### МИНИСТРЕСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ

### Национальный аэрокосмический университет им. Н.Е.Жуковского

### «Харьковский авиационный институт»

### Кафедра 503

### Лабораторная работа №2

на тему «Изучение системных вызовов Win32 API работы с файлами»

### по курсу «Системное программирование»

### Выполнил: студент гр.535-А

### Наздрач В. Ю.

### Проверил: ассистент

### Трубилко А.В.

Постановка задачи:

### Программа 1:

Написать программу, реализующую произвольный доступ к записям в файле двумя способами: с помощью указателя файла (file pointer).

Структура записи:

* номер записи;
* время создания записи (в формате FILETYME);
* текстовая строка заданной длины (80 символов);
* счетчик, показывающий, сколько раз запись изменялась.

Запись может быть пустая (инициализирована нулями).

В заголовке файла хранить количество непустых записей в файле и размер файла. Общее количество записей в файле задается из командной строки. Пользователь должен иметь возможность удалять и модифицировать существующие записи, обращаясь к ним по номеру. Интерфейс с пользователем реализуется на усмотрение студента.

### Программа 2:

Написать программу, реализующую функцию файлового менеджера. Программа должна выдавать приглашение на ввод команды. Поддерживаемые команды:

* Сменить директорию
* Распечатать директорию
* Скопировать файл
* Создать директорию
* Удалить файл (пустую директорию)
* Вывести подробную информацию о файле

**Задание 1**

**Код программы:**

// task1.cpp: определяет точку входа для консольного приложения.

//

/\* TODO:

\*/

#include "stdafx.h"

#include "myDB.h"

#define FILEPATH TEXT("myNotes.ftw")

void readline(char str[], int str\_len);

int \_tmain(int argc, \_TCHAR\*\* argv)

{

setlocale(LC\_ALL, "RUS");

Header\* pH = (Header\*)LocalAlloc(LMEM\_FIXED, sizeof(Header));

INT operation;

HANDLE hF = openFile(FILEPATH, pH);

if (!hF || hF == INVALID\_HANDLE\_VALUE)

{

printf("Невозможно открыть указаный файл, создаем его...\n");

printf("Введите количество записей: ");

DWORD nOfNotes;

if (!scanf\_s("%u", &nOfNotes)) {

printf("Ошибка ввода значения!\n");

scanf("%\*[^\n]");

return 1;

}

hF = createNewFile(FILEPATH, nOfNotes, pH);

if (!hF || hF == INVALID\_HANDLE\_VALUE) {

printf("Невозможно создать файл!\n");

return 1;

}

}

for (;;)

{

DWORD dwID;

printf("\*\*\* Произвольный доступ к записям в файле \*\*\*\n");

printf("Создать запись - 1\nУдалить запись - 2\nИнформация по записи - 3\nВыход из программы - 0\nВыберите операцию: ");

scanf("%i", &operation);

switch (operation)

{

case 0: //Выход из программы

LocalFree(pH);

CloseHandle(hF);

return 0;

case 1: // Создание записи

printf("Введите ID записи: ");

scanf("%u", &dwID);

printf("Введите текст записи: ");

char szStr[STR\_MAX];

readline(szStr, STR\_MAX);

if (writeNote(hF, pH, dwID, szStr)) {

printf("Запись успешно добавлена.\n");

}

else {

printf("Невозможно добавить запись!\n");

}

system("pause");

break;

case 2: // Удаление записи

printf("Введите ID записи: ");

DWORD dwId;

scanf("%u", &dwId);

if (removeNote(hF, pH, dwId)) {

printf("Запись успешно удалена.\n");

}

else {

printf("Невозможно удалить запись!\n");

}

system("pause");

break;

case 3: // Вывод информации по записи

printf("Введите ID записи: ");

scanf("%u", &dwId);

printNoteInfo(hF, pH, dwId);

system("pause");

break;

default: //Неизвестная операция

printf("Выбрана неизвестная операция!\n");

system("pause");

}

system("cls");

}

CloseHandle(hF);

return 0;

}

void readline(char str[], int str\_len)

{

int i = 0, ch;

while (isspace(ch = getchar()));

while (ch != '\n')

{

if (i < str\_len) {

str[i++] = ch;

}

ch = getchar();

}

str[i] = '\0';

}

myDB.cpp

#include "stdafx.h"

#include "myDB.h"

HANDLE openFile(LPCTSTR lpFileName, Header\* pH)

{

HANDLE hFile = CreateFile(lpFileName, GENERIC\_READ | GENERIC\_WRITE, NULL, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if ((INVALID\_HANDLE\_VALUE != hFile) && hFile)

{

readHeader(hFile, pH);

}

return hFile;

}

HANDLE createNewFile(LPCTSTR lpFileName, DWORD dwNumberOfNotes, Header\* pH)

{

HANDLE hFile;

if (INVALID\_HANDLE\_VALUE != (hFile = CreateFile(lpFileName, GENERIC\_READ | GENERIC\_WRITE, NULL, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL)))

{

pH->dwCountNotes = 0;

pH->dwFileSize = dwNumberOfNotes \* sizeof(Note);

writeHeader(hFile, pH);

SetFilePointer(hFile, dwNumberOfNotes \* sizeof(Note) + sizeof(Header), 0, FILE\_BEGIN);

SetEndOfFile(hFile);

}

return hFile;

}

BOOL readHeader(HANDLE hF, Header\* pH)

{

if (SetFilePointer(hF, 0, 0, FILE\_BEGIN) == INVALID\_SET\_FILE\_POINTER) {

return FALSE;

}

if (!ReadFile(hF, &(pH->dwFileSize), sizeof(DWORD), NULL, NULL)) {

return FALSE;

}

return ReadFile(hF, &(pH->dwCountNotes), sizeof(DWORD), NULL, NULL);

}

BOOL writeHeader(HANDLE hF, Header\* pH)

{

if (SetFilePointer(hF, 0, 0, FILE\_BEGIN) == INVALID\_SET\_FILE\_POINTER) {

return FALSE;

}

if (!WriteFile(hF, &pH->dwFileSize, sizeof(DWORD), NULL, NULL)) {

return FALSE;

}

return WriteFile(hF, &pH->dwCountNotes, sizeof(DWORD), NULL, NULL);

}

BOOL readNote(HANDLE hF, Header\* pH, DWORD dwID, Note\* pNote)

{

if (dwID \* sizeof(Note) > pH->dwFileSize) {

return FALSE;

}

if (SetFilePointer(hF, sizeof(Header) + sizeof(Note) \* (dwID - 1), 0, FILE\_BEGIN) == INVALID\_SET\_FILE\_POINTER) {

return FALSE;

}

return ReadFile(hF, pNote, sizeof(Note), NULL, NULL);

}

BOOL writeNote(HANDLE hF, Header\* pH, DWORD dwID, CHAR\* szStr)

{

Note\* pNote;

if (dwID \* sizeof(Note) > pH->dwFileSize) {

return FALSE;

}

if (!(pNote = (Note\*)LocalAlloc(LMEM\_FIXED, sizeof(Note)))) {

return FALSE;

}

if (!readNote(hF, pH, dwID, pNote)) {

return FALSE;

}

if (pNote->dwId == 0) {

pNote->dwId = dwID;

pH->dwCountNotes++;

}

FileTimeToCurrentSystemTime(&pNote->lpLastChangeTime);

pNote->dwCountChanges++;

strcpy(pNote->szNote, szStr);

if (SetFilePointer(hF, sizeof(Header) + sizeof(Note) \* (dwID - 1), 0, FILE\_BEGIN) == INVALID\_SET\_FILE\_POINTER) {

return FALSE;

}

if (!WriteFile(hF, pNote, sizeof(Note), NULL, NULL)) {

return FALSE;

}

LocalFree(pNote);

return writeHeader(hF, pH);

}

BOOL removeNote(HANDLE hF, Header\* pH, DWORD dwID)

{

Note\* pNote;

if (dwID \* sizeof(Note) > pH->dwFileSize) {

return FALSE;

}

if (!(pNote = (Note\*)LocalAlloc(LMEM\_FIXED, sizeof(Note)))) {

return FALSE;

}

ZeroMemory(pNote, sizeof(Note));

if (SetFilePointer(hF, sizeof(Header) + sizeof(Note) \* (dwID - 1), 0, FILE\_BEGIN) == INVALID\_SET\_FILE\_POINTER) {

return FALSE;

}

if (!WriteFile(hF, pNote, sizeof(Note), NULL, NULL)) {

return FALSE;

}

LocalFree(pNote);

return writeHeader(hF, pH);

}

void printNoteInfo(HANDLE hF, Header\* pH, DWORD dwID)

{

setlocale(LC\_ALL, "RUS");

Note\* pNote;

SYSTEMTIME st;

if (!(pNote = (Note\*)LocalAlloc(LMEM\_FIXED, sizeof(Note)))) {

printf("Невозможно выделить память!\n");

return;

}

if (!readNote(hF, pH, dwID, pNote)) {

printf("Невозможно прочитать запись!\n");

return;

}

if (pNote->dwId == 0) {

printf("Запись #%u - пустая!\n", dwID);

return;

}

printf("ID: %u\n", pNote->dwId);

printf("Количество изменений: %u\n", pNote->dwCountChanges);

FileTimeToSystemTime(&pNote->lpLastChangeTime, &st);

printf("Последнее изменение: %d-%02d-%02d %02d:%02d:%02d\n", st.wYear, st.wMonth, st.wDay, st.wHour, st.wMinute, st.wSecond);

printf("Текст записи: %s\n", pNote->szNote);

LocalFree(pNote);

}

void FileTimeToCurrentSystemTime(PFILETIME ft)

{

SYSTEMTIME st;

GetSystemTime(&st);

SystemTimeToFileTime(&st, ft);

}

BOOL isExistFile(LPCTSTR szPath)

{

DWORD dwAttrib = GetFileAttributes(szPath);

return (dwAttrib != INVALID\_FILE\_ATTRIBUTES && !(dwAttrib & FILE\_ATTRIBUTE\_DIRECTORY));

}

myDB.cpp

#pragma once

#define STR\_MAX 81

struct Header

{

DWORD dwFileSize;

DWORD dwCountNotes;

};

struct Note

{

DWORD dwId;

FILETIME lpLastChangeTime;

CHAR szNote[STR\_MAX];

DWORD dwCountChanges;

};

void FileTimeToCurrentSystemTime(PFILETIME ft);

BOOL isExistFile(LPCTSTR szPath);

BOOL readHeader(HANDLE hF, Header\* pH);

BOOL writeHeader(HANDLE hF, Header\* pH);

HANDLE createNewFile(LPCTSTR lpFileName, DWORD dwNumberOfNotes, Header\* pHeader);

HANDLE openFile(LPCTSTR lpFileName, Header\* pHeader);

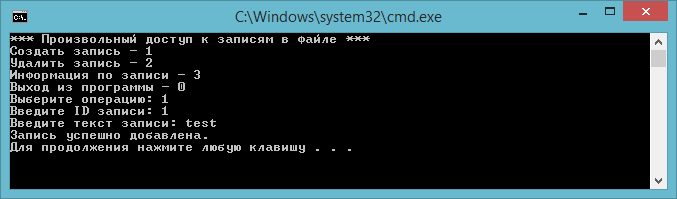
BOOL readNote(HANDLE hF, Header\* pH, DWORD dwID, Note\* pNote);

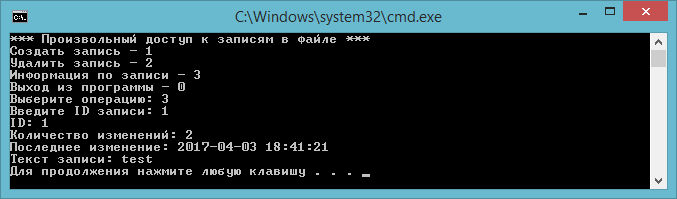
BOOL writeNote(HANDLE hF, Header\* pH, DWORD dwID, CHAR\* szString);

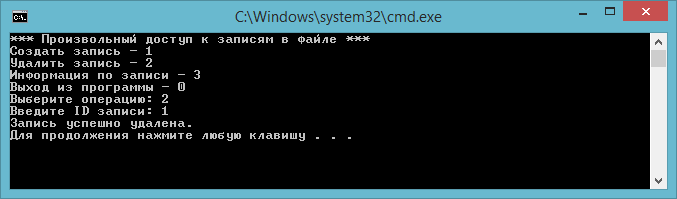
BOOL removeNote(HANDLE hF, Header\* pH, DWORD dwID);

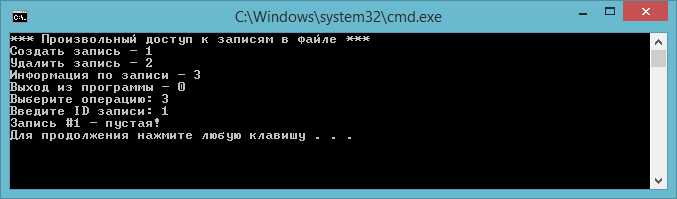
void printNoteInfo(HANDLE hF, Header\* pH, DWORD dwID);

**Скриншоты:**









**Задание 2**

**Код программы:**

// task2.cpp: определяет точку входа для консольного приложения.

//

#include "stdafx.h"

#include "manager.h"

void readline(TCHAR szBuffer[], DWORD nBufferSize);

int main()

{

setlocale(LC\_ALL, "RUS");

TCHAR szBuffer[MAX\_PATH];

TCHAR szBuffer2[MAX\_PATH];

int operation = 0;

while (true)

{

printf("\*\*\* Файловый менеджер \*\*\*\n");

printf("Создать директорию - 1\n");

printf("Сменить директорию - 2\n");

printf("Вывести текущую директорию - 3\n");

printf("Список файлов текущей директории - 4\n");

printf("Информация о файле - 5\n");

printf("Скопировать файл - 6\n");

printf("Удалить файл(или пустую папку)- 7\n");

printf("Выход из прогаммы - 0\n");

printf("Выберите операцию: ");

scanf\_s("%d", &operation);

switch (operation)

{

case 0:

return 0;

case 1:

printf("Введите название директории: ");

readline(szBuffer, MAX\_PATH);

createDir(szBuffer);

break;

case 2:

printf("Изменить директорию на: ");

readline(szBuffer, MAX\_PATH);

setCurDir(szBuffer);

break;

case 3:

printCurDir();

break;

case 4:

printFiles();

break;

case 5:

printf("Введите название директории: ");

readline(szBuffer, MAX\_PATH);

printFileInfo(szBuffer);

break;

case 6:

printf("Введите существующее имя файла: ");

readline(szBuffer, MAX\_PATH);

printf("Введите новое имя файла: ");

readline(szBuffer2, MAX\_PATH);

copyFile(szBuffer, szBuffer2);

break;

case 7:

printf("Введите имя файла: ");

readline(szBuffer, MAX\_PATH);

removeFile(szBuffer);

break;

default:

printf("Неизвестная операция!\n");

break;

}

printf("\n");

system("pause");

system("cls");

}

return 0;

}

void readline(TCHAR szBuffer[], DWORD nBufferSize)

{

DWORD i = 0, ch;

while (isspace(ch = getchar()));

while (ch != '\n')

{

if (i < nBufferSize)

szBuffer[i++] = ch;

ch = getchar();

}

szBuffer[i] = '\0';

}

manager.cpp

#include "stdafx.h"

#include "manager.h"

int FileTimeToString(FILETIME FileTime, DWORD nBufferLength, LPTSTR lpBuffer);

void printErrorMessage(HRESULT hResult = NULL);

void printCurDir()

{

TCHAR szBuffer[MAX\_PATH];

DWORD dwRet;

dwRet = GetCurrentDirectory(MAX\_PATH, szBuffer);

if (dwRet == 0) {

printErrorMessage();

return;

}

if (dwRet > MAX\_PATH) {

printf("Слишком маленький буфер; требуется %d символов\n", dwRet);

return;

}

\_tprintf(TEXT("Текущая директория: %s\n"), szBuffer);

}

void setCurDir(LPCTSTR lpPathName)

{

if (!SetCurrentDirectory(lpPathName)) {

printErrorMessage();

return;

}

\_tprintf(TEXT("Текущая директория изменена на %s\n"), lpPathName);

}

void copyFile(LPCTSTR lpExistingFileName, LPCTSTR lpNewFileName)

{

CopyFileEx(lpExistingFileName, lpNewFileName, NULL, NULL, NULL, NULL);

}

void createDir(LPCTSTR lpPathName)

{

if (!CreateDirectory(lpPathName, NULL)) {

printErrorMessage();

return;

}

}

void removeFile(LPCTSTR lpPathName)

{

WIN32\_FIND\_DATA ffd;

HANDLE hFind = INVALID\_HANDLE\_VALUE;

hFind = FindFirstFile(lpPathName, &ffd);

if (ffd.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY) {

if (!RemoveDirectory(lpPathName)) {

printErrorMessage();

}

}

else {

if (!DeleteFile(lpPathName)) {

printErrorMessage();

}

}

}

void printFiles()

{

WIN32\_FIND\_DATA fileFD;

LARGE\_INTEGER filesize;

HANDLE hFind = INVALID\_HANDLE\_VALUE;

TCHAR szDir[MAX\_PATH];

DWORD dwError = 0;

GetCurrentDirectory(MAX\_PATH, szDir);

StringCchCat(szDir, MAX\_PATH, TEXT("\\\*"));

hFind = FindFirstFile(szDir, &fileFD);

do

{

if (fileFD.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY) {

\_tprintf(TEXT(" %s <DIR>\n"), fileFD.cFileName);

}

else {

filesize.LowPart = fileFD.nFileSizeLow;

filesize.HighPart = fileFD.nFileSizeHigh;

\_tprintf(TEXT(" %s %I64d байт\n"), fileFD.cFileName, filesize.QuadPart);

}

} while (FindNextFile(hFind, &fileFD) != 0);

}

void printFileInfo(LPCTSTR lpPathName)

{

HANDLE hFile = NULL;

if ((hFile = CreateFile(lpPathName, GENERIC\_READ, NULL, NULL, OPEN\_EXISTING, NULL, NULL)) == INVALID\_HANDLE\_VALUE)

{

printErrorMessage();

return;

}

BY\_HANDLE\_FILE\_INFORMATION fileInfo;

if (!GetFileInformationByHandle(hFile, &fileInfo)) {

printErrorMessage();

return;

}

TCHAR szBuffer[MAX\_PATH];

GetFullPathName(lpPathName, MAX\_PATH, szBuffer, NULL);

\_tprintf(TEXT("Полный путь: %s\n"), szBuffer);

FileTimeToString(fileInfo.ftCreationTime, MAX\_PATH, szBuffer);

\_tprintf(TEXT("Время создания: %s\n"), szBuffer);

FileTimeToString(fileInfo.ftLastAccessTime, MAX\_PATH, szBuffer);

\_tprintf(TEXT("Время последнего просмотра: %s\n"), szBuffer);

FileTimeToString(fileInfo.ftLastWriteTime, MAX\_PATH, szBuffer);

\_tprintf(TEXT("Время последнего изменения: %s\n"), szBuffer);

printf("Размер файла: %d байт\n", fileInfo.nFileSizeLow);

CloseHandle(hFile);

}

int FileTimeToString(FILETIME FileTime, DWORD nBufferLength, LPTSTR lpBuffer)

{

SYSTEMTIME st;

FileTimeToSystemTime(&FileTime, &st);

return \_stprintf\_s(lpBuffer, nBufferLength, TEXT("%04d-%02d-%02d %02d:%02d:%02d"), st.wYear, st.wMonth, st.wDay, st.wHour, st.wMinute, st.wSecond);

}

void printErrorMessage(HRESULT hResult)

{

if (hResult == NULL) {

hResult = GetLastError();

}

LPTSTR errorText = NULL;

FormatMessage(

FORMAT\_MESSAGE\_FROM\_SYSTEM | FORMAT\_MESSAGE\_ALLOCATE\_BUFFER | FORMAT\_MESSAGE\_IGNORE\_INSERTS,

NULL,

hResult,

MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT),

(LPTSTR)&errorText,

0,

NULL);

if (NULL != errorText) {

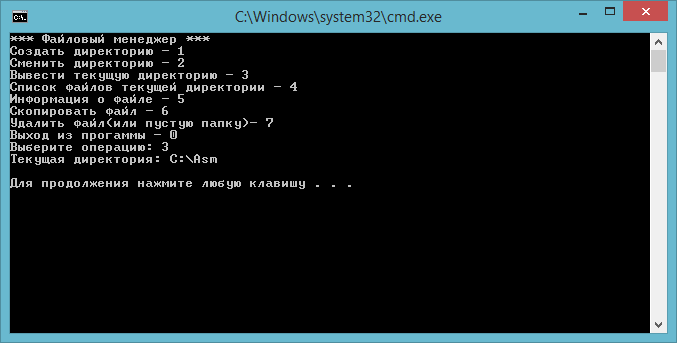
wprintf(L"%s", errorText);

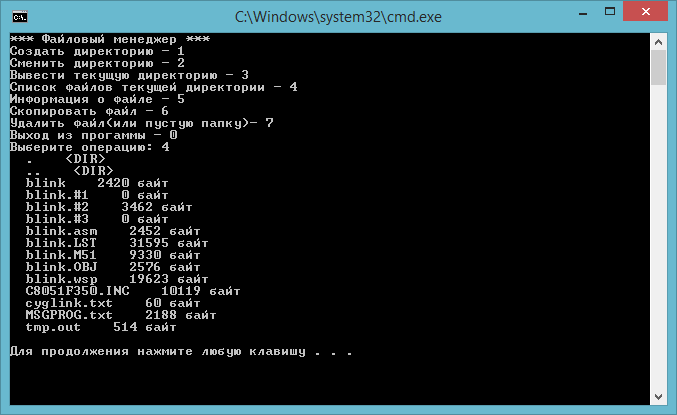
LocalFree(errorText);

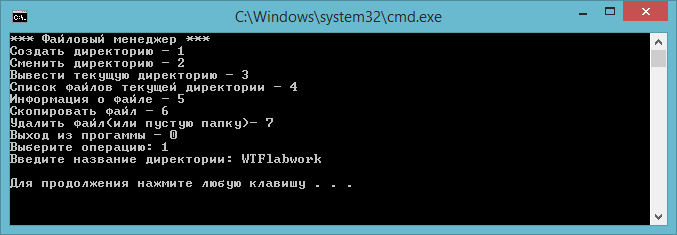
}

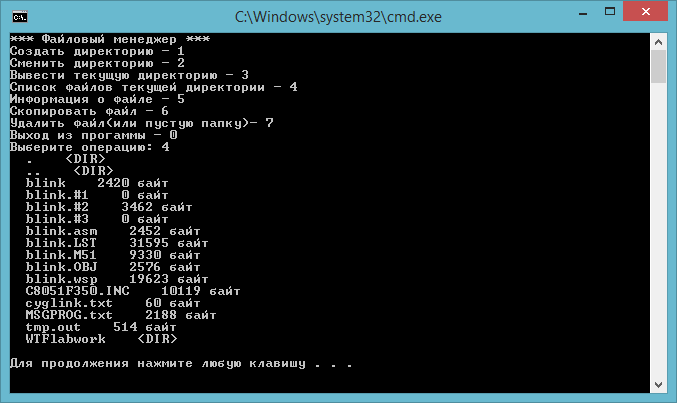
}

**Скриншоты:**









**Выводы:**

я освоил для себя такие новые функции, как FindFistfFile, FindNextFile, SetCurrentDirectoryA, GetCurrentDirectoryA, а также закрепил знание некоторых уже изученных, таких как CreateFileA, CreateFile, понял назначение новых типов, а также с их помощью создал полноченный файловый менеджер и файл структурю