Лабораторна робота № 4

Тема: Робота з файлами та файловими системами.

Організація файлів, що відображені в пам’ять.

Мета: Навчитися розробляти програми реалізації операцій з файлами та каталогами.

Обладнання та програмне забезпечення: ПК, операційна система Windows, середовище розробки Visual Studio.

Вказівки для самостійної підготовки

Під час підготовки необхідно повторити теоретичний матеріал:

1. Операції з файлами і каталогами.
2. Міжпроцесова взаємодія.
3. Файли, що відображаються у пам'ять.
4. Поіменовані канали.

Теоретичні відомості

1 Операції з файлами і каталогами

1.1 Загальні відомості про файлові операції

Основні файлові операції у прикладних програмах:

Відкриття файлу. Після відкриття файлу процес може із ним працювати (робити читання і записування). Операція передбачає завантаження в ОЗП спеціальної структури даних — ***дескриптора файлу,*** який визначає його атрибути та місце розташування на диску. Наступні виклики використовуватимуть цю структуру для доступу до файлу.

Закриття файлу. Структуру даних, створену під час його відкриття, вилучають із пам'яті. Усі не збережені зміни записують на диск.

Створення файлу. Операція створює на диску новий файл нульової довжини. Створений файл автоматично відкривають.

Вилучення файлу. Операція вилучає файл і вивільняє зайнятий ним дисковий простір. Вона недопустима для відкритих файлів.

Читання з **файлу.** Операція пересилає певну кількість байтів із файлу, починаючи із поточної позиції, у заздалегідь виділений для цього буфер пам'яті режиму користувача.

Записування у файл. Дані із поточної позиції записують у файл із заздалегідь виділеного буфера. Якщо на цій позиції вже є дані, вони будуть перезаписані

Переміщення покажчика поточної позиції. Перед операціями читання і записування слід визначити, де у файлі перебувають потрібні дані або куди треба їх записати, задавши за допомогою цієї операції поточну позицію у файлі.

Отримання і завдання атрибутів файлу. Ці дві операції дають змогу зчитувати поточні значення всіх або деяких атрибутів файлу або задавати для них нові значення.

1.2 Файлові операції Win API

1.2.1 Відкриття і створення файлів

Функція ***CreateFile()*** створює файл:

```
HANDLE CreateFileA(
  [in]           LPCSTR                lpFileName,
  [in]           DWORD                 dwDesiredAccess,
  [in]           DWORD                 dwShareMode,
  [in, optional] LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  [in]           DWORD                 dwCreationDisposition,
  [in]           DWORD                 dwFlagsAndAttributes,
  [in, optional] HANDLE                hTemplateFile
);
```

`lpFileName –` назва файлу.

Параметр `dwDesiredAccess` може набувати значень **GENERIC_READ** (читання) і **GENERIC_WRITE** (запис).

Параметр `dwShareMode` може набувати значень:

CREATE_NEW - якщо файл є, повернути помилку, інакше створити новий;

CREATE_ALWAYS - створити новий файл, навіть якщо такий уже є;

OPEN_EXISTING - якщо файл є, відкрити його, якщо немає, повернути помилку;

OPEN_ALWAYS - якщо файл є, відкрити його, інакше створити новий.

Під час створення файлу значенням параметра `dwFlagsAndAttributes` може бути **FILE_ATTRIBUTE_NORMAL**, що означає створення файлу зі стандартними атрибутами.

Ця функція повертає дескриптор відкритого файлу. У разі помилки буде повернуто певне значення **INVALID_HANDLE_VALUE**, рівне -1.

// відкриття наявного файлу

```
HANDLE infile = CreateFile("infile.txt",GENERIC_READ, 0,
NULL,OPEN_EXISTING, 0, 0);
```

// створення нового файлу

```
HANDLE outfile = CreateFi1e("outflle.txt“,GENERIC_WRITE, 0, NULL,
CREATE_ALWAYS,FILE_ATTRIBUTE_NORMAL, 0);
```

1.2.2 Закриття файлу

Для закриття дескриптора файлу застосовують функцію CIoseHandle():

```
BOOL CloseHandle(
  [in] HANDLE hObject
);
```

1.2.3 Читання і запис даних

Для читання з файлу використовують функцію ***ReadFile():***

```
BOOL ReadFile(
  [in]                HANDLE       hFile,
  [out]               LPVOID       lpBuffer,
  [in]                DWORD        nNumberOfBytesToRead,
  [out, optional]     LPDWORD      lpNumberOfBytesRead,
  [in, out, optional] LPOVERLAPPED lpOverlapped
);
```

BOOL ReadFile(

HANDLE `hFile`, // ім'я файлу

LPCVOID `lpBuffer`, // буфер для розміщення прочитаних даних

DWORD `nNumberOfBytesToRead`, // кількість байтів, які потрібно прочитати

LPDWORD `lpNumberOfBytesRead`,

LPOVERLAPPED `lpOverlapped`

Виклик ***ReadFile()*** поверне TRUE у разі успішного читання.

```
char buf[100]; DWORD bytes_read;
ReadFile(outfile, buf, sizeof(buf), &bytes_read,
0);if (bytes_read != sizeof(buf))
printf("Досягнуто кінця файла\n");
```

1.2.4 Копіювання файлів

Приклад реалізації копіювання файлів засобами Win32API

```
char buf[1024];
DWORD bytes_read, bytes_written;
HANDLE infile = CreateFile("infile.txt", GENERIC_READ, 0, 0,
                                                    OPENEXISTING, 0, 0);
if (infile == INVALID_HANDLE_VALUE) {
   printf("Помилка під час відкриття файлу\n"); exit(-1);
   } HANDLE outfile = CreateFile("outfile.txt",
   GENERIC_WRITE, 0, 0,
                    CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0);
do {
   ReadFile(infile, buf, sizeof(buf),
   &bytes_read())if (bytes_read >0)
      WriteFile(outfile, buf, bytes_read, &bytes_written, 0);
} while (bytes_read >0);CloseHandle(infile); CloseHandle(outfile);
```

Для запису у файл використовують функцію ***WriteFile():***

```
BOOL WriteFile(
  [in]                HANDLE       hFile,
  [in]                LPCVOID      lpBuffer,
  [in]                DWORD        nNumberOfBytesToWrite,
  [out, optional]     LPDWORD      lpNumberOfBytesWritten,
  [in, out, optional] LPOVERLAPPED lpOverlapped
);
```

BOOL WriteFile(

HANDLE `hFile`, // буфер, з якого буде йти запис

LPCVOID `lpBuffer`, // буфер, з якого буде йти запис

DWORD `nNumberOfBytesToWrite`, // обсяг записуваних даних

LPDWORD `lpNumberOfBytesWritten`, //адреса записаних байтів

LPOVERLAPPED `lpOverlapped`);

Виклик ***WriteFile()*** поверне TRUE, якщо запис успішний.

```
DWORD bytes_written;
```

```
WriteFile(outfile, "hello", sizeof('hello"), &bytes_written, 0);
```

1.2.5 Прямий доступ до файлу

Прямий доступ до файлу реалізований функцією ***SetFilePointer():***

```
DWORD SetFilePointer(
  [in]                HANDLE hFile,
  [in]                LONG   lDistanceToMove,
  [in, out, optional] PLONG  lpDistanceToMoveHigh,
  [in]                DWORD  dwMoveMethod
);
```

DWORD SetFilePointer(

HANDLE `hFile`,

LONG `lDistanceToMove`, // величина переміщення покажчика

PLONG `lpDistanceToMoveHigh`,

DWORD `dwMoveMethod`); //режим переміщення покажчика

Константи режиму переміщення визначені як **FILE_BEGIN**, **FILE_CURRENT** і **FILE_END**.

```
//переміщення покажчика позиції на 100 байт від початку //файлу
SetFilePointer(outfile, 100, NULL, FILE_BEGIN);
// запис у файл
WriteFile(“outfile, "hello", sizeof('hello"), &bytes_written, 0);
```

1.2.6 Збирання інформації про атрибути файлу

Атрибути файлу визначаються функцією ***GetFileAttributesEx().***

```
BOOL GetFileAttributesExA(
  [in]  LPCSTR                 lpFileName,
  [in]  GET_FILEEX_INFO_LEVELS fInfoLevelId,
  [out] LPVOID                 lpFileInformation
);
```

Вона заповнюєструктуру **WIN32_FILE_ATTRIBUTE_DATA** з полями:

dwFileAttributes — маска прапорців (для каталогів задають прапорець **FILE_ATTRIBUTE_DIRECTORY**);

ftCreationTime, **ftLastAccessTime**, **ftLastWriteTime** — час створення, доступу і

модифікації файлу (структури типу FILETIME);

nFileSizeLow - розмір файлу).

```
WIN32_FILE_ATTRIBUTE_DATA attrs;
// другий параметр завжди задають однаково
GetFileAttributesEx(”myfile”, GetFileExInfoStandard,
&attrs);if (attrs.dwFileAttributes &
FILE_ATTRIBUTE_DIRECTORY)
   printf(“myfile є
каталогом\n”);else
  printf("розмір файла: %d\n", attrs, nFileSizeLow);
```

Для отримання доступу до атрибутів відкритого файлу за його дескриптором використовують функцію ***GetFileInformationByHandle()***. Вона дає змогу отримати повнішу інформацію із файлу. Є також функції, що повертають значення конкретних атрибутів:

```
BOOL GetFileInformationByHandle(
  [in]  HANDLE                      hFile,
  [out] LPBY_HANDLE_FILE_INFORMATION lpFileInformation
);
```

```
long size = FileSize(fh); // розмір файлу за його дескриптором
```

1.3 Операції з каталогами

1. **Створення нового каталогу.** Операція створює новий каталог.
2. **Вилучення каталогу.** На рівні системного виклику операція дозволена тільки для порожніх каталогів.
3. **Відкриття і закриття каталогу.** Каталог має бути відкритий перед використанням і закритий після використання.
4. **Читання елемента каталогу.** Операція зчитує один елемент каталогу і переміщує поточну позицію на наступний елемент. В циклі можна обійти весь каталог.
5. **Перехід у початок каталогу.** Операція переміщує поточну позицію до першого елемента каталогу.

1.4 Робота з каталогами у Win API

Створення каталогу виконує функція ***CreateDirectory().*** Параметри - шлях до каталогу та атрибути безпеки.

```
BOOL CreateDirectory(
  [in]           LPCTSTR               lpPathName,
  [in, optional] LPSECURITY_ATTRIBUTES lpSecurityAttributes
);
```

```
if (!CreateDirectory("c:\\newdir", 0))
    printf("Помилка під час створення каталогу\n");
```

Вилучення порожнього каталогу за його іменем виконує функція ***RemoveDirectory().*** Якщо каталог не порожній, ця функція не вилучає його і повертає *FALSE*.

```
BOOL RemoveDirectoryA(
  [in] LPCSTR lpPathName
);
```

```
if (!RemoveDirectory("c:\\dir"))
printf("Помилка у разі вилучення каталогу\n");
```

Відкривають каталог функцією ***FindFirstFile().***

```
HANDLE FindFirstFileA(
  [in]  LPCSTR             lpFileName,
  [out] LPWIN32_FIND_DATAA lpFindFileData
);
```

`lpFileName` - набір файлів.

`lpFindFileData` —покажчик на структуру, що заповнюється інформацією про знайденийфайл.

Функція повертає *дескриптор пошуку*.

Для доступу до файлу в каталозі використовують функцію ***FindNextFile()***, у яку передають дескриптор пошуку і таку саму структуру, як у FindFirstFile():

```
BOOL FindNextFileA(
  [in]  HANDLE             hFindFile,
  [out] LPWIN32_FIND_DATAA lpFindFileData
);
```

Якщо файлів більше немає, ця функція повертає FALSE.

Після закінчення пошуку потрібно закрити дескриптор пошуку за допомогою ***FindClose()***.

```
BOOL FindClose(
  [in, out] HANDLE hFindFile
);
```

Приклад обходу каталогу в Win32 API:

```
WIN32_FIND_DATA fattrs;
HANDLE findh = FindFirstFile("c:\\mydir\\*",
 &fattrs);do {
printf("%s\n", fattrs.cFileName); //вивід імені елемента
 } while (FindNextFile(findh,
&fattrs));FindClose(findh);
```

2 Міжпроцесова взаємодія

2.1 Файлові блокування

Файлові блокування - засіб синхронізації процесів, які здійснюють доступ до одного й того самого файлу. Процес може заблокувати файл, тоді інші процеси не зможуть отримати доступу до файлу доти, поки з нього не буде зняте блокування.

Є ***консультативне*** і ***обов'язкове*** блокування файлів.

Консультативне блокування підтримується на рівні процесів режиму користувача. Для синхронізації всі процеси перед доступом до файлу перевіряють наявність блокування (якщо блокування відсутнє, процес запроваджує своє блокування, виконує дії з файлом і знімає блокування).

Обов'язкове блокування підтримують на рівні ядра. Коли процес запровадив обов'язкове блокування, жодні операції з файлом будуть неможливі доти, поки блокування не буде зняте.

Для файлових блокувань є режими для читання і для запису.

Файлові блокування у Win

У Windows для обов'язкових файлових блокувань використовують функцію ***LockFileEx(),*** а для консультативних – ***LockFile()***.

```
BOOL LockFileEx(
  [in]      HANDLE       hFile,
  [in]      DWORD        dwFlags,
            DWORD        dwReserved,
  [in]      DWORD        nNumberOfBytesToLockLow,
  [in]      DWORD        nNumberOfBytesToLockHigh,
  [in, out] LPOVERLAPPED lpOverlapped
);
```

BOOL LockFileEx (HANDLE fh, DWORD flags, DWORD dummy, DWORD lcount, DWORD hcount, LPOVERLAPPED ov);

`hFile` — дескриптор відкритого файлу;

`dwFlags` — прапори режиму блокування

LOCKFILE_EXCLUSIVE_LOCK - блокування для запису (якщо значення не задане, встановлюють блокування для читання),

LOCKFILE_FAIL_IMMEDIATELY - повернути нуль, якщо файл уже заблокований;

`nNumberOfBytesToLockLow` — кількість заблокованих байтів;

`lpOverlapped` - покажчик на структуру типу OVERLAPPED, поле **offset** якої визначає зсувзаблокованої ділянки від початку файлу.

LockFileEx() повертає нуль, якщо блокування запровадити не вдалося.

Для розблокування використовують функцію ***UnLockFileEx()*** з тими самими параметрами, за винятком flags.

```
BOOL UnlockFileEx(
  [in]      HANDLE       hFile,
            DWORD        dwReserved,
  [in]      DWORD        nNumberOfBytesToUnlockLow,
  [in]      DWORD        nNumberOfBytesToUnlockHigh,
  [in, out] LPOVERLAPPED lpOverlapped
);
```

```
HANDLE fh = CreateFile("lockfile", GENERIC_WRITE, ... );
OVERLAPPED ov = { 0 };
 long size = FileSize(fh);
// задати блокування всього файлу для запису
 LockFileEx(fh, LOCKFILE_EXCLUSIVE_LOCK, 0, size,
 0, &ov);
// зняти блокування
```

```
UnLockFileEx(fh, 0, size, 0,
&ov);CloseHandle (fh);
```

3 Файли, що відображаються у пам'ять

3.1 Особливості інтерфейсу відображуваної пам'яті Win API

Функція відображення файлу в пам'ять ***MapViewOfFile().*** Параметр - дескриптор спеціального *об'єкта відображення,* що створюється за допомогою функції ***CreateFileMapping().***

Під час створення об'єкта відображення вказують дескриптор відкритого файлу; далі цей об'єкт використовується для відображення кількох регіонів одного файлу.

Особливості функції **CreateFileMapping():**

```
HANDLE CreateFileMappingA(
  [in]           HANDLE                hFile,
  [in, optional] LPSECURITY_ATTRIBUTES lpFileMappingAttributes,
  [in]           DWORD                 flProtect,
  [in]           DWORD                 dwMaximumSizeHigh,
  [in]           DWORD                 dwMaximumSizeLow,
  [in, optional] LPCSTR                lpName
);
```

`hFile` — дескриптор файлу, відкритого для читання;

`flProtect` - режим доступу до пам'яті;

`dwMaximumSizeHigh`, `dwMaximumSizeLow` — розмір файлу;

`lpName` - ім'я відображення.

CreateFileMapping() повертає дескриптор об'єкта відображення.

Після створення об'єкта відображення необхідно задати відображення у пам'ять за допомогою ***MapViewOfFile():***

```
LPVOID MapViewOfFile(
  [in] HANDLE hFileMappingObject,
  [in] DWORD  dwDesiredAccess,
  [in] DWORD  dwFileOffsetHigh,
  [in] DWORD  dwFileOffsetLow,
  [in] SIZE_T dwNumberOfBytesToMap
);
```

PVOID MapViewOfFile (HANDLE fmap, DWORD prot, DWORD off_high, DWORD off_low, SIZE_T len);

`hFileMappingObject` - дескриптор об'єкта відображення;

`dwDesiredAccess` - режим відображення;

`dwFileOffsetLow`, `dwFileOffsetHigh` - зсув відображуваного регіону від початку файлу;

`dwNumberOfBytesToMap` — довжина відображуваного регіону у байтах (якщо len=0, відображаютьділянку до кінця файлу).

Ця функція повертає покажчик на початок відображеної ділянки пам'яті. Відміна відображення виконується функцією ***UnmapViewOfFile():***

```
BOOL UnmapViewOfFile(
  [in] LPCVOID lpBaseAddress
);
```

BOOL UnmapViewOfFile(PVOID start));

start — початкова адреса ділянки відображення.

Після використання дескриптор файлу і дескриптор об'єкта відображення потрібно закрити за допомогою ***CIoseHandle().***

```
BOOL CloseHandle(
  [in] HANDLE hObject
);
```

Дескриптор файлу можна закрити і довиклику ***MapViewOfFile().***

Для роботи з відображуваною пам'яттю у Win необхідно.

1. Відкрити файл для читання і для запису:

HANDLE fh = CreateFile ("myfile.txt", GENERIC_READ| GENERIC_WRITE, 0, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0);

2. Створити об'єкт відображення для файлу:

HANDLE mh = CreateFileMapping (fh, 0, PAGE_READWRITE, 0,len, NULL);

3. Закрити файл: *CloseHandle(fh);*

4. Відобразити файл або його фрагмент у пам'ять:

*int *fmap = (int*)MapViewOfFile(mh, FILE_MAP_WRITE, 0, 0, len);*

5. Організувати доступ до файлу через відображувану пам'ять:

*fmap[0] = 100; fmap[1] = fmap[0] * 2;*

Якщо відображення більше не потрібне — закрити відповідний дескриптор:

CloseHandle(fh);

3.2 Реалізація розподілюваної пам'яті у Win API

Можна задавати об'єкти відображення, які використовують не конкретний дисковий файл, а файл підкачування; для цього слід передати у функцію **CreateFileMapping()** замість дескриптора файлу значення INVALID_HANDLE_VALUE (-1).

Об'єктам відображення під час їхнього створення можна давати імена, після чого вони можуть використовуватися кількома процесами: перший створює об'єкт і задає його ім'я (як параметр **CreateFileMapping()**), а інші відкривають його за допомогою функції **OpenFileMapping(),** передавши як параметр те саме ім'я. Функцію **OpenFileMapping()** використовують для отримання доступу до наявного об'єкта відображення:

```
HANDLE OpenFileMappingW(
  [in] DWORD   dwDesiredAccess,
  [in] BOOL    bInheritHandle,
  [in] LPCWSTR lpName
);
```

`dwDesiredAccess` — аналогічний відповідному параметру MapViewOfFile();

`lpName` — ім'я об'єкта відображення.

3.3 Переваги та недоліки відображуваної пам'яті

Робота із такими файлами зводиться до прямого звертання до пам'яті через покажчики, ніби файл був частиною адресного простору процесу. Під час роботи зі звичайними файлами для виконання одних і тих самих дій потрібно відкривати файл, переміщувати покажчик поточної позиції, виконувати читання і запис та закривати файл.

Робота із відображуваними файлами може бути реалізована ефективніше.

Це пов'язано з тим, що:

1. для роботи з ними достатньо виконати один системний виклик **MapViewOfFile(),** а далі працювати із ділянкою пам'яті в адресному просторі процесу; водночас для відкриття файлу, читання, запис тощо потрібно виконувати окремі системні виклики;
2. не потрібно копіювати дані між системною пам'яттю і буфером режиму користувача, що необхідно для звичайних операцій читання і запису файлу — файл безпосередньо відображається на адресний простір процесу.
3. За допомогою відображуваних файлів можна легко реалізувати розподіл пам'яті між

процесами, якщо відобразити один і той самий файл на адресний простір кількох із них.

4. Головним недоліком цієї технології є те, що відображуваний файл не може бути розширений позиціонуванням покажчика поточної позиції за його кінець і виконанням операції запису (підсистема віртуальної пам'яті у цьому випадку не може визначити точну довжину файлу).

4 Поіменовані канали

4.1 Поіменовані канали Win API

Поіменовані канали Win32 реалізують двобічний обмін повідомленнями і не використовують прямо файли файлової системи. Створення екземпляра каналу на сервері виконує функція ***CreateNamedPipe():***

```
HANDLE CreateNamedPipeA(
  [in]           LPCSTR                lpName,
  [in]           DWORD                 dwOpenMode,
  [in]           DWORD                 dwPipeMode,
  [in]           DWORD                 nMaxInstances,
  [in]           DWORD                 nOutBufferSize,
  [in]           DWORD                 nInBufferSize,
  [in]           DWORD                 nDefaultTimeOut,
  [in, optional] LPSECURITY_ATTRIBUTES lpSecurityAttributes
);
```

`lpName` — ім'я каналу на локальному комп'ютері;

`dwOpenMode` — режим відкриття каналу (прапорець PIPE_ACCESS_DUPLEX означає двобічний зв'язок);

`dwPipeMode` — режим обміну даними із каналом (для читання із каналу PIPE_READMODE_BYTE, для запису - PIPE_TYPE_BYTE);

`nMaxInstances` - максимальна кількість клієнтських з'єднань із цим каналом (необмежена кількість PIPE_UNLIMITED_INSTANCES);

`nDefaultTimeOut` — максимальний час очікування клієнтом доступу до каналу (необмеженеочікування NMPWAIT_WAIT_FOREVER).

CreateNamedPipe() повертає дескриптор створеного каналу:

```
HANDLE ph = CreateNamedPipe("\\\\.\\pipe\\mypipe", PIPE_ACCESS_DUPLEX,
PIPE_READMODE_BYTE, 5, 0, 0, NMPWAIT_WAIT_FOREVER, NULL);
```

Для очікування клієнтського з'єднання необхідно виконати функцію ***Connect***

NamedPipe():

```
BOOL ConnectNamedPipe(
  [in]                HANDLE       hNamedPipe,
  [in, out, optional] LPOVERLAPPED lpOverlapped
);
```

Як тільки клієнт підключиться до цього каналу, очікування буде завершене, і можна читати з каналу або записувати в канал за допомогою **ReadFile()** і **WriteFile().** Після завершення обміну даними сервер закриває з'єднання, викликавши функцію **DisconnectNamedPipe().**

```
BOOL DisconnectNamedPipe(
  [in] HANDLE hNamedPipe
);
```

Після завершення роботи із каналом його дескриптор потрібно закрити за допомогою **CloseHandle()**.

5. Приклади програмної реалізації планування процесів та потоків

Приклад 1. Видалення файлу

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

int main()
{
  //видаляємо файл
  if(!DeleteFile("c:\\demo_file.txt"))
   {
    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.";
  cout << "Press any key to
  finish";cin.get();
```

```
    return 0;
}
```

Приклад 2. Створення та запис файлу

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

int main()
{
  HANDLE hFile;

  //створюємо файл для запису
  данихhFile = CreateFile(
  "d:\\demo_file.tx
  t",
  GENERIC_WRITE,
  0,
  NU
  LL,
  CREATE_NEW,
  FILE_ATTRIBUTE_NORM
  AL,NULL
  );

  //перевіряємо на успішне
  створенняif(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 у
  файлfor(int i=0; i<10;
  ++i)
  {
    DWORD
```

```
        dwBytesWrite;
        if(!WriteFile(
              hFile,
              &i,
              sizeof
              (i),
              &dwBytesWrite,
              (LPOVERLAPPED)
              NULL
              ))
      {
       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
   файлуCloseHandle(hFile);
   cout << "The file is created and written." <<
   endl;cin.get();
   return 0;
}
```

Приклад 3. Створення та читання файлу

```
#include <vcl.h>
#include
<windows.h>
#include
<iostream.h>
int main()
{
  HANDLE hFile;
  //створюємо файл для запису
  данихhFile = CreateFile(
  "d:\\demo_file.tx
  t",
```

```
GENERIC_READ,
0,
NULL,
OPEN_EXISTIN
G,
FILE_ATTRIBUTE_NORM
AL,NULL
);
//перевіряємо на успішне
створенняif(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
 файлуfor(; ;)
 {
   DWORD
   dwBytesRead;int n;
   //читаємо один
   записif(!ReadFile(
         hFile,
         &n,
         sizeof(
         n),
         &dwBytesRead,
         (LPOVERLAPPED)
         NULL
         ))
  {
   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
    файлуif(dwBytesRead == 0)
    //якщо так, то виходимо з
    циклуbreak;
    //інакше виводимо запис на
    консольcout << n << ' ';
  }
    cout << endl;
    //закриваємо дескриптор
    файлуCloseHandle(hFile);
    cout << "The file is opened and read." << endl;
    cin.get();
    return 0;
  }
```

Приклад 4. Копіювання файлу

```
#include <vcl.h>
#include
<windows.h>
#include
<iostream.h>
int main()
{
  //копіюємо
  файл
  if(!CopyFile(
  "d:\\demo_file.txt", "d:\\new_demo_file.txt", 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;cin.get();
   return 0;
}
```

Приклад 5. Переміщення файлу

```
#include <vcl.h>
#include
<windows.h>
#include
<iostream.h>int
main()
{
  //переміщаємо
  файлif(!MoveFile(
  "d:\\demo_file.txt", "d:\\new_demo_file.txt", 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 moved." <<
   endl;cin.get();
   return 0;
}
```

Приклад 6. Установка покажчика позиції файлу за допомогою функції SetFilePointer

```
#include <vcl.h>
#include
<windows.h>
#include
<iostream.h>
int main()
{
  HANDLE
  hFile;long n;
  long p;
  DWORD
  dwBytesRead;int m;
  //відкриваємо файл для читання
  данихhFile = CreateFile(
  "d:\\demo_file.tx
```

```
t",
GENERIC_READ,
0,
NU
LL,
OPEN_EXISTING,
FILE_ATTRIBUTE_NORM
AL,NULL
);
//перевіряємо на успішне
створенняif(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
записуcout << "Input a number from
0 to 9"; cin >> n;
//зміщуємо покажчик позиції файлу
p = SetFilePointer(hFile, n*sizeof(n), 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
файлуcout << "File pointer: " << p << endl;
//читаємо дані з
файлуif(!ReadFile(
        hFile,
        &m,
        sizeof(
        m),
        &dwBytesRead,
```

```
            (LPOVERLAPPED)
            NULL
            ))
        {
         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
        консольcout << "The read number: " <<
        m << endl;
        cin.get();
       //закриваємо дескриптор
       файлуCloseHandle(hFile);
        cin.get();return 0;
    }
```

Приклад 7. Створення каталогу

```
    //створення
    каталогу
    #include <vcl.h>
    #include
    <windows.h>
    #include
    <iostream.h>
    int main()
    {
       //створюємо каталог
       if(!CreateDirectory("c:\\demo_dir", NULL))
       {
              cerr << "Create directory faled. " << endl
              << "The last error code: " << GetLastError()
              << endl;cout << "Press any key tp finish.";
              cin.get();return 0;
       }
       cout   << "The directory is created. "
       << endl;cin.get();
       return 0;
```

```
}
```

Приклад 8. Створення підкаталогу

```
//створення
підкаталогу
#include <vcl.h>
#include
<windows.h>
#include
<iostream.h>

int main()
{
   //створюємо підкаталог
   if(!CreateDirectoryEx("d:\\demo_dir", "d:\\demo_dir\\demo_subdir", NULL))
   {
            cerr << "Create directory faled. " << endl
            << "The last error code: " << GetLastError()
            << endl;cout << "Press any key to finish.";
            cin.get();
            return 0;
   }
   cout   << "The subdirectory is created. "
   << endl;cin.get();
   return 0;
}
```

Приклад 9. Пошук у каталозі

```
//пошук у
каталозі
#include
<vcl.h>
#include
<windows.h>
#include
<iostream.h>

int main()
{
   HANDLE
   hFindFile;
```

```
    WIN32_FIND_D
    ATA fd;
    //знаходимо перший файл
    hFindFile = FindFirstFile("c:\\demo_dir\\*.*", &fd);
    if(hFindFile == INVALID_HANDLE_VALUE)
    {
            cerr << "Finf first file faled. " << endl
            << "The last error code: " << GetLastError()
            << endl;cout << "Press any key tp finish.";
            cin.get();return 0;
    }
    //виводимо на консоль імя першого файлу
    cout    << "The first file name: " << fd.cFileName << endl;
    //знаходимо наступний файл та виводимо на консоль його
    імяwhile (FindNextFile(hFindFile, &fd))
      cout << "The next file name: " << fd.cFileName << endl;
    //закриваємо дескриптор
    пошукуFindClose(hFindFile);
    cin.get();
    return 0;
}
```

Приклад 10. Видалення каталогу

```
//видалення
каталогу
#include <vcl.h>
#include
<windows.h>
#include
<iostream.h>

int main()
{
    //видаляємо каталог
    if(!RemoveDirectory("c:\\demo_dir"))
    {
            cerr << "Remove directory faled. " << endl
            << "The last error code: " << GetLastError()
            << endl;cout << "Press any key tp finish.";
            cin.get();return 0;
```

```
    }
    cout    << "The directory is removed. "
    << endl;cin.get();
    return 0;
}
```

Приклад 11. Видалення каталогу з файлами

```
//видалення каталогу з
файлами#include <vcl.h>
#include
<windows.h>
#include
<iostream.h>

int main()
{
    HANDLE
    hFindFile;
    WIN32_FIND_D
    ATA fd;
    char szFillFileName[MAX_PATH];

    //знаходимо перший файл
    hFindFile = FindFirstFile("c:\\demo_dir\\*.*", &fd);
    if(hFindFile == INVALID_HANDLE_VALUE)
    {
            cerr << "Finf first file faled. " << endl
            << "The last error code: " << GetLastError()
            << endl;cout << "Press any key tp finish.";
            cin.get();
            return 0;
    }

    //виводимо на консоль ім'я першого файлу
    cout    << "The first file name: " << fd.cFileName << endl;

    //видаляємо з каталогу
    файли while
    (FindNextFile(hFindFile,
    &fd))
    {
      //якщо це не підкаталог, то видаляємо його
      if(!(fd.dwFileAttributes &
```

```
        FILE_ATTRIBUTE_DIRECTORY))
      {
         //формуємо ім'я файлу
        sprintf(szFillFileName, "c:\\demo_dir\\%s", fd.cFileName);
       //видаляємо файл
       if(!DeleteFile(szFillFileName))
       {
            cerr << "Delete file faled. " << endl
            << "The last error code: " << GetLastError() << endl;
            cout << "Press any key tp finish.";
            cin.get();
            return 0;
       }
      else
        cout << "The next file: " << fd.cFileName << "is deleted." << endl;
      }
      else
        cout << "The next directory: " << fd.cFileName << "is not deleted." << endl;
   }
      //закриваємо дескриптор
      пошуку
      if(!FindClose(hFindFile))
      {
        cout << "Find close failed. "
        << endl;return 0;
      }
      //видаляємо каталог
      if(!RemoveDirectory("c:\\demo_dir"))
       {
            cerr << "Remove directory faled. " << endl
            << "The last error code: " << GetLastError()
            << endl;cout << "Press any key tp finish.";
            cin.get();
            return 0;
      }
            cout << " The directory is removed. " << endl;
      //закриваємо дескриптор
      пошукуFindClose(hFindFile);
      cin.get();
      return 0;
   }
```

Приклад 12. Переміщення каталогу

```
//переміщення
каталогу#include
<vcl.h>
#include
<windows.h>
#include
<iostream.h>

int main()
{
   //переміщуємо каталог
   if(!MoveFile("c:\\demo_dir",
   "c:\\new_dir"))
   {
          cerr << "Move directory faled. " << endl
          << "The last error code: " << GetLastError()
          << endl;cout << "Press any key tp finish.";
          cin.get();return 0;
   }
   cout   << "The directory is moved. "
   << endl;cin.get();
   return 0;
}
```

Приклад 13. Визначення та установка поточного каталогу

```
//визначення та установка поточного
каталогу#include <vcl.h>
#include
<windows.h>
#include
<iostream.h>

int main()
{
   DWORD
   dwNumberOfChar;
   char
   szDirName[MAX_PATH
   ];
   //знаходимо імя поточного каталогу
   dwNumberOfChar = GetCurrentDirectory(MAX_PATH, szDirName);
```

```
        if(dwNumberOfChar == 0)
    {
        cerr << "Get current directory  faled. " << endl
        << "The last error code: " << GetLastError() << endl;
        cout << "Press any key tp finish.";
        cin.get();
        return 0;
    }
    //виводимо на консоль ім'я поточного каталогу
    cout    << "The current directory  name: " << szDirName << endl;
    //установлюємо поточний каталог для видалення з нього файлів
    if(!SetCurrentDirectory("c:\\demo_dir"))
    {
        cerr << "Set current directory  faled. " << endl
        << "The last error code: " << GetLastError() << endl;
        cout << "Press any key tp finish.";
        cin.get();
        return 0;
    }
    //визначаємо ім'я нового поточного каталогу
    dwNumberOfChar = GetCurrentDirectory(MAX_PATH,
    szDirName);
    {
        cerr << "Get current directory  faled. " << endl
        << "The last error code: " << GetLastError() << endl;
        cout << "Press any key tp finish.";
        cin.get();
        return 0;
    }
    //виводимо на консоль ім'я поточного каталогу
    cout << "The current directory  name: " << szDirName << endl;
    cin.get();
    return 0;
}
```

Завдання для студентів

1. Розробити програми для роботи з файлами і файловою системою. Для програмної реалізації використовувати середовище програмування Microsoft Visual Studio, мова програмування – C/C++, інтерфейс консольний.

2. Скласти звіт про виконання лабораторної роботи. Зміст звіту:

➢ опис функцій для роботи з файловою системою;

➢ постановка задачі;

➢ програмні коди;

➢ результати виконання програм;

➢ висновок.

3. До захисту лабораторної роботи підготувати відповіді на контрольні питання

Контрольні питання

1. Навести загальні відомості про файлові операції.
2. Охарактеризувати файлові операції Win32 API.
3. Як виконується відкриття і створення файлів? Навести функції, що використовуються для цих операцій, вказати параметри функцій.
4. Як виконується закриття файлу? Навести функції, що використовуються для цих операцій, вказати параметри функцій.
5. Як виконується читання і запис даних? Охарактеризувати функції, що використовуються для цих операцій, вказати параметри функцій.
6. Як виконується копіювання файлів? Охарактеризувати функції, що використовуються для цих операцій, вказати параметри функцій.
7. Як виконується прямий доступ до файлу? Охарактеризувати функції, що використовуються для цих операцій, вказати параметри функцій.
8. Як виконується збирання інформації про атрибути файлу? Охарактеризувати функції, що використовуються для цих операцій, вказати параметри функцій.
9. Як виконується створення каталогів? Навести функції, що використовуються для цих операцій, вказати параметри функцій.
10. Як виконується видалення каталогів? Навести функції, що використовуються для цих операцій, вказати параметри функцій.
11. Як виконується відкриття каталогів? Охарактеризувати функції, що використовуються для цих операцій, вказати параметри функцій.
12. Як виконується доступ до файлу у каталозі? Охарактеризувати функції, що використовуються для цих операцій, вказати параметри функцій.
13. Як виконується файлові блокування? Охарактеризувати функції, що використовуються для цих операцій, вказати параметри функцій.
14. Охарактеризувати особливості інтерфейсу відображуваної пам'яті Win32 API.
15. Як виконується реалізація розподілюваної пам'яті у Win32 API?
16. Назвати переваги та недоліки відображуваної пам'яті.
17. Охарактеризувати поіменовані канали.
18. Як виконується реалізація поіменованих каналів Win32 API? Охарактеризувати

функції, що використовуються для цих операцій, вказати параметри функцій.

Варіанти завдань:

1. Розробити програму, в якій будуть реалізовані наступні функції:
   створення файлів;
   вивід вмісту текстового файлу на екран;
   перевірити чи каталог прихований.
2. Розробити програму, в якій будуть реалізовані наступні функції:
   копіювання файлів;
   запис у текстовий файл поточного часу і дати;
   вивід вмісту каталогу на екран.
3. Розробити програму, в якій будуть реалізовані наступні функції:
   видалення файлів;
   перевірити чи файл прихований;
   визначити час і дату створення каталогу.
4. Розробити програму, в якій будуть реалізовані наступні функції:
   створення каталогів;
   перевірити чи файл тільки для читання;
   вивід вмісту каталогу на екран.
5. Розробити програму, в якій будуть реалізовані наступні функції:
   копіювання каталогів;
   перевірити чи файл архівований;
   зробити каталог тільки для читання.
6. Розробити програму, в якій будуть реалізовані наступні функції:
   видалення каталогів;
   перевірити чи файл архівований;
   вивід вмісту каталогу на екран.
7. Розробити програму, в якій будуть реалізовані наступні функції:
   створення файлів;
   перевірити чи файл шифрований;
   визначити час і дату створення каталогу.
8. Розробити програму, в якій будуть реалізовані наступні функції:
   копіювання файлів;
   визначити час і дату створення файлу;
   зробити каталог прихованим.
9. Розробити програму, в якій будуть реалізовані наступні функції:
   видалення файлів;
   визначити час і дату останньої зміни файлу;
   зробити каталог тільки для читання.
10. Розробити програму, в якій будуть реалізовані наступні функції:
    створення каталогів;
    визначити час і дату останнього відкриття файлу;
    вивід вмісту текстового файлу на екран.
11. Розробити програму, в якій будуть реалізовані наступні функції:
    копіювання каталогів;
    зробити файл прихованим;

запис у текстовий файл поточного часу і дати.

12. Розробити програму, в якій будуть реалізовані наступні функції:
    видалення каталогів;
    зробити файл тільки для читання;
    визначити час і дату створення файлу.
13. Розробити програму, в якій будуть реалізовані наступні функції:
    створення файлів;
    зробити файл тільки для читання;
    вивід вмісту каталогу на екран.
14. Розробити програму, в якій будуть реалізовані наступні функції:
    копіювання файлів;
    зробити файл прихованим;
    визначити час і дату створення каталогу.
15. Розробити програму, в якій будуть реалізовані наступні функції:
    видалення файлів;
    визначити час і дату останнього відкриття файлу;
    перевірити чи каталог тільки для читання.
16. Розробити програму, в якій будуть реалізовані наступні функції:
    створення каталогів;
    визначити час і дату останньої зміни файлу;
    перевірити чи файл прихований.
17. Розробити програму, в якій будуть реалізовані наступні функції:
    копіювання каталогів;
    визначити час і дату створення файлу;
    зробити файл прихованим.
18. Розробити програму, в якій будуть реалізовані наступні функції:
    видалення каталогів;
    перевірити чи файл шифрований;
    визначити час і дату останньої зміни файлу.
19. Розробити програму, в якій будуть реалізовані наступні функції:
    створення файлів;
    перевірити чи файл архівований;
    вивід вмісту каталогу на екран.
20. Розробити програму, в якій будуть реалізовані наступні функції:
    копіювання файлів;
    перевірити чи файл системний;
    визначити час і дату створення каталогу.
21. Розробити програму, в якій будуть реалізовані наступні функції:
    видалення файлів;
    перевірити чи файл тільки для читання;
    створення каталогів.
22. Розробити програму, в якій будуть реалізовані наступні функції:
    створення каталогів;
    перевірити чи файл прихований;
    вивід вмісту текстового файлу на екран.
23. Розробити програму, в якій будуть реалізовані наступні функції:
    копіювання каталогів;

перевірити чи файл прихований;
запис у текстовий файл поточного часу і дати.

24. Розробити програму, в якій будуть реалізовані наступні функції:
    видалення каталогів;
    запис у текстовий файл поточного часу і дати;
    перевірити чи файл архівований.

25. Розробити програму, в якій будуть реалізовані наступні функції:
    створення файлів;
    вивід вмісту текстового файлу на екран;
    зробити каталог прихованим.

26. Розробити програму, в якій будуть реалізовані наступні функції:
    копіювання файлів;
    запис у текстовий файл поточного часу і дати;
    зробити каталог тільки для читання.

27. Розробити програму, в якій будуть реалізовані наступні функції:
    видалення файлів;
    перевірити чи файл тільки для читання;
    створення каталогів.

28. Розробити програму, в якій будуть реалізовані наступні функції:
    створення каталогів;
    перевірити чи файл системний;
    визначити час і дату останнього відкриття файлу.

29. Розробити програму, в якій будуть реалізовані наступні функції:
    копіювання каталогів;
    визначити час і дату створення файлу;
    запис у текстовий файл поточного часу і дати.

30. Розробити програму, в якій будуть реалізовані наступні функції:
    видалення каталогів;
    визначити час і дату останньої зміни файлу;
    зробити файл тільки для читання.

Приклад виконання лабораторної роботи:

```
// Програма створює текстовий файл у заданому каталозі,
// записує у нього повідомлення "Hello world!",
// та виводить на екран інформацію про деякі його атрибути.

#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <Windows.h>

int main() {
    const char* fileName = "C:\\delete\\hello.txt";

    // Відкриття файлу для запису
    FILE* file = fopen(fileName, "w");
    if (file == NULL) {
        printf("Error opening file. Check the path and permissions.\n");
        return 1;
    }

    // Запис повідомлення у файл
    fprintf(file, "Hello world!\n");

    // Закриття файлу
    fclose(file);
```

```
    // Отримання інформації про атрибути файлу
    DWORD attributes = GetFileAttributesA(fileName);
    if (attributes != INVALID_FILE_ATTRIBUTES) {
        printf("File attributes:\n");
        if (attributes & FILE_ATTRIBUTE_HIDDEN) {
            printf("Hidden: Yes\n");
        }
        else {
            printf("Hidden: No\n");
        }

        if (attributes & FILE_ATTRIBUTE_ARCHIVE) {
            printf("Archive: Yes\n");
        }
        else {
            printf("Archive: No\n");
        }

        if (attributes & FILE_ATTRIBUTE_READONLY) {
            printf("Read-only: Yes\n");
        }
        else {
            printf("Read-only: No\n");
        }
    }
    else {
        printf("Error getting file attributes. Error code: %d\n", GetLastError());
    }

    return 0;
}
```