**Министерство образования и науки Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ бюджетное образовательное учреждение высшего профессионального образования**

**«Санкт-Петербургский государственный электротехнический университет «ЛЭТИ» им.** **В.И. Ульянова (Ленина)»**

**фАКУЛЬТЕТ КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ИНФОРМАТИКИ**

**КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ**

**Лабораторная работа №1**

**по дисциплине «Операционные системы»**

**на тему «УПРАВЛЕНИЕ ФАЙЛОВОЙ СИСТЕМОЙ»**

Выполнил студент группы 3312

Лебедев И.А.

Принял преподаватель Тимофеев А.В.

Санкт-Петербург

2025

Содержание

[Цель работы 3](#_Toc192988869)

[Задание 1.1 3](#_Toc192988870)

[Примеры запуска программы 3](#_Toc192988871)

[Текст программы 11](#_Toc192988872)

[Выводы 19](#_Toc192988873)

[Задание 1.2 20](#_Toc192988874)

[Примеры запуска программы 20](#_Toc192988875)

[Проверка при разных размерах копируемых блоков 22](#_Toc192988876)

[Проверка при разном количестве перекрывающихся операций 23](#_Toc192988877)

[Текст программы 24](#_Toc192988878)

[Выводы 27](#_Toc192988879)

# Цель работы

Исследовать управление файловой системой с помощью Win32 API.

# Задание 1.1

В этом задании необходимо разработать консольное приложение, использующее *Win32 API* для работы с файловой системой. Программа должна предоставлять пользователю меню, в котором каждая функция реализуется как отдельный пункт.

Приложение выполняет следующие операции:

- Вывод списка логических дисков, используя *GetLogicalDrives()* и *GetLogicalDriveStrings()*.

- Вывод информации о выбранном диске, включая тип носителя, файловую систему и объём свободного пространства (*GetDriveType()*, *GetVolumeInformation()*, *GetDiskFreeSpace()*).

- Создание и удаление каталогов (*CreateDirectory()*, *RemoveDirectory()*).

- Создание новых файлов в каталогах (*CreateFile()*).

- Копирование и перемещение файлов, с обработкой случаев совпадения имён (*CopyFile()*, *MoveFile()*, *MoveFileEx()*).

- Анализ и изменение атрибутов файлов, включая их размер и временные метки (*GetFileAttributes()*, *SetFileAttributes()*, *GetFileInformationByHandle()*, *GetFileTime()*, *SetFileTime()*).

# Примеры запуска программы

Созданное консольное приложение с меню предлагает пользователю выбор из нескольких операций:

1. Список дисков:

Этот пункт меню предназначен для вывода всех доступных дисков в системе. Он позволяет пользователю увидеть подключённые накопители, включая жёсткие диски, флешки и сетевые диски.

Реализация основана на использовании функции *GetLogicalDriveStringsW()*, которая возвращает строку с перечислением всех логических дисков. Сначала вызывается эта функция с нулевым буфером, чтобы определить необходимый размер памяти, затем выделяется динамический массив и повторный вызов *GetLogicalDriveStringsW()* получает список дисков. Циклом *while* программа обходит список и выводит имена дисков, после чего освобождает выделенную память. Если *GetLogicalDriveStringsW()* возвращает 0, программа сообщает об ошибке.

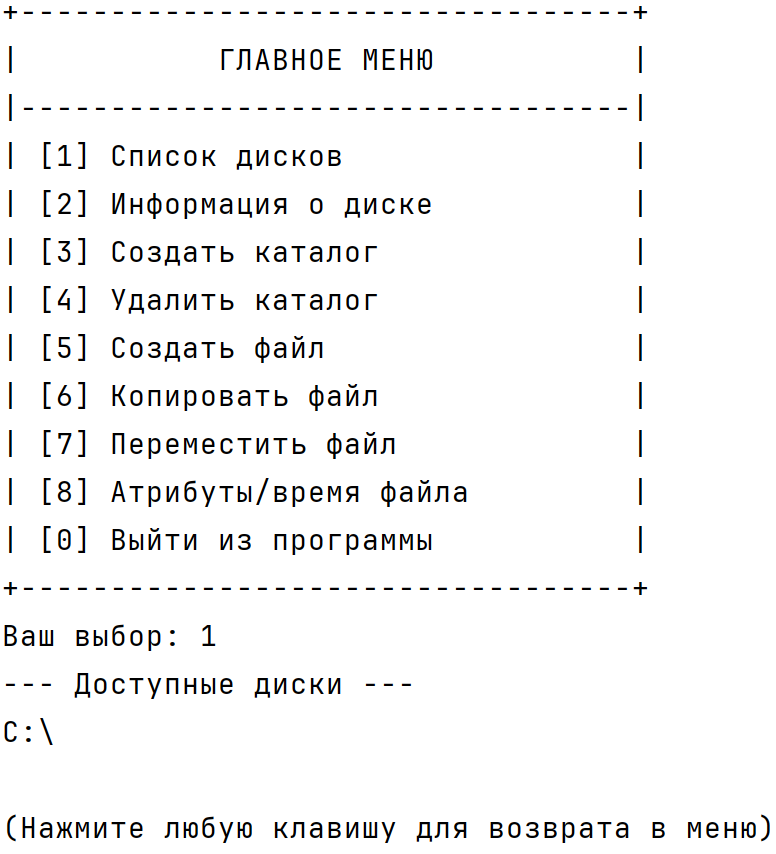


Рисунок 1 – Вывод списка дисков

2. Информация о диске:

Этот пункт меню позволяет получить информацию о выбранном пользователем логическом диске, включая его тип, файловую систему, метку тома, серийный номер, системные флаги и доступное пространство.

Реализация использует несколько функций *WinAPI*. *GetDriveTypeW()* определяет тип носителя (жёсткий диск, съёмный диск, сетевой и т. д.). *GetVolumeInformationW()* получает метку тома, файловую систему и серийный номер, а также системные флаги, которые затем расшифровываются и выводятся. *GetDiskFreeSpaceW()* вычисляет объём свободного места и общий размер диска. Программа запрашивает у пользователя букву диска, автоматически дополняя её до корректного пути. Если *GetVolumeInformationW()* или *GetDiskFreeSpaceW()* возвращают ошибку, программа выводит соответствующее сообщение.

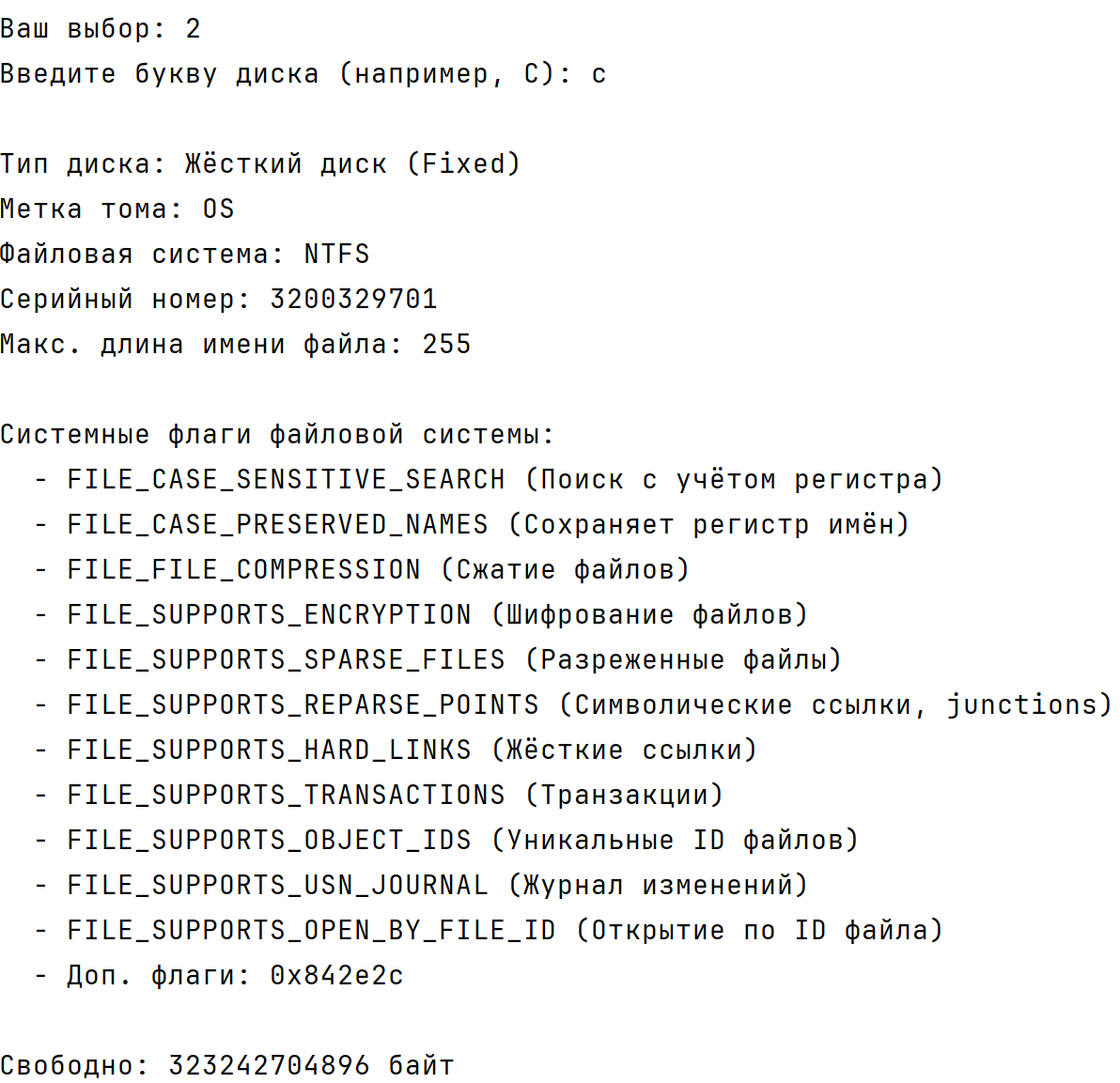


Рисунок 2 – Вывод информации о диске

3. Создание каталога:

Этот пункт меню предназначен для создания нового каталога по указанному пользователем пути.

Реализация основана на функции *CreateDirectoryW()*, которая принимает путь к каталогу и создаёт его. Программа запрашивает путь у пользователя, после чего вызывает эту функцию. Если каталог успешно создан, выводится соответствующее сообщение. В случае ошибки, программа сообщает об этом и выводит код ошибки, полученный с помощью *GetLastError()*. Перед вводом пути используется *wcin.ignore()*, чтобы избежать проблем с буфером ввода после выбора пункта меню.

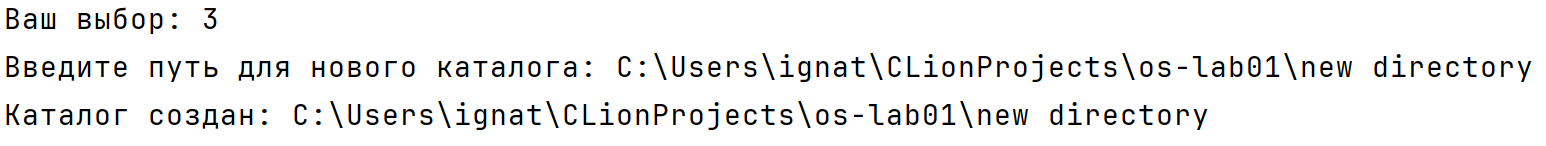


Рисунок 3 – Создание нового каталога

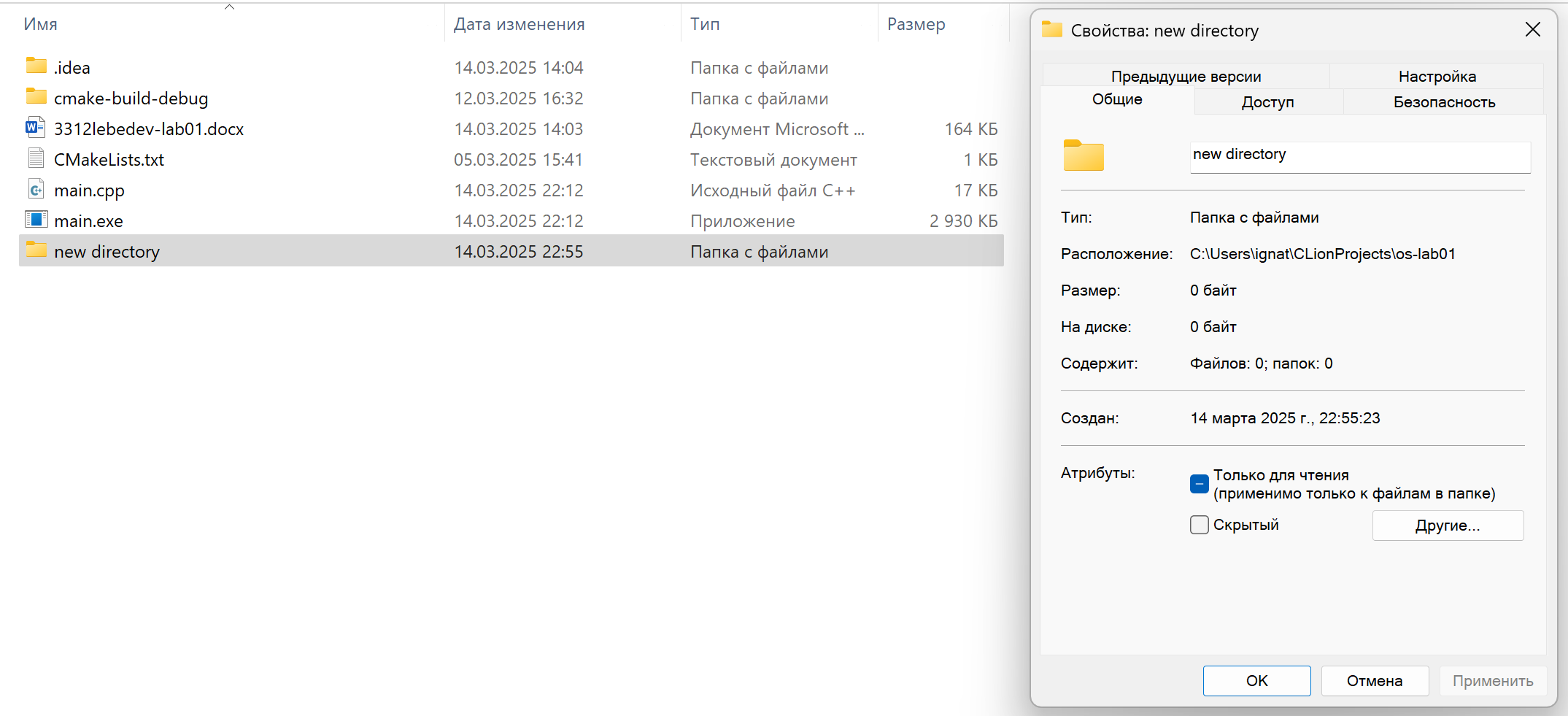


Рисунок 4 – Результат создания нового каталога

4. Удаление каталога:

Этот пункт меню предназначен для удаления указанного пользователем каталога. Поддерживается удаление как пустых, так и непустых директорий, включая все вложенные файлы и каталоги.

Реализация использует *FindFirstFileW()* и *FindNextFileW()* для получения содержимого каталога. Если в ней есть файлы или подкаталоги, они удаляются рекурсивно. Для удаления файлов используется *DeleteFileW()*, а для удаления самих папок — *RemoveDirectoryW()*. Если процесс удаления завершается успешно, программа выводит соответствующее сообщение. В случае ошибки отображается код ошибки через *GetLastError()*.

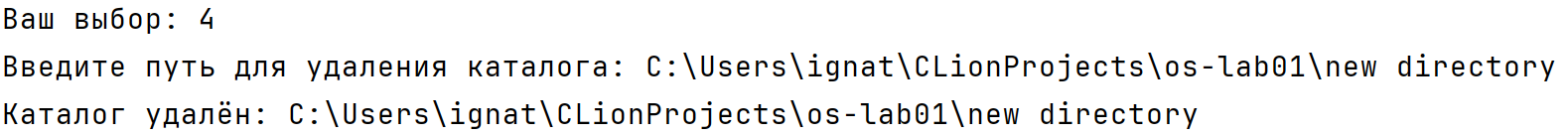


Рисунок 5 – Удаление пустого каталога

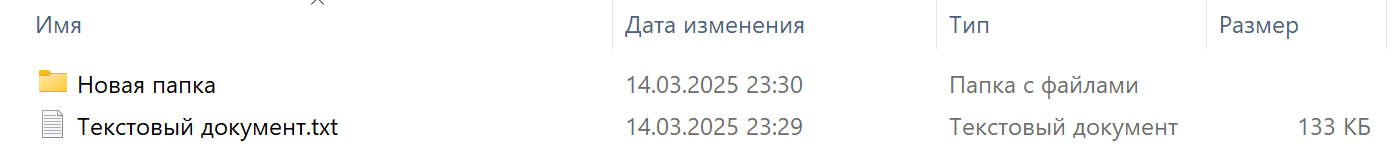


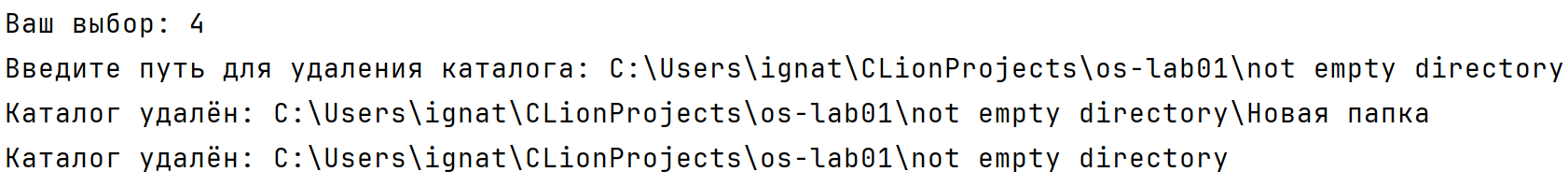
Рисунок 6 – Содержимое непустого каталога для удаления  


Рисунок 7 – Результат удаления непустого каталога

5. Создание файла:

Этот пункт меню позволяет пользователю создать новый файл по указанному пути. Он используется для создания пустых файлов, которые затем можно заполнить данными.

Реализация основана на функции *CreateFileW()*, которая создаёт или открывает файл. Программа запрашивает у пользователя путь для создания файла, затем вызывает *CreateFileW()* с флагами *GENERIC\_WRITE*, *CREATE\_NEW* и *FILE\_ATTRIBUTE\_NORMAL*. Если файл успешно создан, он сразу закрывается с помощью *CloseHandle()*. Если файл уже существует или возникла ошибка, программа выводит сообщение с кодом ошибки, полученным через *GetLastError()*.

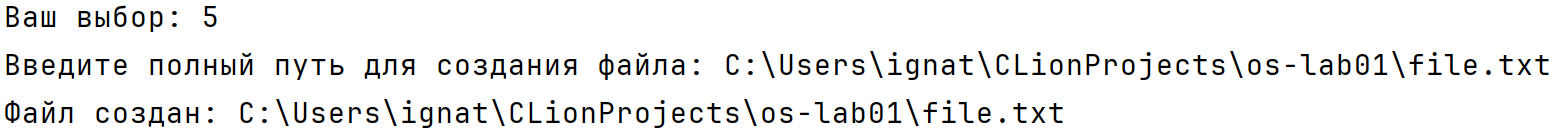


Рисунок 8 – Создание файла

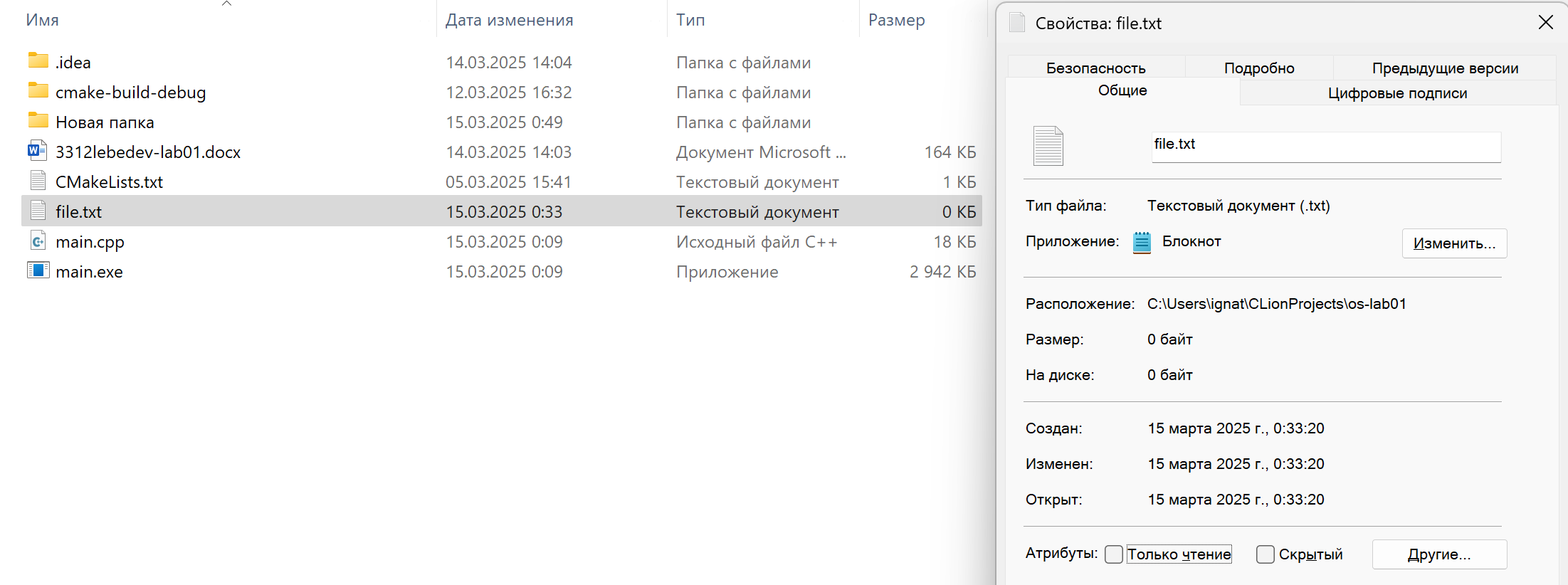


Рисунок 9 – Результат создания файла

6. Копирование файла:

Этот пункт меню предназначен для копирования файла из одного местоположения в другое. Он позволяет пользователю создать дубликат файла, сохраняя его содержимое.

Реализация использует функцию *CopyFileW()*, которая принимает путь к исходному файлу, путь к целевому файлу и флаг, указывающий, можно ли перезаписывать существующий файл. Программа запрашивает у пользователя пути исходного и целевого файлов. Если *CopyFileW()* выполняется успешно, выводится сообщение о завершении операции. В случае ошибки программа сообщает об этом, выводя код ошибки через *GetLastError()*.

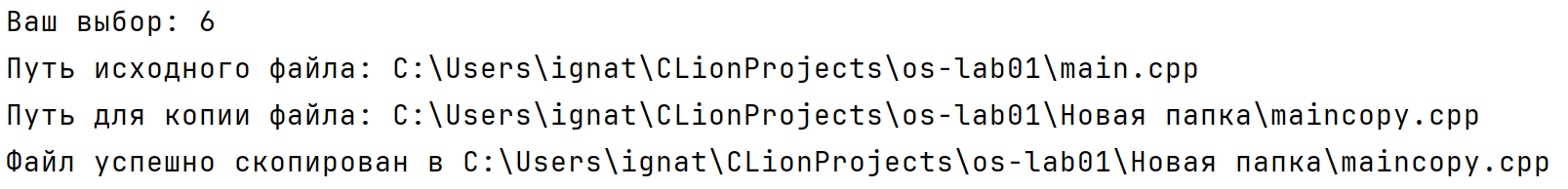


Рисунок 10 – Копирование файла

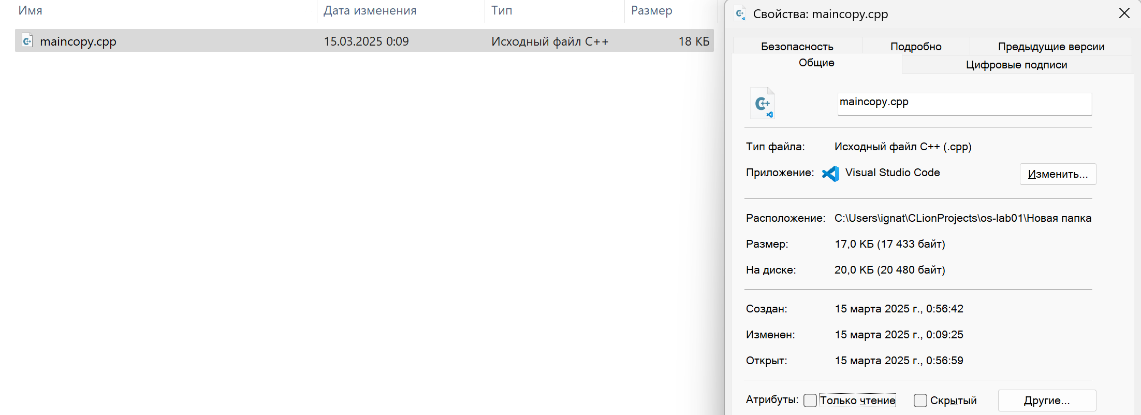


Рисунок 11 – Результат копирования файла

7. Перемещение файла:

Этот пункт меню предназначен для перемещения или переименования файла. Он позволяет пользователю перенести файл в другое местоположение или изменить его имя.

Реализация основана на функции *MoveFileW()*, которая принимает путь к исходному файлу и путь, по которому файл должен быть перемещён. Программа запрашивает у пользователя исходный путь и новый путь файла. Если операция выполнена успешно, программа выводит сообщение о завершении. В случае ошибки выводится код ошибки через *GetLastError()*.

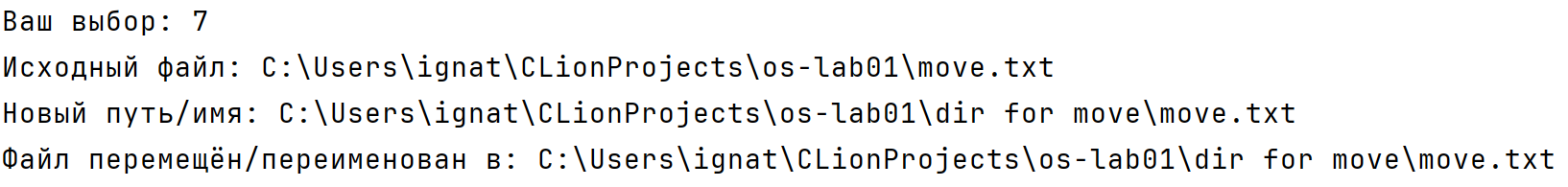


Рисунок 12 – Перемещение файла

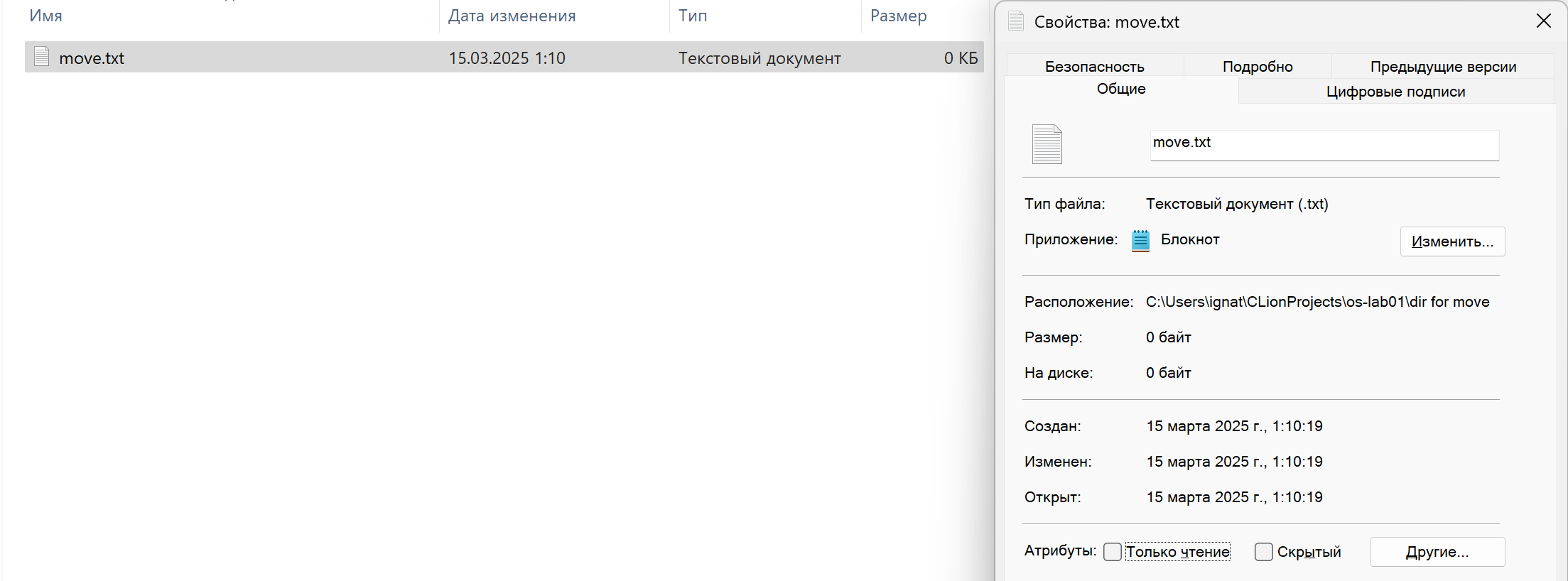


Рисунок 13 – Результат перемещения

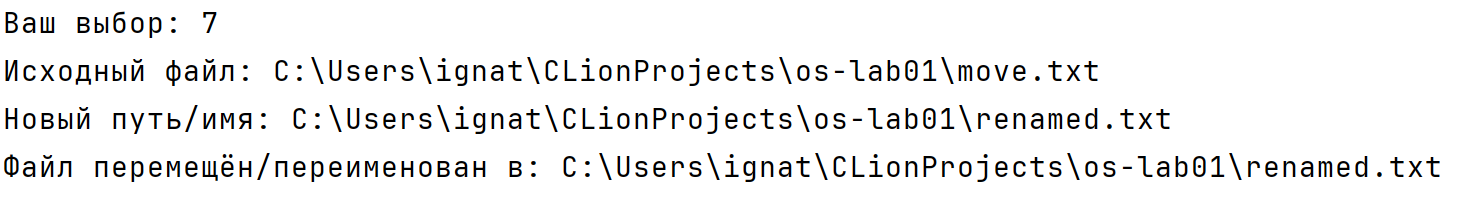


Рисунок 14 – Переименование файла

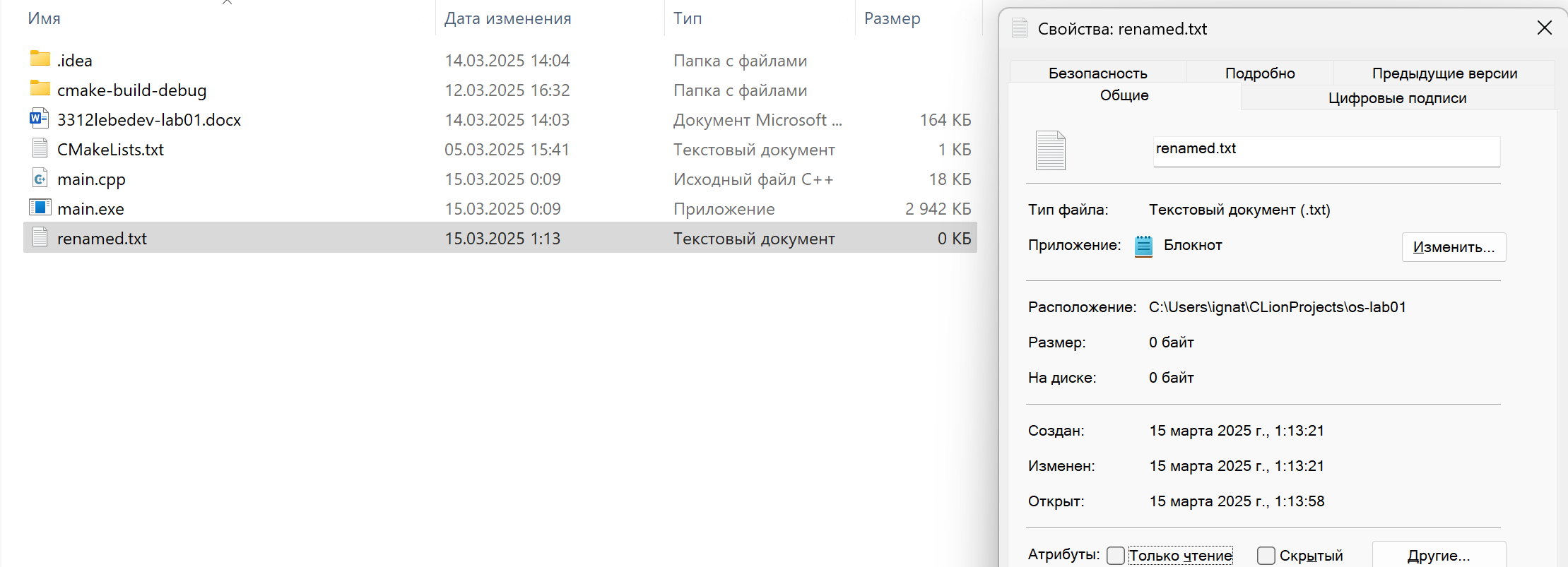


Рисунок 15 – Результат переименования

8. Атрибуты и время файла:

Этот пункт меню позволяет пользователю просмотреть и изменить атрибуты файла, а также узнать и изменить его размер и время создания и последнего изменения.

Реализация начинается с запроса пути к файлу, после чего программа получает его текущие атрибуты с помощью *GetFileAttributesW()* и выводит их в удобном виде. Затем пользователю предлагается изменить скрытый атрибут с использованием *SetFileAttributesW()*. Далее программа открывает файл и получает его размер с помощью *GetFileSizeEx()*, после чего отображает его на экране. Затем с помощью *GetFileTime()* программа получает и выводит время создания и последнего изменения файла в формате даты и времени. После этого пользователю предлагается изменить эти параметры, установив вместо них текущее системное время с использованием *SetFileTime()*. В случае ошибки при получении или изменении атрибутов, размера или времени программа выводит сообщение с кодом ошибки через *GetLastError()*.

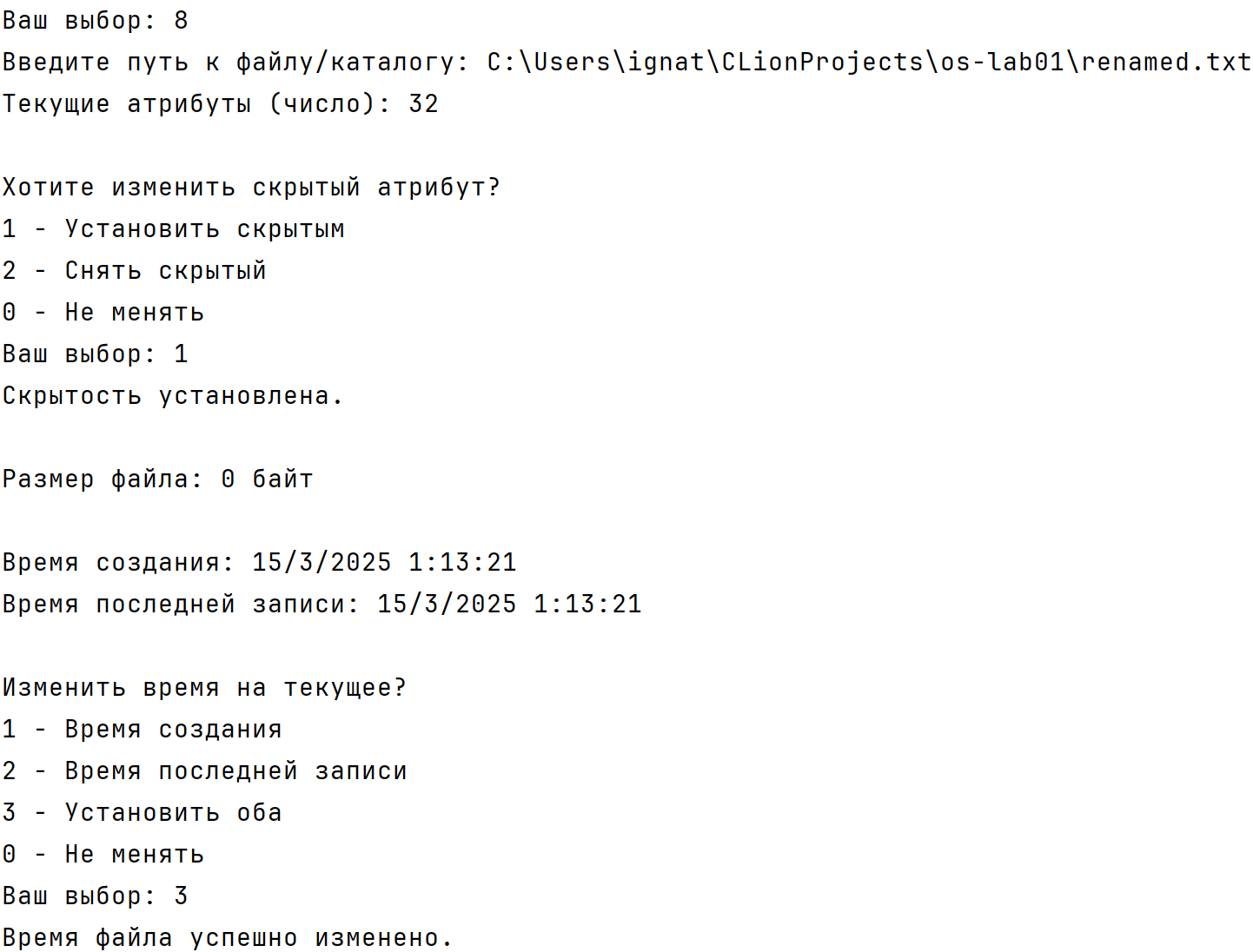


Рисунок 16 – Вывод атрибутов и времени файла и их изменение

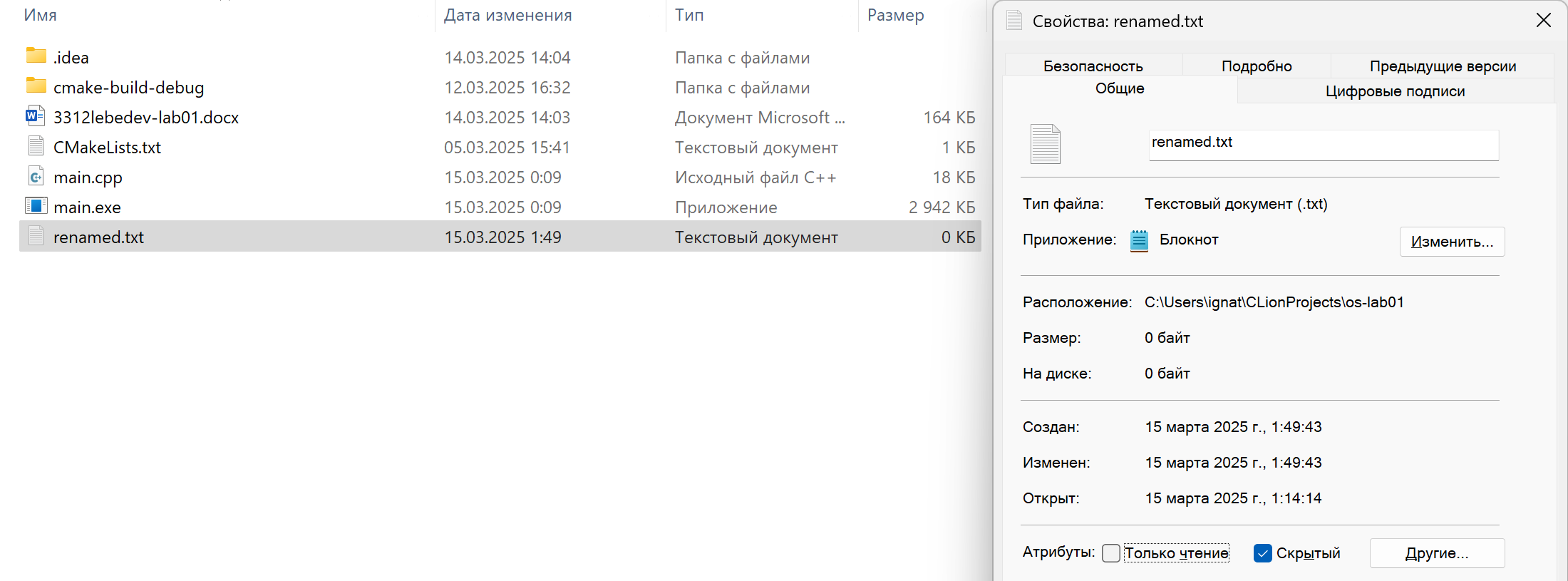


Рисунок 17 – Результаты изменения атрибутов и времени

# Текст программы

**#include <windows.h>  
#include <iostream>  
#include <string>  
#include <functional>  
#include <io.h>  
#include <fcntl.h>  
#include <conio.h>  
#include <limits>  
  
using namespace std;  
  
void SetConsoleToUTF8() {  
 SetConsoleCP(CP\_UTF8);  
 SetConsoleOutputCP(CP\_UTF8);  
 \_setmode(\_fileno(stdout), \_O\_U8TEXT);  
 \_setmode(\_fileno(stdin), \_O\_U8TEXT);  
}  
  
void WaitForKeyPress() {  
 wcout << L"\n(Нажмите любую клавишу для возврата в меню)";  
 \_getch();  
}  
  
void ClearScreen() {  
 system("cls");  
}  
  
void PrintFileSystemFlags(DWORD flags) {  
 if (flags & FILE\_CASE\_SENSITIVE\_SEARCH)  
 wcout << L" - FILE\_CASE\_SENSITIVE\_SEARCH (Поиск с учётом регистра)\n";  
 if (flags & FILE\_CASE\_PRESERVED\_NAMES)  
 wcout << L" - FILE\_CASE\_PRESERVED\_NAMES (Сохраняет регистр имён)\n";  
 if (flags & FILE\_FILE\_COMPRESSION)  
 wcout << L" - FILE\_FILE\_COMPRESSION (Сжатие файлов)\n";  
 if (flags & FILE\_SUPPORTS\_ENCRYPTION)  
 wcout << L" - FILE\_SUPPORTS\_ENCRYPTION (Шифрование файлов)\n";  
 if (flags & FILE\_SUPPORTS\_SPARSE\_FILES)  
 wcout << L" - FILE\_SUPPORTS\_SPARSE\_FILES (Разреженные файлы)\n";  
 if (flags & FILE\_SUPPORTS\_REPARSE\_POINTS)  
 wcout << L" - FILE\_SUPPORTS\_REPARSE\_POINTS (Символические ссылки, junctions)\n";  
 if (flags & FILE\_VOLUME\_IS\_COMPRESSED)  
 wcout << L" - FILE\_VOLUME\_IS\_COMPRESSED (Том сжат)\n";  
 if (flags & FILE\_READ\_ONLY\_VOLUME)  
 wcout << L" - FILE\_READ\_ONLY\_VOLUME (Только для чтения)\n";  
 if (flags & FILE\_SUPPORTS\_HARD\_LINKS)  
 wcout << L" - FILE\_SUPPORTS\_HARD\_LINKS (Жёсткие ссылки)\n";  
 if (flags & FILE\_SUPPORTS\_TRANSACTIONS)  
 wcout << L" - FILE\_SUPPORTS\_TRANSACTIONS (Транзакции)\n";  
 if (flags & FILE\_SUPPORTS\_OBJECT\_IDS)  
 wcout << L" - FILE\_SUPPORTS\_OBJECT\_IDS (Уникальные ID файлов)\n";  
 if (flags & FILE\_SUPPORTS\_USN\_JOURNAL)  
 wcout << L" - FILE\_SUPPORTS\_USN\_JOURNAL (Журнал изменений)\n";  
 if (flags & FILE\_SUPPORTS\_OPEN\_BY\_FILE\_ID)  
 wcout << L" - FILE\_SUPPORTS\_OPEN\_BY\_FILE\_ID (Открытие по ID файла)\n";  
 if (flags & FILE\_SUPPORTS\_INTEGRITY\_STREAMS)  
 wcout << L" - FILE\_SUPPORTS\_INTEGRITY\_STREAMS (Контроль целостности, ReFS)\n";  
 if (flags & FILE\_SUPPORTS\_BLOCK\_REFCOUNTING)  
 wcout << L" - FILE\_SUPPORTS\_BLOCK\_REFCOUNTING (Экономия места, ReFS)\n";  
 if (flags & FILE\_SUPPORTS\_SPARSE\_VDL)  
 wcout << L" - FILE\_SUPPORTS\_SPARSE\_VDL (Оптимизированные sparse, ReFS)\n";  
  
 DWORD knownFlags =  
 FILE\_CASE\_SENSITIVE\_SEARCH | FILE\_CASE\_PRESERVED\_NAMES  
 | FILE\_FILE\_COMPRESSION | FILE\_SUPPORTS\_ENCRYPTION  
 | FILE\_SUPPORTS\_SPARSE\_FILES | FILE\_SUPPORTS\_REPARSE\_POINTS  
 | FILE\_VOLUME\_IS\_COMPRESSED | FILE\_READ\_ONLY\_VOLUME  
 | FILE\_SUPPORTS\_HARD\_LINKS | FILE\_SUPPORTS\_TRANSACTIONS  
 | FILE\_SUPPORTS\_OBJECT\_IDS | FILE\_SUPPORTS\_USN\_JOURNAL  
 | FILE\_SUPPORTS\_OPEN\_BY\_FILE\_ID | FILE\_SUPPORTS\_INTEGRITY\_STREAMS  
 | FILE\_SUPPORTS\_BLOCK\_REFCOUNTING | FILE\_SUPPORTS\_SPARSE\_VDL;  
  
 DWORD unknown = flags & ~knownFlags;  
 if (unknown != 0) {  
 wcout << L" - Доп. флаги: 0x" << std::hex << unknown << std::dec << endl;  
 }  
}  
  
void ListDrives() {  
 DWORD bufSize = GetLogicalDriveStringsW(0, nullptr);  
 if (bufSize == 0) {  
 wcerr << L"Не удалось получить список дисков!" << endl;  
 return;  
 }  
 auto\* buffer = new wchar\_t[bufSize];  
 DWORD result = GetLogicalDriveStringsW(bufSize, buffer);  
 if (result == 0) {  
 wcerr << L"Ошибка при получении списка дисков!" << endl;  
 delete[] buffer;  
 return;  
 }  
 wcout << L"--- Доступные диски ---" << endl;  
 wchar\_t\* current = buffer;  
 while (\*current) {  
 wcout << current << endl;  
 current += wcslen(current) + 1;  
 }  
 delete[] buffer;  
}  
  
void ShowDriveInfo() {  
 wstring drive;  
 wcout << L"Введите букву диска (например, C): ";  
 wcin >> drive;  
  
 if (drive.size() == 1) {  
 drive += L":\\";  
 } else if (drive.size() == 2 && drive[1] == L':') {  
 drive += L"\\";  
 }  
  
 UINT driveType = GetDriveTypeW(drive.c\_str());  
 wcout << L"\nТип диска: ";  
 switch (driveType) {  
 case DRIVE\_UNKNOWN: wcout << L"Неизвестный"; break;  
 case DRIVE\_NO\_ROOT\_DIR: wcout << L"Нет корневого каталога"; break;  
 case DRIVE\_REMOVABLE: wcout << L"Съёмный диск"; break;  
 case DRIVE\_FIXED: wcout << L"Жёсткий диск (Fixed)"; break;  
 case DRIVE\_REMOTE: wcout << L"Сетевой диск"; break;  
 case DRIVE\_CDROM: wcout << L"CD/DVD диск"; break;  
 case DRIVE\_RAMDISK: wcout << L"RAM диск"; break;  
 default: wcout << L"Неизвестный"; break;  
 }  
 wcout << endl;  
  
 wchar\_t volName[MAX\_PATH] = {0};  
 wchar\_t fsName[MAX\_PATH] = {0};  
 DWORD serialNumber = 0, maxLen = 0, fsFlags = 0;  
 if (GetVolumeInformationW(  
 drive.c\_str(),  
 volName, MAX\_PATH,  
 &serialNumber,  
 &maxLen,  
 &fsFlags,  
 fsName, MAX\_PATH))  
 {  
 wcout << L"Метка тома: " << (volName[0] ? volName : L"(нет)") << endl;  
 wcout << L"Файловая система: " << (fsName[0] ? fsName : L"(нет)") << endl;  
 wcout << L"Серийный номер: " << serialNumber << endl;  
 wcout << L"Макс. длина имени файла: " << maxLen << endl;  
  
 wcout << L"\nСистемные флаги файловой системы:\n";  
 PrintFileSystemFlags(fsFlags);  
  
 } else {  
 wcerr << L"Ошибка: не удалось получить информацию о томе." << endl;  
 }  
  
 DWORD sectorsPerCluster, bytesPerSector, freeClusters, totalClusters;  
 if (GetDiskFreeSpaceW(drive.c\_str(),  
 &sectorsPerCluster,  
 &bytesPerSector,  
 &freeClusters,  
 &totalClusters)) {  
 ULONGLONG freeBytes =  
 (ULONGLONG) sectorsPerCluster \* bytesPerSector \* freeClusters;  
 ULONGLONG totalBytes =  
 (ULONGLONG) sectorsPerCluster \* bytesPerSector \* totalClusters;  
 wcout << L"\nСвободно: " << freeBytes << L" байт" << endl;  
 wcout << L"Всего на диске: " << totalBytes << L" байт" << endl;  
 } else {  
 wcerr << L"Не удалось узнать свободное место." << endl;  
 }  
}  
  
void CreateDirectory() {  
 wcin.ignore();  
 wstring dirPath;  
 wcout << L"Введите путь для нового каталога: ";  
 getline(wcin, dirPath);  
  
 if (CreateDirectoryW(dirPath.c\_str(), nullptr)) {  
 wcout << L"Каталог создан: " << dirPath << endl;  
 } else {  
 wcerr << L"Ошибка создания каталога. Код: " << GetLastError() << endl;  
 }  
}  
  
void RemoveDirectory() {  
 wcin.ignore();  
 wstring rootPath;  
 wcout << L"Введите путь для удаления каталога: ";  
 getline(wcin, rootPath);  
  
 DWORD fa = GetFileAttributesW(rootPath.c\_str());  
 if (fa == INVALID\_FILE\_ATTRIBUTES || !(fa & FILE\_ATTRIBUTE\_DIRECTORY)) {  
 wcerr << L"Ошибка: Каталог не найден.\n";  
 return;  
 }  
  
 std::function<void(const wstring&)> removeAll = [&](const wstring& dir) {  
 WIN32\_FIND\_DATAW fd;  
 wstring sp = dir + L"\\\*";  
 HANDLE h = FindFirstFileW(sp.c\_str(), &fd);  
 if (h != INVALID\_HANDLE\_VALUE) {  
 do {  
 wstring name = fd.cFileName;  
 if (name == L"." || name == L"..") continue;  
 wstring full = dir + L"\\" + name;  
  
 if (fd.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY) {  
 removeAll(full);  
 } else {  
 if (!DeleteFileW(full.c\_str())) {  
 wcerr << L"Ошибка удаления файла: " << full  
 << L". Код: " << GetLastError() << endl;  
 }  
 }  
 } while (FindNextFileW(h, &fd));  
 FindClose(h);  
 }  
  
 SetCurrentDirectoryW(L"C:\\");  
 if (!RemoveDirectoryW(dir.c\_str())) {  
 wcerr << L"Ошибка удаления каталога. Код: " << GetLastError() << endl;  
 } else {  
 wcout << L"Каталог удалён: " << dir << endl;  
 }  
 };  
 removeAll(rootPath);  
}  
  
void CreateFile() {  
 wcin.ignore();  
 wstring filePath;  
 wcout << L"Введите полный путь для создания файла: ";  
 getline(wcin, filePath);  
  
 HANDLE hFile = CreateFileW(filePath.c\_str(),  
 GENERIC\_WRITE,  
 0,  
 nullptr,  
 CREATE\_NEW,  
 FILE\_ATTRIBUTE\_NORMAL,  
 nullptr);  
 if (hFile == INVALID\_HANDLE\_VALUE) {  
 wcerr << L"Ошибка создания файла. Код: " << GetLastError() << endl;  
 } else {  
 wcout << L"Файл создан: " << filePath << endl;  
 CloseHandle(hFile);  
 }  
}  
  
void CopyFile() {  
 wcin.ignore();  
 wstring src, dst;  
 wcout << L"Путь исходного файла: ";  
 getline(wcin, src);  
 wcout << L"Путь для копии файла: ";  
 getline(wcin, dst);  
  
 if (CopyFileW(src.c\_str(), dst.c\_str(), FALSE)) {  
 wcout << L"Файл успешно скопирован в " << dst << endl;  
 } else {  
 wcerr << L"Ошибка копирования файла. Код: " << GetLastError() << endl;  
 }  
}  
  
void MoveFile() {  
 wcin.ignore();  
 wstring src, dst;  
 wcout << L"Исходный файл: ";  
 getline(wcin, src);  
 wcout << L"Новый путь/имя: ";  
 getline(wcin, dst);  
  
 if (MoveFileW(src.c\_str(), dst.c\_str())) {  
 wcout << L"Файл перемещён/переименован в: " << dst << endl;  
 } else {  
 wcerr << L"Ошибка перемещения файла. Код: " << GetLastError() << endl;  
 }  
}  
  
void FileAttributes() {  
 wcin.ignore();  
 wstring filePath;  
 wcout << L"Введите путь к файлу/каталогу: ";  
 getline(wcin, filePath);  
  
 DWORD attrs = GetFileAttributesW(filePath.c\_str());  
 if (attrs == INVALID\_FILE\_ATTRIBUTES) {  
 wcerr << L"Ошибка чтения атрибутов. Код: " << GetLastError() << endl;  
 return;  
 }  
 wcout << L"Текущие атрибуты (число): " << attrs << endl;  
  
 wcout << L"\nХотите изменить скрытый атрибут?\n"  
 << L"1 - Установить скрытым\n"  
 << L"2 - Снять скрытый\n"  
 << L"0 - Не менять\n"  
 << L"Ваш выбор: ";  
 int c;  
 wcin >> c;  
 if (c == 1) {  
 attrs |= FILE\_ATTRIBUTE\_HIDDEN;  
 if (!SetFileAttributesW(filePath.c\_str(), attrs)) {  
 wcerr << L"Ошибка установки скрытости. Код: " << GetLastError() << endl;  
 } else {  
 wcout << L"Скрытость установлена." << endl;  
 }  
 } else if (c == 2) {  
 attrs &= ~FILE\_ATTRIBUTE\_HIDDEN;  
 if (!SetFileAttributesW(filePath.c\_str(), attrs)) {  
 wcerr << L"Ошибка снятия скрытости. Код: " << GetLastError() << endl;  
 } else {  
 wcout << L"Скрытый атрибут снят." << endl;  
 }  
 }  
  
 HANDLE hFile = CreateFileW(filePath.c\_str(),  
 GENERIC\_READ | GENERIC\_WRITE,  
 FILE\_SHARE\_READ | FILE\_SHARE\_WRITE,  
 nullptr,  
 OPEN\_EXISTING,  
 FILE\_ATTRIBUTE\_NORMAL,  
 nullptr);  
 if (hFile == INVALID\_HANDLE\_VALUE) {  
 wcerr << L"Ошибка открытия файла для работы со временем. Код: " << GetLastError() << endl;  
 return;  
 }  
  
 LARGE\_INTEGER fsize;  
 if (GetFileSizeEx(hFile, &fsize)) {  
 wcout << L"\nРазмер файла: " << fsize.QuadPart << L" байт" << endl;  
 }  
  
  
 FILETIME ftCreate, ftWrite;  
  
 if (!GetFileTime(hFile, &ftCreate, nullptr, &ftWrite)) {  
 wcerr << L"Не удалось получить время файла. Код: " << GetLastError() << endl;  
 CloseHandle(hFile);  
 return;  
 }  
  
 {  
 SYSTEMTIME stCreateUTC, stCreateLocal;  
 FileTimeToSystemTime(&ftCreate, &stCreateUTC);  
 SystemTimeToTzSpecificLocalTime(nullptr, &stCreateUTC, &stCreateLocal);  
 wcout << L"\nВремя создания: "  
 << stCreateLocal.wDay << L"/" << stCreateLocal.wMonth << L"/" << stCreateLocal.wYear  
 << L" " << stCreateLocal.wHour << L":" << stCreateLocal.wMinute << L":" << stCreateLocal.wSecond << endl;  
 }  
 {  
 SYSTEMTIME stWrUTC, stWrLocal;  
 FileTimeToSystemTime(&ftWrite, &stWrUTC);  
 SystemTimeToTzSpecificLocalTime(nullptr, &stWrUTC, &stWrLocal);  
 wcout << L"Время последней записи: "  
 << stWrLocal.wDay << L"/" << stWrLocal.wMonth << L"/" << stWrLocal.wYear  
 << L" " << stWrLocal.wHour << L":" << stWrLocal.wMinute << L":" << stWrLocal.wSecond << endl;  
 }  
  
 wcout << L"\nИзменить время на текущее?\n"  
 << L"1 - Время создания\n"  
 << L"2 - Время последней записи\n"  
 << L"3 - Установить оба\n"  
 << L"0 - Не менять\n"  
 << L"Ваш выбор: ";  
 int timeCho;  
 wcin >> timeCho;  
 FILETIME \*pC = nullptr;  
 FILETIME \*pW = nullptr;  
  
 if (timeCho != 0) {  
 SYSTEMTIME stUTCNow;  
 GetSystemTime(&stUTCNow);  
 FILETIME ftNow;  
 SystemTimeToFileTime(&stUTCNow, &ftNow);  
  
 if (timeCho == 1) {  
 pC = &ftNow;  
 } else if (timeCho == 2) {  
 pW = &ftNow;  
 } else if (timeCho == 3) {  
 pC = &ftNow;  
 pW = &ftNow;  
 }  
  
 if (!SetFileTime(hFile, pC, nullptr, pW)) {  
 wcerr << L"Не удалось изменить время. Код: " << GetLastError() << endl;  
 } else {  
 wcout << L"Время файла успешно изменено." << endl;  
 }  
 }  
  
 CloseHandle(hFile);  
}  
  
int main() {  
 SetConsoleToUTF8();  
 setlocale(LC\_ALL, "ru\_RU.UTF-8");  
  
 while (true) {  
 wcout << L"\n+----------------------------------+\n";  
 wcout << L"| ГЛАВНОЕ МЕНЮ |\n";  
 wcout << L"|----------------------------------|\n";  
 wcout << L"| [1] Список дисков |\n";  
 wcout << L"| [2] Информация о диске |\n";  
 wcout << L"| [3] Создать каталог |\n";  
 wcout << L"| [4] Удалить каталог |\n";  
 wcout << L"| [5] Создать файл |\n";  
 wcout << L"| [6] Копировать файл |\n";  
 wcout << L"| [7] Переместить файл |\n";  
 wcout << L"| [8] Атрибуты/время файла |\n";  
 wcout << L"| [0] Выйти из программы |\n";  
 wcout << L"+----------------------------------+\n";  
 wcout << L"Ваш выбор: ";  
  
 int choice;  
 if (!(wcin >> choice)) {  
 wcin.clear();  
 wcin.ignore(numeric\_limits<streamsize>::max(), L'\n');  
 ClearScreen();  
 wcout << L"Некорректный ввод. Попробуйте снова.\n";  
 continue;  
 }  
  
 switch (choice) {  
 case 0:  
 wcout << L"Завершение работы.\n";  
 return 0;  
 case 1:  
 ListDrives();  
 break;  
 case 2:  
 ShowDriveInfo();  
 break;  
 case 3:  
 CreateDirectory();  
 break;  
 case 4:  
 RemoveDirectory();  
 break;  
 case 5:  
 CreateFile();  
 break;  
 case 6:  
 CopyFile();  
 break;  
 case 7:  
 MoveFile();  
 break;  
 case 8:  
 FileAttributes();  
 break;  
 default:  
 ClearScreen();  
 wcout << L"Нет такого пункта. Повторите ввод.\n";  
 continue;  
 }  
  
 WaitForKeyPress();  
 ClearScreen();  
 }  
 return 0;  
}**

# Выводы

В ходе работы было разработано консольное приложение для управления файлами и каталогами с использованием *Windows API*. Реализованы функции просмотра логических дисков, получения информации о диске, создания и удаления каталогов, работы с файлами (создание, копирование, перемещение), а также просмотра размера файла и изменения его атрибутов и временных меток.

Использованы основные функции *Windows API*, такие как *GetLogicalDriveStringsW()*, *CreateFileW()*, *CopyFileW()*, *MoveFileW()*, *RemoveDirectoryW()*, *GetFileAttributesW()*, *SetFileTime()*, *GetFileSizeEx()*, что обеспечило прямое взаимодействие с файловой системой.

В результате были освоены методы работы с файловой системой *Windows*, а итоговая программа позволяет удобно выполнять базовые операции с файлами и каталогами через консоль.

# Задание 1.2

Во втором задании необходимо реализовать консольное приложение, выполняющее копирование файла в *Linux* с использованием механизма перекрывающегося ввода-вывода. В процессе работы программа должна одновременно выполнять операций ввода-вывода, используя разбиение на блоки кратные размеру кластера.

Приложение выполняет следующие операции:

- Открытие исходного и целевого файлов.

- Чтение и запись данных блоками, кратными размеру кластера.

- Использование неблокирующего ввода-вывода.

- Ожидание завершения операций.

- Измерение времени выполнения копирования.

После реализации проводится тестирование копирования файлов при разных размерах блоков данных а также при разном количестве операций. На основе полученных данных строится график зависимости скорости копирования от размера блока, а также определяется оптимальное число параллельных операций ввода-вывода.

# Примеры запуска программы

Программа предназначена для копирования файлов в *Linux* с использованием асинхронного ввода-вывода (*AIO*), что позволяет обрабатывать несколько операций чтения и записи одновременно.

Работа программы начинается с получения входных параметров:

- Имя исходного файла.

- Имя файла назначения.

- Размер блока копирования (по умолчанию 4096 байт).

- Число перекрывающихся операций (по умолчанию 1).

Если параметры размера блока или числа операций не указаны, программа устанавливает их в значения по умолчанию.

Затем программа открывает исходный и целевой файлы, получает их размер с помощью *fstat()* и рассчитывает общее количество блоков.

Процесс копирования выполняется с использованием асинхронных операций чтения и записи (*aio\_read()* и *aio\_write()*) в несколько потоков ввода-вывода:

1. Выделяются буферы и создаются управляющие структуры (*aiocb*) для работы с асинхронным вводом-выводом.

2. Выполняются асинхронные операции чтения из исходного файла.

3. Когда чтение завершено, данные асинхронно записываются в целевой файл.

4. Программа использует *aio\_suspend()* для ожидания завершения активных операций и обработки данных по мере их готовности.

5. После завершения всех операций программа закрывает файлы и освобождает память.

После завершения копирования программа выводит статистику:

- Исходный и целевой файлы.

- Размер файла.

- Размер блока копирования.

- Количество перекрывающихся операций.

- Общее количество блоков.

- Время копирования в секундах.

Использование асинхронного ввода-вывода позволяет программе повышать производительность при увеличении количества операций, так как чтение и запись выполняются параллельно.

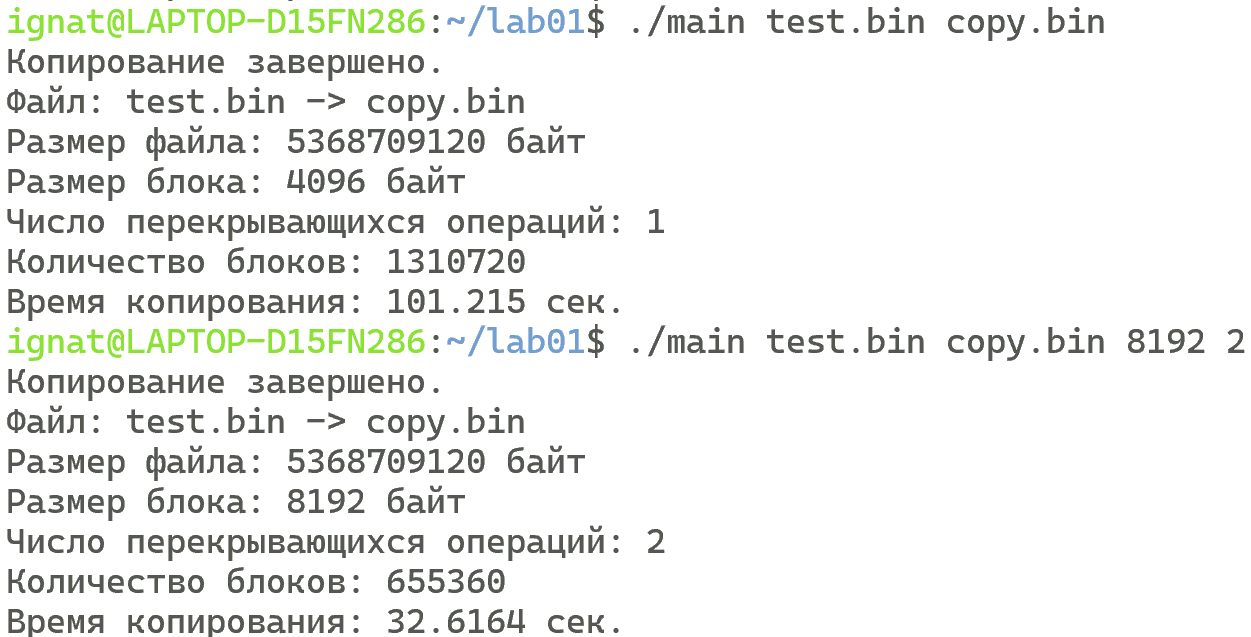


Рисунок 18 – Примеры работы программы

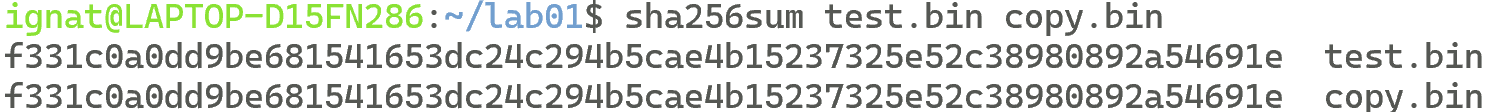


Рисунок 19 – Сравнение хэш-сумм файлов

# Проверка при разных размерах копируемых блоков

Были проведены замеры скорости копирования файла размером ГБ при фиксированном числе операций () и разном размере блока. График показывает, что при малых размерах блока ( КБ) скорость копирования остается низкой из-за большого количества операций ввода-вывода, что приводит к значительным накладным расходам. С увеличением размера блока наблюдается резкий рост производительности, и наибольшая скорость достигается при размере блока МБ, где копирование стабилизируется на уровне около МБ/с.

При дальнейшем увеличении блока прирост скорости становится незначительным, а после определённого порога (~ МБ) наблюдаются небольшие колебания и даже снижение. Это может быть связано с особенностями файловой системы и механизмами кеширования.

Рисунок 20 – Зависимость скорости копирования от размера блока

# Проверка при разном количестве перекрывающихся операций

Были проведены замеры скорости копирования файла размером ГБ при фиксированном размере блока ( КБ) и разном числе перекрывающихся операций (). График показывает, что при увеличении количества операций скорость копирования сначала значительно растёт, но после параллельных операций прирост становится минимальным, а затем выходит на плато.

При малых значениях n ( операции) прирост скорости значителен, так как перекрывающиеся операции позволяют уменьшить время ожидания ввода-вывода. Однако при дальнейшем увеличении числа потоков (более ) система достигает предела параллелизма, и дальнейший рост числа операций практически не даёт ускорения.

После операций наблюдаются незначительные колебания скорости, что может быть связано с перегрузкой подсистемы ввода-вывода или управлением очередями *AIO*. Таким образом, оптимальное число операций для быстрого копирования при блоке КБ – от до , поскольку дальше прирост скорости становится незначительным.

Рисунок 21 – Зависимость скорости копирования от количества операций

# Текст программы

**#include <iostream>  
#include <vector>  
#include <string>  
#include <cstring>  
#include <cerrno>  
#include <cstdlib>  
#include <unistd.h>  
#include <fcntl.h>  
#include <aio.h>  
#include <sys/stat.h>  
#include <sys/types.h>  
#include <chrono>  
  
struct Slot {  
 aiocb cb;  
 bool inUse;  
 bool isWrite;  
 off\_t offset;  
 ssize\_t bytesCount;  
 char\* buffer;  
};  
  
int main(int argc, char\* argv[]) {  
 setlocale(LC\_ALL, "");  
  
 if (argc < 3) {  
 std::cerr << "Использование: " << argv[0]  
 << " <файл\_источник> <файл\_приёмник> [размер\_блока=4096] [число\_операций=1]\n";  
 return 1;  
 }  
  
 std::string inFile = argv[1];  
 std::string outFile = argv[2];  
  
 size\_t blockSize = 4096;  
 int nOps = 1;  
  
 if (argc >= 4) {  
 size\_t val = std::strtoul(argv[3], nullptr, 10);  
 if (val != 0) blockSize = val;  
 }  
  
 if (argc >= 5) {  
 int val = std::atoi(argv[4]);  
 if (val >= 1) nOps = val;  
 }  
  
 int inFd = open(inFile.c\_str(), O\_RDONLY);  
 if (inFd < 0) {  
 std::cerr << "Ошибка при открытии входного файла \"" << inFile  
 << "\": " << std::strerror(errno) << "\n";  
 return 1;  
 }  
  
 int outFd = open(outFile.c\_str(), O\_WRONLY | O\_CREAT | O\_TRUNC, 0666);  
 if (outFd < 0) {  
 std::cerr << "Ошибка при создании выходного файла \"" << outFile  
 << "\": " << std::strerror(errno) << "\n";  
 close(inFd);  
 return 1;  
 }  
  
 struct stat st;  
 if (fstat(inFd, &st) != 0) {  
 std::cerr << "fstat: " << std::strerror(errno) << "\n";  
 close(inFd);  
 close(outFd);  
 return 1;  
 }  
  
 off\_t totalSize = st.st\_size;  
 if (totalSize == 0) {  
 close(inFd);  
 close(outFd);  
 return 0;  
 }  
  
 off\_t totalBlocks = (totalSize + blockSize - 1) / blockSize;  
  
 auto startTime = std::chrono::high\_resolution\_clock::now();  
  
 std::vector<Slot> slots(nOps);  
 std::vector<char\*> buffers(nOps, nullptr);  
  
 for (int i = 0; i < nOps; i++) {  
 buffers[i] = new char[blockSize];  
 std::memset(&slots[i], 0, sizeof(Slot));  
 slots[i].buffer = buffers[i];  
 slots[i].inUse = false;  
 }  
  
 off\_t nextBlockToRead = 0;  
 off\_t blocksDone = 0;  
 bool done = false;  
  
 while (!done) {  
 done = (blocksDone >= totalBlocks && nextBlockToRead >= totalBlocks);  
  
 if (!done) {  
 for (int i = 0; i < nOps; i++) {  
 if (!slots[i].inUse && nextBlockToRead < totalBlocks) {  
 slots[i].inUse = true;  
 slots[i].isWrite = false;  
 slots[i].offset = nextBlockToRead \* blockSize;  
 std::memset(&slots[i].cb, 0, sizeof(aiocb));  
 slots[i].cb.aio\_fildes = inFd;  
 slots[i].cb.aio\_buf = slots[i].buffer;  
 slots[i].cb.aio\_offset = slots[i].offset;  
 slots[i].cb.aio\_nbytes = blockSize;  
 slots[i].cb.aio\_sigevent.sigev\_notify = SIGEV\_NONE;  
  
 if (aio\_read(&slots[i].cb) != 0) {  
 slots[i].inUse = false;  
 } else {  
 nextBlockToRead++;  
 }  
 }  
 }  
 }  
  
 aiocb\* arr[128];  
 int activeCount = 0;  
 for (int i = 0; i < nOps; i++) {  
 if (slots[i].inUse) {  
 arr[activeCount++] = &slots[i].cb;  
 }  
 }  
  
 if (activeCount == 0) {  
 done = (blocksDone >= totalBlocks && nextBlockToRead >= totalBlocks);  
 if (done) break;  
 continue;  
 }  
  
 if (aio\_suspend(arr, activeCount, nullptr) != 0) {  
 if (errno != EINTR) break;  
 }  
  
 for (int i = 0; i < nOps; i++) {  
 if (!slots[i].inUse) continue;  
  
 int err = aio\_error(&slots[i].cb);  
 if (err == EINPROGRESS) continue;  
 if (err != 0) {  
 slots[i].inUse = false;  
 continue;  
 }  
  
 ssize\_t res = aio\_return(&slots[i].cb);  
 if (res < 0) {  
 slots[i].inUse = false;  
 continue;  
 }  
  
 if (!slots[i].isWrite) {  
 if (res == 0) {  
 slots[i].inUse = false;  
 } else {  
 slots[i].isWrite = true;  
 std::memset(&slots[i].cb, 0, sizeof(aiocb));  
 slots[i].cb.aio\_fildes = outFd; *// вместо open(...), используем общий дескриптор* slots[i].cb.aio\_buf = slots[i].buffer;  
 slots[i].cb.aio\_offset = slots[i].offset;  
 slots[i].cb.aio\_nbytes = static\_cast<size\_t>(res);  
 slots[i].cb.aio\_sigevent.sigev\_notify = SIGEV\_NONE;  
  
 if (aio\_write(&slots[i].cb) != 0) {  
 slots[i].inUse = false;  
 }  
 }  
 } else {  
 slots[i].inUse = false;  
 blocksDone++;  
 }  
 }  
 }  
  
 auto endTime = std::chrono::high\_resolution\_clock::now();  
 double totalSec = std::chrono::duration<double>(endTime - startTime).count();  
  
 std::cout << "Копирование завершено.\n";  
 std::cout << "Файл: " << inFile << " -> " << outFile << "\n";  
 std::cout << "Размер файла: " << totalSize << " байт\n";  
 std::cout << "Размер блока: " << blockSize << " байт\n";  
 std::cout << "Число перекрывающихся операций: " << nOps << "\n";  
 std::cout << "Количество блоков: " << totalBlocks << "\n";  
 std::cout << "Время копирования: " << totalSec << " сек.\n";  
  
 for (int i = 0; i < nOps; i++) {  
 delete[] buffers[i];  
 }  
  
 close(inFd);  
 close(outFd);  
 return 0;  
}**

# Выводы

В ходе выполнения второго задания было разработано консольное приложение для копирования файлов в *Linux* с использованием асинхронного ввода-вывода (*AIO*). Реализована поддержка различных размеров блока и количества перекрывающихся операций, что позволяет исследовать влияние этих параметров на скорость копирования.

Программа использует системные вызовы *Linux*, такие как *aio\_read()*, *aio\_write()*, *aio\_suspend()*, а также *open()*, *fstat()*, *close()*, обеспечивая перекрывающиеся операции ввода-вывода. Это позволяет выполнять чтение и запись параллельно, сокращая время ожидания операций с диском и повышая общую скорость копирования.

В результате тестирования установлено, что размер блока МБ и параллельных операций обеспечивают максимальную скорость копирования для файла размером Гб. Дальнейшее увеличение этих параметров не даёт значительного прироста производительности и может даже ухудшать результат из-за накладных расходов.

В ходе работы были освоены методы асинхронного копирования в *Linux*, изучены особенности работы с *AIO*, а также проведён анализ влияния размера блока и параллелизма на скорость копирования. Итоговая программа позволяет гибко настраивать параметры копирования и оценивать их влияние на производительность.