Лабораторна робота № 5

(4 години)

Тема: Програмування файлових операцій засобами Windows API.

Мета: Ознайомитися із засобами Win32 API для роботи з файлами. Навчитися використовувати ці функції для програмування файлових операцій.

Короткі теоретичні відомості

Практично в кожній програмі Windows використовують файлові операції. Найчастіше або початкові дані для програми або дані, одержані в результаті роботи, зберігаються на дисках у вигляді файлів; інколи програмі доводиться створювати ще й тимчасові файли для зберігання поточних даних.

Win32 надає як традиційні засоби роботи з файлами (створення і видалення файлів і каталогів, читання і запис, пошук, зміна характеристик), так і специфічні засоби, наприклад проектування файлів у пам'ять і сторінкову організацію пам'яті із використанням файлів підкачки).

Створення і відкриття файлу

Функція **CreateFile()** створює новий файл або відкриває існуючий. Ця функція використовується також із такими об'єктами як канали, поштові скриньки, диски, консолі, директорії. Функція повертає дескриптор відкритого файлу.

```
HANDLE CreateFile(

  LPCTSTR     lpFileName,                 // вказівник на ім'я файлу
  DWORD       dwDesiredAccess,            // тип доступу
  DWORD       dwShareMode,                // тип спільного використання
  LPSECURITY_ATTRIBUTES  lpSecurityAttributes,  // вказівник на атрибути безпеки
  DWORD       dwCreationDisposition,      // спосіб створення
  DWORD       dwFlagsAndAttributes,       // атрибути нового файлу
  HANDLE      hTemplateFile               // дескриптор шаблону для копіювання
                                          // атрибутів
);
```

Параметри:

dwDesiredAccess `GENERIC_READ | GENERIC_WRITE`

dwShareMode `0 | FILE_SHARE_READ | FILE_SHARE_WRITE`

lpSecurityAttributes `NULL` або структура `SECURITY_ATTRIBUTES`

dwCreationDisposition

`CREATE_NEW`	створює новий файл, якщо існує – помилка.
`CREATE_ALWAYS`	перезаписує існуючий файл.
`OPEN_EXISTING`	якщо не існує – помилка.
`OPEN_ALWAYS`	відкриває файл, створює, якщо не існує.
`TRUNCATE_EXISTING`	встановлює нульовий розмір файлу.

dwFlagsAndAttributes

`FILE_ATTRIBUTE_NORMAL`	
`FILE_ATTRIBUTE_READONLY`	тільки для читання.
`FILE_ATTRIBUTE_HIDDEN`	прихований файл
`FILE_ATTRIBUTE_SYSTEM`	системний файл
`FILE_FLAG_RANDOM_ACCESS`	відкрити для довільного доступу
`FILE_FLAG_DELETE_ON_CLOSE`	знищити після закриття (для тимчасових файлів).

hTemplateFile дескриптор файлу із правами `GENERIC_READ`, із якого копіюються атрибути

Закриття файлу

BOOL CloseHandle(HANDLE hObject**);** закриває будь-який існуючий дескриптор

Читання файлу

Функція читає задану кількість байтів із файлу, і повертає TRUE, якщо читання успішне.

```
BOOL ReadFile(

  HANDLE hFile,                    // дескриптор файлу
  LPVOID lpBuffer,                 // адреса буферу для прийому даних
  DWORD nNumberOfBytesToRead,      // кількість байтів для читання
  LPDWORD lpNumberOfBytesRead,     // адреса числа прочитаних байтів
  LPOVERLAPPED lpOverlapped        // адреса структури перекриття для асинхронних операцій
                                   // або NULL
);
```

Запис у файл

Функція записує задану кількість байтів у файл, і повертає TRUE, якщо запис успішний.

```
BOOL WriteFile(

  HANDLE hFile,                    // дескриптор файлу
  LPCVOID lpBuffer,                // вказівник на буфер із даними для запису
  DWORD nNumberOfBytesToWrite,     // кількість байтів для запису
  LPDWORD lpNumberOfBytesWritten,  // вказівник на число записаних байтів
  LPOVERLAPPED lpOverlapped        // вказівник на структуру перекриття для асинхронних
                                   // операцій або NULL
);
```

Видалення файлу

BOOL DeleteFile (LPCTSTR lpFileName**);**

Копіювання файлу

Windows API має допоміжну функцію **CopyFile()**, яка копіює файл із заданим іменем у файл із новим іменем:

```
BOOL CopyFile(

  LPCTSTR lpExistingFileName,      // вказівник на ім'я існуючого файлу
  LPCTSTR lpNewFileName,           // вказівник на ім'я нового файлу
  BOOL    bFailIfExists            // прапорець для операції, якщо файл існує
);
```

Переміщення (перейменування) файлу

Функція призначена для переміщення або перейменування файлу або каталогу. Новий файл може бути на іншому диску і навіть в іншій файловій системі. Новий каталог повинен бути на тому ж диску.

```
BOOL MoveFile(

  LPCTSTR lpExistingFileName,      // адреса імені існуючого файлу
  LPCTSTR lpNewFileName            // адреса нового імені файлу
);
```

Файл із новим іменем не повинен існувати. Якщо файл вже існує, можна користуватись іншою функцією, яка перейменовує існуючі файли:

```
BOOL MoveFileEx(
```

```
  LPCTSTR lpExistingFileName,        // адреса імені існуючого файлу
  LPCTSTR lpNewFileName,             // адреса нового імені файлу
  DWORD dwFlags                      // набір прапорців, які визначають як переміщувати файл
);
```

Прапорці це будь-яка комбінація наступних символічних констант:

dwFlags MOVEFILE_REPLACE_EXISTING – замінює існуючий файл
MOVEFILE_COPY_ALLOWED – послідовне виконання CopyFile() і DeleteFile()
MOVEFILE_DELAY_UNTIL_REBOOT – переміщення файлу після перезавантаження комп'ютера
MOVEFILE_WRITE_THROUGH – функція чекає завершення переміщення файлу

Керування розміром файлу

Для ***визначення розміру відкритого файлу*** існує функція:

```
DWORD GetFileSize(

  HANDLE hFile,                      // дескриптор файлу
  LPDWORD lpFileSizeHigh             // адреса старшого слова розміру файлу
);
```

Функція повертає молодше подвійне слово розміру файлу. Якщо lpFileSizeHigh = NULL, старше подвійне слово не визначається, інакше lpFileSizeHigh вказує на старше слово розміру файлу.

Є також функція для ***визначення розміру файлу на диску*** за його іменем (невідкритого).

```
DWORD GetCompressedFileSize(

  LPCTSTR lpFileName,                // вказівник на ім'я файлу
  LPDWORD lpFileSizeHigh             // вказівник на DWORD старшої частини розміру файлу
);
```

Функція повертає розмір стисненого файлу на диску, який підтримує стиснення. Якщо файл не стиснений, або диск не підтримує стиснення, повертає актуальний розмір файлу.

Для ***зміни розміру відкритого файлу*** служить функція:

```
BOOL SetEndOfFile(

  HANDLE hFile                       // дескриптор файлу, розмір якого слід змінити
);
```

Функція змінює положення фізичного кінця файлу (EOF). Якщо розмір файлу зменшується, дані в кінці файлу втрачаються. Якщо розмір файлу збільшується, нові дані в кінці файлу не визначені.

Для перевірки наявності файлу або підкаталогу у заданому каталозі призначена функція **FindFirstFile**:

```
HANDLE FindFirstFile(

  LPCTSTR lpFileName,                // вказівник на ім'я файлу для пошуку
  LPWIN32_FIND_DATA lpFindFileData   // вказівник на повернуту інформацію
);
```

Параметр lpFileName вказує на ім'я файлу (допускаються маски пошуку **?** та *****). Параметр lpFindFileData вказує на структуру **WIN32_FIND_DATA**, яка містить повернуту інформацію. Сама функція повертає дескриптор пошуку, який використовується при наступних пошуках. У випадку помилки пошуку повертається значення **INVALID_HANDLE_VALUE**.

Продовжується пошук файлів за допомогою функції **FindNextFile**:

```
BOOL FindNextFile(
```

```
  HANDLE hFindFile,                    // дескриптор пошуку
  LPWIN32_FIND_DATA lpFindFileData     // вказівник на структуру для даних пошуку
);
```

Завершується пошук функцією:

```
BOOL FindClose(

  HANDLE hFindFile                     // дескриптор пошуку
);
```

Керування каталогами

Для ***створення каталогу*** використовують функцію:

```
BOOL CreateDirectory(

  LPCTSTR lpPathName,                          // вказівник на рядок шляху каталогу
  LPSECURITY_ATTRIBUTES lpSecurityAttributes   // вказівник на дескриптор безпеки
);
```

Дескриптор безпеки це структура SECURITY_ATTRIBUTES, яка задає атрибути безпеки для каталогу, що створюється. Якщо lpSecurityAttributes = NULL, використовуються параметри за умовчанням. Файлова система диску, на якому створюється каталог, повинна підтримувати атрибути безпеки.

Для ***видалення каталогу*** використовують функцію:

```
BOOL RemoveDirectory(

  LPCTSTR lpPathName                   // адреса імені каталогу, який потрібно видалити
);
```

Видалити можна лише порожній каталог.

Одержати поточний каталог можна за допомогою функції:

```
DWORD GetCurrentDirectory(

  DWORD nBufferLength,                 // розмір буфера імені каталогу в символах
  LPTSTR lpBuffer                      // адреса буфера для поточного каталогу
);
```

Функція повертає довжину рядка з іменем каталогу. У випадку помилки повертає нуль. Якщо буфер недостатній для прийому імені каталогу, повертає необхідний розмір буфера.

Наступна функція ***встановлює поточний каталог*** для процесу:

```
BOOL SetCurrentDirectory(

  LPCTSTR lpPathName                   // адреса імені нового поточного каталогу
);
```

Шлях може бути абсолютним (C:\masm32\bin) або відносним (\\Thunder\Stuff).

Завдання для виконання роботи

1. Створити програму для копіювання файлів із використанням бібліотечних функцій C **fread**, **fwrite** (див зразок у додатку 1).
2. Створити програму для копіювання файлів із використанням функцій WinAPI **ReadFile()** та **WriteFile()**. (див зразок у додатку 2).
3. Створити програму для копіювання файлів із використанням допоміжної функції Windows **CopyFile()**.
4. Створити програму для пошуку файлу в заданому каталозі.
5. Створити програму для створення підкаталогу в заданому каталозі і копіювання файлу у створений підкаталог.

Контрольні питання

1. Як організована робота з файлами стандартними засобами мови С?
2. Які функції Win API для роботи з файлами ви знаєте?
3. Для чого використовують функцію **CreateFile()** і які вона має параметри?
4. Як діє функція **CreateFile()** у випадку, якщо ім'я створюваного файлу співпадає з існуючим на диску?
5. Які функції Win API застосовують для читання-запису даних у файли?
6. Які ви знаєте алгоритми копіювання файлу і які засоби вам для цього потрібні?
7. Як можна перемістити файл в інший каталог? На інший диск?
8. Як можна визначити розмір існуючого файлу? Чи можна його змінити?
9. Як можна знайти файл із заданим розширенням на диску?
10. Чи можна програмно дізнатися про дату створення файлу?

Завдання для самостійної роботи

1. Ознайомитись із функціями WinAPI **GetSystemTime()**, **GetLocalTime()** та структурою SYSTEMTIME.
2. Створити програму для визначення часу копіювання файлу із одного каталогу в інший.

Додаток 1

Програма копіювання файлів `cpC.c` (з використанням функцій С)

```
/* Basic cpC file copy program. C library functions. */
/* cp file1 file2: Copy file1 to file2. */

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

int main (int argc, char *argv [])
{
   FILE *in_file, *out_file;
   char rec [BUF_SIZE];
   size_t bytes_in, bytes_out;
   if (argc != 3) {
      printf ("Usage: cpC file1 file2\n");
      return 1;
   }
   in_file = fopen (argv [1], "rb");
   if (in_file == NULL) {
      perror (argv [1]);
      return 2;
   }
   out_file = fopen (argv [2], "wb");
   if (out_file == NULL) {
      perror (argv [2]);
      return 3;
   }
/* Process the input file a record at a time. */

   while ((bytes_in = fread (rec, 1, BUF_SIZE, in_file)) > 0) {
      bytes_out = fwrite (rec, 1, bytes_in, out_file);
      if (bytes_out != bytes_in) {
         perror ("Fatal write error.");
         return 4;
      }
   }
   fclose (in_file);
```

```
   fclose (out_file);
   return 0;
}
```

Додаток 2

Програма копіювання файлів `cpW.c` (з використанням функцій WinAPI)

```
/* Basic cpW file copy program. Win32 functions. */
/* cpW file1 file2: Copy file1 to file2. */

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

int main (int argc, LPTSTR argv [])
{
 HANDLE hIn, hOut;
 DWORD nIn, nOut;
 CHAR Buffer [BUF_SIZE];
 if (argc != 3) {
    printf ("Usage: cpW 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) {
    printf ("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) {
    printf ("Cannot open output file. Error: %x\n", GetLastError ());
    return 3;
 }
 while (ReadFile (hIn, Buffer, BUF_SIZE, &nIn, NULL) && nIn > 0) {
    WriteFile (hOut, Buffer, nIn, &nOut, NULL);
    if (nIn != nOut) {
       printf ("Fatal write error: %x\n", GetLastError ());
       return 4;
    }
 }
 CloseHandle (hIn);
 CloseHandle (hOut);
 return 0;
}
```