**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчёт   
ЛАБОРАТОРНАЯ РАБОТА № 1

**по дисциплине «Операционные Системы»**

Тема: **Управление файловой системой**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3311 |  | Шарпинский Д. А. |
| Преподаватель |  | Тимофеев А. В. |

Санкт-Петербург

2025

# ЦЕЛЬ РАБОТЫ:

Исследование управления файловой системой с помощью Win32 API. Копирование файла с помощью операций перекрывающегося ввода-вывода.

# ЗАДАНИЯ НА ЛАБОРАТОРНУЮ РАБОТУ:

**Задание 1.1. Управление дисками, каталогами и файлами.**

1. 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).

2. Запустите приложение и проверьте его работоспособность на

нескольких наборах вводимых данных. Запротоколируйте результаты в отчет.

Дайте свои комментарии в отчете относительно выполнения функций Win32

API.

3. Подготовьте итоговый отчет с развернутыми выводами по

заданию.

**Задание 1.2. Копирование файла с помощью операций перекрывающегося ввода-вывода** (будет изменено на выполнение в

Linux).

Приложение должно копировать существующий файл в новый файл,

«одновременно» выполняя n перекрывающихся операций ввода-вывода

(механизм APC) блоками данных кратными размеру кластера.

Указания к выполнению.

1. Создайте консольное приложение, которое выполняет: −

открытие/создание файлов (функция Win32 API – CreateFile,

обязательно использовать флаги FILE\_FLAG\_NO\_BUFFERING и

FILE\_FLAG\_OVERLAPPED);

∙ файловый ввод-вывод (функции Win32 API – ReadFileEx,

WriteFileEx) блоками кратными размеру кластера;

∙ ожидание срабатывания вызова функции завершения (функция Win32

API – SleepEx);

∙ измерение продолжительности выполнения операции копирования файла

(функция Win32 API – TimeGetTime). 2. Запустите приложение и проверьте

его работоспособность на копировании файлов разного размера для ситуации

с перекрывающимся выполнением одной операции ввода и одной операции

вывода (для сравнения файлов используйте консольную команду FC).

Выполните эксперимент для разного размера копируемых блоков, постройте

график зависимости скорости копирования от размера блока данных.

Определите оптимальный размер блока данных, при котором скорость

копирования наибольшая. Запротоколируйте результаты в отчет. Дайте свои

комментарии в отчете относительно выполнения функций Win32 API.

3. Произведите замеры времени выполнения приложения для разного

числа перекрывающихся операций ввода и вывода (1, 2, 4, 8, 12, 16), не

забывая проверять работоспособность приложения (консольная команда FC).

По результатам измерений постройте график зависимости и определите число

перекрывающихся операций ввода и вывода, при котором достигается

наибольшая скорость копирования файла. Запротоколируйте результаты в

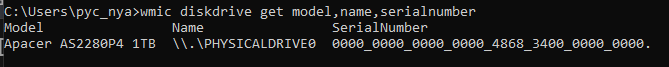
отчет.

4. Подготовьте итоговый отчет с развернутыми выводами по

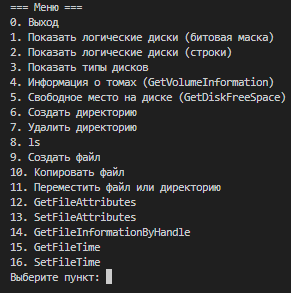
заданию.

# Задание 1.1. Управление дисками, каталогами и файлами.

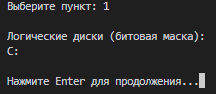
Код запускался на компьютере со следующим SSD:



Главное меню. Демонстрация главного меню функций для работы с файлами:

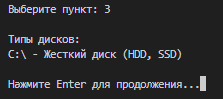


1. Демонстрация работы первой функции главного меню:



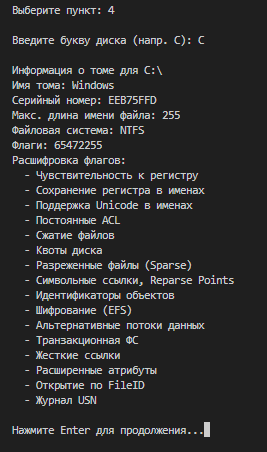
Здесь вызывается showLogicalDrives(), использующая GetLogicalDrives() из WIN32 API.

1. Функция 2 из меню имеет схожий результат работы.
2. Демонстрация работы третьей функции главного меню:

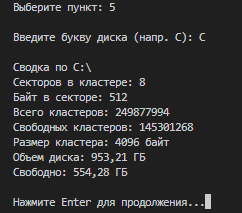


Здесь вызывается showDriveTypes() – выводится тип диска. Использована GetLogicalDriveStrings()

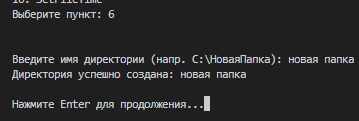
1. Демонстрация работы четвертой функции главного меню:



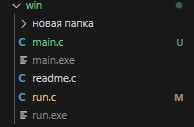
1. Демонстрация работы пятой функции главного меню:



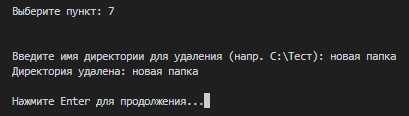
1. Демонстрация работы шестой функции главного меню:



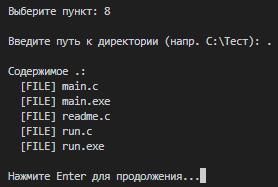
(Была создана новая директория с указанным названием в той же папке, где находился исполняемый файл)



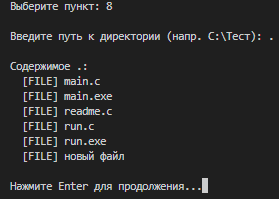
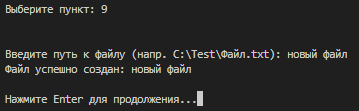
1. Демонстрация работы седьмой функции главного меню:



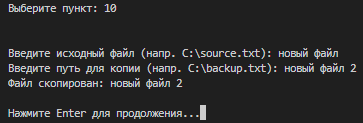
1. Демонстрация работы восьмой функции главного меню:

Данная функция не являлась обязательной к выполнению и была скорее создана для дебага. Результат работы:

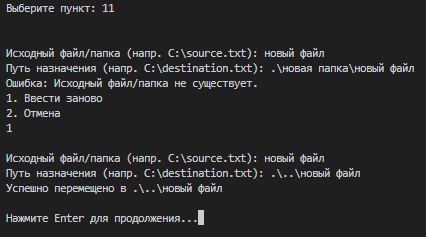
1. Демонстрация работы девятой функции главного меню:



1. Демонстрация работы десятой функции главного меню:

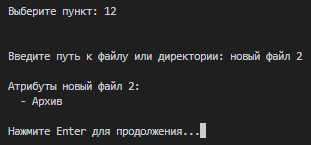


1. Демонстрация работы одиннадцатой функции главного меню:

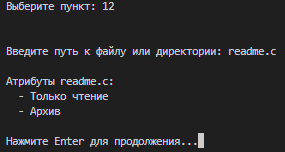
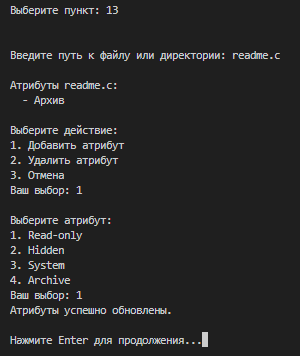


На примере данной функции я бы хотел обратить внимание, что программа устойчива к разным ситуациям и может предложить разные способы решения возможных проблем, как, например, некорректный путь.

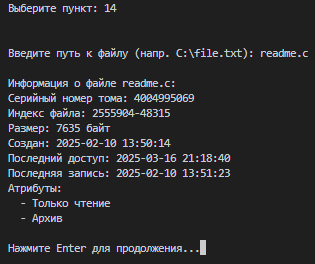
1. Демонстрация работы двенадцатой функции главного меню:



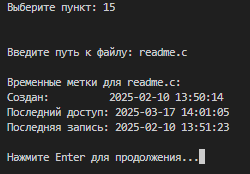
1. Демонстрация работы тринадцатой функции главного меню:



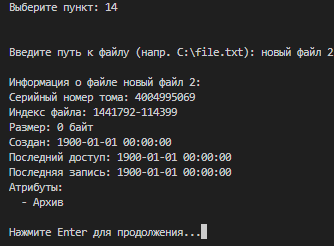
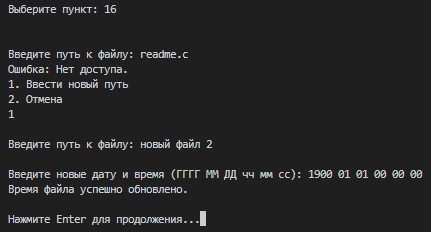
1. Демонстрация работы четырнадцатой функции главного меню:



1. Демонстрация работы пятнадцатой функции главного меню:



1. Демонстрация работы шестнадцатой функции главного меню:



**Итог**

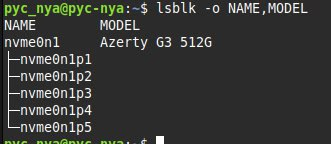
Разработанное консольное приложение успешно решает все пункты задания по работе с дисками, каталогами и файлами в среде Windows. В отчёте приведены примеры тестирования, подтверждающие корректное поведение при разных ситуациях (ошибочных и штатных). Использование Win32 API (CreateFile, CopyFile, MoveFile, GetFileAttributes, SetFileAttributes и проч.) показало, что:

1. Программа способна корректно анализировать состояние файловой системы, работать с атрибутами и атрибутами времени.
2. Обработка ошибок и детальные диалоговые подсказки обеспечивают надёжность и удобство эксплуатации.
3. Все требования задания (от отображения дисков до изменения времени файла) выполнены, что подтверждается экспериментальными проверками.

Данное решение можно считать завершённым и корректным для части 1.1 лабораторной работы.

**Задание 1.2. Копирование файла с помощью операций перекрывающегося ввода-вывода.**

Данная часть работы выполнялась на ноутбуке с операционной системой Linux Mint и следующим SSD:



Так как копирование производилось с помощью асинхронных операций необходимо сначала представить реализацию ожидания завершения операций. В данной программе используются два основных механизма синхронизации – **мьютекс** и **условная переменная** – для корректного ожидания завершения асинхронных операций без излишней загрузки процессора (то есть без активного ожидания). Поскольку в программе используется SIGEV\_THREAD, то для каждой асинхронной операции создается новый поток, а значит использование mutex и conditional variable является подходящим решением.

**Мьютекс** – это объект, который гарантирует, что только один поток в определённый момент времени может получить доступ к критической секции кода или разделяемым данным. В данном задании это свойство было не столь важно, мьютекс использовался потому, что он необходим для инициализации условной переменной.

**Условная переменная** позволяет потоку (или потокам) «заснуть» и ждать наступления определённого условия (без нагрузки на процессор), вместо того чтобы постоянно проверять состояние (что называется активным ожиданием).

У условных переменных есть механизм notify\_all(), который уведомляет о завершении работы данного потока. При получении уведомления вызывается callback, который проверяет, каждый ли поток завершил работу – если так, то копирование считается завершенным, иначе выполнение продолжается.

* Тестирование проводилось на файлах размерами 10mb, 100mb и 1gb. Файлы генерировались случайно с помощью следующей команды:

dd if=/dev/urandom of=random\_file.bin bs=1M count=10

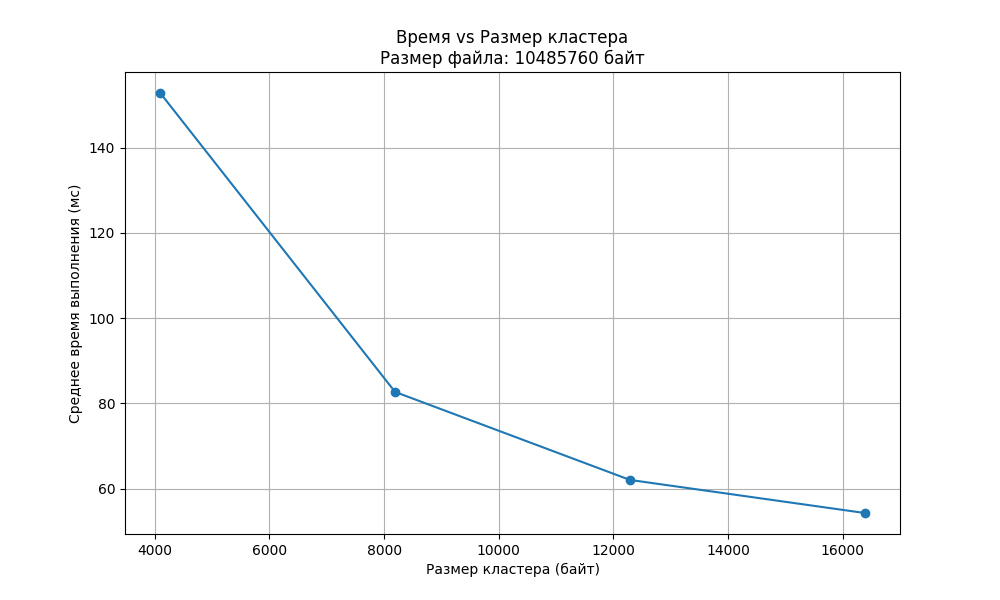
* Для каждого файла проводилось по 5 тестов с одинаковыми входными данными, а именно: количеством байт, отводимых под чтение и запись, и количество одновременных асинхронных операций.
* Между итерациями делался вызов fsync() и sync() для чистоты бенчмарка.
* На начальных этапах была проведена проверка точности копирования с помощью команды:

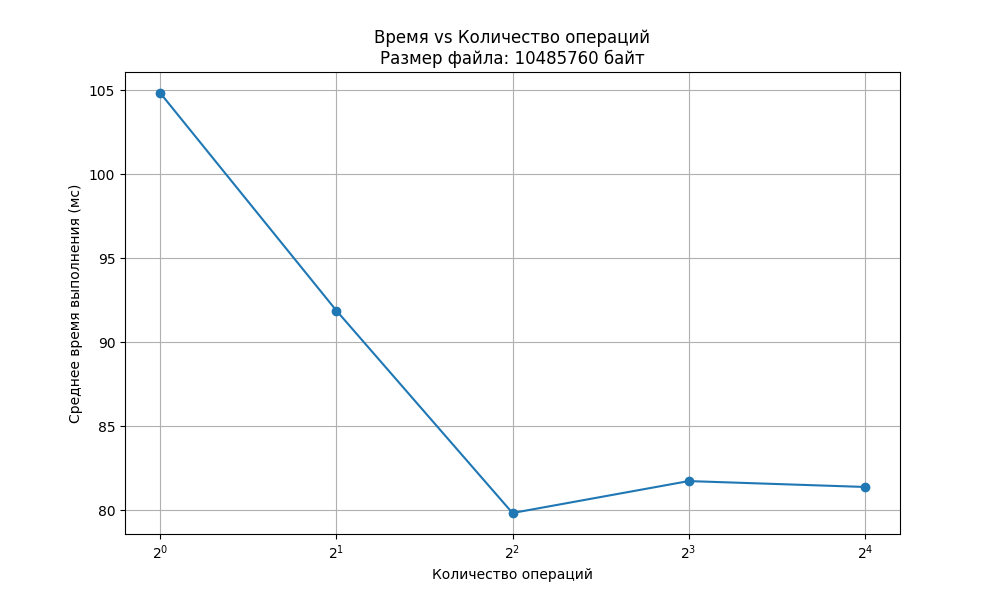
md5sum <file\_name>

Для оригинального файла и его копии. Хэш-сумма совпадала в 100% случаев.

Результаты бенчмарка продемонстрированы на графиках (получены на основании логов программы):

Файл размером 10mb:

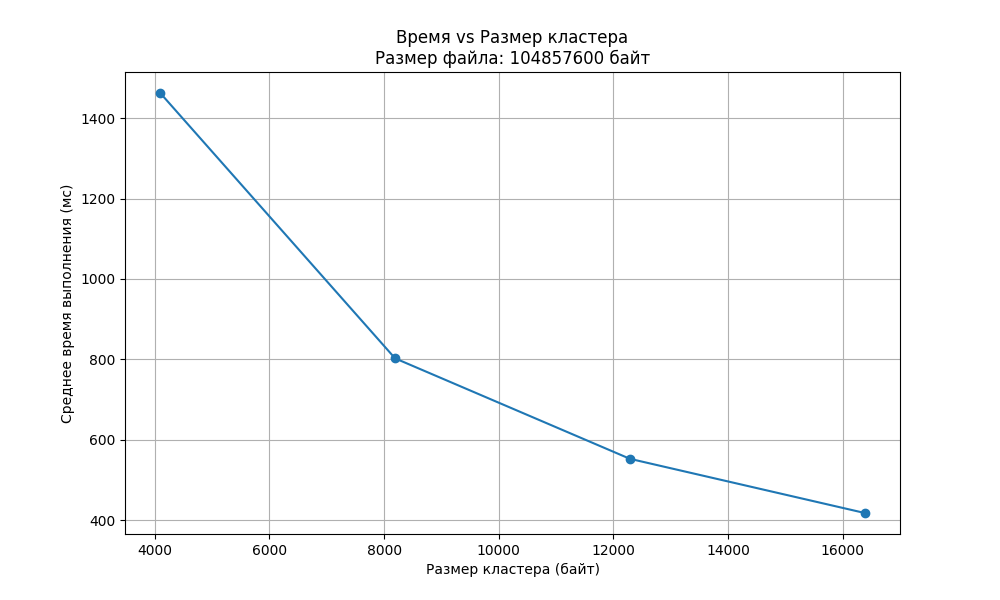


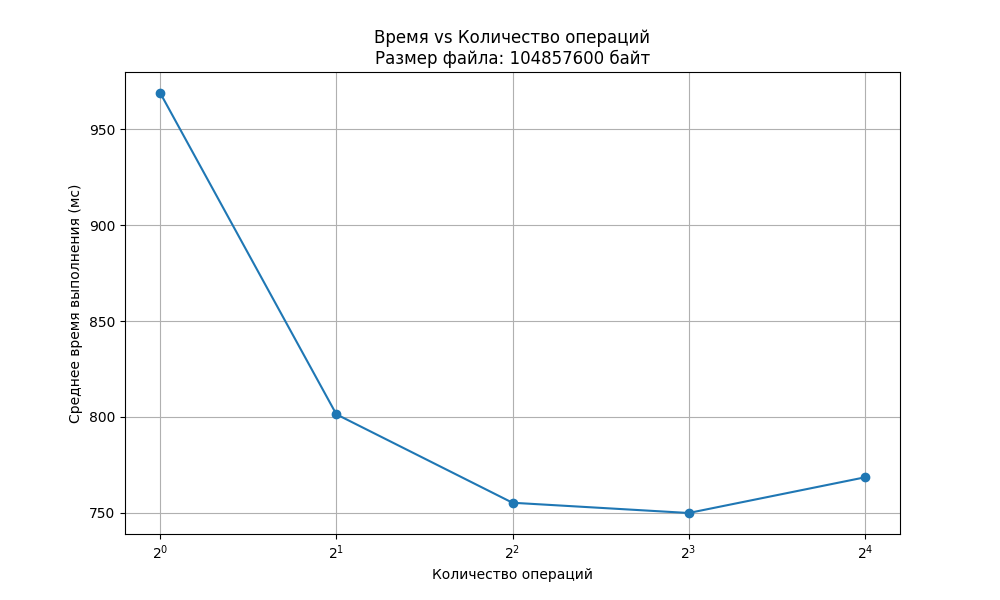


Промежуточный итог:

Для файла размером 10 мегабайт очевидно, что большее количество байт на одну операцию ведет к экспоненциальному понижению времени на копирование. Самыми эффективными показали себя запуски программы с 4 одновременными асинхронными операциями.

Файл размером 100mb:

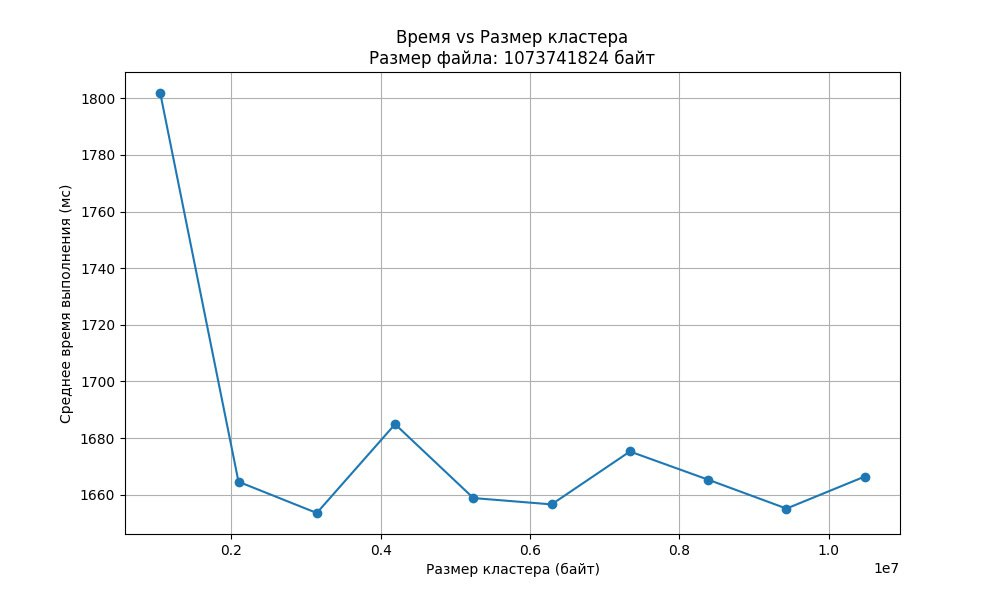


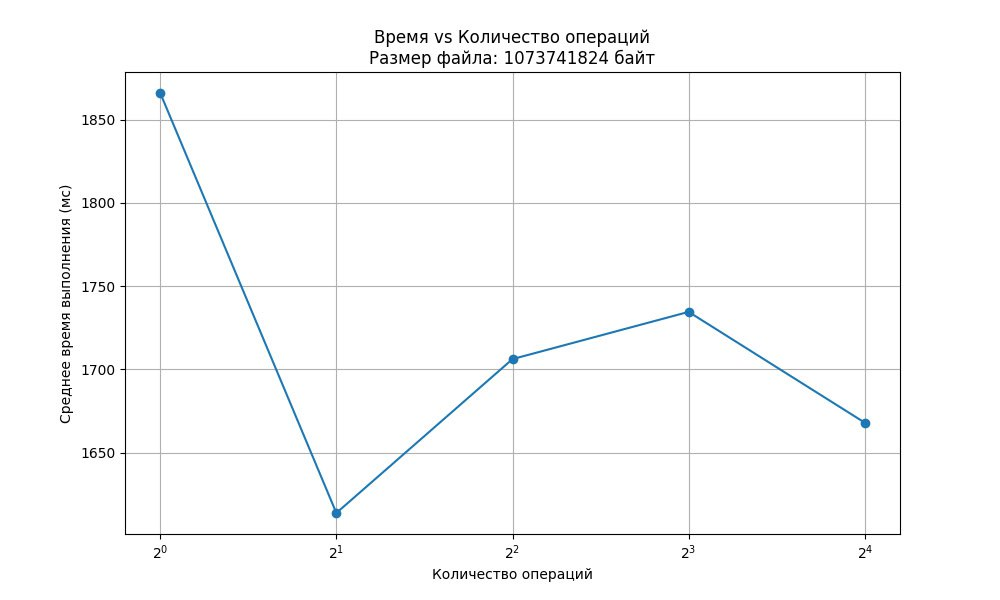


Промежуточный итог:

Для файла размером 100 мегабайт результаты в общем схожи с результатами для файла размером 10мб, но с той разницей, что 8 потоков показали себя более эффективными. Однако разница крайне невелика, так что можно списать это на погрешность.

Файл размером 1gb:





Для проведения бенчмарка для файла такого размера в коде между итерациями был добавлен вызов функции sleep(2), для того, чтобы на 100% гарантировать чистоту эксперимента и независимость каждой итерации.

Промежуточный итог:

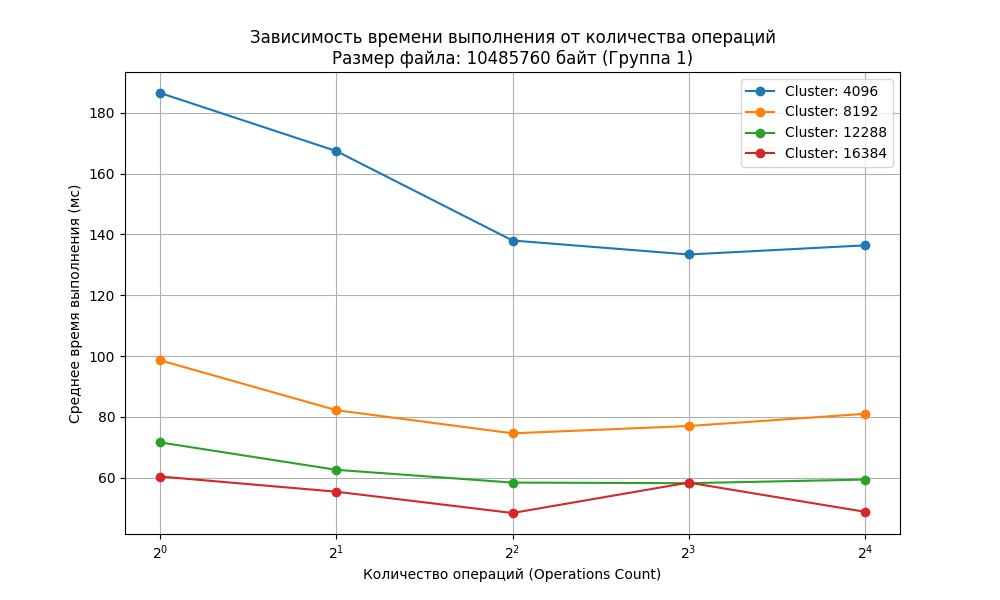
Для файла размером 1 гигабайт результаты в целом сохраняют тенденцию, заданную двумя предыдущими: большему размеру кластера соответствует меньшее время, при большом количестве потоков время может возрастать.

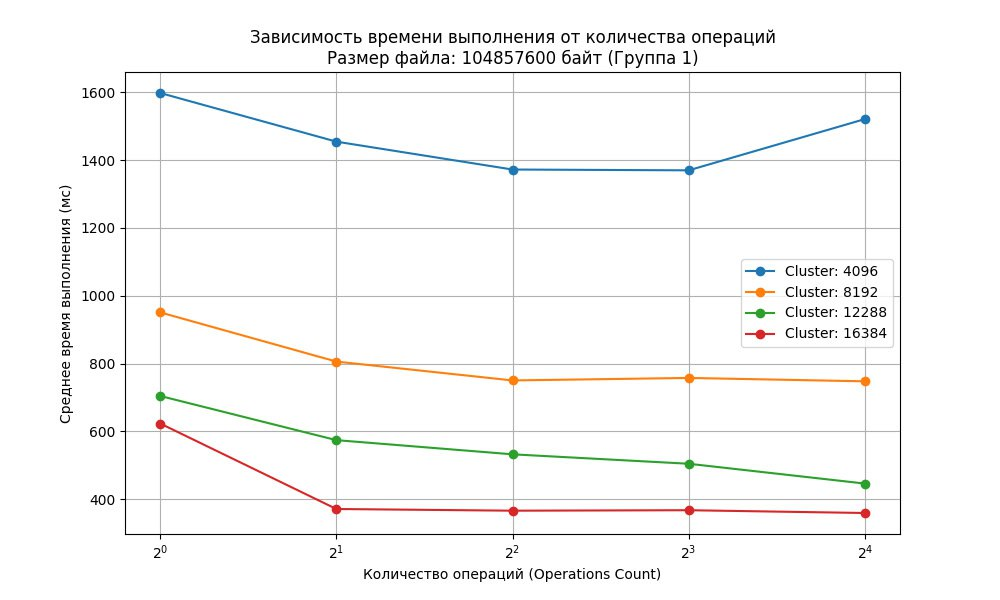
Здесь график кластеров выглядит несколько ломанным – это можно объяснить погрешностью измерений, т к разница колеблется в пределах 20мс, что для файла такого размера несущественно.

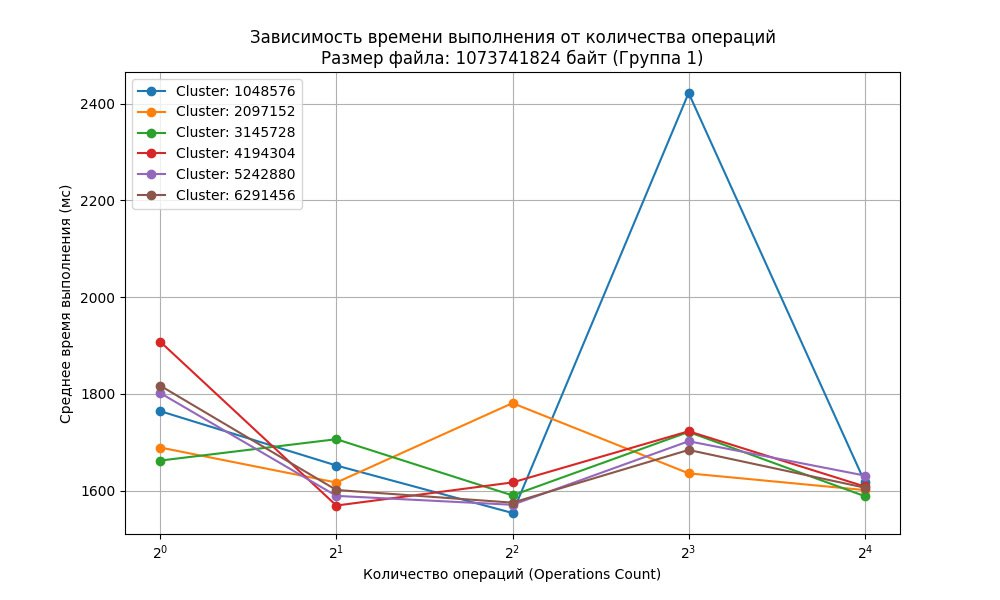
Важно понимать, что для файла размером в 1gb погрешность измерений довольно высока.

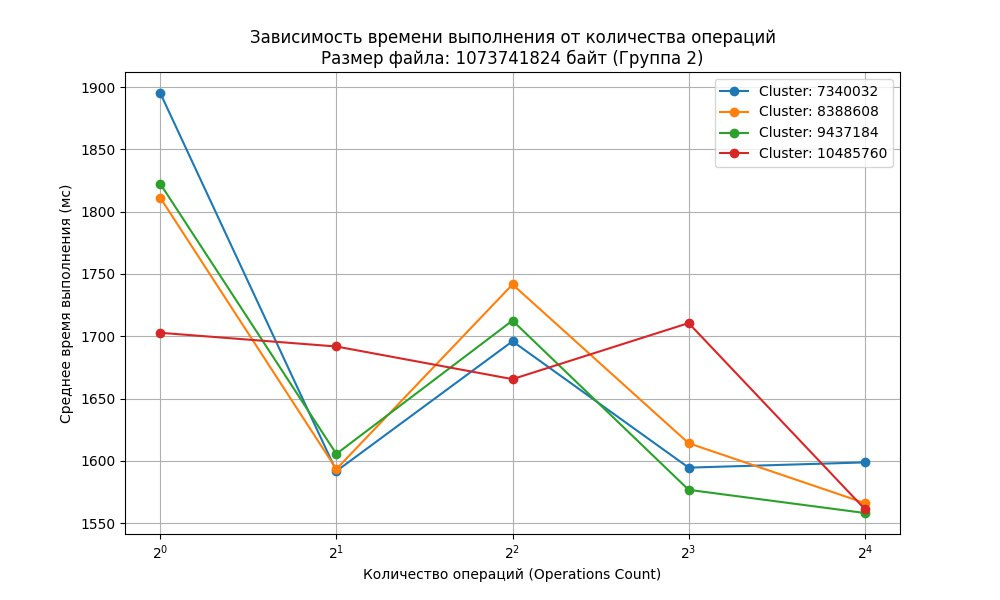
Наибольшую эффективность показали итерации с двумя потоками – это можно объяснить тем, что 2 операции, очевидно, будут давать большую эффективность, чем 1, но, в то же время, не сильно нагружать систему.

Также по полученным логам можно было построить несколько иное отображение результатов:









По графикам можно понять, что наибольшая эффективность работы программы для большинства итераций была при 4 потоках.

**Вывод**

В результате выполнения первого задания лабораторной работы была создана программа, реализующая перечисленные в задании функции из WIN32 API. Программа позволяет посмотреть различную информацию (список дисков, информацию о конкретном диске, информацию о файле), а также предоставляет возможность выполнять действия над файловой системой (редактура атрибутов файла, копирование и создание файла, директории).

В результате выполнения второго задания (для linux) была написана программа, реализующая асинхронное копирование файла путём создания некоторого количества потоков (от 1 до 16), каждый из которых за раз обращается к некоторому объему памяти. Были проведены измерения и составлены графики, а также был выполнен анализ полученных данных. Так, логи программы свидетельствуют о том, что в большинстве случаев нет смысла нагружать SSD диск количеством потоков большим, чем 4, а также о том, что при копировании лучше использовать большой размер чанка (то есть объема памяти на одну операцию чтения/записи): так снизится нагрузка на диск, поскольку создание новых потоков всегда сопряжено с дополнительными расходами, а потому лучше, когда 1 поток читает больший объем данных. Однако при слишком большом количестве потоков эффективность перестает расти, вместо этого лишь возрастают расходы на создание потоков и работу с ними, что снижает производительность.

Ниже представлены исходные коды программ:

1. Код для первого задания:

#include <windows.h>

#include <stdio.h>

#include <wchar.h>

#include <locale.h>

#include <fcntl.h>

#include <io.h>

// Убедимся, что используем UNICODE-версии WinAPI

#ifndef UNICODE

#define UNICODE

#endif

#ifndef \_UNICODE

#define \_UNICODE

#endif

void showLogicalDrives();

void showLogicalDriveStrings();

void showDriveTypes();

void showVolumeInformation();

void showDiskFreeSpace();

void createDirectoryWFunc();

void removeDirectoryWFunc();

void ls();

void createFileWFunc();

void clearStdIn();

void copyFileWFunc();

void moveFileWFunc();

DWORD getFileAttributesWFunc(wchar\_t filePath[MAX\_PATH]);

void setFileAttributesWFunc();

void getFileInformationByHandle();

void printFileTimeW(const FILETIME\* ft);

void getFileTimeWFunc();

void setFileTimeWFunc();

int wmain()

{

    // Устанавливаем локаль (зависит от ОС), чтобы корректно работали wprintf, wscanf и т.п.

    setlocale(LC\_ALL, "");

    // Переключаем потоки ввода/вывода в режим wide-символов (UTF-16)

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

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

    // Опционально меняем кодовые страницы консоли для лучшей совместимости

    SetConsoleCP(CP\_UTF8);

    SetConsoleOutputCP(CP\_UTF8);

    int choice = -1;

    wchar\_t filePath[MAX\_PATH];

    do {

        wprintf(L"=== Меню ===\n");

        wprintf(L"0. Выход\n");

        wprintf(L"1. Показать логические диски (битовая маска)\n");

        wprintf(L"2. Показать логические диски (строки)\n");

        wprintf(L"3. Показать типы дисков\n");

        wprintf(L"4. Информация о томах (GetVolumeInformation)\n");

        wprintf(L"5. Свободное место на диске (GetDiskFreeSpace)\n");

        wprintf(L"6. Создать директорию\n");

        wprintf(L"7. Удалить директорию\n");

        wprintf(L"8. ls\n");

        wprintf(L"9. Создать файл\n");

        wprintf(L"10. Копировать файл\n");

        wprintf(L"11. Переместить файл или директорию\n");

        wprintf(L"12. GetFileAttributes\n");

        wprintf(L"13. SetFileAttributes\n");

        wprintf(L"14. GetFileInformationByHandle\n");

        wprintf(L"15. GetFileTime\n");

        wprintf(L"16. SetFileTime\n");

        wprintf(L"Выберите пункт: ");

        if (wscanf(L"%d", &choice) != 1) {

            wprintf(L"Неверный ввод! Повторите попытку.\n");

            clearStdIn();

            continue;

        }

        clearStdIn();

        wprintf(L"\n");

        switch (choice) {

            case 0:

                break;

            case 1:

                showLogicalDrives();

                break;

            case 2:

                showLogicalDriveStrings();

                break;

            case 3:

                showDriveTypes();

                break;

            case 4:

                showVolumeInformation();

                break;

            case 5:

                showDiskFreeSpace();

                break;

            case 6:

                createDirectoryWFunc();

                break;

            case 7:

                removeDirectoryWFunc();

                break;

            case 8:

                ls();

                break;

            case 9:

                createFileWFunc();

                break;

            case 10:

                copyFileWFunc();

                break;

            case 11:

                moveFileWFunc();

                break;

            case 12:

                getFileAttributesWFunc(filePath);

                break;

            case 13:

                setFileAttributesWFunc();

                break;

            case 14:

                getFileInformationByHandle();

                break;

            case 15:

                getFileTimeWFunc();

                break;

            case 16:

                setFileTimeWFunc();

                break;

            default:

                wprintf(L"Неверный ввод! Повторите попытку.\n");

        }

        if (choice != 0) {

            wprintf(L"\nНажмите Enter для продолжения...");

            clearStdIn();

            getchar();

            \_wsystem(L"cls");

        }

    } while (choice != 0);

    return 0;

}

// -----------------------------------------

void showLogicalDrives()

{

    DWORD drives = GetLogicalDrives();

    if (drives == 0) {

        wprintf(L"Ошибка при получении логических дисков.\n");

        return;

    }

    wprintf(L"Логические диски (битовая маска):\n");

    for (int i = 0; i < 26; i++) {

        if (drives & (1 << i)) {

            wprintf(L"%c:\n", L'A' + i);

        }

    }

}

void showLogicalDriveStrings()

{

    wchar\_t drives[256];

    DWORD length = GetLogicalDriveStringsW(256, drives);

    if (length == 0) {

        wprintf(L"Ошибка при получении строк логических дисков.\n");

        return;

    }

    wprintf(L"Логические диски (строки):\n");

    for (wchar\_t \*drive = drives; \*drive; drive += wcslen(drive) + 1) {

        wprintf(L"%s\n", drive);

    }

}

void showDriveTypes()

{

    wchar\_t drives[256];

    DWORD length = GetLogicalDriveStringsW(256, drives);

    if (length == 0) {

        wprintf(L"Ошибка при получении строк логических дисков.\n");

        return;

    }

    wprintf(L"Типы дисков:\n");

    for (wchar\_t \*drive = drives; \*drive; drive += wcslen(drive) + 1) {

        UINT type = GetDriveTypeW(drive);

        wprintf(L"%s - ", drive);

        switch (type) {

            case DRIVE\_UNKNOWN:

                wprintf(L"Неизвестный\n");

                break;

            case DRIVE\_NO\_ROOT\_DIR:

                wprintf(L"Неверный корневой путь\n");

                break;

            case DRIVE\_REMOVABLE:

                wprintf(L"Съемный (USB, Floppy и т.п.)\n");

                break;

            case DRIVE\_FIXED:

                wprintf(L"Жесткий диск (HDD, SSD)\n");

                break;

            case DRIVE\_REMOTE:

                wprintf(L"Сетевой диск\n");

                break;

            case DRIVE\_CDROM:

                wprintf(L"CD/DVD привод\n");

                break;

            case DRIVE\_RAMDISK:

                wprintf(L"RAM-диск\n");

                break;

            default:

                wprintf(L"Неизвестный тип\n");

        }

    }

}

void showVolumeInformation()

{

    wchar\_t driveLetter;

    wprintf(L"Введите букву диска (напр. C): ");

    if (wscanf(L" %c", &driveLetter) != 1) {

        wprintf(L"Некорректный ввод.\n");

        clearStdIn();

        return;

    }

    clearStdIn();

    wchar\_t drive[4];

    drive[0] = driveLetter;

    drive[1] = L':';

    drive[2] = L'\\';

    drive[3] = L'\0';

    wchar\_t volumeName[MAX\_PATH] = {0};

    DWORD serialNumber = 0;

    DWORD maxComponentLength = 0;

    DWORD fileSystemFlags = 0;

    wchar\_t fileSystemName[MAX\_PATH] = {0};

    if (GetVolumeInformationW(

            drive, volumeName, MAX\_PATH,

            &serialNumber, &maxComponentLength,

            &fileSystemFlags, fileSystemName, MAX\_PATH

        ))

    {

        wprintf(L"\nИнформация о томе для %s\n", drive);

        wprintf(L"Имя тома: %s\n", volumeName[0] ? volumeName : L"—");

        wprintf(L"Серийный номер: %08X\n", serialNumber);

        wprintf(L"Макс. длина имени файла: %lu\n", maxComponentLength);

        wprintf(L"Файловая система: %s\n", fileSystemName);

        wprintf(L"Флаги: %lu\n", fileSystemFlags);

        wprintf(L"Расшифровка флагов:\n");

        if (fileSystemFlags & FILE\_CASE\_SENSITIVE\_SEARCH)

            wprintf(L"  - Чувствительность к регистру\n");

        if (fileSystemFlags & FILE\_CASE\_PRESERVED\_NAMES)

            wprintf(L"  - Сохранение регистра в именах\n");

        if (fileSystemFlags & FILE\_UNICODE\_ON\_DISK)

            wprintf(L"  - Поддержка Unicode в именах\n");

        if (fileSystemFlags & FILE\_PERSISTENT\_ACLS)

            wprintf(L"  - Постоянные ACL\n");

        if (fileSystemFlags & FILE\_FILE\_COMPRESSION)

            wprintf(L"  - Сжатие файлов\n");

        if (fileSystemFlags & FILE\_VOLUME\_QUOTAS)

            wprintf(L"  - Квоты диска\n");

        if (fileSystemFlags & FILE\_SUPPORTS\_SPARSE\_FILES)

            wprintf(L"  - Разреженные файлы (Sparse)\n");

        if (fileSystemFlags & FILE\_SUPPORTS\_REPARSE\_POINTS)

            wprintf(L"  - Символьные ссылки, Reparse Points\n");

        if (fileSystemFlags & FILE\_SUPPORTS\_REMOTE\_STORAGE)

            wprintf(L"  - Удаленное хранение\n");

        if (fileSystemFlags & FILE\_VOLUME\_IS\_COMPRESSED)

            wprintf(L"  - Том сжат\n");

        if (fileSystemFlags & FILE\_SUPPORTS\_OBJECT\_IDS)

            wprintf(L"  - Идентификаторы объектов\n");

        if (fileSystemFlags & FILE\_SUPPORTS\_ENCRYPTION)

            wprintf(L"  - Шифрование (EFS)\n");

        if (fileSystemFlags & FILE\_NAMED\_STREAMS)

            wprintf(L"  - Альтернативные потоки данных\n");

        if (fileSystemFlags & FILE\_READ\_ONLY\_VOLUME)

            wprintf(L"  - Том только для чтения\n");

        if (fileSystemFlags & FILE\_SEQUENTIAL\_WRITE\_ONCE)

            wprintf(L"  - Однократная последовательная запись\n");

        if (fileSystemFlags & FILE\_SUPPORTS\_TRANSACTIONS)

            wprintf(L"  - Транзакционная ФС\n");

        if (fileSystemFlags & FILE\_SUPPORTS\_HARD\_LINKS)

            wprintf(L"  - Жесткие ссылки\n");

        if (fileSystemFlags & FILE\_SUPPORTS\_EXTENDED\_ATTRIBUTES)

            wprintf(L"  - Расширенные атрибуты\n");

        if (fileSystemFlags & FILE\_SUPPORTS\_OPEN\_BY\_FILE\_ID)

            wprintf(L"  - Открытие по FileID\n");

        if (fileSystemFlags & FILE\_SUPPORTS\_USN\_JOURNAL)

            wprintf(L"  - Журнал USN\n");

    } else {

        wprintf(L"Не удалось получить информацию о томе. Код ошибки: %lu\n", GetLastError());

    }

}

void showDiskFreeSpace()

{

    wchar\_t driveLetter;

    wprintf(L"Введите букву диска (напр. C): ");

    if (wscanf(L" %c", &driveLetter) != 1) {

        wprintf(L"Некорректный ввод.\n");

        clearStdIn();

        return;

    }

    clearStdIn();

    wchar\_t drive[4];

    drive[0] = driveLetter;

    drive[1] = L':';

    drive[2] = L'\\';

    drive[3] = L'\0';

    DWORD sectorsPerCluster, bytesPerSector, freeClusters, totalClusters;

    if (GetDiskFreeSpaceW(drive, &sectorsPerCluster, &bytesPerSector, &freeClusters, &totalClusters)) {

        unsigned long long clusterSize = (unsigned long long)sectorsPerCluster \* bytesPerSector;

        unsigned long long freeSpace   = (unsigned long long)freeClusters     \* clusterSize;

        unsigned long long totalSpace  = (unsigned long long)totalClusters    \* clusterSize;

        wprintf(L"\nСводка по %s\n", drive);

        wprintf(L"Секторов в кластере: %lu\n", sectorsPerCluster);

        wprintf(L"Байт в секторе: %lu\n", bytesPerSector);

        wprintf(L"Всего кластеров: %lu\n", totalClusters);

        wprintf(L"Свободных кластеров: %lu\n", freeClusters);

        wprintf(L"Размер кластера: %llu байт\n", clusterSize);

        wprintf(L"Объем диска: %.2f ГБ\n", (double)totalSpace / (1024.0 \* 1024.0 \* 1024.0));

        wprintf(L"Свободно: %.2f ГБ\n", (double)freeSpace / (1024.0 \* 1024.0 \* 1024.0));

    } else {

        wprintf(L"Не удалось получить информацию о свободном пространстве. Код ошибки: %lu\n", GetLastError());

    }

}

void createDirectoryWFunc()

{

    while (1) {

        wchar\_t dirName[MAX\_PATH];

        wprintf(L"\nВведите имя директории (напр. C:\\НоваяПапка): ");

        if (!fgetws(dirName, MAX\_PATH, stdin)) {

            wprintf(L"Ошибка ввода.\n");

            return;

        }

        // Удалим перевод строки, если он есть

        size\_t len = wcslen(dirName);

        if (len > 0 && dirName[len - 1] == L'\n') {

            dirName[len - 1] = L'\0';

        }

        if (CreateDirectoryW(dirName, NULL)) {

            wprintf(L"Директория успешно создана: %s\n", dirName);

            return;

        } else {

            DWORD error = GetLastError();

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

                wprintf(L"Ошибка: Директория уже существует.\n");

                wprintf(L"1. Оставить как есть\n");

                wprintf(L"2. Ввести новый путь\n");

                wprintf(L"3. Удалить существующую и создать новую\n");

                wprintf(L"Выберите опцию: ");

                int choice;

                if (wscanf(L"%d", &choice) != 1) {

                    clearStdIn();

                    return;

                }

                clearStdIn();

                if (choice == 1) {

                    return;

                } else if (choice == 2) {

                    continue;

                } else if (choice == 3) {

                    if (RemoveDirectoryW(dirName)) {

                        wprintf(L"Старая директория удалена. Создаем новую...\n");

                        if (CreateDirectoryW(dirName, NULL)) {

                            wprintf(L"Директория успешно создана: %s\n", dirName);

                        } else {

                            wprintf(L"Не удалось создать директорию. Код ошибки: %lu\n", GetLastError());

                        }

                    } else {

                        wprintf(L"Не удалось удалить старую директорию. Код ошибки: %lu\n", GetLastError());

                    }

                    return;

                }

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

                wprintf(L"Ошибка: Родительская директория не существует.\n");

                wprintf(L"1. Создать недостающие родительские директории\n");

                wprintf(L"2. Ввести новый путь\n");

                wprintf(L"3. Отмена\n");

                wprintf(L"Выберите опцию: ");

                int choice;

                if (wscanf(L"%d", &choice) != 1) {

                    clearStdIn();

                    return;

                }

                clearStdIn();

                if (choice == 1) {

                    // Пытаемся создать родительскую директорию

                    wchar\_t parentDir[MAX\_PATH];

                    wcscpy\_s(parentDir, MAX\_PATH, dirName);

                    // Ищем последний слеш, чтобы отсечь имя конечной папки

                    for (int i = (int)wcslen(parentDir) - 1; i >= 0; i--) {

                        if (parentDir[i] == L'\\') {

                            parentDir[i] = L'\0';

                            break;

                        }

                    }

                    if (CreateDirectoryW(parentDir, NULL)) {

                        wprintf(L"Родительская директория создана: %s\n", parentDir);

                        // И снова пробуем создать конечную

                        continue;

                    } else {

                        wprintf(L"Не удалось создать родительскую директорию. Код ошибки: %lu\n", GetLastError());

                    }

                } else if (choice == 2) {

                    continue;

                } else {

                    return;

                }

            } else {

                wprintf(L"Не удалось создать директорию. Код ошибки: %lu\n", error);

                return;

            }

        }

    }

}

void removeDirectoryWFunc()

{

    while (1) {

        wchar\_t dirName[MAX\_PATH];

        wprintf(L"\nВведите имя директории для удаления (напр. C:\\Тест): ");

        if (!fgetws(dirName, MAX\_PATH, stdin)) {

            wprintf(L"Ошибка ввода.\n");

            return;

        }

        // Удалим перевод строки

        size\_t len = wcslen(dirName);

        if (len > 0 && dirName[len - 1] == L'\n') {

            dirName[len - 1] = L'\0';

        }

        if (RemoveDirectoryW(dirName)) {

            wprintf(L"Директория удалена: %s\n", dirName);

            return;

        } else {

            DWORD error = GetLastError();

            if (error == ERROR\_FILE\_NOT\_FOUND || error == ERROR\_PATH\_NOT\_FOUND) {

                wprintf(L"Ошибка: Директория не существует.\n");

                wprintf(L"1. Ввести новый путь\n");

                wprintf(L"2. Отмена\n");

                wprintf(L"Выберите опцию: ");

                int choice;

                if (wscanf(L"%d", &choice) != 1) {

                    clearStdIn();

                    return;

                }

                clearStdIn();

                if (choice == 1) {

                    continue;

                } else {

                    return;

                }

            } else if (error == ERROR\_DIR\_NOT\_EMPTY) {

                wprintf(L"Ошибка: Директория не пуста.\n");

                wprintf(L"1. Удалить все содержимое и директорию\n");

                wprintf(L"2. Ввести новый путь\n");

                wprintf(L"3. Отмена\n");

                wprintf(L"Выберите опцию: ");

                int choice;

                if (wscanf(L"%d", &choice) != 1) {

                    clearStdIn();

                    return;

                }

                clearStdIn();

                if (choice == 1) {

                    // Удаляем рекурсивно

                    wchar\_t cmd[MAX\_PATH + 50];

                    // Используем rmdir /s /q

                    swprintf(cmd, MAX\_PATH + 50, L"rmdir /s /q \"%s\"", dirName);

                    \_wsystem(cmd);

                    if (RemoveDirectoryW(dirName)) {

                        wprintf(L"Директория удалена: %s\n", dirName);

                    } else {

                        wprintf(L"Не удалось удалить директорию. Код ошибки: %lu\n", GetLastError());

                    }

                    return;

                } else if (choice == 2) {

                    continue;

                } else {

                    return;

                }

            } else if (error == ERROR\_ACCESS\_DENIED) {

                wprintf(L"Ошибка: Отказано в доступе.\n");

                wprintf(L"1. Повторить с правами администратора\n");

                wprintf(L"2. Ввести новый путь\n");

                wprintf(L"3. Отмена\n");

                wprintf(L"Выберите опцию: ");

                int choice;

                if (wscanf(L"%d", &choice) != 1) {

                    clearStdIn();

                    return;

                }

                clearStdIn();

                if (choice == 1) {

                    wprintf(L"Перезапустите программу от имени администратора.\n");

                    return;

                } else if (choice == 2) {

                    continue;

                } else {

                    return;

                }

            } else {

                wprintf(L"Не удалось удалить директорию. Код ошибки: %lu\n", error);

                return;

            }

        }

    }

}

void ls()

{

    wchar\_t dirPath[MAX\_PATH];

    wprintf(L"Введите путь к директории (напр. C:\\Тест): ");

    if (!fgetws(dirPath, MAX\_PATH, stdin)) {

        wprintf(L"Ошибка ввода.\n");

        return;

    }

    // Удалим перевод строки

    size\_t len = wcslen(dirPath);

    if (len > 0 && dirPath[len - 1] == L'\n') {

        dirPath[len - 1] = L'\0';

    }

    wchar\_t searchPath[MAX\_PATH];

    swprintf(searchPath, MAX\_PATH, L"%s\\\*", dirPath);

    WIN32\_FIND\_DATAW findData;

    HANDLE hFind = FindFirstFileW(searchPath, &findData);

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

        wprintf(L"Ошибка: Невозможно открыть директорию или она не существует.\n");

        return;

    }

    wprintf(L"\nСодержимое %s:\n", dirPath);

    do {

        if (wcscmp(findData.cFileName, L".") != 0 && wcscmp(findData.cFileName, L"..") != 0) {

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

                wprintf(L"  [DIR]  %s\n", findData.cFileName);

            } else {

                wprintf(L"  [FILE] %s\n", findData.cFileName);

            }

        }

    } while (FindNextFileW(hFind, &findData));

    FindClose(hFind);

}

void createFileWFunc()

{

    while (1) {

        wchar\_t filePath[MAX\_PATH];

        wprintf(L"\nВведите путь к файлу (напр. C:\\Test\\Файл.txt): ");

        if (!fgetws(filePath, MAX\_PATH, stdin)) {

            wprintf(L"Ошибка ввода.\n");

            return;

        }

        size\_t len = wcslen(filePath);

        if (len > 0 && filePath[len - 1] == L'\n') {

            filePath[len - 1] = L'\0';

        }

        HANDLE hFile = CreateFileW(

            filePath, GENERIC\_WRITE, 0, NULL,

            CREATE\_NEW, FILE\_ATTRIBUTE\_NORMAL, NULL

        );

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

            DWORD error = GetLastError();

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

                wprintf(L"Ошибка: Файл уже существует.\n");

                wprintf(L"1. Перезаписать файл\n");

                wprintf(L"2. Ввести новый путь\n");

                wprintf(L"3. Отмена\n");

                int choice;

                if (wscanf(L"%d", &choice) != 1) {

                    clearStdIn();

                    return;

                }

                clearStdIn();

                if (choice == 1) {

                    hFile = CreateFileW(

                        filePath, GENERIC\_WRITE, 0, NULL,

                        CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL

                    );

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

                        wprintf(L"Не удалось перезаписать файл. Код ошибки: %lu\n", GetLastError());

                    } else {

                        wprintf(L"Файл перезаписан: %s\n", filePath);

                        CloseHandle(hFile);

                    }

                    return;

                } else if (choice == 2) {

                    continue;

                } else {

                    return;

                }

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

                wprintf(L"Ошибка: Неверный путь или директория не существует.\n");

                wprintf(L"1. Создать недостающую директорию\n");

                wprintf(L"2. Ввести новый путь\n");

                wprintf(L"3. Отмена\n");

                int choice;

                if (wscanf(L"%d", &choice) != 1) {

                    clearStdIn();

                    return;

                }

                clearStdIn();

                if (choice == 1) {

                    // Отсекаем имя файла, пытаемся создать каталог

                    wchar\_t dirPath[MAX\_PATH];

                    wcscpy\_s(dirPath, MAX\_PATH, filePath);

                    for (int i = (int)wcslen(dirPath) - 1; i >= 0; i--) {

                        if (dirPath[i] == L'\\') {

                            dirPath[i] = L'\0';

                            break;

                        }

                    }

                    if (CreateDirectoryW(dirPath, NULL)) {

                        wprintf(L"Директория создана: %s\n", dirPath);

                    } else {

                        wprintf(L"Не удалось создать директорию. Код ошибки: %lu\n", GetLastError());

                    }

                    continue;

                } else if (choice == 2) {

                    continue;

                } else {

                    return;

                }

            } else {

                wprintf(L"Не удалось создать файл. Код ошибки: %lu\n", error);

                return;

            }

        } else {

            wprintf(L"Файл успешно создан: %s\n", filePath);

            CloseHandle(hFile);

            return;

        }

    }

}

void clearStdIn()

{

    // Считываем символы до перевода строки

    int c;

    while ((c = getwchar()) != L'\n' && c != WEOF) {}

}

void copyFileWFunc()

{

    while (1) {

        wchar\_t sourcePath[MAX\_PATH];

        wchar\_t destinationPath[MAX\_PATH];

        wprintf(L"\nВведите исходный файл (напр. C:\\source.txt): ");

        if (!fgetws(sourcePath, MAX\_PATH, stdin)) return;

        size\_t len = wcslen(sourcePath);

        if (len > 0 && sourcePath[len - 1] == L'\n') {

            sourcePath[len - 1] = L'\0';

        }

        wprintf(L"Введите путь для копии (напр. C:\\backup.txt): ");

        if (!fgetws(destinationPath, MAX\_PATH, stdin)) return;

        len = wcslen(destinationPath);

        if (len > 0 && destinationPath[len - 1] == L'\n') {

            destinationPath[len - 1] = L'\0';

        }

        if (CopyFileW(sourcePath, destinationPath, FALSE)) {

            wprintf(L"Файл скопирован: %s\n", destinationPath);

            return;

        } else {

            DWORD error = GetLastError();

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

                wprintf(L"Ошибка: Исходный файл не существует.\n");

                wprintf(L"1. Ввести новый исходный файл\n");

                wprintf(L"2. Отмена\n");

                int choice;

                if (wscanf(L"%d", &choice) != 1) {

                    clearStdIn();

                    return;

                }

                clearStdIn();

                if (choice == 1) continue;

                else return;

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

                wprintf(L"Ошибка: Неверный путь назначения.\n");

                wprintf(L"1. Ввести новый путь\n");

                wprintf(L"2. Отмена\n");

                int choice;

                if (wscanf(L"%d", &choice) != 1) {

                    clearStdIn();

                    return;

                }

                clearStdIn();

                if (choice == 1) continue;

                else return;

            } else if (error == ERROR\_ACCESS\_DENIED) {

                wprintf(L"Ошибка: Доступ запрещен.\n");

                wprintf(L"1. Повторить с другим путем\n");

                wprintf(L"2. Отмена\n");

                int choice;

                if (wscanf(L"%d", &choice) != 1) {

                    clearStdIn();

                    return;

                }

                clearStdIn();

                if (choice == 1) continue;

                else return;

            } else if (error == ERROR\_ALREADY\_EXISTS) {

                wprintf(L"Ошибка: Файл назначения уже существует.\n");

                wprintf(L"1. Перезаписать\n");

                wprintf(L"2. Ввести новый путь\n");

                wprintf(L"3. Отмена\n");

                int choice;

                if (wscanf(L"%d", &choice) != 1) {

                    clearStdIn();

                    return;

                }

                clearStdIn();

                if (choice == 1) {

                    // Попытка перезаписать

                    if (CopyFileW(sourcePath, destinationPath, TRUE)) {

                        wprintf(L"Файл перезаписан: %s\n", destinationPath);

                    } else {

                        wprintf(L"Не удалось перезаписать файл. Код ошибки: %lu\n", GetLastError());

                    }

                    return;

                } else if (choice == 2) {

                    continue;

                } else {

                    return;

                }

            } else {

                wprintf(L"Не удалось скопировать файл. Код ошибки: %lu\n", error);

                return;

            }

        }

    }

}

void moveFileWFunc()

{

    while (1) {

        wchar\_t sourcePath[MAX\_PATH];

        wchar\_t destinationPath[MAX\_PATH];

        wprintf(L"\nИсходный файл/папка (напр. C:\\source.txt): ");

        if (!fgetws(sourcePath, MAX\_PATH, stdin)) return;

        size\_t len = wcslen(sourcePath);

        if (len > 0 && sourcePath[len - 1] == L'\n') {

            sourcePath[len - 1] = L'\0';

        }

        wprintf(L"Путь назначения (напр. C:\\destination.txt): ");

        if (!fgetws(destinationPath, MAX\_PATH, stdin)) return;

        len = wcslen(destinationPath);

        if (len > 0 && destinationPath[len - 1] == L'\n') {

            destinationPath[len - 1] = L'\0';

        }

        if (MoveFileW(sourcePath, destinationPath)) {

            wprintf(L"Успешно перемещено в %s\n", destinationPath);

            return;

        } else {

            DWORD error = GetLastError();

            if (error == ERROR\_FILE\_NOT\_FOUND || error == ERROR\_PATH\_NOT\_FOUND) {

                wprintf(L"Ошибка: Исходный файл/папка не существует.\n");

                wprintf(L"1. Ввести заново\n");

                wprintf(L"2. Отмена\n");

                int choice;

                if (wscanf(L"%d", &choice) != 1) {

                    clearStdIn();

                    return;

                }

                clearStdIn();

                if (choice == 1) continue;

                else return;

            } else if (error == ERROR\_ALREADY\_EXISTS) {

                wprintf(L"Ошибка: Уже есть элемент с таким именем в папке назначения.\n");

                wprintf(L"1. Заменить\n");

                wprintf(L"2. Ввести новый путь\n");

                wprintf(L"3. Отмена\n");

                int choice;

                if (wscanf(L"%d", &choice) != 1) {

                    clearStdIn();

                    return;

                }

                clearStdIn();

                if (choice == 1) {

                    if (MoveFileExW(sourcePath, destinationPath, MOVEFILE\_REPLACE\_EXISTING)) {

                        wprintf(L"Успешно перемещено (заменено): %s\n", destinationPath);

                    } else {

                        wprintf(L"Не удалось заменить. Код ошибки: %lu\n", GetLastError());

                    }

                    return;

                } else if (choice == 2) {

                    continue;

                } else {

                    return;

                }

            } else if (error == ERROR\_ACCESS\_DENIED) {

                wprintf(L"Ошибка: Нет доступа.\n");

                wprintf(L"1. Повторить с другим путем\n");

                wprintf(L"2. Отмена\n");

                int choice;

                if (wscanf(L"%d", &choice) != 1) {

                    clearStdIn();

                    return;

                }

                clearStdIn();

                if (choice == 1) continue;

                else return;

            } else {

                wprintf(L"Не удалось переместить. Код ошибки: %lu\n", error);

                return;

            }

        }

    }

}

DWORD getFileAttributesWFunc(wchar\_t filePath[MAX\_PATH])

{

    while (1) {

        wprintf(L"\nВведите путь к файлу или директории: ");

        if (!fgetws(filePath, MAX\_PATH, stdin)) {

            wprintf(L"Ошибка ввода.\n");

            return 0;

        }

        size\_t len = wcslen(filePath);

        if (len > 0 && filePath[len - 1] == L'\n') {

            filePath[len - 1] = L'\0';

        }

        DWORD attributes = GetFileAttributesW(filePath);

        if (attributes == INVALID\_FILE\_ATTRIBUTES) {

            DWORD error = GetLastError();

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

                wprintf(L"Ошибка: Файл не существует.\n");

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

                wprintf(L"Ошибка: Неверный путь.\n");

            } else if (error == ERROR\_ACCESS\_DENIED) {

                wprintf(L"Ошибка: Нет доступа.\n");

            } else {

                wprintf(L"Не удалось получить атрибуты. Код: %lu\n", error);

            }

            wprintf(L"1. Ввести путь заново\n");

            wprintf(L"2. Отмена\n");

            int choice;

            if (wscanf(L"%d", &choice) != 1) {

                clearStdIn();

                return 0;

            }

            clearStdIn();

            if (choice == 1) {

                continue;

            } else {

                return 0;

            }

        }

        wprintf(L"\nАтрибуты %s:\n", filePath);

        if (attributes & FILE\_ATTRIBUTE\_DIRECTORY) wprintf(L"  - Директория\n");

        if (attributes & FILE\_ATTRIBUTE\_READONLY)  wprintf(L"  - Только чтение\n");

        if (attributes & FILE\_ATTRIBUTE\_HIDDEN)    wprintf(L"  - Скрытый\n");

        if (attributes & FILE\_ATTRIBUTE\_SYSTEM)    wprintf(L"  - Системный\n");

        if (attributes & FILE\_ATTRIBUTE\_ARCHIVE)   wprintf(L"  - Архив\n");

        if (attributes & FILE\_ATTRIBUTE\_ENCRYPTED) wprintf(L"  - Зашифрован\n");

        if (attributes & FILE\_ATTRIBUTE\_COMPRESSED)wprintf(L"  - Сжат\n");

        if (attributes & FILE\_ATTRIBUTE\_TEMPORARY) wprintf(L"  - Временный\n");

        if (attributes & FILE\_ATTRIBUTE\_OFFLINE)   wprintf(L"  - Оффлайн\n");

        return attributes;

    }

}

void setFileAttributesWFunc()

{

    wchar\_t filePath[MAX\_PATH];

    while (1) {

        DWORD attributes = getFileAttributesWFunc(filePath);

        if (attributes == 0) return; // Пользователь прервал

        wprintf(L"\nВыберите действие:\n");

        wprintf(L"1. Добавить атрибут\n");

        wprintf(L"2. Удалить атрибут\n");

        wprintf(L"3. Отмена\n");

        wprintf(L"Ваш выбор: ");

        int action;

        if (wscanf(L"%d", &action) != 1) {

            clearStdIn();

            return;

        }

        clearStdIn();

        if (action == 3) return;

        wprintf(L"\nВыберите атрибут:\n");

        wprintf(L"1. Read-only\n");

        wprintf(L"2. Hidden\n");

        wprintf(L"3. System\n");

        wprintf(L"4. Archive\n");

        wprintf(L"Ваш выбор: ");

        int attrChoice;

        if (wscanf(L"%d", &attrChoice) != 1) {

            clearStdIn();

            return;

        }

        clearStdIn();

        DWORD attrFlag = 0;

        if      (attrChoice == 1) attrFlag = FILE\_ATTRIBUTE\_READONLY;

        else if (attrChoice == 2) attrFlag = FILE\_ATTRIBUTE\_HIDDEN;

        else if (attrChoice == 3) attrFlag = FILE\_ATTRIBUTE\_SYSTEM;

        else if (attrChoice == 4) attrFlag = FILE\_ATTRIBUTE\_ARCHIVE;

        else {

            wprintf(L"Неверный выбор.\n");

            return;

        }

        if (action == 1) {

            attributes |= attrFlag;   // Добавляем флаг

        } else if (action == 2) {

            attributes &= ~attrFlag;  // Снимаем флаг

        }

        if (SetFileAttributesW(filePath, attributes)) {

            wprintf(L"Атрибуты успешно обновлены.\n");

        } else {

            wprintf(L"Не удалось обновить атрибуты. Код ошибки: %lu\n", GetLastError());

        }

        return;

    }

}

void printFileTimeW(const FILETIME\* ft)

{

    SYSTEMTIME st;

    FileTimeToSystemTime(ft, &st);

    wprintf(L"%04d-%02d-%02d %02d:%02d:%02d\n",

        st.wYear, st.wMonth, st.wDay,

        st.wHour, st.wMinute, st.wSecond

    );

}

void getFileInformationByHandle()

{

    while (1) {

        wchar\_t filePath[MAX\_PATH];

        wprintf(L"\nВведите путь к файлу (напр. C:\\file.txt): ");

        if (!fgetws(filePath, MAX\_PATH, stdin)) {

            wprintf(L"Ошибка ввода.\n");

            return;

        }

        size\_t len = wcslen(filePath);

        if (len > 0 && filePath[len - 1] == L'\n') {

            filePath[len - 1] = L'\0';

        }

        HANDLE hFile = CreateFileW(

            filePath, GENERIC\_READ, FILE\_SHARE\_READ, NULL,

            OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL

        );

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

            DWORD error = GetLastError();

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

                wprintf(L"Ошибка: Файл не существует.\n");

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

                wprintf(L"Ошибка: Неверный путь.\n");

            } else if (error == ERROR\_ACCESS\_DENIED) {

                wprintf(L"Ошибка: Нет доступа.\n");

            } else {

                wprintf(L"Ошибка при открытии файла. Код: %lu\n", error);

            }

            wprintf(L"1. Ввести новый путь\n");

            wprintf(L"2. Отмена\n");

            int choice;

            if (wscanf(L"%d", &choice) != 1) {

                clearStdIn();

                return;

            }

            clearStdIn();

            if (choice == 1) {

                continue;

            } else {

                return;

            }

        }

        BY\_HANDLE\_FILE\_INFORMATION fileInfo;

        if (GetFileInformationByHandle(hFile, &fileInfo)) {

            wprintf(L"\nИнформация о файле %s:\n", filePath);

            wprintf(L"Серийный номер тома: %lu\n", fileInfo.dwVolumeSerialNumber);

            wprintf(L"Индекс файла: %lu-%lu\n", fileInfo.nFileIndexHigh, fileInfo.nFileIndexLow);

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

            wprintf(L"Размер: %llu байт\n", size);

            wprintf(L"Создан: ");       printFileTimeW(&fileInfo.ftCreationTime);

            wprintf(L"Последний доступ: ");   printFileTimeW(&fileInfo.ftLastAccessTime);

            wprintf(L"Последняя запись: ");   printFileTimeW(&fileInfo.ftLastWriteTime);

            wprintf(L"Атрибуты:\n");

            if (fileInfo.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY) wprintf(L"  - Директория\n");

            if (fileInfo.dwFileAttributes & FILE\_ATTRIBUTE\_READONLY)  wprintf(L"  - Только чтение\n");

            if (fileInfo.dwFileAttributes & FILE\_ATTRIBUTE\_HIDDEN)    wprintf(L"  - Скрытый\n");

            if (fileInfo.dwFileAttributes & FILE\_ATTRIBUTE\_SYSTEM)    wprintf(L"  - Системный\n");

            if (fileInfo.dwFileAttributes & FILE\_ATTRIBUTE\_ARCHIVE)   wprintf(L"  - Архив\n");

            if (fileInfo.dwFileAttributes & FILE\_ATTRIBUTE\_COMPRESSED)wprintf(L"  - Сжат\n");

            if (fileInfo.dwFileAttributes & FILE\_ATTRIBUTE\_ENCRYPTED) wprintf(L"  - Зашифрован\n");

        } else {

            wprintf(L"Не удалось получить информацию о файле. Код ошибки: %lu\n", GetLastError());

        }

        CloseHandle(hFile);

        return;

    }

}

void getFileTimeWFunc()

{

    while (1) {

        wchar\_t filePath[MAX\_PATH];

        wprintf(L"\nВведите путь к файлу: ");

        if (!fgetws(filePath, MAX\_PATH, stdin)) {

            wprintf(L"Ошибка ввода.\n");

            return;

        }

        size\_t len = wcslen(filePath);

        if (len > 0 && filePath[len - 1] == L'\n') {

            filePath[len - 1] = L'\0';

        }

        HANDLE hFile = CreateFileW(

            filePath, GENERIC\_READ, FILE\_SHARE\_READ, NULL,

            OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL

        );

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

            DWORD error = GetLastError();

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

                wprintf(L"Ошибка: Файл не существует.\n");

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

                wprintf(L"Ошибка: Неверный путь.\n");

            } else if (error == ERROR\_ACCESS\_DENIED) {

                wprintf(L"Ошибка: Нет доступа.\n");

            } else {

                wprintf(L"Ошибка при открытии файла. Код: %lu\n", error);

            }

            wprintf(L"1. Ввести новый путь\n");

            wprintf(L"2. Отмена\n");

            int choice;

            if (wscanf(L"%d", &choice) != 1) {

                clearStdIn();

                return;

            }

            clearStdIn();

            if (choice == 1) {

                continue;

            } else {

                return;

            }

        }

        FILETIME creationTime, accessTime, writeTime;

        if (GetFileTime(hFile, &creationTime, &accessTime, &writeTime)) {

            wprintf(L"\nВременные метки для %s:\n", filePath);

            wprintf(L"Создан:          "); printFileTimeW(&creationTime);

            wprintf(L"Последний доступ: "); printFileTimeW(&accessTime);

            wprintf(L"Последняя запись: "); printFileTimeW(&writeTime);

        } else {

            wprintf(L"Не удалось получить время файла. Код: %lu\n", GetLastError());

        }

        CloseHandle(hFile);

        return;

    }

}

void setFileTimeWFunc()

{

    while (1) {

        wchar\_t filePath[MAX\_PATH];

        wprintf(L"\nВведите путь к файлу: ");

        if (!fgetws(filePath, MAX\_PATH, stdin)) {

            wprintf(L"Ошибка ввода.\n");

            return;

        }

        size\_t len = wcslen(filePath);

        if (len > 0 && filePath[len - 1] == L'\n') {

            filePath[len - 1] = L'\0';

        }

        HANDLE hFile = CreateFileW(

            filePath, GENERIC\_WRITE, FILE\_SHARE\_READ, NULL,

            OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL

        );

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

            DWORD error = GetLastError();

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

                wprintf(L"Ошибка: Файл не существует.\n");

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

                wprintf(L"Ошибка: Неверный путь.\n");

            } else if (error == ERROR\_ACCESS\_DENIED) {

                wprintf(L"Ошибка: Нет доступа.\n");

            } else {

                wprintf(L"Ошибка при открытии файла. Код: %lu\n", error);

            }

            wprintf(L"1. Ввести новый путь\n");

            wprintf(L"2. Отмена\n");

            int choice;

            if (wscanf(L"%d", &choice) != 1) {

                clearStdIn();

                return;

            }

            clearStdIn();

            if (choice == 1) {

                continue;

            } else {

                return;

            }

        }

        wprintf(L"\nВведите новые дату и время (ГГГГ ММ ДД чч мм cc): ");

        SYSTEMTIME st;

        if (wscanf(L"%hu %hu %hu %hu %hu %hu",

            &st.wYear, &st.wMonth, &st.wDay,

            &st.wHour, &st.wMinute, &st.wSecond) != 6)

        {

            wprintf(L"Неверный формат даты.\n");

            clearStdIn();

            CloseHandle(hFile);

            return;

        }

        clearStdIn();

        FILETIME newTime;

        if (!SystemTimeToFileTime(&st, &newTime)) {

            wprintf(L"Неверная дата.\n");

            CloseHandle(hFile);

            return;

        }

        if (SetFileTime(hFile, &newTime, &newTime, &newTime)) {

            wprintf(L"Время файла успешно обновлено.\n");

        } else {

            wprintf(L"Ошибка при обновлении времени. Код: %lu\n", GetLastError());

        }

        CloseHandle(hFile);

        return;

    }

}

Код для второго задания:

#include <iostream>

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <aio.h>

#include <fcntl.h>

#include <unistd.h>

#include <signal.h>

#include <string.h>

#include <errno.h>

#include <chrono>

#include <sys/types.h>

#include <sys/stat.h>

#include <stdint.h>

#include <inttypes.h>

#include <mutex>

#include <condition\_variable>

#include <fstream>      // Для работы с файлом-логом

#include <iomanip>      // Для std::put\_time или strftime

using namespace std;

using namespace std::chrono;

#define kb4 4096

#define kb16 16384

#define mb 1048576

#define mb10 10485760

// Глобальные переменные

int chunk;              // Текущий размер «кластера» для чтения/записи

int fileReadHandler;

int fileWriteHandler;

int operationsCount;    // Текущее количество одновременных aio-операций

int fileSize;

std::mutex gMutex;

std::condition\_variable gCv;

struct AioOperation {

    struct aiocb\* aio;       // Указатель на структуру асинхронной операции ввода/вывода

    char \*buffer;            // Буфер данных

    int  isReading;          // Флаг: 1 - операция чтения, 0 - операция записи

    int  isWorking;          // Флаг: 1 - операция ещё активна, 0 - завершена

};

// Прототипы функций

void freeMemory(struct AioOperation\* operations, int n);

void aioCompletionHandler(sigval\_t sigval);

void initOperations(struct AioOperation\* operations, int cluster);

void startOperations(struct AioOperation\* operations);

void await(struct AioOperation\* operations, int count);

int main() {

    struct stat fileStat;

    struct AioOperation\* operations;

    string pathRead = "1g\0", pathWrite = "1gc\0";

    // cout << "Введите путь к копируемому файлу: ";

    // cin >> pathRead;

    // cout << "Введите путь к файлу записи: ";

    // cin >> pathWrite;

    int tempFileHandler = open(pathRead.c\_str(), O\_RDONLY);

    if (tempFileHandler == -1) {

        cerr << "Ошибка: Невозможно открыть исходный файл!" << endl;

        return -1;

    }

    if (fstat(tempFileHandler, &fileStat) == -1) {

        cerr << "Ошибка: Нет доступа к информации о файле!" << endl;

        close(tempFileHandler);

        return -1;

    }

    fileSize = fileStat.st\_size;

    close(tempFileHandler);

    cout << "Размер файла: " << fileSize << " байт" << endl;

    for (int currentChunk = mb; currentChunk <= mb10; currentChunk += mb) {

        for (int j = 0; j <= 4; j++) {

            chunk = currentChunk;

            operationsCount = (1 << j);

            for (int iteration = 0; iteration < 5; iteration++) {

                cout << "\n=== Размер кластера: " << chunk

                     << ", операций: " << operationsCount

                     << ", повторение #" << (iteration+1) << " ===\n";

                operations = (struct AioOperation\*)calloc(operationsCount, sizeof(struct AioOperation));

                auto startTime = high\_resolution\_clock::now();

                fileReadHandler = open(pathRead.c\_str(), O\_RDONLY  | O\_NONBLOCK, 0666);

                fileWriteHandler = open(pathWrite.c\_str(), O\_CREAT  | O\_WRONLY | O\_TRUNC | O\_NONBLOCK, 0666);

                if (fileReadHandler == -1 || fileWriteHandler == -1) {

                    cerr << "Ошибка: Невозможно открыть один из файлов (чтения или записи)!" << endl;

                    free(operations);

                    continue;

                }

                initOperations(operations, chunk);

                startOperations(operations);

                if (fsync(fileWriteHandler) == -1) {

                    cerr << "Ошибка: fsync не удался: " << strerror(errno) << endl;

                }

                sync();

                close(fileReadHandler);

                close(fileWriteHandler);

                auto endTime = high\_resolution\_clock::now();

                auto durationMs = duration\_cast<milliseconds>(endTime - startTime).count();

                cout << "Копирование завершено за " << durationMs << " мс" << endl;

                {

                    ofstream logFile("lab.log", ios::app);

                    if (!logFile) {

                        cerr << "Ошибка: невозможно открыть файл lab.log для записи!\n";

                    } else {

                        auto t = std::time(nullptr);

                        std::tm tmBuf;

                        localtime\_r(&t, &tmBuf);

                        char timeStr[64];

                        std::strftime(timeStr, sizeof(timeStr), "%Y-%m-%d %H:%M:%S", &tmBuf);

                        logFile << "[" << timeStr << "] "

                                << fileSize << " "

                                << chunk << " "

                                << operationsCount << " "

                                << durationMs

                                << "\n";

                    }

                }

                freeMemory(operations, operationsCount);

                sleep(2);

            }

        }

    }

    return 0;

}

void freeMemory(struct AioOperation\* operations, int n) {

    for (int i = 0; i < n; i++) {

        free(operations[i].buffer);

        free(operations[i].aio);

    }

    free(operations);

}

void aioCompletionHandler(sigval\_t sigval) {

    struct AioOperation \*aioOp = (struct AioOperation \*)sigval.sival\_ptr;

    if (aioOp->isReading) {

        aioOp->isReading = 0;

        aioOp->aio->aio\_fildes = fileWriteHandler;

        if (aio\_write(aioOp->aio) == -1) {

            cerr << "Ошибка: Асинхронная запись невозможна!" << endl;

            aioOp->isWorking = 0;

            gCv.notify\_all();

            return;

        }

    } else {

        if (aioOp->isWorking) {

            aioOp->aio->aio\_offset += chunk \* operationsCount;

            aioOp->aio->aio\_fildes = fileReadHandler;

            if (aioOp->aio->aio\_offset >= fileSize) {

                aioOp->isWorking = 0;

            } else {

                aioOp->isReading = 1;

                if (aioOp->aio->aio\_offset + chunk > fileSize) {

                    aioOp->aio->aio\_nbytes = fileSize - aioOp->aio->aio\_offset;

                }

                if (aio\_read(aioOp->aio) == -1) {

                    cerr << "Ошибка: Асинхронное чтение невозможно!" << endl;

                    aioOp->isWorking = 0;

                    gCv.notify\_all();

                    return;

                }

            }

        }

    }

    gCv.notify\_all();

}

void initOperations(struct AioOperation\* operations, int cluster) {

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

        operations[i].aio = (struct aiocb\*)calloc(1, sizeof(struct aiocb));

        memset(operations[i].aio, 0, sizeof(\*(operations[i].aio)));

        operations[i].buffer = (char\*)calloc(cluster, sizeof(char));

        operations[i].aio->aio\_sigevent.sigev\_notify = SIGEV\_THREAD;

        operations[i].aio->aio\_sigevent.sigev\_notify\_function = aioCompletionHandler;

        operations[i].aio->aio\_sigevent.sigev\_value.sival\_ptr = &operations[i];

        if (cluster > fileSize) {

            operations[i].aio->aio\_nbytes = fileSize;

        } else {

            operations[i].aio->aio\_nbytes = cluster;

        }

        operations[i].aio->aio\_fildes = fileReadHandler;

        operations[i].aio->aio\_offset = cluster \* i;

        operations[i].aio->aio\_buf = operations[i].buffer;

        operations[i].isWorking = 1;

        operations[i].isReading = 1;

    }

}

void startOperations(struct AioOperation\* operations) {

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

        if (operations[i].aio->aio\_offset >= fileSize) {

            operations[i].isWorking = 0;

        } else {

            if (operations[i].aio->aio\_offset + chunk > fileSize) {

                operations[i].aio->aio\_nbytes = fileSize - operations[i].aio->aio\_offset;

            }

            if (aio\_read(operations[i].aio) == -1) {

                cerr << "Ошибка: Асинхронное чтение невозможно!" << endl;

                operations[i].isWorking = 0;

            }

        }

    }

    await(operations, operationsCount);

}

void await(struct AioOperation\* operations, int count) {

    unique\_lock<mutex> lock(gMutex);

    auto allDone = [&]() {

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

            if (operations[i].isWorking != 0) {

                return false;

            }

        }

        return true;

    };

    gCv.wait(lock, allDone);

}