Лабораторна робота №7

ТЕМА: ПРОГРАМУВАННЯ В WIN32 API В ОС WINDOWS

МЕТА: Навчитися організовувати програми та коректно розв'язувати складні задачі, пов'язані зі створенням об’єктів файлів та маніпулюванням ними, застосовуючи системні служби.

ЗНАТИ: Принципи програмування в Win32 API в ОС Windows.

ВМІТИ: Застосовувати API-функції createFile, closeHandle, DeleteFile, writeFile, FlushFileBuffers, ReadFile, CopyFile, MoveFile, ReplaceFile, SetFilePointer, setFilePointerEx, GetFileAttributes, SetFileAttributes, GetFiieSize, GetFilesizeEx, setEndofFile, LockFile, UnlockFile, GetFileInformationByHandle, FileTimeToSystemTime, GetFileType, GetBinaryType для роботи з файлами в ОС Windows.

ТЕОРЕТИЧНІ ВІДОМОСТІ

Дескриптор-(handle)-це 32 розрядне ціле число без знака, яке унікальним чином ідентифікує деякий об’єкт в системі, наприклад, растрове зображення, будь-який пристрій введення-виведення.

Робота з файлами в Windows

1.1 Іменування файлів у Windows

В операційних системах Windows повне ім'я файлу представляється рядком, який закінчується порожнім символом. Причому довжина такого рядка не може перевищувати max_ратн символів. Файлові системи FAT32 і NTFS підтримують повні імена файлів довжиною до 255 символів. Такі імена називаються довгими. Файлова система FAT 16, яка використовувалася в операційній системі MS-DOS, підтримує файлові імена завдовжки до 8

символів плюс 3 символи на розширення файлу. Повне ім'я файлу складається з компонент, кожна з яких поділяється символом \ (зворотна нахилена). Кожна не остання компонента повного імені файлу задає ім'я каталогу, в якому знаходиться файл, а остання компонента задає ім'я самого файлу. Тому повне ім'я файлу також часто називають шляхом до файлу, тому що повне ім'я файлу фактично описує шлях по дереву каталогів до його листа, який і представляє сам файл. При формуванні шляху до файлу потрібно дотримуватися певних правил, які перераховані нижче:

• імена каталогів та файлів не повинні містити символів, ASCII-коди яких знаходяться в діапазоні від 0 до 31 (це службові символи);

• імена каталогів та файлів не повинні містити символи <,>,:,",/, \ і |;

• імена каталогів та файлів можуть містити символи з розширеної множини, яке включає символи з кодами від 128 до 255;

• для позначення поточного каталогу в якості компоненти шляху використовується символ «.» (крапка);

• для позначення батьківського каталогу для поточного каталогу в якості компоненти шляху використовуються символи «. .» (дві крапки);

• у якості компонент шляху не можна використовувати імена пристроїв, як, наприклад, aux, con, lpt1 і prn.

Крім того, імена файлів нечутливі до регістрів клавіатури. Тобто файлова система не розрізняє імена файлів, які відрізняються тільки прописним або заголовним написанням літер. Наприклад, імена Demo, demo і DEMO є однаковими.

1.2. Створення і відкриття файлів

Для створення нових або відкриття вже існуючих файлів використовується **функція CreateFile**, яка має наступний прототип:

```
HANDLE CreateFile (
LPCTSTR lpFileName, / / ім'я файлу
DWORD dwDesiredAccess, / / спосіб доступу
```

```
DWORD dwShareMode, / / режими спільного використання
LPSECUTITY_ATTRIBUTES IpSecurutyAttributes, / / атрибути захисту
DWORD dwCreationDisposition, / / створення або відкриття файлу
DWORD dwFlagsAndAttributes, / / прапори і атрибути
HANDLE hTemplateFile / / файл атрибутів
);
```

У разі успішного завершення функція повертає дескриптор створеного або відкритого файлу, а в разі невдачі - значення invalid_handle_value. У параметрі lpFileName задається покажчик на символьний рядок, який містить повне ім'я створюваного файлу або файлу, що відкривається. Якщо повне ім'я файлу не вказано, то файл із заданим ім'ям створюється або шукається у поточному каталозі. Параметр dwDesiredAccess задає спосіб доступу до файлу і може приймати будь-яку комбінацію наступних значень:

o - додаток може тільки визначати атрибути пристрою;

geNeric_read - допускається тільки читання даних з файлу;

generic_write - допускається тільки запис даних у файл.

Це загальні або родові режими доступу до файлу. Існують також і інші режими, значення яких залежать від доступу, заданого при визначенні атрибутів захисту файлу. Будемо вважати, що файл отримує атрибути захисту за замовчуванням, що дозволяє виконувати над ним усі існуючі операції.

Параметр dwShareMode задає режими спільного доступу до файлу. Якщо значення цього параметра дорівнює нулю, то файл не може використовуватися для спільного доступу. Інакше параметр dwShareMode може приймати будь-яку комбінацію наступних значень:

file_share_read - файл може використовуватися тільки для спільного читання кількома програмами;

file_share_write - файл може використовуватися тільки для спільного запису кількома програмами;

FILE_SHARE_DELETE - файл може використовуватися декількома програмами за умови, що кожна з них має дозвіл на видалення цього файлу.

Відзначимо, що останнє значення може використовуватися тільки в операційних системах Windows NT/2000.

Параметр ipsecurutyAttributes повинен задавати атрибути захисту файлу. Поки що цей параметр будемо встановлювати в null. Це означає, що атрибути захисту файлу встановлюються за замовчуванням, тобто дескриптор файлу не є успадковуваним і файл відкритий для доступу всім користувачам.

Параметр dwCreationDisposition задає дії, які потрібно виконати при створенні або відкритті файлу. Цей параметр може приймати одне з наступних значень:

create_new - створити новий файл, якщо файл із заданим ім'ям вже існує, то функція закінчується невдачею;

create_always - створити новий файл, якщо файл із заданим ім'ям вже існує, то він знищується і створюється новий файл;

open_existing - відкрити існуючий файл, якщо файл із заданим ім'ям не існує, то функція закінчується невдачею;

open_always - відкрити файл, якщо файл із заданим ім'ям не існує, то створюється новий файл;

truncate_existing - відкрити файл і знищити його вміст, якщо файл із заданим ім'ям не існує, то функція закінчується невдачею.

Відзначимо, що в останньому випадку викликаючий процес повинен мати права запису у файл, тобто в параметрі dwDesiredAccess повинен бути встановлений прапор GENERIC_WRITE.

У параметрі dwFiagsAndAttributes повинні бути задані прапори і атрибути створюваного або відкриваємого файлу. Атрибути файлу управляють його властивостями і можуть приймати будь-яку комбінацію наступних значень:

file_attribute_archive - архівний файл, який містить службову інформацію;

file_attribute_encrypted - зашифрований файл;

FILE_ATTRIBUTE_HIDDEN - прихований файл;

FILE_ATTRIBUTE_NORMAL - звичайний файл, який не має інших атрибутів;

FILE_ATTRIBUTE_NOT_CONTENT_INDEXED - вміст файлу не індексований;

file_attribute_offline - файл знаходиться в допоміжній пам'яті;

FILE_ATTRIBUTE_READONLY - файл можна лише читати;

file_attribute_system - файл використовується операційною системою;

file_attribute_temporary - файл використовується для тимчасового зберігання даних.

Зробимо декілька зауважень щодо деяких атрибутів файлів. Спочатку відзначимо, що зашифровані файли не можуть мати також атрибут file_attribute_system. Тепер зауважимо, що атрибут file_attribute_normal повинен використовуватися тільки один, а не в комбінації з іншими атрибутами.

Крім того, у параметрі dwFlagsAndAttributes може бути встановлена будь-яка комбінація наступних керуючих прапорів:

file_flag_write_through - запис даних безпосередньо на диск, не використовуючи кешування;

FILE_FLAG_OVERLAPPED - забезпечується асинхронне виконання операцій читання і запису;

file_flag_no_buffering - не використовувати буферизацію при доступі до файлу;

file_flag_random_access - програма передбачає вибирати записи з файлу випадковим чином;

file_flag_sequential_scan - програма буде сканувати файл послідовно;

file_flag_delete_on_close - файл буде видалено після того, як всі дескриптори цього файлу будуть закриті;

FILE_FLAG_BACKUP_SEMANTICS - резервний файл;

FILE_FLAG_POSIX_SEMANTICS - доступ до файлу буде здійснюватися за стандартом POSIX;

file_flag_open_reparse_point - при доступі до файлу використовується системний фільтр;

FILE_FLAG_OPEN_NO_RECALL - при використанні ієрархічної системи управління пам'яттю файл не повинен читатися в оперативну пам'ять, а ставати на нижньому рівні ієрархії.

Відзначимо, що прапор file_flag_backup_semantics може використовуватися тільки в операційних системах Windows NT/2000.

Параметр MempiateFiie використовується при створенні файлу, атрибути якого повинні відповідати атрибутам раніше створеного файлу. У цьому випадку параметр hTempiateFile повинен містити дескриптор файлу, атрибути якого копіюються в атрибути створюваного файлу.

В інших параметрах, окрім параметра dwCreationDisposition, може бути встановлено значення 0. На платформах Windows 98/ME в цьому параметрі повинно бути встановлено значення null.

На завершення цього розділу скажімо, що приклад використання функції createFile наведений у розділі 1.4.

1.3. Закриття та видалення файлів

Для закриття доступу до файлу, як і для закриття доступу до будь-якого іншого об'єкта ядра, використовується **функція closeHandle**, єдиним параметром якої є дескриптор відкритого файлу.

Для фізичного видалення файлу з диска використовується **функція DeleteFile**, яка має наступний прототип:

```
     bool DeieteFile (
lpctstr lpFileName / / ім'я файлу
);
```

Єдиний параметр lpFileName є покажчиком на рядок, що вказує повний шлях до файлу. При успішному завершенні функція повертає ненульове значення, а при невдачі - false.

У лістингу 1.1 приведена програма, яка видаляє файл з ім'ям demo_file.dat, який розташований в кореневому каталозі на диску C:.

Лістинг 1.1 Видалення файлу

```
#include <windows.h>
#include <iostream.h>

int main()
{
  // видаляємо файл
  if(!DeleteFile("C:\\demo_file.dat"))
  {
    cerr << "Delete file failed." << endl
      << "The last error code: " << GetLastError() << endl;
    cout << "Press any key to finish.";
    cin.get();
    return 0;
  }
  cout << "The file is deleted." << endl;

  return 0;
}
```

1.4. Запис даних у файл

Для запису даних у файл, використовується **функція writeFile**, причому відзначимо, що ця функція може використовуватися як для синхронного, так і для асинхронного запису даних. У цьому розділі буде розглянуто тільки синхронний запис даних у файл. У цьому випадку дані записуються в файл послідовно - байт за байтом, і покажчик файлу пересувається по мірі запису даних на нову позицію.

Тепер розглянемо функцію WriteFile більш докладно. Ця функція має наступний прототип

```
BOOL   WriteFile(
handle     hFile,          // дескриптор файлу
LPCVOID    lpBuffer, // покажчик на буфер даних
```

```
    DWORD       nNumberOfBytesToWrite,      // кількість записуваних
байтів LPDWORD   lpNumberOfBytesWritten,      // кількість записаних
байтів LPOVERLAPPED    lpOverlapped      // використовується при
асинхронному записі
    ) ;
```

У разі успішного завершення функція повертає ненульове значення, а в разі невдачі - значення false.

Параметр hFile повинен містити дескриптор файлу, причому файл повинен бути відкритий у режимі запису.

Параметр lpBuffer повинен вказувати на область пам'яті, в яку будуть читатися дані.

Параметр nNumberOfBytesToWrite повинен містити кількість байт, які передбачається записати у файл за допомогою виклику функції WriteFile.

Параметр lpNumberOfBytesWritten повинен містити адресу пам'яті, до якої функція WriteFile помістить кількість фактично записаних байт. При роботі на платформі Windows 98 цей параметр повинен мати значення, відмінне від null. При виконанні функції WriteFile операційна система записує за цією адресою нуль, перш ніж виконати запис даних у файл.

Так як у прикладі не буде розглядатися асинхронний запис даних у файл, то параметр lpOverlapped буде встановлюватися в null. У лістингу 1.2 приведена програма, яка створює файл і записує в нього послідовність цілих чисел.

Лістинг 1.2 Створення файлу і запис до нього даних

```
#include <windows.h>
#include <iostream.h>

int main()
{
  HANDLE hFile;

  // створюємо файл для запису даних
  hFile = CreateFile(
```

```
    "C:\\demo_file.dat",    // ім'я файлу
    GENERIC_WRITE,          // запис в файл
    0,                      // монопольний доступ до файлу
    NULL,                   // захисту немає
    CREATE_NEW,             // створюємо новий файл
    FILE_ATTRIBUTE_NORMAL, // звичайний файл
    NULL                    // шаблону немає
);
  // перевіряємо на успішне створення
if (hFile == INVALID_HANDLE_VALUE)
  {
    cerr << "Create file failed." << endl
      << "The last error code: " << GetLastError() << endl;
    cout << "Press any key to finish.";
    cin.get();
    return 0;
  }

  // записуємо дані до файлу
  for (int i = 0; i < 10; ++i)
  {
    DWORD dwBytesWrite;

    if (!WriteFile(
        hFile,           // дескриптор файлу
     &i,                  // адреса буфера, звідки йде запис
     sizeof(i),           // кількість записуваних байтів
     &dwBytesWrite,       // кількість записаних байтів
        (LPOVERLAPPED)NULL   // запис синхронний
      ))
    {
      cerr << "Write file failed." << endl
        << "The last error code: " << GetLastError() << endl;
      CloseHandle(hFile);
      cout << "Press any key to finish.";
      cin.get();
      return 0;
    }
```

```
  }
  // закриваємо дескриптор файла
  CloseHandle(hFile);

  cout << "The file is created and written." << endl;

  return 0;
}
```

1.5. Звільнення буферів файлу

Часто кілька додатків мають спільний доступ до одного і того ж файлу. При цьому може вимагатися, щоб додаток, який читає дані з файлу, мав доступ до останньої версії цього файлу. Для цього необхідно, щоб додаток, який змінює вміст файлу, фіксував зміну файлу після обробки потрібних записів. Так як не виключена можливість того, що останні оброблені записи зберігаються в буфері файлу, то в цьому випадку необхідно звільнити буфер від записів.

Виконати цю операцію можна за допомогою **функції** яка має наступний прототип:

```
FlushFileBuffers,
BOOL   FlushFileBuffers(
HANDLE   hFile       // дескриптор файла
) ;
```

У разі вдалого завершення ця функція повертає ненульове значення, а в разі невдачі - false.

У лістингу 1.3 Наведено програму, в якій функція FlushFileBuffers використовується для скидання даних з буферів в файл після запису половини файлу.

```
#include <windows.h>
#include <iostream.h>
int main()
{
```

```
HANDLE hFile;

// створюємо файл для запису даних
hFile = CreateFile(
  "C:\\demo_file.dat",     // ім'я файлу
  GENERIC_WRITE,           // запис до файлу
  FILE_SHARE_READ,         // читання файлу, що розділяється
  NULL,                    // захисту немає
  CREATE_ALWAYS,           // створюємо новий файл
  FILE_ATTRIBUTE_NORMAL,   // звичайний файл
  NULL                     // шаблона немає
);
// перевіряєма на успішне створення
if (hFile == INVALID_HANDLE_VALUE)
{
  cerr << "Create file failed." << endl
    << "The last error code: " << GetLastError() << endl;
  cout << "Press any key to finish.";
  cin.get();
  return 0;
}
  // пишемо дані до файлу
for (int i = 0; i < 10; ++i)
{
  DWORD dwBytesWrite;
  if (!WriteFile(
      hFile,               // дескриптор файлу
      &i,                  // адреса буфера, звідки йде запис
      sizeof(i),           // кількість записуваних байтів
   &dwBytesWrite,       // кількість записаних байтів
   (LPOVERLAPPED)NULL   // запис синхронний
    ))
  {
    cerr << "Write file failed." << endl
      << "The last error code: " << GetLastError() << endl;
    CloseHandle(hFile);
    cout << "Press any key to finish.";
    cin.get();
```

```
      return 0;
    }
    // якщо досягли середини файлу, то звільняємо буфер
    if(i == 5)
    {
      if(!FlushFileBuffers(hFile))
      {
        cerr << "Flush file buffers failed." << endl
          << "The last error code: " << GetLastError() << endl;
        CloseHandle(hFile);
        cout << "Press any key to finish.";
        cin.get();
        return 0;
      }
      // тепер можна проглянути вміст файлу
      cout << "A half of the file is written." << endl
        << "Press any key to continue.";
      cin.get();
    }
  }
  // закриваємо дескриптор файлу
  CloseHandle(hFile);
  cout << "The file is created and written." << endl;
  return 0;
}
```

Можна скасувати режим буферизації файлу, встановивши прапор file_flag_no_buffering у параметрі dwFiagsAndAttributes функції createFile. Однак у цьому випадку довжина записуваних або зчитуваних даних з файлу повинна бути кратна розміру сектора. Наприклад, в операційних системах Windows довжина сектора дорівнює 512 байт.

1.6. Читання даних з файлу

Для читання даних з файлу служить **функція ReadFile**, яка може використовуватися як для синхронного, так і асинхронного читання даних. Розглянемо лише синхронне читання даних з файлу. У цьому випадку дані читаються з файлу послідовно байт за байтом, і покажчик файлу

пересувається по мірі читання даних на нову позицію. Тепер більш детально розглянемо функцію ReadFile, яка має наступний прототип:

```
      BOOL   ReadFile(
      HANDLE     hFile,          // дескриптор файлу
      LPVOID     lpBuffer,  // покажчик на буфер даних
      DWORD       nNumberOfBytesToRead,      // кількість читаних
байтів LPDWORD   lpNumberOfBytesRead,        // кількість прочитаних
байтів LPOVERLAPPED   lpOverlapped    // використовується при
асинхронному записі
      );
```

У разі успішного завершення функція повертає ненульове значення, а в разі невдачі - значення false.

Параметр hFile повинен містити дескриптор файлу, причому файл повинен бути відкритий в режимі читання.

Параметр lpBuffer повинен вказувати на область пам'яті, з якої будуть читатися дані.

Параметр nNumberOfBytesToRead повинен містити кількість байт, які передбачається читати з файлу за допомогою виклику функції ReadFile.

Параметр lpNumberOfBytesRead повинен містити адресу пам'яті, до якої функція ReadFile помістить кількість фактично прочитаних з файлу байтів. Як і в випадку функції writeFile, при роботі на платформі Windows 98 цей параметр повинен мати значення, відмінне від null. При виконанні функції ReadFile операційна система записує за цією адресою 0, перш ніж виконати читання даних з файлу.Так як ми не розглядатимемо асинхронне читання даних з файлу, то параметр lpoveriapped буде встановлюватися в null.

<u>У лістингу 1.4 приведена програма, яка читає дані з файлу, створеного програмою з лістингу 1.2, і виводить ці дані на консоль</u>

```
#Include <windows.h>
#Include <iostream.h>
int main ()
{
  HANDLE  hFile;
```

```
// Відкриваємо файл для читання
hFile = CreateFile(
  z,                          // ім'я файлу
  GENERIC_READ,               // читання з файлу
  0,                          // монопольний доступ до файлу
  NULL,                       // захисту немає
  OPEN_EXISTING,              // відкриваємо існуючий файл
  FILE_ATTRIBUTE_NORMAL,      // звичайний файл
  NULL                        // шаблону немає
);
// Перевіряємо на успішне відкриття
if (hFile == INVALID_HANDLE_VALUE)
{
  cerr << "Create file failed." << endl
    << "The last error code: " << GetLastError() << endl;
  cout << "Press any key to finish.";
  cin.get();
  return 0;
}
// Читаємо дані з файлу
for (;;)
{
  DWORD  dwBytesRead;
  int    n;
  // Читаємо один запис
  if (!ReadFile(
         hFile,               // дескриптор файлу
        & N,                  // адреса буфера, куди читаємо дані
        sizeof (n),           // кількість читаних байтів
        & DwBytesRead,        // кількість прочитаних байтів
        (LPOVERLAPPED) NULL   // читання синхронне
    ))
  {
    cerr << "Read file failed." << endl
      << "The last error code: " << GetLastError() << endl;
    CloseHandle(hFile);
    cout << "Press any key to finish.";
    cin.get();
```

```
      return 0;
    }
    // Перевіряємо на кінець файлу
    if (dwBytesRead == 0)
      // Якщо так, то виходимо з циклу
      break;
     else
      // Інакше виводимо запис на консоль
      cout << n << ' ';
  }
  cout << endl;
  // Закриваємо дескриптор файлу
  CloseHandle(hFile);
  cout << "The file is opened and read." << endl;
  return 0;
     }
```

У зв'язку з цією програмою відзначимо обробку кінця файлу. При досягненні кінця файлу, а також запит на читання запису функція ReadFile повертає ненульове значення і при цьому встановлює значення кількості прочитаних байтів в 0.

1.7. Копіювання файлу

Для копіювання файлів використовується **функція CopyFile**, яка має наступний прототип:

```
BOOL CopyFile (
LPCTSTR IpExistingFileName, / / ім'я наявного файлу
LPCTSTR lpNewFileName, / / ім'я нового файлу
bool bFaillfExists / / дії в разі існування файлу
);
```

У разі успішного завершення функція повертає ненульове значення, а в разі невдачі - значення false. При цьому зазначимо, що функція copyFile копіює для нового файлу також і атрибути доступу старого файлу, але атрибути безпеки не копіюються.

Параметри цієї функції мають наступне призначення.

Параметр lpExistingFiieName повинен вказувати на рядок, що містить ім'я копійованого файлу.

Параметр lpNewFileName повинен вказувати на рядок з ім'ям файлу, у який буде копіюватися існуючий файл. При цьому зазначимо, що новий файл створюється самою функцією CopyFile.

Параметр bFaillfExists визначає дії, які потрібно здійснити в разі, якщо файл, у який виконується копіювання, вже існує. Якщо значення цього параметра дорівнює false, то функція перезаписує існуючий файл. Якщо ж значення цього параметра дорівнює true, то виконання функції в цьому випадку закінчується невдачею.

У лістингу 1.5 приведена програма, яка виконує копіювання файлу, використовуючи функцію CopyFile.

Лістинг 1.5 Копіювання файлу

```
# Include <windows.h>
# Include <iostream.h>

int main ()
{     char qqq[20];
  // Копіюємо файл
      cout<< " Введіть куди копіювати(C:\\...)"<<endl;
      gets(qqq);
  if(!CopyFile(qq, qqq, FALSE))
  {
    cerr << "Copy file failed." << endl
      << "The last error code: " << GetLastError() << endl;
    cout << "Press any key to finish.";
    cin.get();
    return 0;
  }

  cout << "The file is copied." << endl;

  return 0;
}
```

В операційних системах сімейства Windows NT (4.0 \ 2000 \ XP) для копіювання файлів може використовуватися функція copyFiieEx, що надає більш широкі можливості щодо копіювання файлів.

1.8. Переміщення файлу

Тепер розберемо **функцію MoveFile**, яка служить для переміщення файлів. Переміщення файлу відрізняється від копіювання файлу тільки тим, що старий файл після його переміщення видаляється.

Функція MoveFile має наступний прототип:

```
BOOL MoveFile (
LPCTSTR IpExistingFileName, / / ім'я наявного файлу
LPCTSTR lpNewFileName / / ім'я нового файлу
);
```

У разі успішного завершення функція повертає ненульове значення, а в разі невдачі - значення false. Відзначимо, що функція MoveFile зберігає всі атрибути переміщуваного файлу. Параметри цієї функції мають наступне призначення.

Параметр lpExistingFileName повинен вказувати на рядок, що містить ім'я переміщуваного файлу.

Параметр lpNewFileName повинен вказувати на рядок з ім'ям файлу, у який буде переміщатися існуючий файл. При цьому зазначимо, що новий файл створюється самою функцією MoveFile.

У лістингу 1.6 приведена програма, яка виконує переміщення файлу, використовуючи функцію MoveFile.

Лістинг 1.6. Переміщення файлу

```
#Include <windows.h>
#Include <iostream.h>
int main ()
{     char zzz[20];
```

```
// Переміщаємо файл
    cout<< " Введіть куди перемістити(C:\\...)"<<endl;
    gets(zzz);
if(!MoveFile(zz, zzz))
{
  cerr << "Move file failed." << endl
    << "The last error code: " << GetLastError() << endl;
  cout << "Press any key to finish.";
  cin.get();
  return 0;
}

cout << "The file is moved." << endl;

return 0;
    }
```

В операційних системах сімейства Windows NT (4.0 \ 2000 \ XP) для переміщення файлів може також використовуватися функція MoveFiieEx, в якій можна задавати режими переміщення файлів.

1.9. Заміщення файлу

У Windows 2000 і Windows XP визначена **функція ReplaceFile**, яка призначена для заміщення файлів. Істотна відмінність цієї функції від функцій CopyFile і MoveFile полягає в тому, що вона копіює в заміщуваний файл не лише атрибути доступу, але також і атрибути безпеки файлу-заступника. Відзначимо також, що функція RepiaceFile працює тільки з файлами, які знаходяться на одному томі (логічному диску). Функція RepiaceFile має наступний прототип:

```
bool RepiaceFile (
LPCTSTR lpReplacedFileName, / / ім'я файлу, що заміщається
LPCTSTR lpReplacementFileName, / / ім'я файлу-заступника
LPCTSTR lpBackupFileName, / / ім'я резервної копії файла-заступника
DWORD dwReplaceFlags, / / опції заміщення
LPVOID IpExclude, / / не використовується
```

```
lpvoid lpReserved / / не використовується
);
```

Функція заміщення файлів у разі успішного завершення повертає ненульове значення, а в разі невдачі - false. Коротко опишемо призначення параметрів цієї функції.

Параметр lpReplacedFileName повинен вказувати на рядок, який містить ім'я заміщаємого файлу. При цьому зазначимо, що функція RepiaceFile може як створювати новий файл, так і використовувати вже існуючий. Параметр lpReplacementFileName повинен вказувати на рядок, що містить ім'я файлу-заступника. Цей файл замінить файл, на який вказує параметр lpReplacedFileName. При цьому зазначимо, що сам файл-заступник видаляється з диска.

У параметрі lpBackupFiieName встановлюється адреса рядка, яка містить ім'я файлу, який містить резервну копію файлу-заступника. Якщо цей параметр встановлений в null, то резервна копія файлу-заступника не створюється.

У параметрі dwRepiaceFiags встановлюються прапори, які вказують режими заміщення файлу. Можна встановити будь-яку комбінацію наступних прапорів:

REPLACEFILE_WRITE_THROUGH - звільнити буфери перед виходом з функції;
REPLACEFILE_IGNORE_MERGE_ERRORS - ігнорувати помилки при копіюванні даних.

Параметри lpExciude і lpReserved не використовуються. Тому вони повинні бути встановлені в null.

У лістингу 1.7 приведена програма, в якій показано приклад використання функції RepiaceFile для заміщення файлу.

Лістинг 1.7. Заміщення файлу

```
# Define _WIN32_WINNT 0x0500
# Include <windows.h>
# Include <iostream>
using namespace std;
```

```
int main ()
{
  / / Переміщаємо файл
  if (! ReplaceFile (
    "C: \ \ demo_file.dat", / / ім'я файлу, що заміщається
    "C: \ \ new_file.dat", / / ім'я файлу- заступника
    "C: \ \ back_file.dat", / / ім'я резервного файлу
    REPLACEFILE_WRITE_THROUGH, / / звільнити буфери
    NULL, NULL / / не використовуються
    ))
  {
    cerr <<"Replace file failed." <<Endl
      <<"The last error code:" <<GetLastError () <<endl;
    cout <<"Press any key to finish.";
    cin.get ();
    return 0;
  }

  cout <<"The file is replaced." <<Endl;

  return 0;
}
```

1.10. Робота з покажчиком позиції файлу

Перш ніж розбиратися з функцією для роботи з покажчиком позиції файлу, розглянемо формат самого покажчика позиції файлу. **Покажчик позиції файлу** складається з двох значень типу dword, які будуть називатися старша і молодша частина покажчика позиції файлу відповідно. Отже, покажчик позиції має довжину в 64 біта. Якщо довжина файлу не перевищує двох гігабайт без двох байт, тобто 231 - 2 байти, то в покажчику позиції використовується тільки його молодша частина, яка розглядається як ціле число зі знаком, але при цьому старша частина покажчика позиції повинна бути встановлена в null .

Для роботи з покажчиком позиції файлу служить **функція SetFilePointer**, яка має наступний прототип:

```
DWORD SetFilePointer (
HANDLE hFile, / / дескриптор файлу
LONG IDistanceToMove, / / молодша частина зсуву покажчика в байтах
PLONG lpDistanceToMoveHigh, / / вказівник на старшу частину зсуву
/ / Покажчика в байтах
DWORD dwMoveMethod / / початкова точка зсуву
);
```

У разі вдалого завершення ця функція повертає молодшу частину нової позиції покажчика файлу, а за адресою, заданою параметром lpDistanceToMoveHigh, записує старшу частину нової позиції покажчика файлу. Якщо функція встановлює старшу частину покажчика позиції в null, то молодша частина покажчика позиції представлена позитивним цілим числом. У випадку невдалого завершення функція SetFilePointer повертає значення -1 і при цьому встановлює значення параметра lpDistanceToMoveHigh в null. Якщо ж значення цього параметра не встановлено в null, то повернене значення -1 може бути і дійсною молодшою частиною покажчика позиції. У цьому випадку потрібно перевірити код останньої помилки, який повертає функція GetLastError. Якщо цей код дорівнює no_error, то помилки немає, а в іншому випадку виконання функції SetFilePointer завершиться невдачею. Призначення параметрів функції SetFilePointer:

Параметр hFile повинен містити дескриптор файлу, причому сам файл повинен бути відкритий в режимі читання або запису.

Параметр lDistanceToMove повинен містити молодшу частину зсуву для покажчика позиції файлу. Якщо значення параметра lpDistanceToMoveHigh встановлено в null, то значення цього параметра розглядається як ціле число зі знаком. У разі позитивного числа функція виконує зсув вказівника вперед на задану кількість байт, а в разі негативного числа виконується зсув тому.

Параметр lpDistanceToMoveHigh повинен містити адресу старшої частини зсуву для покажчика позиції файлу. Старша і молодша частини покажчика позиції розглядаються як ціле число зі знаком. Якщо значення цього параметра дорівнює null, то зсув задається тільки молодшою частиною. Відзначимо, що в операційній системі Windows 98 значення цього параметра може дорівнювати тільки одному з наступних значень: null, 0 і 1.

Параметр dwMoveMethod задає початкову точку, від якої виконується зсув вказівника позиції. Цей параметр може приймати тільки одне з наступних значень:

file_begin - зсув від початку файлу;

file_current - зсув від поточної позиції файлу;

file_end - зсув від кінця файлу.

У лістингу 1.8 приведена програма, яка читає запис файлу, попередньо встановивши на цей запис покажчик. До речі, таке читання записів файлу називається прямим доступом до файлу. У загальному випадку прямий доступ до файлу має на увазі читання запису із заданим значенням ключа, який визначається вмістом одного або декількох полів запису. Правда, в цьому випадку потрібно знати залежність покажчика позиції файлу від значення ключа запису.

Лістинг 1.8. Установка покажчика позиції файлу за допомогою функції `SetFilePoixiter`

```
# Include <windows.h>
# Include <iostream.h>

int main ()
{
  HANDLE hFile; / / дескриптор файлу
  long n; / / для номера запису
  long p; / / для покажчика позиції
  DWORD dwBytesRead; / / кількість прочитаних байтів
  int m; / / прочитане число
```

```
/ / Відкриваємо файл для читання
hFile = CreateFile (
  "C: \ \ demo_file.dat", / / ім'я файлу
  GENERIC_READ, / / читання з файлу
  0, / / монопольний доступ до файлу
  NULL, / / захисту немає
  OPEN_EXISTING, / / відкриваємо існуючий файл
  FILE_ATTRIBUTE_NORMAL, / / звичайний файл
  NULL / / шаблону немає
);
/ / Перевіряємо на успішне відкриття
if (hFile == INVALID_HANDLE_VALUE)
{
  cerr <<"Create file failed." <<Endl
    <<"The last error code:" <<GetLastError () <<endl;
  cout <<"Press any key to finish.";
  cin.get ();
  return 0;
}

/ / Вводимо номер потрібного запису
cout <<"Input a number from 0 to 9:";
cin>> n;
/ / Зрушує покажчик позиції файлу
p = SetFilePointer (hFile, n * sizeof (int), NULL, FILE_BEGIN);
if (p == -1)
{
  cerr <<"Set file pointer failed." <<Endl
    <<"The last error code:" <<GetLastError () <<endl;
  CloseHandle (hFile);
  cout <<"Press any key to finish.";
  cin.get ();
  return 0;
}
/ / Виводимо на консоль значення покажчика позиції файлу
cout <<"File pointer:" <<p <<endl;
/ / Читаємо дані з файлу
```

```
  if (! ReadFile (
      hFile, / / дескриптор файлу
      & M, / / адреса буфера, куди читаємо дані
      sizeof (m), / / кількість читаних байтів
      & DwBytesRead, / / кількість прочитаних байтів
      (LPOVERLAPPED) NULL / / читання синхронне
    ))
  {
    cerr <<"Read file failed." <<Endl
      <<"The last error code:" <<GetLastError () <<endl;
    CloseHandle (hFile);
    cout <<"Press any key to finish.";
    cin.get ();
    return 0;
  }
  / / Виводимо прочитане число на консоль
  cout <<"The read number:" <<m <<endl;
  / / Закриваємо дескриптор файлу
  CloseHandle (hFile);
  return 0;
}
```

За допомогою функції setFilePointer можна також визначити поточний стан покажчика позиції файлу. Для цього потрібно просто зрушити покажчик файлу від поточної позиції на нульову кількість байт. У результаті функція SetFilePointer поверне поточний стан покажчика позиції файлу.

Починаючи з операційної системи Windows 2000, для роботи з покажчиком позиції файлу можна використовувати **функцію setFiiePointerEx**, яка більш проста у використанні і має наступний прототип:

```
     bool SetFiiePointerEx (
HANDLE hFile, / / дескриптор файлу
LARGE_INTEGER liDistanceToMove, / / зрушення в байтах
PLARGE_INTEGER lpNewFilePointer, / / новий вказівник позиції файлу
DWORD dwMoveMethod / / початкова точка зсуву
);
```

У разі вдалого завершення ця функція повертає ненульове значення, а в разі невдачі - false. Перший і останній параметри цієї функції мають таке ж призначення, як і у функції SetFiiePointer. Тому коротко опишемо тільки параметри, які залишилися.

Параметр liDistanceToMove задає зсув вказівника позиції файлу, який розглядається як ціле число зі знаком. У разі позитивного числа покажчик позиції зрушується вперед, а в разі негативного - назад. Параметр lpNewFiiePointer повинен вказувати на об'єднання типу LARGE_INTEGER, в яке функція SetFiiePointerEx поверне нове значення індикатора позиції файлу. Об'єднання типу large_integer має наступний формат:

```
     typedef union _LARGE_INTEGER (
struct
(
DWORD LowPart; / / молодша частина
LONG HighPart; / / старша частина
);
LONGLONG QuadPart; / / всі частини
) LARGE_INTEGER, * PLARGE_INTEGER;
```

Якщо значення параметра lpNewFiiePointer дорівнює null, то нове значення індикатора позиції не буде повертатися.

У лістингу 1.9 приведена програма, яка для встановлення індикатора позиції використовує функцію SetFiiePointerEx, а потім читає з файлу запис, на який встановлений індикатор позиції.

Лістинг 1.9 Установка покажчика позиції файлу за допомогою функції SetFiiePointerEx

```
// # Define _WIN32_WINNT 0x0500 / / або визначити цей макрос замість
функції, якщо нова платформа
# Include <windows.h>
# Include <iostream>
using namespace std;
extern "C" WINBASEAPI BOOL WINAPI SetFilePointerEx (
    HANDLE hFile,
```

```
    LARGE_INTEGER liDistanceToMove,
    PLARGE_INTEGER lpNewFilePointer,
    DWORD dwMoveMethod
    );
int main ()
{
  HANDLE hFile; / / дескриптор файлу
  int n; / / для номера запису
  LARGE_INTEGER p, q; / / для покажчика позиції
  DWORD dwBytesRead; / / кількість прочитаних байтів
  int m; / / прочитане число
  / / Відкриваємо файл для читання
  hFile = CreateFile (
    "C: \ \ demo_file.dat", / / ім'я файлу
    GENERIC_READ, / / читання з файлу
    0, / / монопольний доступ до файлу
    NULL, / / захисту немає
    OPEN_EXISTING, / / відкриваємо існуючий файл
    FILE_ATTRIBUTE_NORMAL, / / звичайний файл
    NULL / / шаблону немає
  );
  / / Перевіряємо на успішне відкриття
  if (hFile == INVALID_HANDLE_VALUE)
  {
    cerr <<"Create file failed." <<Endl
      <<"The last error code:" <<GetLastError () <<endl;
    cout <<"Press any key to finish.";
    cin.get ();
    return 0;
  }

  / / Вводимо номер потрібного запису
  cout <<"Input a number from 0 to 9:";
  cin>> n;
  q.HighPart = 0;
  q.LowPart = n * sizeof (int);
  / / Зрушуємо покажчик позиції файлу
  if (! SetFilePointerEx (hFile, q, & p, FILE_BEGIN))
```

```
  {
    cerr <<"Set file pointer failed." <<Endl
      <<"The last error code:" <<GetLastError () <<endl;
    CloseHandle (hFile);
    cout <<"Press any key to finish.";
    cin.get ();
    return 0;
  }
  / / Виводимо на консоль значення покажчика позиції файлу
  cout <<"File pointer:" <<p.HighPart <<'' <<p.LowPart <<endl;
  / / Читаємо дані з файлу
  if (! ReadFile (
      hFile, / / дескриптор файлу
      & M, / / адреса буфера, куди читаємо дані
      sizeof (m), / / кількість читаних байтів
      & DwBytesRead, / / кількість прочитаних байтів
      (LPOVERLAPPED) NULL / / читання синхронне
    ))
  {
    cerr <<"Read file failed." <<Endl
      <<"The last error code:" <<GetLastError () <<endl;
    CloseHandle (hFile);
    cout <<"Press any key to finish.";
    cin.get ();
    return 0;
  }
  / / Виводимо прочитане число на консоль
  cout <<"The read number:" <<m <<endl;
  / / Закриваємо дескриптор файлу
  CloseHandle (hFile);
  return 0;
}
```

1.11. Визначення та зміна атрибутів файлу

Дізнатися атрибути файлу можна за допомогою **функції GetFileAttributes**, яка має наступний прототип:

```
DWORD GetFileAttributes (
LPCTSTR lpFileName / / ім'я файлу
);
```

У разі успішного завершення ця функція повертає атрибути файлу, а в разі невдачі - значення -1. Єдиний параметр цієї функції повинен містити ім'я файлу, а в значенні, що повертається, встановлюються атрибути файлу. Ці атрибути можна перевірити, використовуючи наступні прапори:

file_attribute_archive - архівний файл;

file_attribute_compressed - стиснений файл;

file_attribute_directory - файл є каталогом;

FILE_ATTRIBUTE_ENCRYPTED - незашифрований файл;

FILE_ATTRIBUTE_HIDDEN - прихований файл;

file_attribute_normal - нормальний файл;

FILE_ATTRIBUTE_NOT_CONTENT_INDEXED - файл не індексується;

file_attribute_offline - файл у зовнішній пам'яті;

FILE_ATTRIBUTE_READONLY - файл лише для читання;

FILE_ATTRIBUTE_REPARSE_POINT - файл вимагає інтерпретації;

file_attribute_sparse_file - розріджений файл;

FILE_ATTRIBUTE_SYSTEM - системний файл;

FILE_ATTRIBUTE_TEMPORARY - тимчасовий файл.

Інші атрибути файлу, які не увійшли до вищенаведеного списку, можна дізнатися, використовуючи функцію GetFileAttributesEx. Змінити атрибути файлу можна за допомогою **функції SetFileAttributes**, яка має наступний прототип:

```
BOOL SetFileAttributes (
LPCTSTR lpFileName, / / ім'я файлу
DWORD dwFileAttributes / / атрибути файлу
);
```

У разі успішного завершення ця функція повертає ненульове значення, а в разі невдачі - false. Параметр lpFileName повинен містити ім'я файлу, а в параметрі dwFiieAttributes можна встановити такі атрибути файлу:

file_attribute_archive - архівний файл;

FiLE_ATTRiBUTE_HiDDEN - прихований файл;

file_attribute_normal - нормальний файл;

file_attribute_not_content_indexed - файл не індексується;

file_attribute_offline - файл у зовнішній пам'яті;

FiLE_ATTRiBUTE_READONLY - файл лише для читання;

FILE_ATTRIBUTE_SYSTEM - системний файл;

FILE_ATTRIBUTE_TEMPORARY - тимчасовий файл.

При цьому, якщо встановлюється прапор file_attribute_normal, то він повинен бути один, тому що всі інші атрибути, анулюють цей атрибут.

У лістингу 1.10 наведена програма, яка читає і змінює атрибути файлу.

Лістінг 1.10 Читання та зміна атрибутів файлу

```
# Include <windows.h>
# Include <iostream.h>
int main ()
{
  DWORD  file_attr;
  // Читаємо атрибути файлу
  file_attr = GetFileAttributes(izia);
  if(file_attr == -1)
  {
    cerr << "Get file attributes failed." << endl
      << "The last error code: " << GetLastError() << endl;
    cout << "Press any key to finish.";
    cin.get();
    return 0;
  }
  // Перевіряємо, чи є файл нормальним
  if(file_attr == FILE_ATTRIBUTE_NORMAL)
    cout << "This is a normal file." << endl;
```

```
else
{
  cout << "This is a not normal file." << endl;
  return 0;
}
// Встановлюємо атрибут прихованого файлу
if(!SetFileAttributes(izia, FILE_ATTRIBUTE_HIDDEN))
{
  cerr << "Set file attributes failed." << endl
    << "The last error code: " << GetLastError() << endl;
  cout << "Press any key to finish.";
  cin.get();
  return 0;
}
// Тепер можна перевірити, що файл зник
cout << "Now the file is hidden." << endl
  << "Press any key to continue.";
cin.get();
// Зворотно робимо файл звичайним
if(!SetFileAttributes(izia, FILE_ATTRIBUTE_NORMAL))
{
  cerr << "Set file attributes failed." << endl
    << "The last error code: " << GetLastError() << endl;
  cout << "Press any key to finish.";
  cin.get();
  return 0;
}
cout << "Now the file is again normal." << endl;

return 0;
    }
```

1.12. Визначення та зміна розмірів файлу

Розмір файлу є цілим числом і для його зберігання потрібні два значення типу dword або, іншими словами, два подвійних слова. Перше з цих подвійних слів містить старшу частину розміру файлу, а друге - молодшу.

Якщо розмір файлу входить тільки в молодшу частину, то значення старшої встановлюється в null.

Визначити розмір файлу можна за допомогою **функції GetFiieSize**, яка має наступний прототип:

```
     DWORD GetFiieSize (
HANDLE hFile, // дескриптор файлу
LPDWORD lpFileSizeHigh / / вказівник на старшу частину розміру файлу
);
```

У разі успішного завершення ця функція повертає молодшу частину розміру файлу, а за адресою, вказаною у параметрі lpFileSizeHigh, записує старшу частину розміру файлу. У разі невдачі функція GetFiieSize повертає значення -1, якщо значення адреси, заданої параметром lpFileSizeHigh, встановлено в null. Якщо ж значення цієї адреси не дорівнює null і функція закінчилася невдачею, то вона повертає значення -1 і функція коду останньої помилки GetLastError поверне значення, відмінне від NO_ERROR.

Відзначимо, що для коректного функціонування GetFiieSize необхідно, щоб файл був відкритий в режимі читання або запису.

У лістингу 1.11 наведена програма, яка визначає розмір файлу, використовуючи функцію GetFiieSize.

Лістинг 1.11. Визначення розміру файлу за допомогою функції GetFiieSize

```
# Include <windows.h>
# Include <iostream.h>
int main ()
{
  HANDLE  hFile;
  DWORD  dwFileSize;   // молодша частина розміру файлу
  // Відкриваємо файл для читання
  hFile = CreateFile(
    zak,                        // ім'я файлу
    GENERIC_READ,               // читання з файлу
      0,                        // монопольний доступ до файлу
      NULL,                     // захисту немає
      OPEN_EXISTING,            // відкриваємо існуючий файл
```

```
      FILE_ATTRIBUTE_NORMAL,          // звичайний файл
      NULL                            // шаблону немає
   );
  // Перевіряємо на успішне відкриття
  if (hFile == INVALID_HANDLE_VALUE)
  {
    cerr << "Create file failed." << endl
      << "The last error code: " << GetLastError() << endl;
    cout << "Press any key to finish.";
    cin.get();
    return 0;
  }
  // Визначаємо розмір файлу
  dwFileSize = GetFileSize(hFile, NULL);
  if (dwFileSize == -1)
  {
    cerr << "Get file size failed." << endl
      << "The last error code: " << GetLastError() << endl;
    CloseHandle(hFile);
    cout << "Press any key to finish.";
    cin.get();
    return 0;
  }
  // Виводимо розмір файлу
  cout << "File size: " << dwFileSize << endl;
  // Закриваємо дескриптор файлу
  CloseHandle(hFile);
  return 0;
     }
```

Визначити розмір файлу можна також за допомогою **функції GetFilesizeEx**, яка підтримується тільки операційними системами Windows 2000 і Windows XP. Ця функція має наступний прототип:

```
     BOOL GetFileSizeEx (
HANDLE hFile, / / дескриптор файлу
PLARGE_INTEGER lpFileSize / / розмір файлу
);
```

У разі успішного завершення функція повертає ненульове значення, а в разі невдачі - значення false.

Параметр hFile цієї функції повинен містити дескриптор файлу, а параметр lpFileSize вказує на об'єднання типу large_integer, в яке функція GetFilesizeEx записує розмір файлу. Формат об'єднання типу large_INTEGER наведений у розділі 1.10.

У лістингу 1.12 наведена програма, яка визначає розмір файлу, використовуючи функцію GetFileSizeEx.

Лістинг 1.12 Визначення розміру файлу за допомогою функції GetFileSizeEx

```
# Include <windows.h>
# Include <iostream.h>
int main ()
{
  HANDLE hFile;
  LARGE_INTEGER liFileSize; / / розмір файлу
  / / Відкриваємо файл для читання
  hFile = CreateFile (
    "C: \ \ demo_file.dat", / / ім'я файлу
    GENERIC_READ, / / читання з файлу
    0, / / монопольний доступ до файлу
    NULL, / / захисту немає
    OPEN_EXISTING, / / відкриваємо існуючий файл
    FILE_ATTRIBUTE_NORMAL, / / звичайний файл
    NULL / / шаблону немає
  );
  / / Перевіряємо на успішне відкриття
  if (hFile == INVALID_HANDLE_VALUE)
  {
    cerr <<"Create file failed." <<Endl
      <<"The last error code:" <<GetLastError () <<endl;
    cout <<"Press any key to finish.";
    cin.get ();
    return 0;
  }
```

```
  // Визначаємо розмір файлу
  if (! GetFileSizeEx (hFile, & liFileSize))
    {
cerr <<"Get file size failed." <<Endl
      <<"The last error code:" <<GetLastError () <<endl;
    CloseHandle (hFile);
    cout <<"Press any key to finish.";
    cin.get ();
    return 0;
  }
  / / Виводимо розмір файлу
  cout <<"File size:" <<liFileSize.LowPart <<endl;
  / / Закриваємо дескриптор файлу
  CloseHandle (hFile);
  return 0;
}
```

Змінити розмір файлу можна за допомогою **функції setEndofFile**, яка має наступний прототип:

```
BOOL SetEndOfFile (
HANDLE hFile / / дескриптор файлу );
```

У разі успішного завершення ця функція повертає ненульове значення, а в разі невдачі - false. Єдиним параметром функції є дескриптор файлу, розмір якого змінюється. При цьому відзначимо, що файл повинен бути відкритий у режимі запису.

Функція setEndofFile працює таким чином. Вона пересуває маркер кінця файлу eof на позицію, яку містить покажчик позиції файлу. Якщо покажчик позиції файлу вказує на запис, який не є останнім, то всі записи, які знаходяться за поточним записом, відкидаються. Таким чином, в цьому випадку вміст файлу урізається. Якщо ж покажчик позиції файлу вказує за межі файлу, то обсяг файлу розширюється за рахунок додавання нових кластерів до нього. При цьому вміст доданих кластерів не визначено. У лістингу 1.13 приведений приклад програми, яка зменшує розмір файлу в два рази.

Лістинг 1.13. Зміна розміру файлу

```
# Include <windows.h>
# Include <iostream.h>
int main ()
{
 HANDLE hFile;              // дескриптор файлу
  DWORD dwFileSize;         // розмір файлу
  long p;                   // вказівник позиції

  // Відкриваємо файл для читання
  hFile = CreateFile(
  zakys,                          // ім'я файлу
    GENERIC_WRITE,          // запис в файл
    0,                      // монопольний доступ до файлу
    NULL,                   // захисту немає
    OPEN_EXISTING,          // відкриваємо існуючий файл
    FILE_ATTRIBUTE_NORMAL,       // звичайний файл
    NULL                    // шаблону немає
  );
  // Перевіряємо на успішне відкриття
  if (hFile == INVALID_HANDLE_VALUE)
  {
    cerr << "Create file failed." << endl
      << "The last error code: " << GetLastError() << endl;
    cout << "Press any key to finish.";
    cin.get();

    return 0;
  }
  // Визначаємо розмір файлу
  dwFileSize = GetFileSize(hFile, NULL);
  if (dwFileSize == -1)
  {
    cerr << "Get file size failed." << endl
      << "The last error code: " << GetLastError() << endl;
    CloseHandle(hFile);
    cout << "Press any key to finish.";
    cin.get();
```

```
  return 0;
}
// Виводимо на консоль розмір файлу
cout << "Old file size: " << dwFileSize << endl;
// Зменшуємо розмір файлу вдвічі
dwFileSize /= 2;
// Здвигаємо покажчик позиції файлу
p = SetFilePointer(hFile, dwFileSize, NULL, FILE_BEGIN);
if(p == -1)
{
  cerr << "Set file pointer failed." << endl
    << "The last error code: " << GetLastError() << endl;
  CloseHandle(hFile);
  cout << "Press any key to finish.";
  cin.get();
  return 0;
}
// Встановлюємо новий розмір файлу
if (!SetEndOfFile(hFile))
{
  cerr << "Set end of file failed." << endl
    << "The last error code: " << GetLastError() << endl;
  CloseHandle(hFile);
  cout << "Press any key to finish.";
  cin.get();
  return 0;
}
// Визначаємо новий розмір файлу
dwFileSize = GetFileSize(hFile, NULL);
if (dwFileSize == -1)
{
  cerr << "Get file size failed." << endl
    << "The last error code: " << GetLastError() << endl;
  CloseHandle(hFile);
  cout << "Press any key to finish.";
  cin.get();
  return 0;
```

```
}
// Виводимо на консоль розмір файлу
cout << "New file size: " << dwFileSize << endl;
/ / Закриваємо дескриптор файлу
CloseHandle(hFile);
return 0;
    }
```

1.13. Блокування файлу

Часто кілька додатків мають спільний доступ до одного і того ж файлу. При цьому може знадобитися, щоб додаток, який змінює дані у файлі, мав до цього файлу монопольний доступ. Для цього необхідно блокувати весь файл або тільки його частину для доступу іншим додаткам. Для цих цілей використовується **функція LockFile**, яка має наступний прототип:

```
     BOOL LockFile (
HANDLE hFile, / / дескриптор файлу
DWORD dwFileOffsetLow, / / молодша частина зміщення
DWORD dwFileOffsetHigh, / / старша частина зміщення
DWORD nNumberOfBytesToLockLow, / / молодша частину кількості байтів
DWORD nNumberOfBytesTiLockHigh / / старша частину кількості байтів
);
```

У разі успішного завершення функція повертає ненульове значення, а в разі невдачі - false. Коротко опишемо призначення параметрів цієї функції.

Параметр hFile повинен містити дескриптор файлу. Причому цей файл повинен бути відкритий в режимі запису або читання.

У параметрах dwFileOffsetLow і dwFileOffsetHigh повинні бути встановлені відповідно молодша та старша частини зміщення від початку файлу в байтах. Для операційної системи Windows 98 значення параметра dwFileOffsetHigh повинно бути встановлено в 0.

У параметрах nNumberOfBytesToLockLow і nNumberOfBytesTiLockHigh повинні бути встановлені відповідно старша і молодша частини довжини області файла, яка блокується для монопольного доступу додатком.

Для скасування блокування області файла використовується **функція UnlockFile**, яка має наступний прототип:

```
     BOOL UnlockFile (
HANDLE hFile, / / дескриптор файлу
DWORD dwFileOffsetLow, / / молодша частина зміщення
DWORD dwFileOffsetHigh, / / старша частина зміщення
DWORD nNumberOfBytesToLockLow, / / молодша частину кількості байтів
DWORD nNumberOfBytesTiLockHigh / / старша частину кількості байтів
);
```

У разі успішного завершення функція повертає ненульове значення, а в разі невдачі - значення false. Всі параметри цієї функції аналогічні параметрам функції LockFiie.

У лістингу 1.14 наведена програма, яка спочатку блокує доступ до файлу, а потім розблокує його.

Лістинг 1.14. Блокування та розблокування файлу

```
# Include <windows.h>
# Include <iostream.h>
int main ()
{
  HANDLE  hFile;
  DWORD  dwFileSize;
  // Відкриваємо файл для запису
  hFile = CreateFile(
  mania,                              // ім'я файлу
    GENERIC_WRITE,                 // запис в файл
    0,                             // монопольний доступ до файлу
    NULL,                          // захисту немає
    OPEN_EXISTING,                 // відкриваємо існуючий файл
    FILE_ATTRIBUTE_NORMAL,         // звичайний файл
    NULL                           // шаблону немає
  );
  // Перевіряємо на успішне відкриття
  if (hFile == INVALID_HANDLE_VALUE)
  {
    cerr << "Create file failed." << endl
      << "The last error code: " << GetLastError() << endl;
```

```
  cout << "Press any key to finish.";
  cin.get();
  return 0;
}
// Визначаємо розмір файлу
dwFileSize = GetFileSize(hFile, NULL);
if (dwFileSize == -1)
{
  cerr << "Get file size failed." << endl
    << "The last error code: " << GetLastError() << endl;
  CloseHandle(hFile);
  cout << "Press any key to finish.";
  cin.get();
  return 0;
}
// Блокуємо файл
if (!LockFile(hFile, 0, 0, dwFileSize, 0))
{
  cerr << "Lock file failed." << endl
    << "The last error code: " << GetLastError() << endl;
  CloseHandle(hFile);
  cout << "Press any key to finish.";
  cin.get();
  return 0;
}
cout << "Now the file is locked." << endl
  << "Press any key to continue." << endl;
cin.get();
// Розблокуємо файл
if (!UnlockFile(hFile, 0, 0, dwFileSize, 0))
{
  cerr << "Unlock file failed." << endl
    << "The last error code: " << GetLastError() << endl;
  CloseHandle(hFile);
  cout << "Press any key to finish.";
  cin.get();
  return 0;
}
```

```
cout << "Now the file is unlocked." << endl
  << "Press any key to continue." << endl;
cin.get();
// Закриваємо дескриптор файлу
CloseHandle(hFile);
return 0;
    }
```

Перш за все, відзначимо, що якщо додаток видає запит на блокування області файла, яка вже заблокована самим додатком, то виконання функції LockFile завершиться невдачею. Крім того, блокування та розблокування файлів повинно співпадати по областях.

1.14. Отримання інформації про файл

Щоб отримати інформацію про файл, можна використовувати **функцію GetFileInformationByHandle**, яка має наступний прототип

```
BOOL GetFileInformationByHandle (
HANDLE hFile, / / дескриптор файлу
/ / Вказівник на інформацію
LPBY_HANDLE_FILE_INFORMATION lpFilelnf ormation
);
```

У разі успішного завершення ця функція повертає ненульове значення, а в разі невдачі - false.

У параметрі hFile цієї функції повинен бути встановлений дескриптор файлу, про який ви хочете отримати. Відзначимо, що цей файл може бути відкритий у будь-якому режимі доступу.

Параметр lpFiieinformation повинен вказувати на структуру типу BY_HANDLE_FILE_INFORMATION, в яку функція запише інформацію про файл. Ця структура має наступний формат:

typedef struct _BY_HANDLE_FILE_INFORMATION (

DWORD dwFileAttributes; / / атрибути файлу

FILETIME ftCreationTime; / / час створення файлу

FILETIME ftLastAccessTime; / / час останнього доступу до файлу

FILETIME ftLastWriteTime; / / час останнього запису в файл

DWORD dwVolumeSerialNumber; / / серійний номер тому

DWORD nFileSizeHigh; / / старша частина розміру файлу

DWORD nFileSizeLow; / / молодша частина розміру файлу

DWORD nNumberOfLinks; / / кількість посилань на файл

DWORD nFileIndexHigh; / / старша частину індексу файлу

DWORD nFileIndexLow; / / молодша частину індексу файлу

) BY_HANDLE_FILE_INFORMATION, * LPBY_HANDLE_FILE_INFORMATION;

У лістингу 1.15 наведена програма, яка отримує інформацію про файл і роздруковує її.

Лістинг 1.15 Отримання інформації про файл

```
# Include <windows.h>
# Include <iostream.h>
int main ()
{
  HANDLE  hFile;
  BY_HANDLE_FILE_INFORMATION  bhfi;  // інформація про файл
  // Відкриваємо файл для читання
  hFile = CreateFile(
  vania,                                  // ім'я файлу
    0,                            // отримання інформації про файл
    0,                            // монопольний доступ до файлу
    NULL,                         // захисту немає
    OPEN_EXISTING,                // відкриваємо існуючий файл
    FILE_ATTRIBUTE_NORMAL,        // звичайний файл
    NULL                          // шаблону немає
  );
  // Перевіряємо на успішне відкриття
  if (hFile == INVALID_HANDLE_VALUE)
  {
    cerr << "Create file failed." << endl
      << "The last error code: " << GetLastError() << endl;
    cout << "Press any key to finish.";
    cin.get();
```

```
  return 0;
}
// Отримуємо інформацію про файл
if (!GetFileInformationByHandle(hFile, &bhfi))
{
  cerr << "Get file information by handle failed." << endl
    << "The last error code: " << GetLastError() << endl;
  cout << "Press any key to finish.";
  cin.get();
  return 0;
}
// Роздруковуємо інформацію про файл
cout << "File attributes: " << bhfi.dwFileAttributes << endl
  << "Creation time: high date: "
    << bhfi.ftCreationTime.dwHighDateTime << endl
  << "Creation time: low date: "
    << bhfi.ftCreationTime.dwLowDateTime << endl
  << "Last access time: high date: "
    << bhfi.ftLastAccessTime.dwHighDateTime << endl
  << "Last access time: low date: "
    << bhfi.ftLastAccessTime.dwLowDateTime << endl
  << "Last write time: high date: "
    << bhfi.ftLastWriteTime.dwHighDateTime << endl
  << "Last write time: low date: "
    << bhfi.ftLastWriteTime.dwLowDateTime << endl
  << "Volume serial number: " << bhfi.dwVolumeSerialNumber << endl
  << "File size high: " << bhfi.nFileSizeHigh << endl
  << "File size low: " << bhfi.nFileSizeLow << endl
  << "Number of links: " << bhfi.nNumberOfLinks << endl
  << "File index high: " << bhfi.nFileIndexHigh << endl
  << "File index low: " << bhfi.nFileIndexLow << endl;
// Закриваємо дескриптор файлу
CloseHandle(hFile);
return 0;
    }
```

Структура типу by_handle_file_information містить структуру типу filetime, яка служить для збереження часу. Ця структура має наступний формат:

```
      typedef struct _FILETIME (
DWORD dwLowDateTime; / / молодша частину часу
DWORD dwHighDateTime; / / старша частину часу
) FILETIME, * PFILETIME;
```

Саме час задається в інтервалах, кожен з яких дорівнює 100 наносекунд. Природно, що такий час незручно проглядати користувачеві. Тому для перекладу часу в більш зручну форму існує **функція FileTimeToSystemTime**, яка має наступний прототип.

```
      BOOL FileTimeToSystemTime (
CONST FILETIME * lpFileTime; / / вказівник на час у форматі "файл"
LPSYSTEMTIME IpSystemTime / / вказівник на час у форматі "система"
);
```

У разі успішного завершення ця функція повертає ненульове значення, а в разі невдачі - false.

Параметр lpFiieTime цієї функції повинен вказувати на структуру типу filetime, яка містить час у форматі, який використовується для зберігання у файловій системі.

Параметр lpSystemTime повинен вказувати на структуру типу systemtime, яка має наступний формат:

```
      typedef struct _SYSTEMTIME (
WOR  wYear;     //
WOR  wMonth;    //
WOR  wDayOfWee //
WOR  wDay;      //
WOR  wHour;     //
WOR  wMinute;   //
WOR  wSecond;   //
WOR        wMillisec onds;

) SYSTEMTIME, * LPSYSTEMTIME;
```

У лістингу 1.16 наведена програма, яка використовує функцію FileTimeToSystemTime для перекладу часу з формату файлової системи в системний формат.

Лістинг 1.16. Перетворення часу в системний формат

```
# Include <windows.h>
# Include <iostream.h>
int main ()
{
  HANDLE hFile;
  BY_HANDLE_FILE_INFORMATION bhfi; / / інформація про файл
  SYSTEMTIME st; / / системний час
  / / Відкриваємо файл для читання
  hFile = CreateFile (
    "C: \ \ demo_file.dat", / / ім'я файлу
    0, / / отримання інформації про файл
    0, / / монопольний доступ до файлу
    NULL, / / захисту немає
    OPEN_EXISTING, / / відкриваємо існуючий файл
    FILE_ATTRIBUTE_NORMAL, / / звичайний файл
    NULL / / шаблону немає
  );
  / / Перевіряємо на успішне відкриття
  if (hFile == INVALID_HANDLE_VALUE)
  {
    cerr <<"Create file failed." <<Endl
      <<"The last error code:" <<GetLastError () <<endl;
    cout <<"Press any key to finish.";
    cin.get ();
    return 0;
  }
  / / Отримуємо інформацію про файл
  if (! GetFileInformationByHandle (hFile, & bhfi))
  {
    cerr <<"Get file information by handle failed." <<Endl
      <<"The last error code:" <<GetLastError () <<endl;
    cout <<"Press any key to finish.";
    cin.get ();
    return 0;
  }
  / / Переводимо час створення файлу в системний час
  if (! FileTimeToSystemTime (& (bhfi.ftCreationTime), & st))
  {
```

```
    cerr <<"File time to system time failed." <<Endl
      <<"The last error code:" <<GetLastError () <<endl;
    cout <<"Press any key to finish.";
    cin.get ();
    return 0;
  }
  / / Роздруковуємо системний час
  cout <<"File creation time in system format:" <<endl
    <<"\ TYear:" <<st.wYear <<endl
    <<"\ TMonth:" <<st.wMonth <<endl
    <<"\ TDay of week:" <<st.wDayOfWeek <<endl
    <<"\ TDay:" <<st.wDay <<endl
    <<"\ THour:" <<st.wHour <<endl
    <<"\ TMinute:" <<st.wMinute <<endl
    <<"\ TSecond:" <<st.wSecond <<endl
    <<"\ TMilliseconds:" <<st.wMilliseconds <<endl;
  / / Закриваємо дескриптор файлу
  CloseHandle (hFile);
  return 0;
}
```

Визначити тип файлу можна за допомогою **функції GetFileType**, яка має наступний прототип:

```
DWORD GetFileType (
HANDLE hFile / / дескриптор файлу
);
```

Єдиним параметром цієї функції є дескриптор файлу. Функція GetFileType повертає одне з наступних значень:

file_type_unknown - невідомий тип файлу;

file_type_disk - дисковий файл;

file_type_char - символьний файл;

file_type_pipe - іменований або анонімний канал.

Відзначимо, що під символьним файлом звичайно розуміється принтер або консоль.

У лістингу 1.17 наведена програма, яка визначає тип файлу, використовуючи функцію GetFileType.

Лістинг 1.17 Визначення типу файлу

```
# Include <windows.h>
# Include <iostream.h>
int main ()
{
  HANDLE  hFile;
  DWORD  dwFileType;
  // Відкриваємо файл для читання
  hFile = CreateFile(
   gava,                          // ім'я файлу
    0,                            // отримання інформації про файл
    0,                            // монопольний доступ до файлу
    NULL,                         // захисту немає
    OPEN_EXISTING,                // відкриваємо існуючий файл
    FILE_ATTRIBUTE_NORMAL,        // звичайний файл
    NULL                          // шаблону немає
  );
  // Перевіряємо на успішне відкриття
  if (hFile == INVALID_HANDLE_VALUE)
  {
    cerr << "Create file failed." << endl
      << "The last error code: " << GetLastError() << endl;
    cout << "Press any key to finish.";
    cin.get();
    return 0;
  }
  // Визначаємо тип файлу
  dwFileType = GetFileType(hFile);
    // Роздруковуємо тип файлу
  switch (dwFileType)
  {
  case FILE_TYPE_UNKNOWN:
    cout << "Unknown type file." << endl;
    break;
  case FILE_TYPE_DISK:
```

```
    cout << "Disk type file." << endl;
    break;
  case FILE_TYPE_CHAR:
    cout << "Char type file." << endl;
    break;
  case FILE_TYPE_PIPE:
    cout << "Pipe type file." << endl;
    break;
  default:
    break;
  }
  return 0;
      }
```

На платформі Windows NT можна визначити, чи є файл виконуваним. Для цього треба використовувати **функцію GetBinaryType**, яка має наступний прототип:

```
     BOOL GetBinaryType (
LPCTSTR lpApplicationName, / / назва програми
LPDWORD lpBinaryType / / тип виконуваного файлу
);
```

Якщо файл є виконуваним, то ця функція повертає ненульове значення, в іншому випадку - false.

У параметрі lpApplicationName встановлюється вказівник на рядок, що містить ім'я файлу, що перевіряється.

Параметр lpBinaryType повинен вказувати на змінну типу DWORD, в яку функція GetBinaryType поміщає тип виконуваного файлу. Цей тип може приймати одне з наступних значень:

scs_32bit_binary - додаток Win32;

scs_dos_binary - додаток MS-DOS;

scs_os216_binary - 16-бітове додаток OS / 2;

scs_pif_binary - PIF-файл;

scs_posix_binary - додаток POSIX;

scs_wow_binary - 16-бітове програму Windows.

У лістингу 1.18 наведена програма, яка визначає тип виконуваного файлу, використовуючи функцію GetBinaryType.

Лістинг 1.18 Визначення типу виконуваного файлу

```
# include <windows.h>
# include <iostream.h>
int main ()
{
  DWORD dwBinaryType;
  // Визначаємо тип файлу
  if (! GetBinaryType ("C: \ \ temp.exe", & dwBinaryType))
  {
    cerr <<"Get binary type failed." <<Endl
      <<"The file may not be executable." <<Endl
      <<"The last error code:" <<GetLastError () <<endl;
    cout <<"Press any key to finish.";
    cin.get ();
    return 0;
  }
  // Роздруковуємо тип файлу
  if (dwBinaryType == SCS_32BIT_BINARY)
    cout <<"The file is Win32 based application." <<Endl;
  else
    cout <<"The file is not Win32 based application." <<Endl;
  return 0;
}
```

Завдання:

1. Створити програму, яка видаляє файл з ім'ям demo_file.dat, який розташований в кореневому каталозі на диску С.

2. Створити програму, яка створює файл і записує в нього послідовність цілих чисел.

3. Створити програму, в якій функція FlushFileBuffers використовується для скидання даних з буферів в файл після запису половини файлу.

4. Створити програму, яка читає дані з файлу, створеного програмою з лістингу 1.2, і виводить ці дані на консоль.

5. Створити програму, яка виконує копіювання файлу, використовуючи функцію CopyFile.

6. Створити програму, яка виконує переміщення файлу, використовуючи функцію MoveFile.

7. Створити програму, в якій використати функцію RepiaceFile для заміщення файлу.

8. Створити програму, яка читає запис файлу, попередньо встановивши на цей запис покажчик.

9. Створити програму, яка читає і змінює атрибути файлу.

10. програма, яка визначає розмір файлу, використовуючи функцію GetFiieSize.

11. Створити програму, яка визначає розмір файлу, використовуючи функцію GetFileSizeEx.

12. Створити програму, яка зменшує розмір файлу в два рази.

13. Створити програму, яка спочатку блокує доступ до файлу, а потім розблокує його.

14. Створити програму, яка отримує інформацію про файл і роздруковує її.

15. Створити програму, яка використовує функцію FileTimeToSystemTime для перекладу часу з формату файлової системи в системний формат.

16. Створити програму, яка визначає тип файлу, використовуючи функцію GetFileType.

17. Розробити програму, яка визначає тип виконуваного файлу, використовуючи функцію GetBinaryType.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Які імена файлів називають довгими?

2. Що представляє собою дескриптор?

3. Що представляє собою повне ім’я файлу?

4.. З яких частин складається покажчик позиції файлу?

5. Яка функція використовується для створення нових або відкриття вже існуючих файлів?

6. Яка функція використовується для визначення типу файла?

7. Яка функція використовується для визначення, чи є файл виконуваним?

8. Яка функція використовується для перекладу часу в більш зручну форму?

9. Яка функція використовується для визначення атрибутів файлу?

10. Для яких цілей використовується функція LockFile?

11. Яка функція використовується для роботи з покажчиком позиції файлу?

12. Яка функція використовується для скасування блокування області файла?

13. Яка функція використовується для визначення розміру файлу?

14. Які відмінності функції ReplaceFile від функцій CopyFile і MoveFile.

15. Яка функція використовується для переміщення файлів?

16. Яка функція використовується для копіювання файлів?

17. Яка функція використовується для читання даних з файлу?

18. Яка функція використовується для запису даних у файл?

19. Перелічіть загальні або родові режими доступу до файлу.