`: Pointer la structura `CONSOLE_READCONSOLE_CONTROL`. Folosim NULL deocamdată

- Ambele funcții returnează
  - TRUE în caz de succes și
  - FALSE în caz de eroare

## Scrierea este similară cu WriteFile – cu parametri interpretați analogic cu ReadConsole

```
BOOL WINAPI WriteConsole(  
    HANDLE hConsoleOutput,   
    const VOID *lpBuffer,   
    DWORD nNumberOfCharsToWrite,   
    LPDWORD lpNumberOfCharsWritten,   
    LPVOID lpReserved );
```

Diagram illustrating the parameters of `WriteConsole`:

- `HANDLE hConsoleOutput`: Handle-ul consolei de ieșire cu drept de scriere (`GENERIC_WRITE`)
- `const VOID *lpBuffer`: Pointer către bufferul care conține datele de scris
- `DWORD nNumberOfCharsToWrite`: Numărul de caractere TCHAR de scris
- `LPDWORD lpNumberOfCharsWritten`: Numărul de caractere TCHAR scrise (se returnează în acest parametru)
- `LPVOID lpReserved`: Rezervat: trebuie să fie NULL

# Exemplu: lucrul cu consola

## Programul utilitar PrintMsg.c

```
/* PrintMsg.c: ConsolePrompt, PrintStrings, PrintMsg */

#include "Everything.h"
#include <stdarg.h>

BOOL PrintStrings(HANDLE hOut, ...)
/* Write the messages to the output handle. Frequently hOut
will be standard out or error, but this is not required.
Use WriteConsole (to handle Unicode) first, as the
output will normally be the console. If that fails, use WriteFile.

hOut:Handle for output file.
... :Variable argument list containing TCHAR strings.
The list must be terminated with NULL. */
{
    DWORD msgLen, count;
    LPCTSTR pMsg;
    va_list pMsgList; /* Current message string. */

    va_start(pMsgList, hOut); /* Start processing msgs. */

    while ((pMsg = va_arg(pMsgList, LPCTSTR)) != NULL) {
        msgLen = lstrlen(pMsg);
        if (!WriteConsole(hOut, pMsg, msgLen, &count, NULL)
            && !WriteFile(hOut, pMsg, msgLen * sizeof(TCHAR), &count, NULL)) {
            va_end(pMsgList);
            return FALSE;
        }
    }
    va_end(pMsgList);
    return TRUE;
}

BOOL PrintMsg(HANDLE hOut, LPCTSTR pMsg) {
/* For convenience only - Single message version of PrintStrings so that
you do not have to remember the NULL arg list terminator. */
    return PrintStrings(hOut, pMsg, NULL);
}
```

```
BOOL ConsolePrompt(LPCTSTR pPromptMsg, LPTSTR pResponse, DWORD maxChar, BOOL echo)
/* Prompt the user at the console and get a response
which can be up to maxChar generic characters.

pPromptMsg:Message displayed to user.
pResponse:Programmer-supplied buffer that receives the response.
maxChar:Maximum size of the user buffer, measured as generic characters.
echo:Do not display the user's response if this flag is FALSE. */
{
    HANDLE hIn, hOut;
    DWORD charIn, echoFlag;
    BOOL success;
    hIn = CreateFile(_T("CONIN$"), GENERIC_READ | GENERIC_WRITE, 0,
                    NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    hOut = CreateFile(_T("CONOUT$"), GENERIC_WRITE, 0,
                     NULL, OPEN_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);

    /* Should the input be echoed? */
    echoFlag = echo ? ENABLE_ECHO_INPUT : 0;

    /* API "and" chain. If any test or system call fails, the rest of the expression
    is not evaluated, and the subsequent functions are not called.
    GetStdError () will return the result of the failed call. */

    success = SetConsoleMode(hIn, ENABLE_LINE_INPUT | echoFlag | ENABLE_PROCESSED_INPUT)
        && SetConsoleMode(hOut, ENABLE_WRAP_AT_EOL_OUTPUT | ENABLE_PROCESSED_OUTPUT)
        && PrintStrings(hOut, pPromptMsg, NULL)
        && ReadConsole(hIn, pResponse, maxChar - 2, &charIn, NULL);

    /* Replace the CR-LF by the null character. */
    if (success) pResponse[charIn - 2] = _T('\0');
    else ReportError(_T("ConsolePrompt failure."), 0, TRUE);

    CloseHandle(hIn);
    CloseHandle(hOut);
    return success;
}
```

# Repoziționarea în fișier

## Pointerul de fișier

- Fiecare fișier deschis are asociat un pointer care indică poziția curentă în fișier
  - La deschiderea fișierului pointerul este zero – indică începutul de fișier
- Operațiile de citire / scriere transferă date de la / la această locație în mod secvențial
  - Și incrementează valoarea pointerului cu numărul de octeți transferați
- Accesul direct la locații arbitrare necesită repoziționarea pointerului

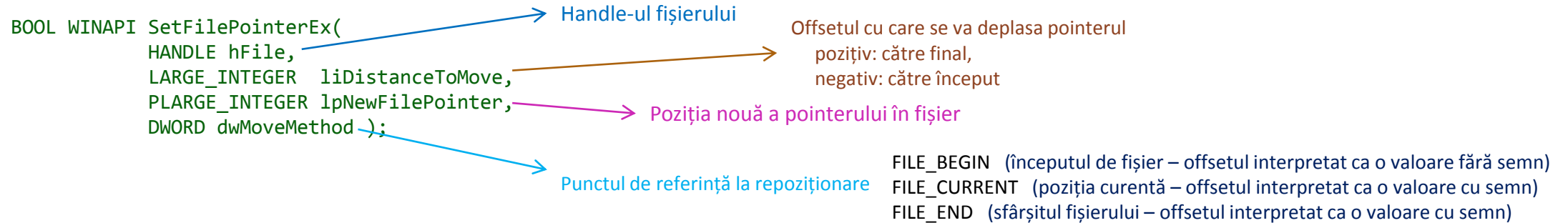
## Adresa locației din fișier

- Windows NT oferă suport pentru sistemele de fișiere pe 64 de biți
  - În principiu fișierele pot ajunge la dimensiuni teoretice de  $2^{64}$  de octeți
  - SF cu adrese pe 32 de biți au limitare pentru dimensiunea unui fișier de  $2^{32}$  octeți – adică 4 GB
    - Multe aplicații gestionează fișiere mai mari

## Repoziționarea pointerului

- Funcția **SetFilePointer** este învechit: tratarea adreselor pe 64 de biți este anevoioasă
- Funcția **SetFilePointerEx** oferă suport pentru adrese pe 64 de biți – descriem și folosim această funcție
  - Necesită parametri de tip LARGE\_INTEGER - uniune de componente
    - Valoare pe 64 de biți
      - LONGLONG QuadPart → Acesta este componenta care va reține adresa din cadrul fișierului
    - Două valori pe câte 32 de biți
      - DWORD LowPart
      - LONG HighPart

# Repoziționarea în fișier



- Returnează TRUE în caz de succes și FALSE în caz de eșec
- Prin repoziționare la finalul fișierului putem determina dimensiunea fișierului
  - Efect secundar: se mută pointerul la final
  - Alternativa: funcția GetFileSizeEx
- Este posibilă repoziționarea după finalul de fișier, dar dimensiunea se modifică doar la efectuarea
  - unei operații de scriere (octeții din “gaură” rămân neinițializați ) sau
  - folosind funcția SetEndOfFile

## Repoziționarea la citire și scriere poate fi realizată folosind structura OVERLAPPED

- Ultimul parametru al funcțiilor ReadFile și WriteFile – până acum foloseam NULL
  - Componentele Offset și OffsetHigh

# Exemplu: citirea unei înregistrări

## Programul getn.c

```
/* Chapter 3. getn command. */
/* Get a specified fixed size record from the file.
   The user is prompted for record numbers. Each requested record is
   retrieved and displayed until a negative record number is requested.
   Fixed size, text records are assumed. */
/* There is a maximum line size (MAX_LINE_SIZE). */

/* This program illustrates:
1. Setting the file pointer.
2. LARGE_INTEGER arithmetic and using the 64-bit file positions. */

#include <windows.h>
#include <tchar.h>
#include <stdio.h>

#define MAX_LINE_SIZE 256

int _tmain(int argc, LPTSTR argv[]) {
    HANDLE hInFile;
    LARGE_INTEGER CurPtr;
    DWORD nRead, RecSize;
    TCHAR buffer[MAX_LINE_SIZE + 1];
    BOOL Exit = FALSE;
    LPTSTR p;
    int RecNo;

    if (argc != 2)
        ReportError(_T("Usage: getn file"), 1, FALSE);
    hInFile = CreateFile(argv[1], GENERIC_READ, 0, NULL,
                        OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
    if (hInFile == INVALID_HANDLE_VALUE)
        ReportError(_T("getn error: Cannot open file."), 2, TRUE);
```

```
    RecSize = 0;
    /* Get the record size from the first record.
       Read the max line size and look for a CR/LF. */

    if (!ReadFile(hInFile, buffer, sizeof(buffer), &nRead, NULL) || nRead == 0)
        ReportError(_T("Nothing to read."), 1, FALSE);
    if ((p = _tcsstr(buffer, _T("\r\n"))) == NULL)
        ReportError(_T("No end of line found."), 2, FALSE);

    /* Ignore Win64 warning about possible loss of data */
    RecSize = (DWORD)(p - buffer + 2); /* 2 for the CRLF. */

    _tprintf(_T("Record size is: %d\n"), RecSize);

    while (TRUE) {
        _tprintf(_T("Enter record number. Negative to exit: "));
        _tscanf_s(_T("%d"), &RecNo);
        if (RecNo < 0) break;

        CurPtr.QuadPart = (LONGLONG)RecNo * RecSize;
        if (!SetFilePointerEx(hInFile, CurPtr, NULL, FILE_BEGIN))
            /* Alternative: Use an overlapped structure */
            ReportError(_T("getn Error: Set Pointer."), 3, TRUE);
        if (!ReadFile(hInFile, buffer, RecSize, &nRead, NULL) || (nRead != RecSize))
            ReportError(_T("Error reading n-th record."), 0, TRUE);
        buffer[RecSize] = _T('\0');
        _tprintf(_T("%s\n"), buffer);
    }
    CloseHandle(hInFile);
    return 0;
}
```

# Dimensiunea fișierului

## Determinarea dimensiunii fișierului

```
BOOL WINAPI GetFileSizeEx(  
    HANDLE hFile,  
    PLARGE_INTEGER lpFileSize );
```

→ Handle-ul fișierului

→ Dimensiunea este returnată în acest parametru

- Pentru obținerea dimensiunii fișierului după nume (fișier care nu este deschis) se pot folosi funcțiile `GetCompressedFileSize` sau `FindFirstFile`

## Setarea finalului de fișier

- Fișierul poate fi extins sau trunchiat - finalul fișierului va fi la poziția curentă a pointerului de fișier

```
BOOL WINAPI SetEndOfFile(  
    HANDLE hFile );
```

→ Handle-ul fișierului

- Finalul fizic poate fi setat după finalul logic – formând o coadă sau o gaură (conținutul dintre finalul logic și finalul fizic)
- Fișierul poate fi extins și prin mai multe apeluri de scriere succesive – însă acesta rezultă în alocare mai fragmentată
- Fișierele rare consumă spațiu de pe disc doar pe măsură ce sunt scrise
  - Funcția `DeviceIoControl` cu flagul `FSCTL_SET_SPARSE` poate specifica un fișier ca fiind rar (Sparse file)
- NTFS inițializează fișierele și cozile/găurile cu 0 din motive de securitate

# Exemplu: actualizarea înregistrărilor

```
#include "Everything.h"

#define STRING_SIZE 256

typedef struct _RECORD { /* File record structure */
    DWORD referenceCount; /* 0 means an empty record */
    SYSTEMTIME recordCreationTime;
    SYSTEMTIME recordLastReferenceTime;
    SYSTEMTIME recordUpdateTime;
    TCHAR dataString[STRING_SIZE];
} RECORD;

typedef struct _HEADER { /* File header descriptor */
    DWORD numRecords;
    DWORD numNonEmptyRecords;
} HEADER;

int _tmain (int argc, LPTSTR argv[]) {
    HANDLE hFile;
    LARGE_INTEGER currentPtr;
    DWORD OpenOption, nXfer, recNo;
    RECORD record;
    TCHAR string[STRING_SIZE], command, extra;
    OVERLAPPED ov = {0, 0, 0, 0, NULL}, ovZero = {0, 0, 0, 0, NULL};
    HEADER header = {0, 0};
    SYSTEMTIME currentTime;
    BOOLEAN headerChange, recordChange;

    int prompt = (argc <= 3) ? 1 : 0;

    if (argc < 2)
        ReportError (_T("Usage: RecordAccess file [nrec [prompt]]"), 1, FALSE);

    OpenOption = ((argc > 2 && _ttoi(argv[2]) <= 0) || argc <= 2)
        ? OPEN_EXISTING : CREATE_ALWAYS;
    hFile = CreateFile (argv[1], GENERIC_READ | GENERIC_WRITE,
        0, NULL, OpenOption, FILE_FLAG_RANDOM_ACCESS, NULL);
    if (hFile == INVALID_HANDLE_VALUE)
        ReportError (_T("RecordAccess error: Cannot open existing file."), 2, TRUE);
```

```
    if (argc >= 3 && _ttoi(argv[2]) > 0) {
        /* Write the header (all header and pre-size the new file) */

        header.numRecords = _ttoi(argv[2]);
        if (!WriteFile(hFile, &header, sizeof (header), &nXfer, &ovZero))
            ReportError (_T("RecordAccess Error: WriteFile header."), 4, TRUE);

        currentPtr.QuadPart = (LONGLONG)sizeof(RECORD) *
            _ttoi(argv[2]) + sizeof(HEADER);
        if (!SetFilePointerEx (hFile, currentPtr, NULL, FILE_BEGIN))
            ReportError (_T("RecordAccess Error: Set Pointer."), 4, TRUE);
        if (!SetEndOfFile(hFile))
            ReportError (_T("RecordAccess Error: Set End of File."), 5, TRUE);
        if (prompt)
            _tprintf (_T("Empty file with %d records created.\n"),
                header.numRecords);

        return 0;
    }
```

# Exemplu: actualizarea înregistrărilor

```
/* Read the file header to find the number of records and non-empty records */
if (!ReadFile(hFile, &header, sizeof(HEADER), &nXfer, &ovZero))
    ReportError(_T("RecordAccess Error: ReadFile header."), 6, TRUE);
if (prompt) _tprintf(_T("File %s contains %d non-empty records of size %d.\n Total capacity: %d\n"),
                    argv[1], header.numNonEmptyRecords, sizeof(RECORD), header.numRecords);

/* Prompt the user to read or write a numbered record */
while (TRUE) {
    headerChange = FALSE; recordChange = FALSE;
    if (prompt) _tprintf(_T("Enter r(ead)/w(rite)/d(elete)/qu(it) record#\n"));
    _tscanf(_T("%c%u%c"), &command, &recNo, &extra);
    if (command == _T('q')) break;
    if (recNo >= header.numRecords) {
        if (prompt) _tprintf(_T("record Number is too large. Try again.\n"));
        continue;
    }
    currentPtr.QuadPart = (LONGLONG)recNo * sizeof(RECORD) + sizeof(HEADER);
    ov.Offset = currentPtr.LowPart;
    ov.OffsetHigh = currentPtr.HighPart;
    if (!ReadFile(hFile, &record, sizeof(RECORD), &nXfer, &ov))
        ReportError(_T("RecordAccess: ReadFile failure."), 7, FALSE);
    GetSystemTime(&currentTime); /* Use to update record time fields */
    record.recordLastRefernceTime = currentTime;
    if (command == _T('r') || command == _T('d')) { /* Report record contents, if any */
        if (record.referenceCount == 0) {
            if (prompt) _tprintf(_T("record Number %d is empty.\n"), recNo);
            continue;
        } else {
            if (prompt) _tprintf(_T("record Number %d. Reference Count: %d \n"),
                                recNo, record.referenceCount);
            if (prompt) _tprintf(_T("Data: %s\n"), record.dataString);
            /* Exercise: Display times. See ls.c for an example */
        }
    }
    if (command == _T('d')) { /* Delete the record */
        record.referenceCount = 0;
        header.numNonEmptyRecords--;
        headerChange = TRUE;
        recordChange = TRUE;
    }
}
```



# Exemplu: actualizarea înregistrărilor

```
else if (command == _T('w')) { /* Write the record, even if for the first time */
    if (prompt) _tprintf (_T("Enter new data string for the record.\n"));
    _fgetts (string, sizeof(string), stdin); // Don't use _getts (potential buffer overflow)
    string[_tcslen(string)-1] = _T('\0'); // remove the newline character
    if (record.referenceCount == 0) {
        record.recordCreationTime = currentTime;
        header.numNonEmptyRecords++;
        headerChange = TRUE;
    }
    record.recordUpdateTime = currentTime;
    record.referenceCount++;
    _tcsncpy (record.dataString, string, STRING_SIZE-1);
    recordChange = TRUE;
}
else {
    if (prompt) _tprintf (_T("command must be r, w, or d. Try again.\n"));
}
/* Update the record in place if any record contents have changed. */
if (recordChange && !WriteFile (hFile, &record, sizeof (RECORD), &Xfer, &ov))
    ReportError (_T("RecordAccess: WriteFile update failure."), 8, FALSE);
/* Update the number of non-empty records if required */
if (headerChange) {
    if (!WriteFile (hFile, &header, sizeof (header), &Xfer, &ovZero))
        ReportError (_T("RecordAccess: WriteFile update failure."), 9, FALSE);
}
} /* while */

if (prompt)
    _tprintf (_T("Computed number of non-empty records is: %d\n"), header.numNonEmptyRecords);
if (!ReadFile(hFile, &header, sizeof (HEADER), &Xfer, &ovZero))
    ReportError (_T("RecordAccess Error: ReadFile header."), 10, TRUE);
if (prompt)
    _tprintf (_T("File %s NOW contains %d non-empty records.\nTotal capacity is: %d\n"),
        argv[1], header.numNonEmptyRecords, header.numRecords);

CloseHandle (hFile);
return 0;
}
```



```
Command Prompt
C:\WSP4_Examples\run8>RecordAccess large.ra 20000000
Empty file with 20000000 records created.

C:\WSP4_Examples\run8>RecordAccess large.ra
File large.ra contains 0 non-empty records of size 308.
Total capacity: 20000000
Enter r(ead)/w(rite)/d(elete)/qu(it) Record#
w 2
Enter new data string for the record.
This is record number 2
Enter r(ead)/w(rite)/d(elete)/qu(it) Record#
w 19000000
Enter new data string for the record.
Record 19 million. Write takes over a minute. Please wait!
Enter r(ead)/w(rite)/d(elete)/qu(it) Record#
w 2
Enter new data string for the record.
Record number 2, Version 2
Enter r(ead)/w(rite)/d(elete)/qu(it) Record#
r 19000000
Record Number 19000000. Reference Count: 1
Data: Record 19 million. Write takes over a minute. Please wait!
Enter r(ead)/w(rite)/d(elete)/qu(it) Record#
r 2
Record Number 2. Reference Count: 2
Data: Record number 2, Version 2
Enter r(ead)/w(rite)/d(elete)/qu(it) Record#
r 20000001
Record Number is too large. Try again.
Enter r(ead)/w(rite)/d(elete)/qu(it) Record#
r 1
Record Number 1 is empty.
Enter r(ead)/w(rite)/d(elete)/qu(it) Record#
d 2
Record Number 2. Reference Count: 2
Data: Record number 2, Version 2
Enter r(ead)/w(rite)/d(elete)/qu(it) Record#
r 2
Record Number 2 is empty.
Enter r(ead)/w(rite)/d(elete)/qu(it) Record#
qu
Computed number of non-empty records is: 1
File large.ra NOW contains 1 non-empty records.
Total capacity is: 20000000

C:\WSP4_Examples\run8>dir *.ra
Volume in drive C is HP
Volume Serial Number is 521E-9D92

Directory of C:\WSP4_Examples\run8

09/13/2009  12:26 PM        6,160,000,000 large.ra
               1 File(s)        6,160,000,000 bytes
               0 Dir(s)        469,504,638,976 bytes free
```

# Căutarea unui fișier/subdirector

## Căutarea fișierelor/subdirectoarelor care satisfac un șablon de nume și obținerea atributelor

- Căutarea necesită un handle-de căutare – obținut print FindFirstFile
- Se parcurg toate fișierele care satisfac criteriul – folosind funcția FindNextFile
- Se termină căutarea închizând handle-ul de căutare – cu funcția FindClose

```
HANDLE WINAPI FindFirstFile(  
    LPCTSTR lpFileName,  
    LPWIN32_FIND_DATA lpFindFileData );
```

Directorul sau calea care poate conține  
meta-caractere (wildcard), cum ar fi ? sau \*

Informații despre primul fișier/subdirector  
care satisface criteriile (dacă s-a găsit)

- Returnează un handle de căutare sau INVALID\_HANDLE\_VALUE

```
BOOL WINAPI FindNextFile(  
    HANDLE hFindFile,  
    LPWIN32_FIND_DATA lpFindFileData );
```

Handle-ul obținut cu funcția FindFirstFile

Informații despre următorul fișier/subdirector  
care satisface criteriile (dacă s-a găsit)

- Returnează FALSE dacă a eșuat sau dacă nu s-au găsit alte fișiere (caz în care GetLastError returnează ERROR\_NO\_MORE\_FILES)

```
BOOL WINAPI FindClose(  
    HANDLE hFindFile );
```

Handle-ul obținut cu funcția FindFirstFile

- Atenție: nu folosim CloseHandle – ar genera o excepție. Handle-ul obținut prin FindFirstFile nu este un handle de obiect kernel

```
typedef struct _WIN32_FIND_DATA {  
    DWORD dwFileAttributes;  
    FILETIME ftCreationTime;  
    FILETIME ftLastAccessTime;  
    FILETIME ftLastWriteTime;  
    DWORD nFileSizeHigh;  
    DWORD nFileSizeLow;  
    DWORD dwReserved0;  
    DWORD dwReserved1;  
    TCHAR cFileName[MAX_PATH];  
    TCHAR cAlternateFileName[14];  
} WIN32_FIND_DATA;
```

# Exemplu: FindFirstFile

---

```
#include <windows.h>
#include <tchar.h>
#include <stdio.h>

void _tmain(int argc, TCHAR *argv[]) {
    WIN32_FIND_DATA FindFileData;
    HANDLE hFind;

    if (argc != 2) {
        _tprintf(TEXT("Usage: %s [target_file]\n"), argv[0]);
        return;
    }

    _tprintf(TEXT("Target file is %s\n"), argv[1]);
    hFind = FindFirstFile(argv[1], &FindFileData);
    if (hFind == INVALID_HANDLE_VALUE) {
        printf("FindFirstFile failed (%d)\n", GetLastError());
        return;
    } else {
        _tprintf(TEXT("The first file found is %s\n"),
            FindFileData.cFileName);
        FindClose(hFind);
    }
}
```

# Exemplu: listarea fișierelor din director

Listarea conținutului unui director cu câteva atribute

```
#include <windows.h>
#include <tchar.h>
#include <stdio.h>
#include <strsafe.h>

void DisplayErrorBox(LPTSTR lpszFunction);

int _tmain(int argc, TCHAR *argv[]) {
    WIN32_FIND_DATA ffd;
    LARGE_INTEGER filesize;
    TCHAR szDir[MAX_PATH];
    size_t length_of_arg;
    HANDLE hFind = INVALID_HANDLE_VALUE;
    DWORD dwError = 0;

    // If the directory is not specified as a command-line argument,
    // print usage.

    if (argc != 2) {
        _tprintf(TEXT("\nUsage: %s <directory name>\n"), argv[0]);
        return (-1);
    }

    // Check that the input path plus 3 is not longer than MAX_PATH.
    // Three characters are for the "\*" plus NULL appended below.
    StringCchLength(argv[1], MAX_PATH, &length_of_arg);

    if (length_of_arg > (MAX_PATH - 3)) {
        _tprintf(TEXT("\nDirectory path is too long.\n"));
        return (-1);
    }

    _tprintf(TEXT("\nTarget directory is %s\n\n"), argv[1]);
```

```
    // Prepare string for use with FindFile functions. First, copy the
    // string to a buffer, then append '\*' to the directory name.
    StringCchCopy(szDir, MAX_PATH, argv[1]);
    StringCchCat(szDir, MAX_PATH, TEXT("\\*"));

    // Find the first file in the directory.
    hFind = FindFirstFile(szDir, &ffd);

    if (INVALID_HANDLE_VALUE == hFind) {
        DisplayErrorBox(TEXT("FindFirstFile"));
        return dwError;
    }

    // List all the files in the directory with some info about them.
    do {
        if (ffd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
            _tprintf(TEXT(" %s <DIR>\n"), ffd.cFileName);
        }
        else {
            filesize.LowPart = ffd.nFileSizeLow;
            filesize.HighPart = ffd.nFileSizeHigh;
            _tprintf(TEXT(" %s %ld bytes\n"), ffd.cFileName, filesize.QuadPart);
        }
    } while (FindNextFile(hFind, &ffd) != 0);

    dwError = GetLastError();
    if (dwError != ERROR_NO_MORE_FILES) {
        DisplayErrorBox(TEXT("FindFirstFile"));
    }

    FindClose(hFind);
    return dwError;
}
```

# Exemplu: listarea fișierelor din director

Listarea conținutului unui director cu câteva atribute – funcția DisplayErrorBox

```
void DisplayErrorBox(LPTSTR lpzFunction) {
    // Retrieve the system error message for the last-error code

    LPVOID lpMsgBuf;
    LPVOID lpDisplayBuf;
    DWORD dw = GetLastError();

    FormatMessage(
        FORMAT_MESSAGE_ALLOCATE_BUFFER |
        FORMAT_MESSAGE_FROM_SYSTEM |
        FORMAT_MESSAGE_IGNORE_INSERTS,
        NULL,
        dw,
        MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
        (LPTSTR)&lpMsgBuf,
        0, NULL);

    // Display the error message and clean up

    lpDisplayBuf = (LPVOID)LocalAlloc(LMEM_ZEROINIT,
        (lstrlen((LPCTSTR)lpMsgBuf) + lstrlen((LPCTSTR)lpzFunction) + 40) * sizeof(TCHAR));
    StringCchPrintf((LPTSTR)lpDisplayBuf, LocalSize(lpDisplayBuf) / sizeof(TCHAR),
        TEXT("%s failed with error %d: %s"), lpzFunction, dw, lpMsgBuf);
    MessageBox(NULL, (LPCTSTR)lpDisplayBuf, TEXT("Error"), MB_OK);

    LocalFree(lpMsgBuf);
    LocalFree(lpDisplayBuf);
}
```

# Alte attribute asociate unui fișier

Atributele de timp asociate unui fișier deschis pot fi obținute cu funcția `GetFileTime`

```
BOOL WINAPI GetFileTime(  
    HANDLE      hFile,  
    LPFILETIME  lpCreationTime,  
    LPFILETIME  lpLastAccessTime,  
    LPFILETIME  lpLastWriteTime );
```

Handle-ul fișierului sau directorului, obținut cu funcția `CreateFile`  
și având drept de acces `GENERIC_READ`

Timpul de creare

Timpul ultimului access

Timpul ultimei modificări

Timpul ca întreg pe 64 de biți - timpul scurs din 1 ianuarie 1601  
– în unități de 100 de nanosecunde ( $10^7$  unități pe secundă)

```
#include <windows.h>  
#include <tchar.h>  
#include <strsafe.h>  
  
// GetLastWriteTime - Retrieves the last-write time and converts the time to a string  
// Return value - TRUE if successful, FALSE otherwise  
// hFile - Valid file handle  
// lpszString - Pointer to buffer to receive string  
BOOL GetLastWriteTime(HANDLE hFile, LPTSTR lpszString, DWORD dwSize) {  
    FILETIME ftCreate, ftAccess, ftWrite;  
    SYSTEMTIME stUTC, stLocal;  
    DWORD dwRet;  
  
    // Retrieve the file times for the file.  
    if (!GetFileTime(hFile, &ftCreate, &ftAccess, &ftWrite)) return FALSE;  
  
    // Convert the last-write time to local time.  
    FileTimeToSystemTime(&ftWrite, &stUTC);  
    SystemTimeToTzSpecificLocalTime(NULL, &stUTC, &stLocal);  
  
    // Build a string showing the date and time.  
    dwRet = StringCchPrintf(lpszString, dwSize, TEXT("%02d/%02d/%d %02d:%02d"),  
                           stLocal.wMonth, stLocal.wDay, stLocal.wYear,  
                           stLocal.wHour, stLocal.wMinute);  
  
    if (S_OK == dwRet) return TRUE;  
    else return FALSE;  
}
```

```
int _tmain(int argc, TCHAR *argv[]) {  
    HANDLE hFile;  
    TCHAR szBuf[MAX_PATH];  
  
    if (argc != 2) {  
        printf("This sample takes a file name as a parameter\n");  
        return 0;  
    }  
    hFile = CreateFile(argv[1], GENERIC_READ, FILE_SHARE_READ, NULL,  
                      OPEN_EXISTING, 0, NULL);  
  
    if (hFile == INVALID_HANDLE_VALUE) {  
        printf("CreateFile failed with %d\n", GetLastError());  
        return 0;  
    }  
    if (GetLastWriteTime(hFile, szBuf, MAX_PATH))  
        _tprintf(TEXT("Last write time is: %s\n"), szBuf);  
  
    CloseHandle(hFile);  
}
```

`FileTimeToSystemTime` – desparte timpul în an, luna, zi ora, etc.  
`SystemTimeToTzSpecificLocalTime` – ia în considerare timpul local

# Alte attribute asociate unui fișier

## Obținerea atributelor pe baza numelui

```
DWORD WINAPI GetFileAttributes(  
    LPCTSTR lpFileName );
```

Numele fișierului sau directorului

- Returnează attributele fișierului sau INVALID\_FILE\_ATTRIBUTES în caz de eșec

Cateva valori de masca pentru attribute:

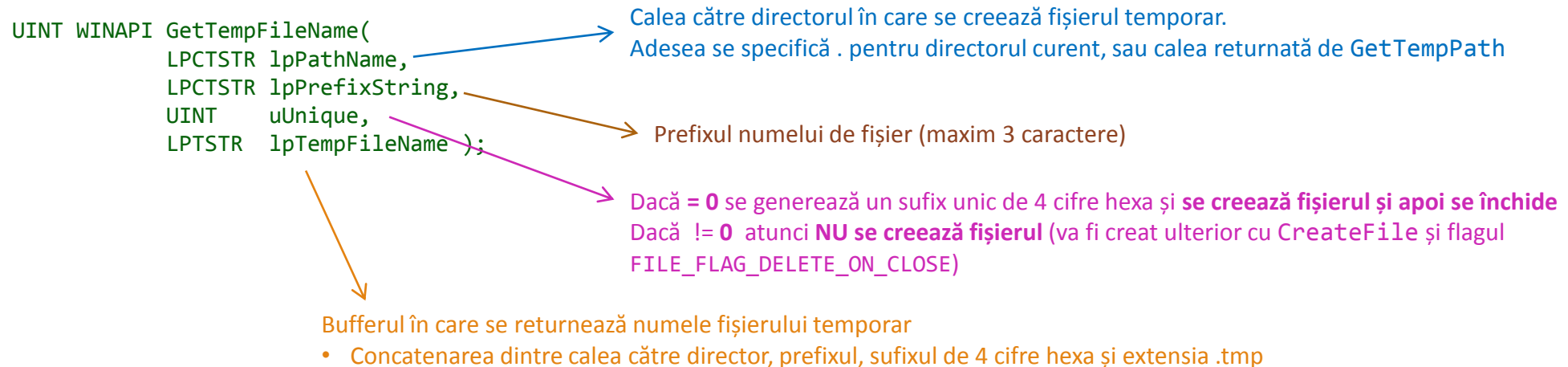
FILE\_ATTRIBUTE\_DIRECTORY  
FILE\_ATTRIBUTE\_READONLY  
FILE\_ATTRIBUTE\_TEMPORARY

```
#include <windows.h>  
#include <tchar.h>  
#include <stdio.h>  
#include <strsafe.h>  
  
void _tmain(int argc, TCHAR* argv[]) {  
    WIN32_FIND_DATA FileData;  
    HANDLE hSearch;  
    DWORD dwAttrs;  
    TCHAR szNewPath[MAX_PATH];  
    BOOL fFinished = FALSE;  
  
    if (argc != 2) {  
        _tprintf(TEXT("Usage: %s <dir>\n"), argv[0]);  
        return;  
    }  
  
    // Create a new directory.  
    if (!CreateDirectory(argv[1], NULL)) {  
        printf("CreateDirectory failed (%d)\n", GetLastError());  
        return;  
    }  
  
    // Start searching for text files in the current directory.  
    hSearch = FindFirstFile(TEXT("*.txt"), &FileData);  
    if (hSearch == INVALID_HANDLE_VALUE) {  
        printf("No text files found.\n");  
        return;  
    }  
}
```

```
// Copy each .TXT file to the new directory and change it to read only, if not already.  
while (!fFinished) {  
    StringCchPrintf(szNewPath, sizeof(szNewPath) / sizeof(szNewPath[0]),  
        TEXT("%s\\%s"), argv[1], FileData.cFileName);  
  
    if (CopyFile(FileData.cFileName, szNewPath, FALSE)) {  
        dwAttrs = GetFileAttributes(FileData.cFileName);  
        if (dwAttrs == INVALID_FILE_ATTRIBUTES) return;  
  
        if (!(dwAttrs & FILE_ATTRIBUTE_READONLY)) {  
            SetFileAttributes(szNewPath, dwAttrs | FILE_ATTRIBUTE_READONLY);  
        }  
    } else {  
        printf("Could not copy file.\n");  
        return;  
    }  
  
    if (!FindNextFile(hSearch, &FileData)) {  
        if (GetLastError() == ERROR_NO_MORE_FILES) {  
            _tprintf(TEXT("Copied *.txt to %s\n"), argv[1]);  
            fFinished = TRUE;  
        } else {  
            printf("Could not find next file.\n");  
            return;  
        }  
    }  
}  
// Close the search handle.  
FindClose(hSearch);  
}
```

# Nume de fișiere temporare

Numele fișierului temporar poate fi în orice director specificat și trebuie să fie unic



- Returnează valoarea numerică unică folosită în crearea numelui de fișier, sau 0 în caz de eșec
- lpTempFileName are forma: `<path>\<pre><uuuu>.TMP`
- Fișierele create cu GetTempFileName nu se șterg automat → programatorul trebuie să apeleze DeleteFile explicit



# Exemplu: utilizarea fișierului temporar

```
/* Open a file specified by the user and use a temporary file
   to convert the file to upper case letters. */

#include <windows.h>
#include <tchar.h>
#include <stdio.h>

#define BUFSIZE 1024

void PrintError(LPCTSTR errDesc);

int _tmain(int argc, TCHAR *argv[]) {
    HANDLE hFile = INVALID_HANDLE_VALUE;
    HANDLE hTempFile = INVALID_HANDLE_VALUE;

    BOOL fSuccess = FALSE;
    DWORD dwRetVal = 0;
    UINT uRetVal = 0;

    DWORD dwBytesRead = 0;
    DWORD dwBytesWritten = 0;

    TCHAR szTempFileName[MAX_PATH];
    TCHAR lpTempPathBuffer[MAX_PATH];
    char chBuffer[BUFSIZE];

    LPCTSTR errMsg;

    if (argc != 2) {
        _tprintf(TEXT("Usage: %s <file>\n"), argv[0]);
        return -1;
    }
}
```

```
// Opens the existing file.
hFile = CreateFile(argv[1], // file name
    GENERIC_READ,           // open for reading
    0,                       // do not share
    NULL,                   // default security
    OPEN_EXISTING,          // existing file only
    FILE_ATTRIBUTE_NORMAL,  // normal file
    NULL);                  // no template

if (hFile == INVALID_HANDLE_VALUE) {
    PrintError(TEXT("First CreateFile failed"));
    return (1);
}

// Gets the temp path env string (no guarantee it's a valid path).
dwRetVal = GetTempPath(MAX_PATH, // length of the buffer
    lpTempPathBuffer);           // buffer for path

if (dwRetVal > MAX_PATH || (dwRetVal == 0)) {
    PrintError(TEXT("GetTempPath failed"));
    if (!CloseHandle(hFile)) {
        PrintError(TEXT("CloseHandle(hFile) failed"));
        return (7);
    }
    return (2);
}
```

# Exemplu: utilizarea fișierului temporar

```
// Generates a temporary file name.
uRetVal = GetTempFileName(
    lpTempPathBuffer, // directory for tmp files
    TEXT("DEMO"),     // temp file name prefix
    0,                // create unique name
    szTempFileName);  // buffer for name
if (uRetVal == 0) {
    PrintError(TEXT("GetTempFileName failed"));
    if (!CloseHandle(hFile)) {
        PrintError(TEXT("CloseHandle(hFile) failed"));
        return (7);
    }
    return (3);
}

// Creates the new file to write to for the upper-case version.
hTempFile = CreateFile((LPTSTR)szTempFileName, // file name
    GENERIC_WRITE, // open for write
    0,             // do not share
    NULL,          // default security
    CREATE_ALWAYS, // overwrite existing
    FILE_ATTRIBUTE_NORMAL, // normal file
    NULL);         // no template

if (hTempFile == INVALID_HANDLE_VALUE) {
    PrintError(TEXT("Second CreateFile failed"));
    if (!CloseHandle(hFile)) {
        PrintError(TEXT("CloseHandle(hFile) failed"));
        return (7);
    }
    return (4);
}
```

```
// Reads BUFSIZE blocks to the buffer and converts all characters
// to upper case, then writes the buffer to the temporary file.
do
{
    if (ReadFile(hFile, chBuffer, BUFSIZE, &dwBytesRead, NULL)) {
        // Replaces lower case letters with upper case
        // in place (using the same buffer). The return
        // value is the number of replacements performed,
        // which we aren't interested in for this demo.
        CharUpperBuffA(chBuffer, dwBytesRead);

        fSuccess = WriteFile( hTempFile,
                               chBuffer,
                               dwBytesRead,
                               &dwBytesWritten,
                               NULL);

        if (!fSuccess) {
            PrintError(TEXT("WriteFile failed"));
            return (5);
        }
    }
    else {
        PrintError(TEXT("ReadFile failed"));
        return (6);
    }
    // Continues until the whole file is processed.
} while (dwBytesRead == BUFSIZE);
```

# Exemplu: utilizarea fișierului temporar

```
// The handles to the files are no longer needed, so
// they are closed prior to moving the new file.
if (!CloseHandle(hFile)) {
    PrintError(TEXT("CloseHandle(hFile) failed"));
    return (7);
}

if (!CloseHandle(hTempFile)) {
    PrintError(TEXT("CloseHandle(hTempFile) failed"));
    return (8);
}

// Moves the temporary file to the new text file, allowing for differnt
// drive letters or volume names.
fSuccess = MoveFileEx( szTempFileName,
                      TEXT("AllCaps.txt"),
                      MOVEFILE_REPLACE_EXISTING | MOVEFILE_COPY_ALLOWED);

if (!fSuccess) {
    PrintError(TEXT("MoveFileEx failed"));
    return (9);
}
else
    _tprintf(_T("All-caps version of %s written to AllCaps.txt\n"), argv[1]);
    return (0);
}
```

```
// ErrorMessage support function.
// Retrieves the system error message for GetLastError().
// Note: caller must use LocalFree() on the returned LPCTSTR buffer.
LPCTSTR ErrorMessage(DWORD error) {
    LPVOID lpMsgBuf;

    FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER
                | FORMAT_MESSAGE_FROM_SYSTEM
                | FORMAT_MESSAGE_IGNORE_INSERTS,
                NULL,
                error,
                MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
                (LPTSTR)&lpMsgBuf,
                0,
                NULL);

    return((LPCTSTR)lpMsgBuf);
}

// PrintError support function.
// Simple wrapper function for error output.
void PrintError(LPCTSTR errDesc) {
    LPCTSTR errMsg = ErrorMessage(GetLastError());
    _tprintf(TEXT("\n** ERROR ** %s: %s\n"), errDesc, errMsg);
    LocalFree((LPVOID)errMsg);
}
```

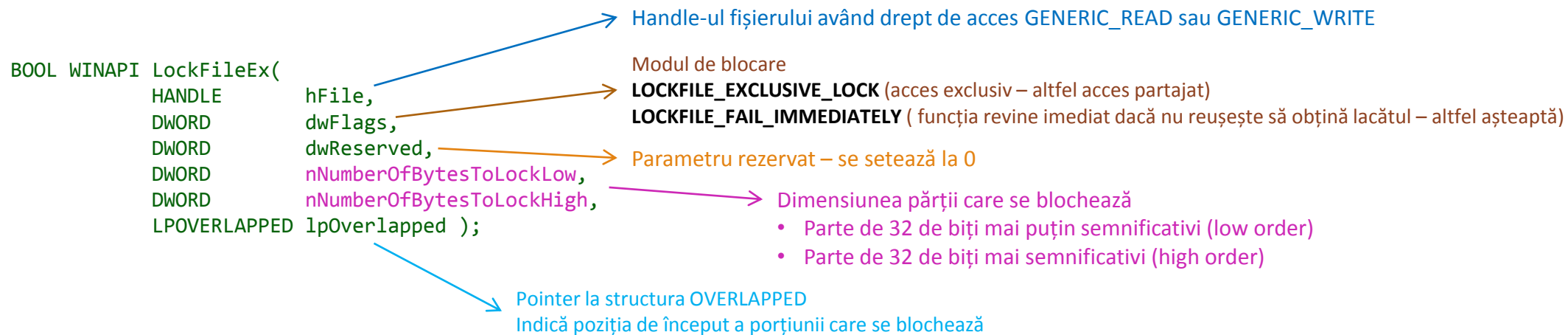
# Lacăte pentru blocarea fișierului

- O formă limitată de sincornizare a proceselor și threadurilor concurente

Windows oferă posibilitatea blocării unui fișier în întregime sau a unei părți din fișier pentru ca nici un alt proces sau thread să nu poată accesa simultan porțiunea protejată de lacăt

- Lacătele de fișier pot fi
  - doar în citire (read-only) – lacăte partajate (shared) sau
  - în citire-sciere – lacăte exclusive
- Lacătul aparține procesului care îl setează

Blocarea lacătului pe o porțiune specificată dintr-un fișier deschis pentru acces partajat (mai mulți cititori) sau acces exclusiv (un singur cititor-scriitor)



# Lacăte pentru blocarea fișierului

## Eliberarea lacătului

- Parametrii funcției `UnlockFileEx` sunt identici cu parametrii funcției `LockFileEx` dar lipsește `dwFlags`

```
BOOL WINAPI UnlockFileEx(  
    HANDLE      hFile,  
    DWORD       dwReserved,  
    DWORD       nNumberOfBytesToUnlockLow,  
    DWORD       nNumberOfBytesToUnlockHigh,  
    LPOVERLAPPED lpOverlapped );
```

Lock Request Logic		
Existing Lock	Requested Lock Type	
	Shared Lock	Exclusive Lock
None	Granted	Granted
Shared lock (one or more)	Granted	Refused
Exclusive lock	Refused	Refused

- Pentru a debloca un lacăt blocat anterior trebuie să se specifice exact aceleași valori la parametri – aceeași poziție de început și dimensiune, etc.
- Se pot bloca porțiuni care să depășească dimensiunea fișierului – de ex când se dorește extinderea fișierului
- Lacătele nu se moștenesc de procesele nou create
- Nu se poate folosi parametru `LARGE_INTEGER`, poziția de început și dimensiunea porțiunii se specifică separat ca valori pe 32 de biți

Atenție: blocarea poate duce la înfometare

- Unele procese/threaduri nu primesc resursele după care așteaptă  
→ afectează negativ performanța

Este important să deblocăm lacătul imediat ce blocarea nu mai este necesară

Locks and I/O Operation		
Existing Lock	I/O Operation	
	Read	Write
None	Succeeds	Succeeds
Shared lock (one or more)	Succeeds. It is not necessary for the calling process to own a lock on the file region.	Fails
Exclusive lock	Succeeds if the calling process owns the lock. Fails otherwise.	Succeeds if the calling process owns the lock. Fails otherwise.

# Materiale de studiu

---

- Johnson HART, Windows System Programming, 4th edition, Addison Wesley, 2010
  - Capitolul 2 si prima parte din capitolul 3
  - Exemplele incluse sunt din cartea de mai sus – vezi arhiva de pe moodle: WSP4\_Examples.zip