**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

# Кафедра Вычислительной техники

**ОТЧЕТ**

# по лабораторной работе № 1.1

**по дисциплине «Операционные системы» Тема: «Управление файловой системой»**

Студент гр. 3311

Пасечный

Л.В.

Преподаватель

Тимофеев

А. В.

Санкт-Петербург

2024

# Введение

**Цель работы:**

исследовать управление файловой системой с помощью Win32 API.

# Постановка задачи:

Создайте консольное приложение с меню (каждая выполняемая функция и/или операция должна быть доступна по отдельному пункту меню),

которое выполняет:

* вывод списка дисков (функции Win32 API – GetLogicalDrives, GetLogicalDriveStrings);
* для одного из выбранных дисков вывод информации о диске и размер

свободного пространства (функции Win32 API –

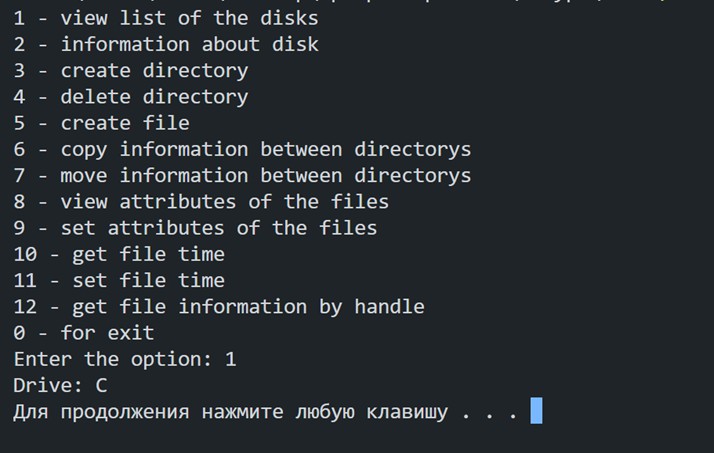
GetDriveType, GetVolumeInformation, GetDiskFreeSpace); ∙ создание и удаление заданных каталогов (функции Win32 API – CreateDirectory, RemoveDirectory);

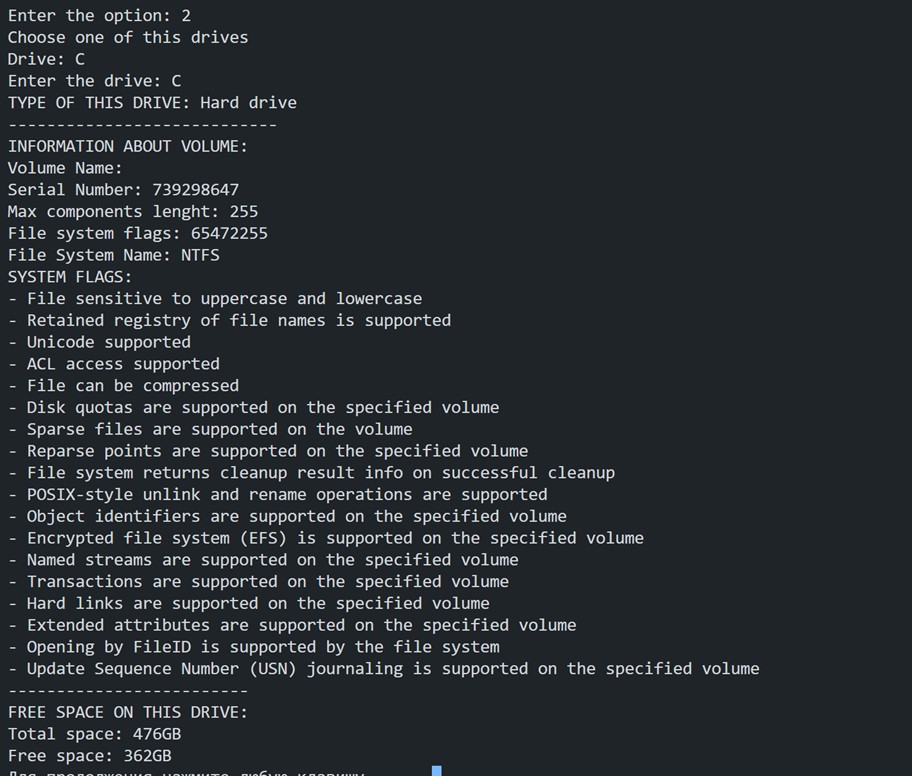
* создание файлов в новых каталогах (функция Win32 API – CreateFile);
* копирование и перемещение файлов между каталогами с возможностью выявления попытки работы с файлами, имеющими совпадающие

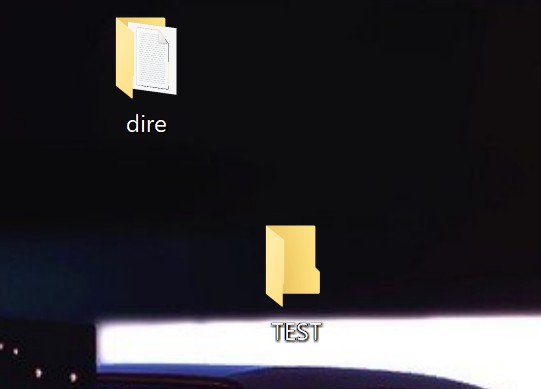
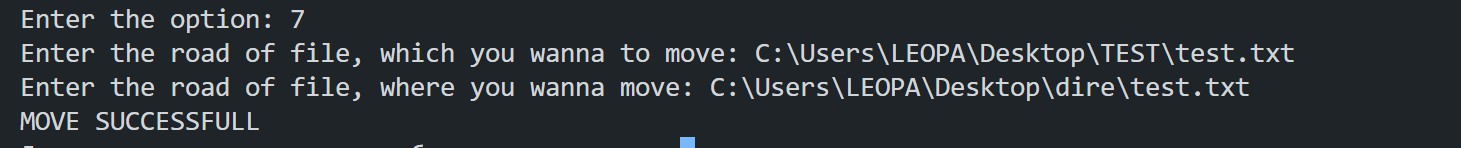
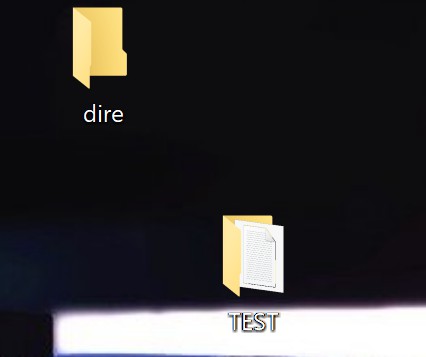
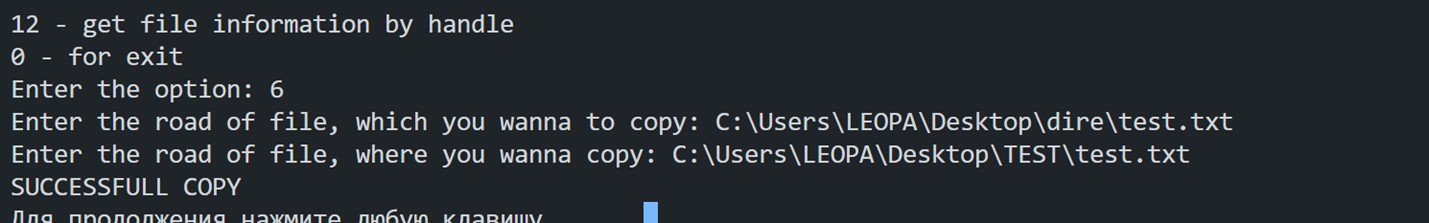
имена (функции Win32 API – CopyFile, MoveFile, MoveFileEx);

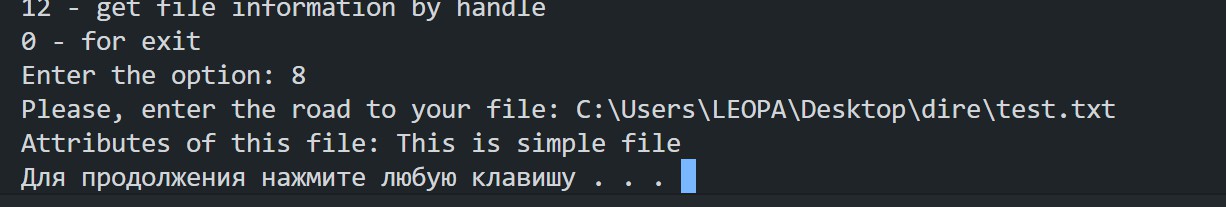
* анализ и изменение атрибутов файлов (функции Win32 API – GetFileAttributes, SetFileAttributes, GetFileInformationByHandle, GetFileTime, SetFileTime).

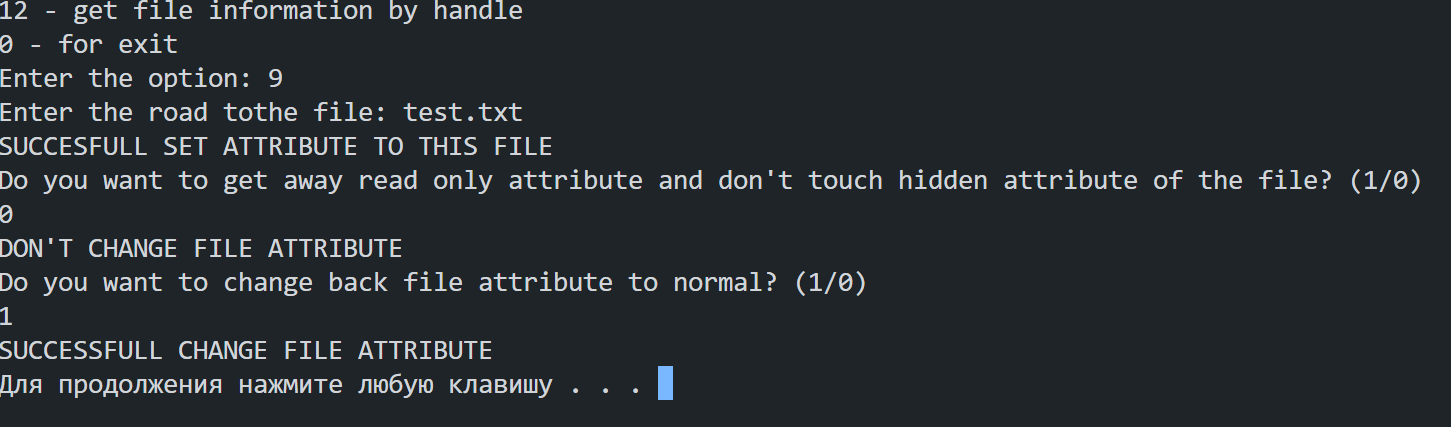
# Результаты:

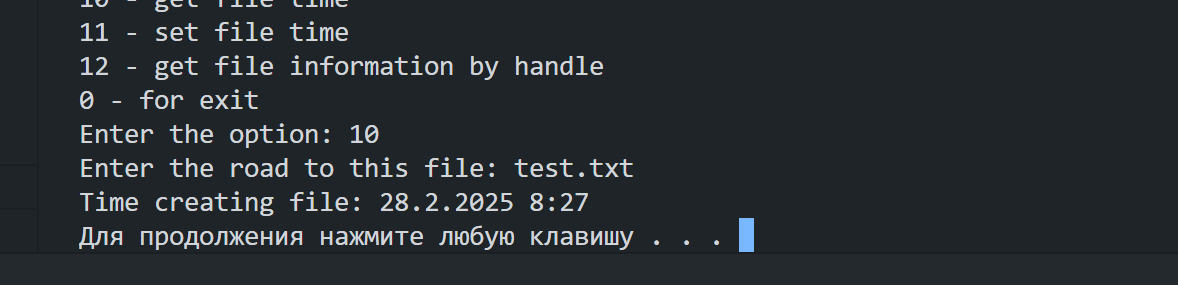
****

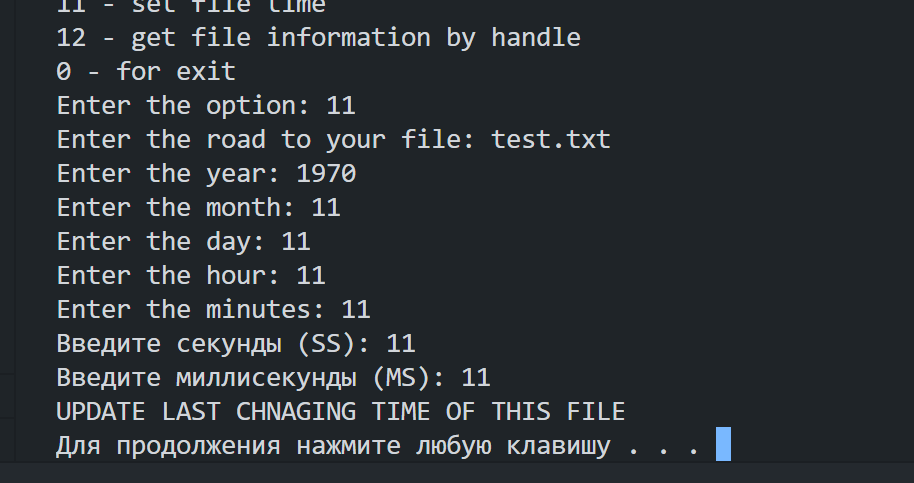


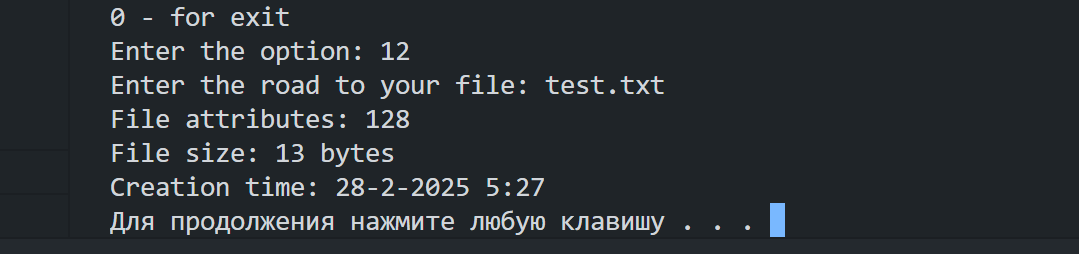
****











**Заключение**

Лабораторная работа была посвящена изучению возможностей Win32 API для управления файловой системой. Win32 API — это мощный и гибкий набор функций, предоставляющий низкоуровневый доступ к ресурсам операционной системы Windows, включая файлы, процессы, память и многое другое. Он позволяет разработчикам точно контролировать систему, но требует аккуратного обращения с дескрипторами, ресурсами и обязательной обработки ошибок. Несмотря на сложность, Win32 API остается востребованным инструментом для системного программирования и создания высокопроизводительных приложений благодаря своей функциональности и обширной документации от Microsoft.

# Код программы

#include <iostream> #include <Windows.h> #include <string> #include <locale> #include <fcntl.h> #include <io.h>

using namespace std;

// Дескрипторы типа HANDLE нужны для безопасной работы с файловой системой компьютера, они посредники между нами и ядром ОС.

// Если бы их не было, то система стала бы более уязвимой.

void menu(){

wcout << L"1 - view list of the disks" << endl; wcout << L"2 - information about disk" << endl; wcout << L"3 - create directory" << endl;

wcout << L"4 - delete directory" << endl; wcout << L"5 - create file" << endl;

wcout << L"6 - copy information between directorys" << endl; wcout << L"7 - move information between directorys" << endl; wcout << L"8 - view attributes of the files" << endl;

wcout << L"9 - set attributes of the files" << endl; wcout << L"10 - get file time" << endl;

wcout << L"11 - set file time" << endl;

wcout << L"12 - get file information by handle" << endl; wcout << L"0 - for exit" << endl;

}

void print\_flags(DWORD flags){

wcout << L"SYSTEM FLAGS:" << endl;

if (flags & FILE\_CASE\_SENSITIVE\_SEARCH) wcout << L"- File sensitive to uppercase and lowercase" << endl;

if (flags & FILE\_CASE\_PRESERVED\_NAMES) wcout << L"- Retained registry of file names is supported" << endl;

if (flags & FILE\_UNICODE\_ON\_DISK) wcout << L"- Unicode supported" << endl;

if (flags & FILE\_PERSISTENT\_ACLS) wcout << L"- ACL access supported" << endl;

if (flags & FILE\_FILE\_COMPRESSION) wcout << L"- File can be compressed" << endl; if (flags & FILE\_VOLUME\_QUOTAS) wcout << L"- Disk quotas are supported on the

specified volume" << endl;

if (flags & FILE\_SUPPORTS\_SPARSE\_FILES) wcout << L"- Sparse files are supported on the volume" << endl;

if (flags & FILE\_SUPPORTS\_REPARSE\_POINTS) wcout << L"- Reparse points are supported on the specified volume" << endl;

if (flags & FILE\_SUPPORTS\_REMOTE\_STORAGE) wcout << L"- Remote storage is supported by the file system" << endl;

if (flags & FILE\_RETURNS\_CLEANUP\_RESULT\_INFO) wcout << L"- File system returns cleanup result info on successful cleanup" << endl;

if (flags & FILE\_SUPPORTS\_POSIX\_UNLINK\_RENAME) wcout << L"- POSIX-style unlink and rename operations are supported" << endl;

if (flags & FILE\_VOLUME\_IS\_COMPRESSED) wcout << L"- The specified volume is a compressed volume" << endl;

if (flags & FILE\_SUPPORTS\_OBJECT\_IDS) wcout << L"- Object identifiers are supported on the specified volume" << endl;

if (flags & FILE\_SUPPORTS\_ENCRYPTION) wcout << L"- Encrypted file system (EFS) is supported on the specified volume" << endl;

if (flags & FILE\_NAMED\_STREAMS) wcout << L"- Named streams are supported on the specified volume" << endl;

if (flags & FILE\_READ\_ONLY\_VOLUME) wcout << L"- The specified volume is read- only" << endl;

if (flags & FILE\_SEQUENTIAL\_WRITE\_ONCE) wcout << L"- Sequential write-once is supported on the specified volume" << endl;

if (flags & FILE\_SUPPORTS\_TRANSACTIONS) wcout << L"- Transactions are supported on the specified volume" << endl;

if (flags & FILE\_SUPPORTS\_HARD\_LINKS) wcout << L"- Hard links are supported on the specified volume" << endl;

if (flags & FILE\_SUPPORTS\_EXTENDED\_ATTRIBUTES) wcout << L"- Extended attributes are supported on the specified volume" << endl;

if (flags & FILE\_SUPPORTS\_OPEN\_BY\_FILE\_ID) wcout << L"- Opening by FileID is supported by the file system" << endl;

if (flags & FILE\_SUPPORTS\_USN\_JOURNAL) wcout << L"- Update Sequence Number (USN) journaling is supported on the specified volume" << endl;

if (flags & FILE\_SUPPORTS\_INTEGRITY\_STREAMS) wcout << L"- Integrity streams are supported by the file system" << endl;

if (flags & FILE\_SUPPORTS\_BLOCK\_REFCOUNTING) wcout << L"- Logical cluster sharing between files on the same volume is supported" << endl;

if (flags & FILE\_SUPPORTS\_SPARSE\_VDL) wcout << L"- Sparse valid data length (VDL) tracking is supported by the file system" << endl;

if (flags & FILE\_DAX\_VOLUME) wcout << L"- The specified volume is a Direct Access (DAX) volume" << endl;

if (flags & FILE\_SUPPORTS\_GHOSTING) wcout << L"- Ghosting is supported by the file system" << endl;

}

void ChooseDisk(){ int disk;

WCHAR final\_disk;

UINT driveType;

wcout << L"Choose one of this drives" << endl; if (GetLogicalDrives == 0){

wcerr << "DRIVES ERROR " << GetLastError() << endl;

}

for (int i = 0; i < 26; i++){

if (GetLogicalDrives() & 1<<i){ char driveLetter = 'A'+i;

wcout << L"Drive: " << driveLetter << endl;

}

}

wcout << L"Enter the drive: "; wcin >> final\_disk;

wstring road = wstring(1, static\_cast<wchar\_t>(final\_disk)) + L":\\"; // static\_cast - оператор явного преобразования видов

driveType = GetDriveTypeW(road.c\_str()); // указатель на массив символов.

wcout << L"TYPE OF THIS DRIVE: ";

switch (driveType){

case DRIVE\_UNKNOWN:

wcout << L"Unknown drive" << endl; break;

case DRIVE\_NO\_ROOT\_DIR:

wcout << L"Not root directory" << endl; break;

case DRIVE\_REMOVABLE:

wcout << L"Removable drive" << endl; break;

case DRIVE\_FIXED:

wcout << L"Hard drive" << endl; break;

case DRIVE\_REMOTE:

wcout << L"Net drive" << endl; break;

case DRIVE\_CDROM:

wcout << L"CD/DVD - drive" << endl; break;

case DRIVE\_RAMDISK:

wcout << L"RAM-drive" << endl; break;

default:

wcout << L"I don't know what is the drive" << endl;

break;

}

wcout << L" " << endl;

wcout << L"INFORMATION ABOUT VOLUME: " << endl;

wchar\_t volumeNameBuffer[MAX\_PATH];

DWORD serialNumber, maxComponentLen, fileSystemFlags; wchar\_t fileSystemNameBuffer[MAX\_PATH];

//const WCHAR\* road2 = road; if (GetVolumeInformationW(

road.c\_str(), // Путь к диску

volumeNameBuffer, // Буфер для сохранения имени нашего диска

MAX\_PATH, // Размер именного буфера объёма

&serialNumber, // Буфер для сохранения серийного номера нашего диска &maxComponentLen, // Сохранение максимальной длины компонентов. &fileSystemFlags, //Сохранение флагов файловой системы. fileSystemNameBuffer, // СОхранение имени файловой системы MAX\_PATH

)){

wcout << L"Volume Name: " << volumeNameBuffer << endl; wcout << L"Serial Number: " << serialNumber << endl;

wcout << L"Max components lenght: " << maxComponentLen << endl; wcout << L"File system flags: " << fileSystemFlags << endl;

wcout << L"File System Name: " << fileSystemNameBuffer << endl; print\_flags(fileSystemFlags);

}else{

wcerr << "GET VOLUME INFORMATION ERROR" << endl;

}

wcout << L" " << endl;

wcout << L"FREE SPACE ON THIS DRIVE: " << endl;

DWORD sectorsPerClusters, bytesPerSector, freeClusters, totalClusters;

if (GetDiskFreeSpaceW( road.c\_str(),

&sectorsPerClusters, // сохранения количества секторов в кластере &bytesPerSector, // сохранение кол-ва байтов в секторе &freeClusters, // свободные кластеры

&totalClusters // всего кластеров

)){

ULONGLONG totalSpace = static\_cast<ULONGLONG>(totalClusters) \* static\_cast<ULONGLONG>(sectorsPerClusters) \* static\_cast<ULONGLONG>(bytesPerSector);

ULONGLONG freeSpace = static\_cast<ULONGLONG>(freeClusters)\* static\_cast<ULONGLONG>(sectorsPerClusters)\* static\_cast<ULONGLONG>(bytesPerSector);

wcout << L"Total space: " << totalSpace / (1024 \* 1024 \* 1024) << "GB" << endl; wcout << L"Free space: " << freeSpace / (1024 \* 1024 \* 1024) << "GB" << endl;

}else{

cerr << "SPACE FIND ERROR" << endl;

}

system("pause");

}

void view(){

DWORD drives = GetLogicalDrives(); // DWORD - 32-битный unsigned integer(uint32\_t) for (int i = 0; i < 26; i++){

if (drives & 1<<i){

char driveLetter = 'A'+i;

wcout << L"Drive: " << driveLetter << endl;

}

}

system("pause");

}

void Createdirectory(){ wstring road;

wcout << L"Enter the road to file: " << endl; getline(wcin, road);

if (CreateDirectoryW(road.c\_str(), NULL)){ wcout << L"Katalog sozdan" << endl;

}else{

wcerr << L"ERROR" << endl;

}

system("pause");

}

void createfile(){ wstring road;

wcout << L"Enter the road to file (C:\\example.txt)" << endl; getline(wcin, road);

HANDLE hFile = CreateFileW( road.c\_str(), // Имя файла GENERIC\_WRITE, // Доступ на запись 0, // Без совместного доступа

NULL, // Атрибуты безопасности по умолчанию CREATE\_ALWAYS, // Создать новый файл, перезаписать текущий FILE\_ATTRIBUTE\_NORMAL, // Обычные аттрибуты файла NULL // Без файла шаблона

);

const char\* data = "Hello, world!"; DWORD bytesWritten;

BOOL result = WriteFile(

hFile, //Дескриптор файла data, // Данные для записи strlen(data), // Размер данных

&bytesWritten, // Количество записанрных байт

NULL // Без перекрытия(синхронная операция)

);

if (!result){

wcerr << "WRITTEN ERROR "<< endl;

}else{

wcout << L"Written " << bytesWritten << " bytes" << endl;

}

CloseHandle(hFile); system("pause");

}

void Remove\_dir(){ wstring dirPath;

wcout << L"Enter the road to directory: "; getline(wcin, dirPath);

if (RemoveDirectoryW(dirPath.c\_str())){ wcout << L"SUCCESSFULL" << endl;

}else{

wcerr << "DELETE ERROR" << endl;

}

system("pause");

}

void copyfile(){ wstring road1; wstring road2;

wcout << L"Enter the road of file, which you wanna to copy(Указать файл который копируется): ";

getline(wcin, road1);

wcout << L"Enter the road of file, where you wanna copy: "; getline(wcin, road2);

BOOL result = CopyFileW( road1.c\_str(), // откуда road2.c\_str(), // куда

TRUE // Не перезаписывать файл

);

if (!result){

DWORD error = GetLastError();

if (error == ERROR\_FILE\_EXISTS){

wcout << L"ERROR: THIS FILE ALREADY STAY HERE" << endl;

}else if (error == ERROR\_FILE\_NOT\_FOUND){

wcout << L"ERROR: YOUR SOURCE FILE NOT FOUND" << endl;

}else{

wcout << L"ERROR OF COPY FILE" << endl;

}

}else{

wcout << L"SUCCESSFULL COPY" << endl;

}

system("pause");

}

void movefile(){ wstring road1; wstring road2;

wcout << L"Enter the road of file, which you wanna to move: ";

getline(wcin, road1);

wcout << L"Enter the road of file, where you wanna move: "; getline(wcin, road2);

BOOL result = MoveFileExW( road1.c\_str(), // откуда road2.c\_str(), // куда

MOVEFILE\_REPLACE\_EXISTING | MOVEFILE\_COPY\_ALLOWED // Перезаписать

и разрешить копирование

);

if (!result){

DWORD error = GetLastError();

wcout << L"ERROR MOVING FILE " << error << endl;

}else{

wcout << L"MOVE SUCCESSFULL" << endl;

}

system("pause");

}

void getatt(){ wstring road;

wcout << L"Please, enter the road to your file: "; getline(wcin, road);

DWORD fileattributes = GetFileAttributesW(road.c\_str());

if (fileattributes == INVALID\_FILE\_ATTRIBUTES){ DWORD error = GetLastError();

wcerr << "ERROR: " << error << endl;

}

wcout << L"Attributes of this file: ";

if (fileattributes & FILE\_ATTRIBUTE\_READONLY){ wcout << L"File have attributte read only" << endl;

}else if (fileattributes & FILE\_ATTRIBUTE\_HIDDEN){ wcout << L"File is hidden" << endl;

}else if (fileattributes & FILE\_ATTRIBUTE\_SYSTEM){ wcout << L"It's system file" << endl;

}else if (fileattributes & FILE\_ATTRIBUTE\_DIRECTORY){ wcout << L"This file is directory" << endl;

}else {

wcout << L"This is simple file" << endl;

}

system("pause");

}

void setatt(){ wstring road;

wcout << L"Enter the road tothe file: "; getline(wcin, road);

DWORD fileattribute = GetFileAttributesW(road.c\_str()); if (fileattribute == INVALID\_FILE\_ATTRIBUTES){

DWORD error = GetLastError();

wcerr << L"ERROR GET FILE ATTRIBUTE: " << error << endl;

}else{

// Устанавливаем атрибуты только для чтения

DWORD attributeToSet = FILE\_ATTRIBUTE\_READONLY | FILE\_ATTRIBUTE\_HIDDEN;

if (!SetFileAttributesW(road.c\_str(), attributeToSet)){ DWORD error = GetLastError();

wcerr << L"ERROR SET FILE ATTRIBUTE: " << error << endl;

}else{

wcout << L"SUCCESFULL SET ATTRIBUTE TO THIS FILE" << endl;

//Проверяем установленны ли атрибуты

DWORD testattribute = GetFileAttributesW(road.c\_str()); if (testattribute == INVALID\_FILE\_ATTRIBUTES){

if (testattribute & FILE\_ATTRIBUTE\_READONLY){

wcout << L"Now this file have attribute read only" << endl;

}

if (testattribute & FILE\_ATTRIBUTE\_HIDDEN){ wcout << L"Now this file is hidden" << endl;

}

}

char flag;

wcout << L"Do you want to get away read only attribute and don't touch hidden attribute of the file? (Y/N)";

cin >> flag;

if (flag == 'Y'){

DWORD newattribute = attributeToSet | ~FILE\_ATTRIBUTE\_READONLY; if (!SetFileAttributesW(road.c\_str(), newattribute)){

DWORD error = GetLastError();

wcout << L"ERROR CHANGE FILE ATTRIBUTE: " << error << endl;

}else{

wcout << L"SUCCESSFULL CHANGE FILE ATTRIBUTE" << endl;

}

}else{

wcout << L"DON'T CHANGE FILE ATTRIBUTE" << endl;

}

wcout << L"Do you want to change back file attribute to normal? (Y/N)" << endl; cin >> flag;

if (flag == 'Y'){

DWORD attributeback = FILE\_ATTRIBUTE\_NORMAL; if (!SetFileAttributesW(road.c\_str(), attributeback)){

DWORD error = GetLastError();

wcerr << L"ERROR OF CHANGE THE ATTRIBUTE: " << error << endl;

}else{

wcout << L"SUCCESSFULL CHANGE FILE ATTRIBUTE" << endl;

}

}

}

}

system("pause");

}

void getfiletime(){ wstring road;

wcout << L"Enter the road to this file: "; wcin.ignore();

getline(wcin, road);

HANDLE hfile = CreateFileW( road.c\_str(),

FILE\_READ\_ATTRIBUTES, //Требуемые права FILE\_SHARE\_READ, //Режим общего доступа NULL, //Защита файла по умолчанию OPEN\_EXISTING, // Открыть существующий файл

FILE\_ATTRIBUTE\_NORMAL, // Обычные флаги атрибутов

NULL

);

if (hfile == INVALID\_HANDLE\_VALUE){

DWORD error = GetLastError();

wcerr << L"ERROR OPENNING FILE: " << error << endl;

}else{

FILETIME creationtime, accesstime, writetime;

if (GetFileTime(hfile, &creationtime, &accesstime, &writetime)){ SYSTEMTIME systime; FileTimeToSystemTime(&creationtime, &systime);

wcout << L"Time creating file: " << systime.wDay << "." << systime.wMonth << "." << systime.wYear

<< " " << systime.wHour + 3<< ":" << systime.wMinute << endl;

}else{

wcout << L"ERROR GET FILE TIME: " << GetLastError << endl;

}

}

CloseHandle(hfile); system("pause");

}

void setfiletime(const wstring& filename){ HANDLE hfile = CreateFileW(

filename.c\_str(),

GENERIC\_WRITE, // права на запись FILE\_SHARE\_READ, // общий доступ для записи NULL, // безопасность файла по умолчанию OPEN\_EXISTING, // открыть уже существующий файл

FILE\_ATTRIBUTE\_NORMAL, // классические атрибуты файла

NULL

);

if (hfile == INVALID\_HANDLE\_VALUE){

wcerr << "HANDLE ERROR: " << GetLastError() << endl;

}else{

if (hfile == INVALID\_HANDLE\_VALUE){

wcerr << "HANDLE ERROR: " << GetLastError() << endl;

}else{

SYSTEMTIME newwritetime; wcout << L"Enter the year: "; wcin >> newwritetime.wYear; wcout << L"Enter the month: "; wcin >> newwritetime.wMonth; wcout << L"Enter the day: "; wcin >> newwritetime.wDay; wcout << L"Enter the hour: "; wcin >> newwritetime.wHour; wcout << L"Enter the minutes: "; wcin >> newwritetime.wMinute;

std::wcout << L"Введите секунды (SS): "; std::wcin >> newwritetime.wSecond;

std::wcout << L"Введите миллисекунды (MS): "; std::wcin >> newwritetime.wMilliseconds; FILETIME newfiletime;

SystemTimeToFileTime(&newwritetime, &newfiletime);

if (SetFileTime(hfile, NULL, NULL, &newfiletime)){

wcout << L"UPDATE LAST CHNAGING TIME OF THIS FILE" << endl;

}else{

wcerr << L"ERROR SET NEW TIME: " << GetLastError() << endl;

}

}

}

CloseHandle(hfile); system("pause");

}

void getinfbh(const wstring &filename){ HANDLE hFile = CreateFileW(

filename.c\_str(), GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL, NULL

);

if (hFile == INVALID\_HANDLE\_VALUE){

wcerr << L"HANDLE ERROR: " << GetLastError() << endl;

}else{

BY\_HANDLE\_FILE\_INFORMATION fileinf;

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

wcerr << L"GET INFORMATION ERROR: " << GetLastError() << endl;

}else{

wcout << L"File attributes: " << fileinf.dwFileAttributes << endl; if (fileinf.dwFileAttributes | FILE\_ATTRIBUTE\_DIRECTORY){

wcout << L"This file is directory" << endl;

}

if (fileinf.dwFileAttributes | FILE\_ATTRIBUTE\_READONLY){ wcout << L"This file has attribute read only" << endl;

}

ULONGLONG filesize = ((ULONGLONG)fileinf.nFileSizeHigh << 32) | fileinf.nFileSizeLow;

wcout << L"File size: " << filesize << " bytes" << endl;

SYSTEMTIME creationtime; FileTimeToSystemTime(&fileinf.ftCreationTime, &creationtime); wcout << L"Creation time: "

<< creationtime.wDay << L"-"

<< creationtime.wMonth << L"-"

<< creationtime.wYear << L" "

<< creationtime.wHour << L":"

<< creationtime.wMinute << endl;

}

}

CloseHandle(hFile); system("pause");

}

int main(){ int options;

wstring filename;

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

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

do{

menu();

wcout << L"Enter the option: "; wcin >> options;

wcin.ignore(); switch(options){

case(1):

view(); break;

case (2): ChooseDisk(); break;

case (3): Createdirectory(); break;

case (4): Remove\_dir(); break;

case (5): createfile(); break;

case (6): copyfile(); break;

case (7): movefile(); break;

case (8):

getatt(); break;

case (9):

setatt(); break;

case (10): getfiletime(); break;

case (11):

wcout << L"Enter the road to your file: "; wcin.ignore();

getline(wcin, filename); setfiletime(filename); break;

case (12):

wcout << L"Enter the road to your file: "; wcin.ignore();

getline(wcin, filename); getinfbh(filename); break;

case (0): break;

default:

wcout << L"Please try again" << endl;

}

}while (options != 0); return 0;

}