Изучение системных вызовов Win32 API работы с файлами.

***Задача 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.

**Код программы:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<windows.h>

#include<string>

#include<stdio.h>

#include<string.h>

#include<tchar.h>

#define STEP 100

#define INFO\_SIZE 80

#define FREE\_ROW\_ID -1

#define LINE\_T\_SIZE sizeof(LINE\_T)

using namespace std;

typedef struct {

DWORD id;

FILETIME time;

CHAR info[INFO\_SIZE];

DWORD changed;

}LINE\_T;

typedef struct {

DWORD lastId;

DWORD fileSize;

DWORD exists;

}FIRST\_LINE\_T;

void ModifyLine(INT);

DWORD FindLine(INT);

void DeleteLineFromFile(INT);

void ShowInformationFile();

void InitData();

void WriteLineToFile();

HFILE file;

int \_tmain(INT argc, TCHAR\*\* argv) {

HANDLE std = GetStdHandle(STD\_INPUT\_HANDLE);

DWORD mode;

GetConsoleMode(std, &mode);

SetConsoleMode(std, mode | ENABLE\_PROCESSED\_INPUT);

LPTSTR act = NULL;

LPCTSTR fileName = L"../fileData.txt";

try {

if (argv[1] == NULL) throw 1;

//Create file

file = (HFILE)CreateFile(

(LPCTSTR)fileName, GENERIC\_WRITE | GENERIC\_READ, 0,

NULL, OPEN\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, 0);

int sizeOfFile = GetFileSize((HANDLE)file, NULL);

if (sizeOfFile == 0) {

InitData();

}

BYTE commmand = (BYTE)act[1];

INT num;

if (commmand == 's') ShowInformationFile();

else if (commmand == 'i') while (TRUE) WriteLineToFile();

else if (commmand == 'd') {

argv[2] == NULL ? throw 2 : num = \_ttoi(argv[2]); DeleteLineFromFile(num);

}

else if (commmand == 'm') {

argv[2] == NULL ? throw 2 : num = \_ttoi(argv[2]);

cout << "Input new info: ";

ModifyLine(num);

}

}

catch (int e) {

cout << "Error: argument is wrong " << endl;

}

CloseHandle((HANDLE)file);

}

DWORD FindLine(INT num) {

//Find free row

DWORD count;

BOOL res = TRUE;

LINE\_T line = {};

SetFilePointer((HANDLE)file, LINE\_T\_SIZE, NULL, FILE\_BEGIN);

do res = ReadFile((HANDLE)file, (LPVOID)&line, LINE\_T\_SIZE, &count, NULL);

while ((res & count != 0) && (line.id != num));

if (line.id == num) {

int offset = (LINE\_T\_SIZE);

return SetFilePointer((HANDLE)file, -offset, NULL, FILE\_CURRENT);

}

else {

SetFilePointer((HANDLE)file, 0, NULL, FILE\_END);

}

return FALSE;

}

void ModifyLine(INT num) {

DWORD id = -1, count, newChang;

if (FindLine(num)) {

LINE\_T line = {};

ReadFile((HANDLE)file, (LPVOID)&line, LINE\_T\_SIZE, &count, NULL);

//new inf

if (!fgets(line.info, INFO\_SIZE, stdin)) {

return;

}

//set count of changes

line.changed++;

int offset = (LINE\_T\_SIZE);

SetFilePointer((HANDLE)file, -offset, NULL, FILE\_CURRENT);

WriteFile((HANDLE)file, (LPVOID)&line, LINE\_T\_SIZE, &count, NULL);

}

else {

cout << "Record doesn't exist." << endl;

}

}

void DeleteLineFromFile(INT num) {

DWORD id = FREE\_ROW\_ID, count;

if (FindLine(num)) {

WriteFile((HANDLE)file, (LPVOID)&id, sizeof(DWORD), &count, NULL);

}

else {

cout << "Record doesn't exist." << endl;

return;

}

//dec index

FIRST\_LINE\_T fline = {};

DWORD red;

SetFilePointer((HANDLE)file, 0, NULL, FILE\_BEGIN);

ReadFile((HANDLE)file, (LPVOID)&fline, sizeof(FIRST\_LINE\_T), &red, NULL);

fline.exists--;

SetFilePointer((HANDLE)file, 0, NULL, FILE\_BEGIN);

WriteFile((HANDLE)file, (LPVOID)&fline, sizeof(FIRST\_LINE\_T), &red, NULL);

}

void WriteLineToFile() {

LINE\_T fileline = {};

FIRST\_LINE\_T fline = {};

//init tmp memory

DWORD red;

SetFilePointer((HANDLE)file, 0, NULL, FILE\_BEGIN);

ReadFile((HANDLE)file, (LPVOID)&fline, sizeof(FIRST\_LINE\_T), &red, NULL);

//correct dif between first and others lines

SetFilePointer((HANDLE)file, LINE\_T\_SIZE, NULL, FILE\_BEGIN);

//get info from file

if (!fgets(fileline.info, INFO\_SIZE, stdin)) {

return;

}

//update info

fline.exists++;

fline.lastId++;

fline.fileSize = GetFileSize((HANDLE)file, NULL);

//apply first line

SetFilePointer((HANDLE)file, 0, NULL, FILE\_BEGIN);

WriteFile((HANDLE)file, (LPVOID)&fline, LINE\_T\_SIZE, &red, NULL);

Veronika Abarnikova, [09.05.20 00:26]

//struct init

fileline.id = fline.lastId;

//get time

SYSTEMTIME st;

GetSystemTime(&st); // gets current time

SystemTimeToFileTime(&st, &fileline.time); // converts to file time format

//changed are 0 already

//find free row

FindLine(FREE\_ROW\_ID);

WriteFile((HANDLE)file, (LPVOID)&fileline, LINE\_T\_SIZE, &red, NULL);

}

void ShowInformationFile() {

DWORD count = 0;

FIRST\_LINE\_T fline = {};

//set file pointer

SetFilePointer((HANDLE)file, 0, NULL, FILE\_BEGIN);

//read file

ReadFile((HANDLE)file, (LPVOID)&fline, sizeof(FIRST\_LINE\_T), &count, NULL);

//correct dif between first and others lines

SetFilePointer((HANDLE)file, LINE\_T\_SIZE, NULL, FILE\_BEGIN);

BOOL result = FALSE;

LINE\_T line = {};

while (TRUE) {

result = ReadFile((HANDLE)file, (LPVOID)&line, LINE\_T\_SIZE, &count, NULL);

if (result & count == 0) {

break;

}

if (line.id == FREE\_ROW\_ID) {

continue;

}

SYSTEMTIME st = {};

FileTimeToSystemTime(&line.time, (LPSYSTEMTIME)&st);

cout << "#" << line.id << endl;

cout << "CreatedTime: " << st.wHour << ":" << st.wMinute

<< " " << st.wDay << "." << st.wMonth << "." << st.wYear << endl;

cout << "Message: " << line.info << endl;

cout << "Number of change : " << line.changed << " times" << endl;

cout << "\n\t\*\*\*\n";

}

cout << "\n-----------------" << endl;

cout << "Total: " << fline.exists << " records, " << fline.fileSize << " bytes." << endl;

}

//data initialize

void InitData() {

LPSTR bufer = new CHAR[LINE\_T\_SIZE];

memset(bufer, 0, LINE\_T\_SIZE);

DWORD written;

BOOL res = WriteFile((HANDLE)file, bufer, LINE\_T\_SIZE, &written, NULL);

delete[](bufer);

}

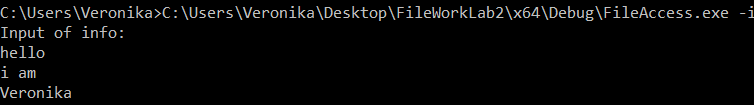


Рисунок 1 – Ввод инф в файл.

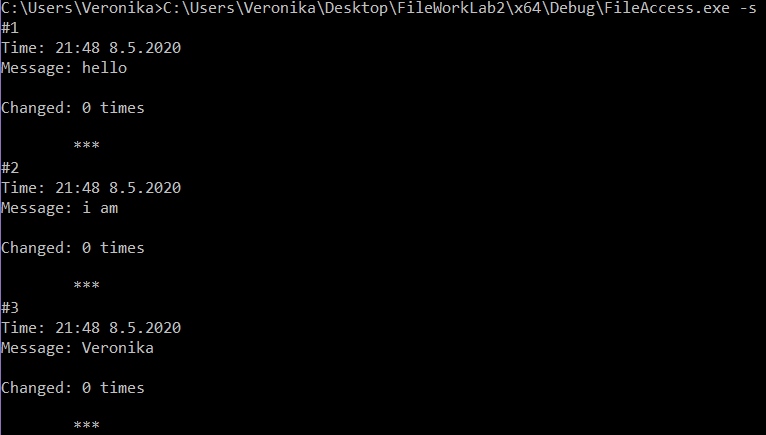


Рисунок 2 – Вывод содержимого.



Рисунок 3 – Модификация записи.

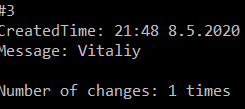


Рисунок 4 – Вывод содержимого.



Рисунок 5 – Удаление первой записи.

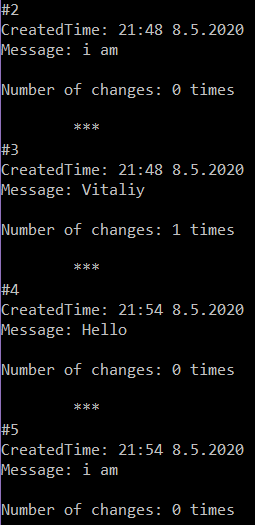


Рисунок 6 – Вывод содержимого.

***Задача 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

**Код программы**

#define \_CRT\_SECURE\_NO\_WARNINGS

#define \_CRT\_NON\_CONFORMING\_WCSTOK

#define \_CRT\_NON\_CONFORMING\_SWPRINTFS

#include <windows.h>

#include <string.h>

#include <tchar.h>

#include <iomanip>

#include <iostream>

#include <fcntl.h>

#include <io.h>

#define MAX\_BUFFER 1000

#define N\_MEASURES 4

#define MAX\_ARGC 3

#define FILE\_NAME 40

#define DISP\_STEP 15

using namespace std;

//Creating file

void CreateDir(LPTSTR dirPath) {

if (!\*dirPath) throw (INT)2;

if (!CreateDirectory(dirPath, NULL)) {

wcout << "Sorry, error occured while creating directory." << endl;

throw GetLastError();

}

wcout << "File was created" << endl;

}

//Change directory

void ChangeDir(LPTSTR dirPath){

if (!\*dirPath) throw (INT)2;

SetCurrentDirectory(dirPath);

GetCurrentDirectory(MAX\_BUFFER, dirPath);

wcout << "Directory has been changed to " << dirPath << endl;

}

// Copy file

void Copyfile(LPTSTR dirPath, LPTSTR src) {

if (!\*dirPath || !\*src) throw (INT)1;

if (!CopyFile(src, dirPath, FALSE)) {

wcout << "Sorry, can't copy the file." << endl;

throw GetLastError();

}

wcout << "Successful operation" << endl;

}

//Delete file or Directory

void DeleteFileOrDirectory(LPTSTR dirPath) {

if (!\*dirPath) throw (INT)2;

DWORD attr = GetFileAttributes(dirPath);

if (attr & FILE\_ATTRIBUTE\_DIRECTORY) {

if (!RemoveDirectory(dirPath)) {

throw GetLastError();

}

}

else {

if (!DeleteFile(dirPath)) {

throw GetLastError();

}

}

wcout << "File or directory was deleted!" << endl;

}

// Directory inf

void DirectoryInformaion(LPTSTR dirPath) {

dirPath = new TCHAR[MAX\_BUFFER];

GetCurrentDirectory(MAX\_BUFFER, dirPath);

wcout << dirPath << endl;

}

//Check path

BOOL IsNormal(LPCTSTR path) {

INT i = 0;

while (path[i] != TEXT('\0')) {

if (path[i] == TEXT(':')) {

return TRUE;

}

i++;

}

return FALSE;

}

//Decoding information

BOOL SplitInputAndDecode(LPTSTR input, LPTSTR com, LPTSTR dirPath, LPTSTR src, LPTSTR pathfile) {

//split input string

LPWSTR sep = LPWSTR(L" \n");

LPTSTR token;

INT argc = 0;

LPTSTR argv[MAX\_ARGC \* sizeof(LPWSTR)] = {};

token = wcstok(input, sep);

if (!token) return FALSE;

while (token) {

argv[argc] = token;

argc++;

token = wcstok(NULL, sep);

}

wcscpy(com, argv[0]);

if (argc > 1) {

if (!IsNormal(argv[1])) {

GetFullPathName(argv[1], MAX\_BUFFER, dirPath, &pathfile);

}

else {

wcscpy(dirPath, argv[1]);

}

}

if (argc > 2) {

if (!IsNormal(argv[2])) {

GetFullPathName(argv[2], MAX\_BUFFER, src, &pathfile);

}

else {

wcscpy(src, argv[2]);

}

}

return TRUE;

}

//File inf

void FileInf(LPTSTR dirPath) {

if (!\*dirPath) throw (INT)2;

WIN32\_FIND\_DATA info;

if (FindFirstFile(dirPath, &info) == INVALID\_HANDLE\_VALUE) throw (DWORD)2;

wcout << "File information:\n" << endl;

//if it's a directory

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

wcout << "<directory>" << endl;

}

wcout << "NameFile: " << info.cFileName << endl;

wcout << "Attribs mask: " << info.dwFileAttributes << endl;

wcout << "Alter: " << info.cAlternateFileName << endl;

unsigned long long size = (unsigned long long)info.nFileSizeHigh << 32 | info.nFileSizeLow;

LPCSTR measures[] = { "B", "KB" ,"MB" ,"GB" };

INT i = 0;

while (size >> 10 \* ++i);

i--;

wcout << "SizeFile: " << (size >> 10 \* i) << " " << measures[i] << endl;

SYSTEMTIME stime;

FileTimeToSystemTime(&info.ftCreationTime, &stime);

wcout << "Created: " << stime.wHour << ":" << stime.wMinute << " "

<< stime.wDay << "." << stime.wMonth << "." << stime.wYear << endl;

}

void PrDirectory(LPCTSTR cdir) {

LPTSTR dir = new TCHAR[MAX\_BUFFER];

memcpy(dir, cdir, MAX\_BUFFER \* sizeof(TCHAR));

wcscat(dir, TEXT("/\*.\*"));

WIN32\_FIND\_DATA fd;

HANDLE hFind = ::FindFirstFile(dir, &fd);

if (hFind != INVALID\_HANDLE\_VALUE) {

do {

// read all (real) files in current folder

// , delete '!' read other 2 default folder . and ..

//do printing

wcout << left << setw(FILE\_NAME) << fd.cFileName << " ";

unsigned long long sizeOfFile = (unsigned long long)fd.nFileSizeHigh << 32 | fd.nFileSizeLow;

LPCTSTR measures[] = { TEXT("B"), TEXT("KB") ,TEXT("MB") ,TEXT("GB") };

INT i = 0;

while (sizeOfFile >> 10 \* ++i);

i--;

wcout << left << setw(DISP\_STEP);

if (!(fd.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)) {

TCHAR buf[50];

memset(buf, 0, 100);

swprintf(buf, TEXT("%d "), (sizeOfFile >> 10 \* i));

wcscat(buf, measures[i]);

wcout << buf;

}

else {

wcout << "<dir>";

}

SYSTEMTIME stime;

FileTimeToSystemTime(&fd.ftCreationTime, &stime);

wcout<< stime.wDay << "." << stime.wMonth << "." << stime.wYear;

wcout << endl;

} while (::FindNextFile(hFind, &fd));

FindClose(hFind);

}

else {

wcout << "Error code :" << GetLastError() << endl;

}

delete dir;

}

//Print directory

void PrintDir(LPTSTR dirPath) {

if (!GetCurrentDirectory(MAX\_BUFFER, dirPath)) {

throw;

}

PrDirectory(dirPath);

}

int \_tmain(INT argc, TCHAR\*\* argv)

{

// enabling all Unicode chars in console

\_setmode(\_fileno(stdout), \_O\_U16TEXT);

LPTSTR dirPath = new TCHAR[MAX\_BUFFER], src = new TCHAR[MAX\_BUFFER], pfile = new TCHAR[MAX\_BUFFER];

LPTSTR com = new TCHAR[MAX\_BUFFER];

LPTSTR input = new TCHAR[MAX\_BUFFER];

int operation;

memset(src, 0, MAX\_BUFFER \* sizeof(TCHAR));

memset(dirPath, 0, MAX\_BUFFER \* sizeof(TCHAR));

memset(input, 0, MAX\_BUFFER \* sizeof(TCHAR));

memset(com, 0, MAX\_BUFFER \* sizeof(TCHAR));

memset(pfile, 0, MAX\_BUFFER \* sizeof(TCHAR));

wcout << "\t <File Manager>" << endl;

wcout << " Create dirirectory : mkdir name " << endl;

wcout << " Change current directory : cd dirPath " << endl;

wcout << " Information about directory : infdir" << endl;

wcout << " Print current directory : ls " << endl;

wcout << " Copy file : cp dirPath src " << endl;

wcout << " Delete file or empty directory : rm name " << endl;

wcout << " Full info about file : inffile name " << endl;

while (TRUE) {

try {

int k =0 ;

if (!fgetws(input, MAX\_BUFFER, stdin)) {

continue;

}

if (!SplitInputAndDecode(input, com, dirPath, src, pfile)) {

continue;

}

int i = 0;

string strCommand;

while (com[i]!= '\0')

{

strCommand += com[i];

i++;

}

if (strCommand == "mkdir") {

CreateDir(dirPath);

continue;

}

else if (strCommand == "cd") {

ChangeDir(dirPath);

}

else if (strCommand == "ls") {

DirectoryInformaion(dirPath);

}

else if (strCommand == "cp") {

Copyfile(dirPath, src);

}

else if (strCommand == "rm") {

DeleteFileOrDirectory(dirPath);

}

else if (strCommand == "infdir") {

PrintDir(dirPath);

}

else if (strCommand == "inffile") {

FileInf(dirPath);

}

else {

wcout << L"Command doesn't exist." << endl;

}

}

catch (INT e) {

switch (e) {

case 1:

wcout << "Source folder not found." << endl;

break;

case 2:

wcout << "Destination folder not found." << endl;

break;

}

}

catch (DWORD e) {

LPTSTR message = new TCHAR[MAX\_BUFFER];

FormatMessage(

// use system message tables to retrieve error text

FORMAT\_MESSAGE\_FROM\_SYSTEM

// allocate buffer on local heap for error text

| FORMAT\_MESSAGE\_ALLOCATE\_BUFFER

| FORMAT\_MESSAGE\_IGNORE\_INSERTS,

NULL, e,

MAKELANGID(LANG\_NEUTRAL, SUBLANG\_DEFAULT),

(LPWSTR)&message,

0, NULL);

wcout << L"Error: " << message << endl;

}

}

getchar();

return 0;

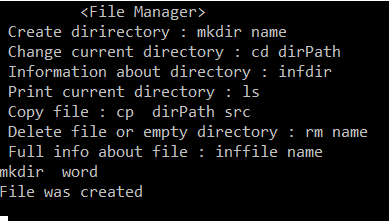


Рисунок 1 – Cоздать директорию.

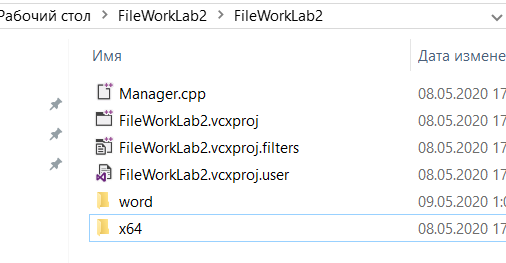


Рисунок 2 – Cоздать директорию.



Рисунок 3 – Сменить директорию.

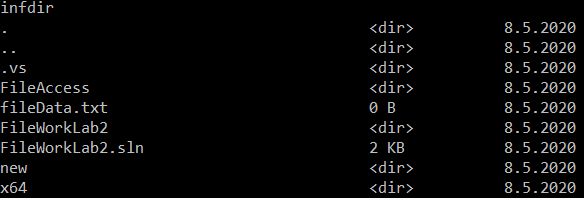


Рисунок 4 – информация про директорию.



Рисунок 5 – Текущая директория.



Рисунок 6 – Удаление директориии.

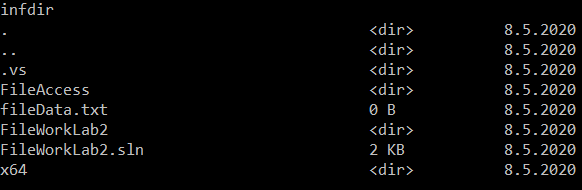


Рисунок 7 – Удаление директориии.

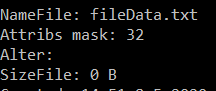


Рисунок 8 – Информация о файле.