МИНОБРНАУКИ РОССИИ

Санкт-Петербургский государственный

электротехнический университет

«ЛЭТИ» им. В.И. Ульянова (Ленина)

Кафедра Вычислительной техники

**отчет**

по лабораторной работе №1

по дисциплине «Операционные системы»

по теме «Управление файловой системой»

|  |  |  |
| --- | --- | --- |
| Студентки гр. 2307 |  | Левушкина С.И. |
| Преподаватель |  | Тимофеев А.В. |

Санкт-Петербург, 2023

**СОДЕРЖАНИЕ**

[ЦЕЛЬ РАБОТЫ 3](#_Toc160889050)

[ЗАДАНИЯ 3](#_Toc160889051)

[1. Управление дисками, каталогами и файлами. 3](#_Toc160889052)

[выполнению. 3](#_Toc160889053)

[2. Копирование файла с помощью операций 4](#_Toc160889054)

[перекрывающегося ввода-вывода в Linux. 4](#_Toc160889055)

[ЗАДАНИЕ 1 6](#_Toc160889056)

[Описание работы программы и примеры 6](#_Toc160889057)

[Вывод 12](#_Toc160889058)

[ЗАДАНИЕ 2 14](#_Toc160889059)

[Описание работы программы 14](#_Toc160889060)

[Примеры работы программы 16](#_Toc160889061)

[Графики зависимости 17](#_Toc160889062)

[Анализ графиков 20](#_Toc160889063)

[Вывод 21](#_Toc160889064)

[ИСХОДНЫЕ КОДЫ ПРОГРАММ 21](#_Toc160889065)

[Задание 1 21](#_Toc160889066)

[Задание 2 (основная программа на с++) 40](#_Toc160889067)

[Разное количество операций 46](#_Toc160889068)

[Разные множители для размера блока 47](#_Toc160889069)

ЦЕЛЬ РАБОТЫ

Исследовать управление файловой системой с

помощью Win32 API, исследовать механизм асинхронных операций AIO в Linux.

ЗАДАНИЯ

1. Управление дисками, каталогами и файлами.

выполнению.

* 1. Создать консольное приложение с меню (каждая выполняемая

функция и/или операция должна быть доступна по отдельному пункту меню), которое выполняет:

1. Вывод списка дисков (функции Win32 API – GetLogicalDrives,

GetLogicalDriveStrings);

1. Для одного из выбранных дисков вывод информации о диске и размер

свободного пространства (функции Win32 API –

GetDriveType, GetVolumeInformation, GetDiskFreeSpace);

1. Создание и удаление заданных каталогов (функции Win32 API –

CreateDirectory, RemoveDirectory);

1. Создание файлов в новых каталогах (функция Win32 API –

CreateFile);

1. Копирование и перемещение файлов между каталогами с возможностью

выявления попытки работы с файлами, имеющими совпадающие

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

1. Анализ и изменение атрибутов файлов (функции Win32 API –

GetFileAttributes, SetFileAttributes, GetFileInformationByHandle,

GetFileTime, SetFileTime).

* 1. Запустить приложение и проверьте его работоспособность на нескольких наборах вводимых данных. Запротоколировать результаты в отчет. Дать свои комментарии в отчете относительно выполнения функций Win32 API.
  2. Подготовить отчет.

1. Копирование файла с помощью операций

перекрывающегося ввода-вывода в Linux.

Приложение должно копировать существующий файл в новый файл,

«одновременно» выполняя n перекрывающихся операций ввода-вывода

(механизм APC) блоками данных кратными размеру кластера.

* 1. Создать консольное приложение, которое выполняет

1. Файловый ввод-вывод блоками кратными размеру кластера;
2. Ожидание срабатывания вызова функции завершения
3. Измерение продолжительности выполнения операции копирования файла
   1. Запустить приложение и проверить его работоспособность на копировании файлов разного размера для ситуации с перекрывающимся выполнением одной операции ввода и одной операции вывода.

Выполнить эксперимент для разного размера копируемых блоков, построить график зависимости скорости копирования от размера блока данных. Определить оптимальный размер блока данных, при котором скорость копирования наибольшая. Запротоколировать результаты в отчет. Дать свои комментарии в отчете относительно выполнения функций Win32 API.

* 1. Произвести 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 построить график зависимости и определить число перекрывающихся операций ввода и вывода, при котором достигается

наибольшая скорость копирования файла. Запротоколировать результаты в

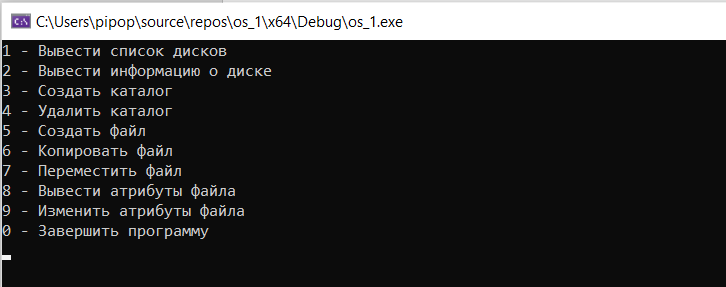
отчет.

* 1. Подготовить итоговый отчет.

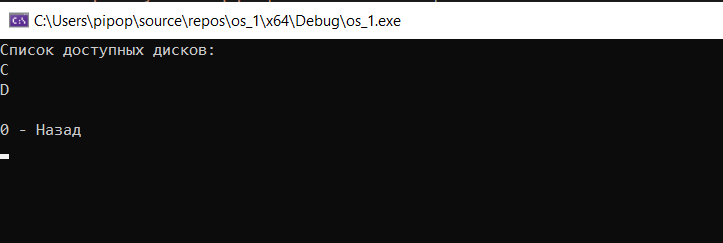
ЗАДАНИЕ 1

Описание работы программы и примеры

* 1. Меню (функция choice):

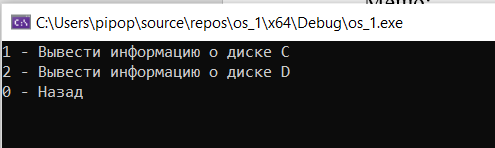


* 1. Список доступных дисков (функция show\_disks):

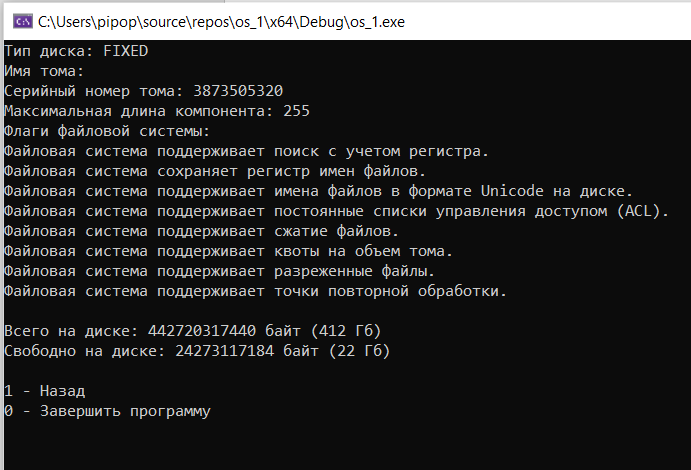


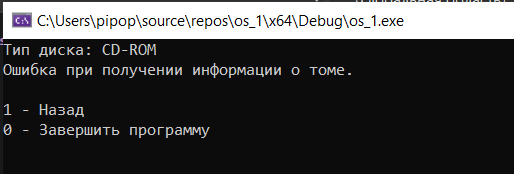
Функция show\_disks получает список доступных дисков на компьютере и выводит их на экран. Используется GetLogicalDrives - функция Windows API возвращает битовую маску, в которой каждый бит представляет доступность соответствующего диска.

Выбор диска (функция disk\_choice):



Информация о диске (функция show\_disk\_info):





Функция show\_disk\_info получает информацию о диске, на котором расположен указанный путь. При передаче значения disk = 1 функция работает с диском "C:\\", в противном случае с диском "D:\\".

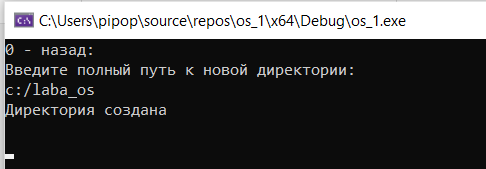
Использованные функции Windows API:

1. GetDriveType: Эта функция определяет тип диска (REMOVABLE, FIXED, REMOTE, CD-ROM, RAMDISK) по указанному пути.

2. GetVolumeInformation: Данная функция позволяет получить информацию о томе диска, включая его имя, серийный номер, максимальную длину компонента и флаги файловой системы.

1. GetDiskFreeSpaceEx: Эта функция используется для получения информации о доступном и свободном пространстве на диске в байтах и гигабайтах.

Создание каталога (функция new\_folder):

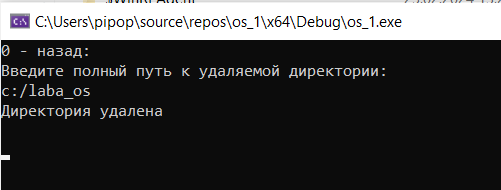


Функция new\_folder использует из Windows API:

1. CreateDirectory: Эта функция Windows API используется для создания новой директории. Она принимает путь к директории в формате LPCWSTR (широкие символы) и создает директорию с этим путем.

Если директория успешно создана, программа выводит сообщение об успешном создании и ждет 2 секунды (с помощью Sleep(2000)), затем вызывает функцию choice() для перехода к другому функционалу программы.

Удаление каталога (функция delete\_folder):

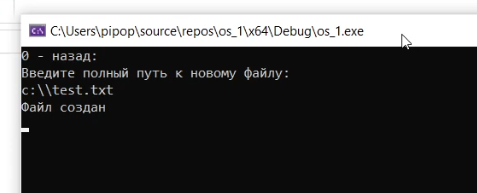


Функция delete\_folder удаляет выбранный каталог.

Использует функции Windows API:  
1. RemoveDirectory: Эта функция Windows API используется для удаления директории. Она принимает путь к директории в формате LPCWSTR (широкие символы) и удаляет директорию с этим путем.

Если директория успешно удалена, программа выводит сообщение об успешном удалении и ждет 2 секунды (с помощью Sleep(2000)), затем вызывает функцию choice() для перехода к другому функционалу программы.

Создание файла (функция new\_file):

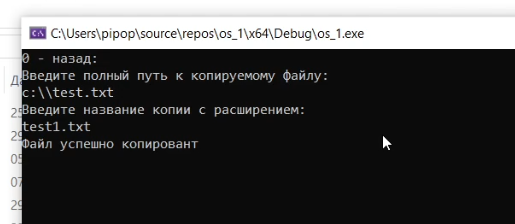


Функция new\_file создает новый файл в выбранной директории.

Использует функции Windows API:  
 1. CreateFile: Эта функция Windows API используется для создания файла. Она принимает путь к файлу в формате LPCWSTR (широкие символы), указывает режим доступа (GENERIC\_READ), режим совместного доступа (FILE\_SHARE\_READ), указывает, что файл должен быть создан, если он не существует (CREATE\_ALWAYS), и другие параметры.

Если файл успешно создан, программа выводит сообщение о создании файла, закрывает дескриптор файла с помощью CloseHandle(hFile), ждет 2 секунды (с помощью Sleep(2000)) и вызывает функцию choice() для перехода к другому функционалу программы.

Копирование файла (функция copy\_file):



Функция copy\_file копирует файл в выбранную директорию.

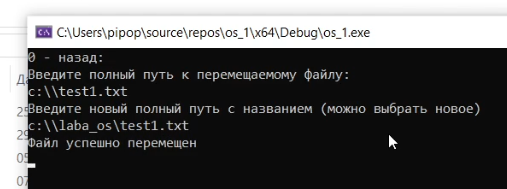
Использует функции Windows API:

1. CopyFile: Эта функция Windows API используется для копирования файла. Она принимает путь к исходному файлу и путь к целевому файлу (в формате LPCWSTR), а также флаг, указывающий, нужно ли перезаписать целевой файл, если он уже существует.

Если копирование прошло успешно, программа выводит сообщение о успешном копировании файла, ждет 2 секунды с помощью Sleep(2000) и вызывает функцию choice() для перехода к другому функционалу программы.

Если возникла ошибка при копировании файла, программа получает код ошибки с помощью GetLastError(), выводит его на экран и также вызывает Sleep(2000) и choice() для перехода к другому функционалу программы.

Перемещение файла (функция move\_file):



Функция move\_file перемещает файл в выбранную директорию.

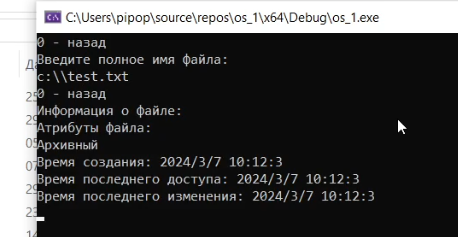
Использует функции Windows API:

1. Проверяется существование файла по новому пути с помощью функции PathFileExists. Если файл с таким именем уже существует, программа спрашивает пользователя, хочет ли он заменить существующий файл.

1. В зависимости от выбора пользователя ("y" или "n"), используется функция MoveFile или MoveFileEx для перемещения файла. Функция MoveFile используется, если файл по новому пути не существует, а MoveFileEx используется, если нужно заменить существующий файл.

Если перемещение прошло успешно, программа выводит сообщение о успешном перемещении файла. Если возникла ошибка при перемещении файла, программа выводит код ошибки с помощью GetLastError() и вызывает Sleep(2000) и choice() для перехода к другому функционалу программы.

Вывод атрибутов файла (функция show\_file\_attr)



Функция show\_file\_attr выводит атрибуты выбранного файла.

Использует функции Windows API:

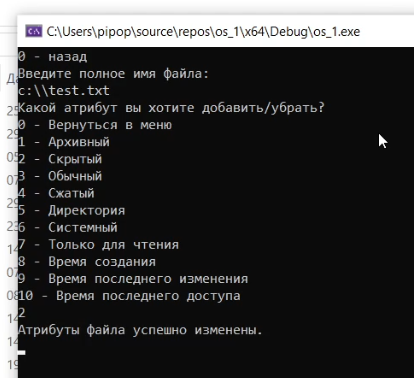
1. Получаются атрибуты файла с помощью функции GetFileAttributes. Если атрибуты получены успешно, программа выводит информацию об атрибутах файла: архивный, сжатый, директория, скрытый, обычный, только для чтения, системный.

1. Создается дескриптор файла с помощью CreateFile для получения информации о времени создания, последнего доступа и последнего изменения файла.

1. Если дескриптор файла создан успешно, программа использует GetFileInformationByHandle для получения информации о времени создания, последнего доступа и последнего изменения файла. Затем эта информация конвертируется в локальное время и выводится на экран.

Если возникает ошибка при открытии файла или получении информации, программа выводит соответствующее сообщение с кодом ошибки и вызывает Sleep(2000) и choice() для перехода к другому функционалу программы.

Изменение атрибутов файла (функция



Вывод

Программа выполняет работу с файловой системой, обеспечивает обработку ошибок при операциях с файлами, использует функции Windows API и позволяет изменять атрибуты файлов для управления доступом и безопасностью. Имеет дружественный интерфейс и возможность удобно перемещаться по меню для выполнения любых представленных операций. Также проверяет правильность ввода данных пользователем для избегания ошибок.

См. код программы в конце отчета.

ЗАДАНИЕ 2

Описание работы программы

Программа выполняет асинхронное копирование файлов с использованием механизма ввода-вывода с асинхронными операциями (AIO) в UNIX-подобных операционных системах (проверка работы на Ubuntu 20.04)

**Структура aio\_operation:**

Структура, которая содержит информацию о каждой асинхронной операции чтения и записи.

Включает в себя структуры aiocb для чтения и записи, буферы для чтения и записи, а также информацию о типе операции (чтение или запись).

**get\_file\_size(const char\* filename):**

Эта функция используется для получения размера файла по его имени.

Она вызывает функцию stat, которая возвращает информацию о файле, включая его размер.

В случае успеха функция возвращает размер файла, а в случае ошибки возвращает -1.

**aio\_completion\_handler(sigval\_t sigval):**

Это обработчик, который вызывается при завершении асинхронной операции чтения.

Он принимает указатель sigval на структуру aio\_operation.

При вызове обработчик создает буфер для записи и копирует в него прочитанные данные.

Затем он запускает асинхронную операцию записи с прочитанными данными.

**wait\_for\_all\_aio\_operations(struct aiocb \*aio\_list[], int num\_operations):**

Эта функция используется для ожидания завершения всех асинхронных операций в списке.

Она циклически проверяет статус каждой операции в списке.

Когда все операции завершены, функция выходит из цикла.

**input\_source():**

Эта функция запрашивает у пользователя путь к исходному файлу.

Она использует статический массив для хранения введенного пути.

Пользовательский ввод считывается с помощью cin.getline().

**input\_destination():**

Эта функция запрашивает у пользователя путь к новому местоположению файла.

Она также использует статический массив для хранения введенного пути.

Пользовательский ввод считывается с помощью cin.getline().

**input\_params():**

Эта функция запрашивает у пользователя параметры операции: количество параллельных операций и множитель блока данных.

Она возвращает указатель на массив, содержащий эти параметры.

**main():**

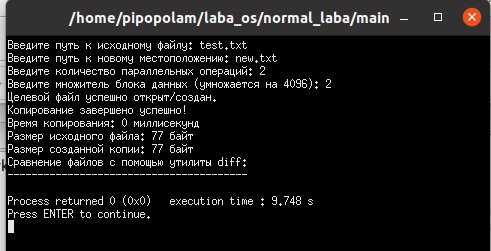
Это основная функция программы.

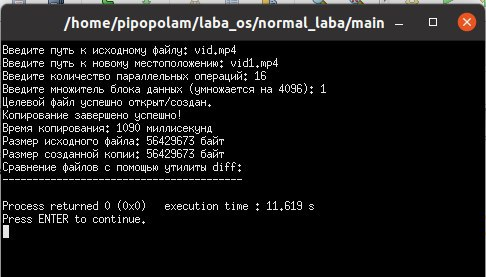
Она вызывает вышеперечисленные функции для получения пользовательского ввода и настройки параметров копирования.

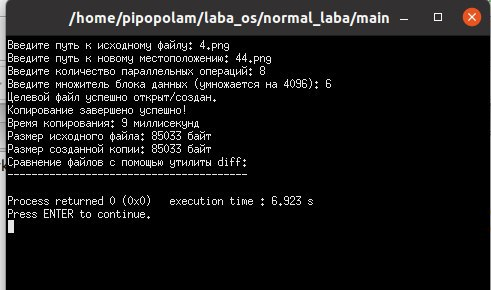
Затем она открывает исходный и целевой файлы, инициализирует асинхронные операции и выполняет копирование данных.

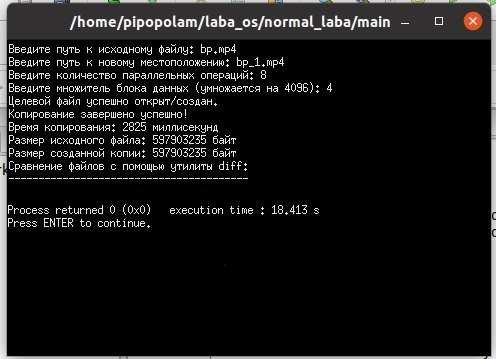
В конце функция измеряет время выполнения, выводит информацию о завершении копирования и сравнивает файлы с помощью утилиты diff.

Примеры работы программы





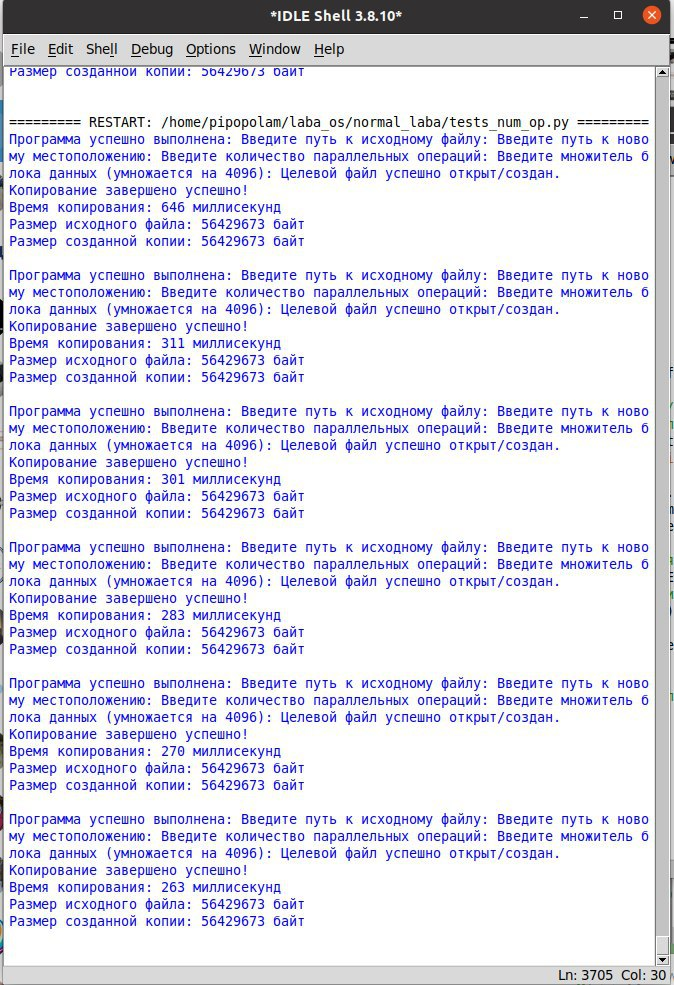


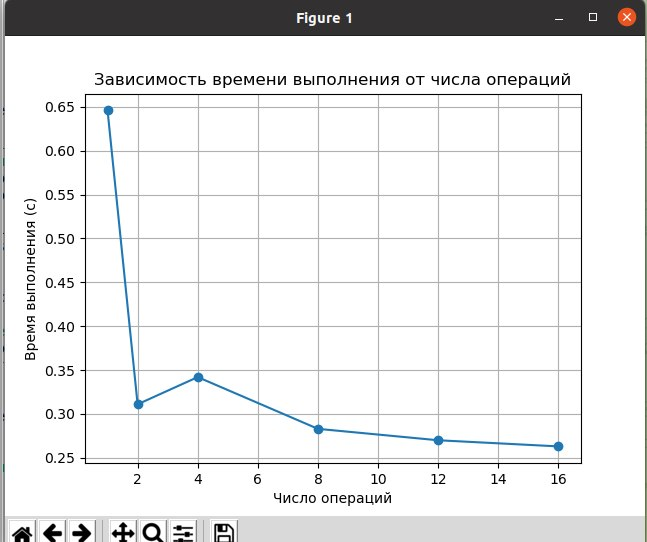


Графики зависимости

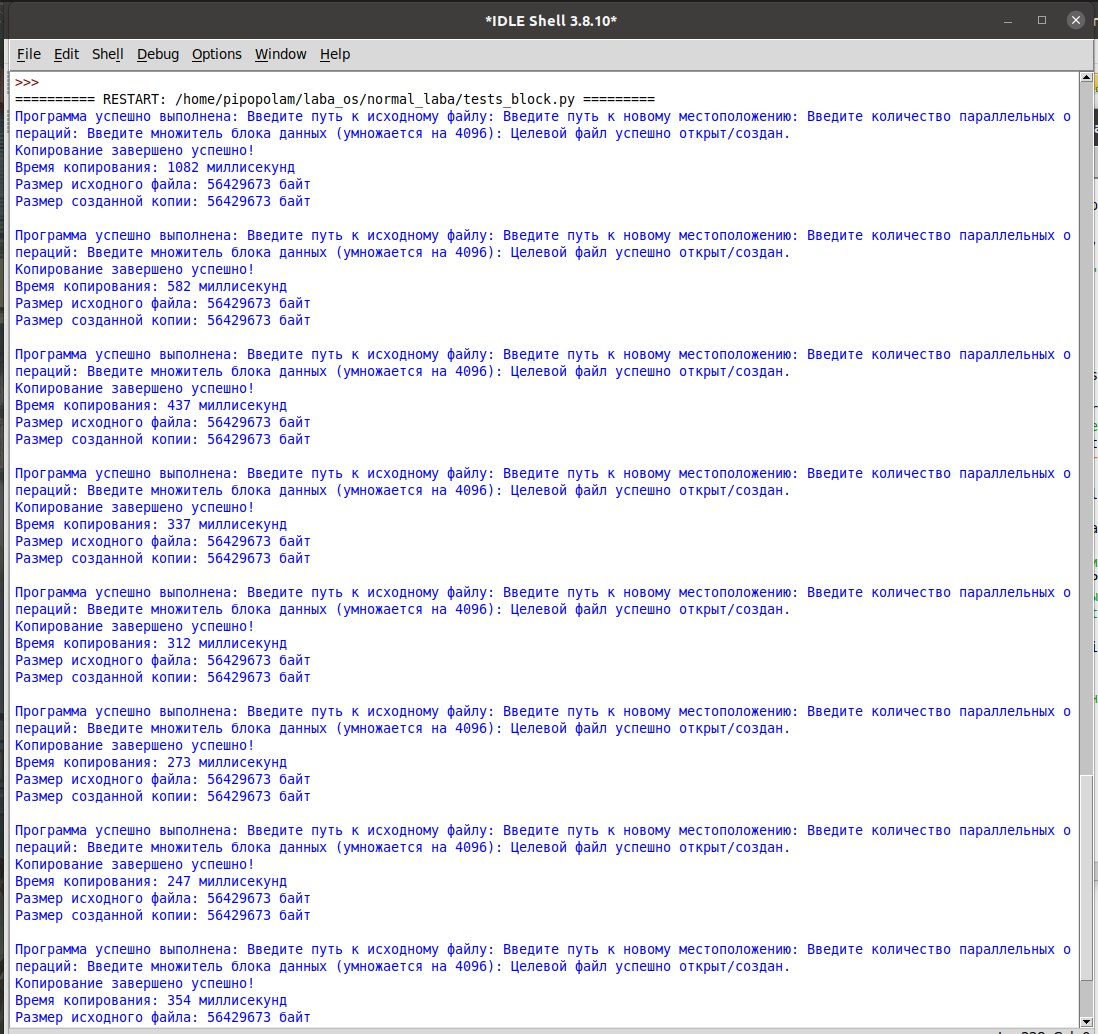
Для построения графиков зависимости времени копирования от количества перекрывающихся операций и размера блока для одной операции были написаны программы на python, которые запускают программу и имитируют ввод в консоль различных тестовых данных (разные значения количества операций и множитель размера блока). Из вывода консоли выделяется время выполнение, посчитанное в программе на с++ и записывается в список, на основе которого с помощью инструментов библиотеки matplotlib строится график зависимости.

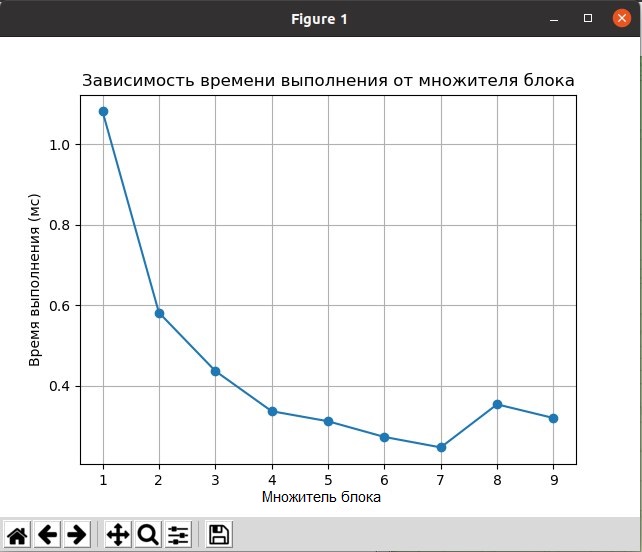
1. Зависимость времени от числа операций





1. Зависимость времени от множителя блока





Анализ графиков

В общем виде графики убывают с увеличением числа выполняемых операций и множителей блока. Вначале график убывает резко, но ближе к верхним значениям - все мягче. В графике с числом операций наблюдается скачок на значении 4-х операций. Вероятно, это связано с процессом, происходящим вне программы, который занял часть вычислительной мощности процесса. То же самое со скачком при множителях размера блока 8-9. Уменьшение скорости постепенно прекращается на обоих графиках, так как вычислительная мощность имеет предел и при значениях, больше представленных на графике, достигнет предела и не будет меняться с дальнейшим увеличением параметров.

Вывод

Анализ времени выполнения программы показал, что существует минимальный предел времени выполнения программы, который не меняется с увеличением размера блока и количества перекрывающихся операций. В данном случае – это размер блока 7\*4096 байт и 8-12 перекрывающихся операций. Эти параметры и являются оптимальными для работы программы, так как дальнейшее их увеличение приведет к увеличению расхода памяти, что является проблемой. Поэтому стоит использовать минимальные значения параметров, при которых достигается минимальное время выполнения.

ИСХОДНЫЕ КОДЫ ПРОГРАММ

Задание 1

#include <iostream>

#include <stdlib.h>

#include <windows.h>

#include <atlbase.h>

#include <atlconv.h>

#include <string>

#include <vector>

using namespace std;

void disk\_choice();

void choice();

void show\_disks();

void new\_folder();

FILETIME StringToFileTime(const string& timeStr) {

FILETIME fileTime;

SYSTEMTIME systemTime;

sscanf\_s(timeStr.c\_str(), "%d:%d:%d", &systemTime.wHour, &systemTime.wMinute, &systemTime.wSecond);

SystemTimeToFileTime(&systemTime, &fileTime);

return fileTime;

}

wstring string\_to\_wstring(const string& str)

{

wstring wStr(str.begin(), str.end());

return wStr;

}

string trimToLastDelimiter(const string& str) {

size\_t lastDelimiterPos = str.find\_last\_of('\\');

if (lastDelimiterPos != string::npos) {

return str.substr(0, lastDelimiterPos);

}

return str;

}

void show\_disk\_info(int disk) {

string path;

wstring wpath;

if (disk == 1) {

path = "C:\\";

} else {

path = "D:\\";

}

wpath = string\_to\_wstring(path);

LPWSTR volume\_name = new WCHAR[MAX\_PATH + 1];

DWORD VolumeSerialNum, MaxCompomtLen, SystemFlags;

UINT drive\_type = GetDriveType(wpath.c\_str());

system("cls");

cout << "Тип диска: ";

if (drive\_type == DRIVE\_REMOVABLE)

cout << "REMOVABLE\n";

else if (drive\_type == DRIVE\_FIXED)

cout << "FIXED\n";

else if (drive\_type == DRIVE\_REMOTE)

cout << "REMOTE\n";

else if (drive\_type == DRIVE\_CDROM)

cout << "CD-ROM\n";

else if (drive\_type == DRIVE\_RAMDISK)

cout << "RAMDISK\n";

else

cout << "НЕИЗВЕСТНЫЙ\_ТИП\n";

if (GetVolumeInformation(

wpath.c\_str(),

volume\_name,

MAX\_PATH + 1,

&VolumeSerialNum,

&MaxCompomtLen,

&SystemFlags,

NULL,

0

)) {

wcout << L"Имя тома: " << volume\_name << endl;

wcout << L"Серийный номер тома: " << VolumeSerialNum << endl;

wcout << L"Максимальная длина компонента: " << MaxCompomtLen << endl;

wcout << L"Флаги файловой системы: " << endl;

if (SystemFlags & FILE\_CASE\_SENSITIVE\_SEARCH) {

wcout << L"Файловая система поддерживает поиск с учетом регистра.\n";

}

if (SystemFlags & FILE\_CASE\_PRESERVED\_NAMES) {

wcout << L"Файловая система сохраняет регистр имен файлов.\n";

}

if (SystemFlags & FILE\_UNICODE\_ON\_DISK) {

wcout << L"Файловая система поддерживает имена файлов в формате Unicode на диске.\n";

}

if (SystemFlags & FILE\_PERSISTENT\_ACLS) {

wcout << L"Файловая система поддерживает постоянные списки управления доступом (ACL).\n";

}

if (SystemFlags & FILE\_FILE\_COMPRESSION) {

wcout << L"Файловая система поддерживает сжатие файлов.\n";

}

if (SystemFlags & FILE\_VOLUME\_QUOTAS) {

wcout << L"Файловая система поддерживает квоты на объем тома.\n";

}

if (SystemFlags & FILE\_SUPPORTS\_SPARSE\_FILES) {

wcout << L"Файловая система поддерживает разреженные файлы.\n";

}

if (SystemFlags & FILE\_SUPPORTS\_REPARSE\_POINTS) {

wcout << L"Файловая система поддерживает точки повторной обработки.\n";

}

}

else {

cout << "Ошибка при получении информации о томе." << endl;

}

if (drive\_type == DRIVE\_FIXED) {

ULARGE\_INTEGER FreeBytesAvailable, TotalNumberOfBytes, TotalNumberOfFreeBytes;

if (GetDiskFreeSpaceEx(

wpath.c\_str(),

&FreeBytesAvailable,

&TotalNumberOfBytes,

&TotalNumberOfFreeBytes

)) {

cout << "\nВсего на диске: " << TotalNumberOfBytes.QuadPart << " байт (" << TotalNumberOfBytes.QuadPart / (1024 \* 1024 \* 1024) << " Гб)\n";

cout << "Свободно на диске: " << TotalNumberOfFreeBytes.QuadPart << " байт (" << TotalNumberOfFreeBytes.QuadPart / (1024 \* 1024 \* 1024) << " Гб)\n";

}

else {

cout << "Ошибка при получении информации о свободном месте на диске." << endl;

}

}

cout << "\n1 - Назад\n0 - Завершить программу\n";

int flag = -1;

cin >> flag;

while (flag != 1 || flag != 0)

{

if (flag == 1)

{

choice();

}

else if (flag == 0)

{

exit(0);

}

}

}

void disk\_choice()

{

system("cls");

int disk = -1;

cout << "1 - Вывести информацию о диске С\n";

cout << "2 - Вывести информацию о диске D\n";

cout << "0 - Назад\n";

cin >> disk;

while (disk < -1 || disk > 3)

{

cout << "Ошибка. Выберите существующий диск (1 - С, 2 - D, 0 - назад)\n";

cin >> disk;

}

if (disk != 0)

{

show\_disk\_info(disk);

}

else

{

choice();

}

}

void show\_disks()

{

system("cls");

int flag = -1;

DWORD drives = GetLogicalDrives();

cout << "Список доступных дисков:" << endl;

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

if (drives & (1 << i)) {

cout << static\_cast<char>('A' + i) << "\n";

}

}

cout << endl << "0 - Назад\n";

flag = -1;

cin >> flag;

while (flag != 0)

{

cout << "Введите корректное число (0)\n";

cin >> flag;

}

choice();

}

void new\_folder()

{

system("cls");

string path;

cout << "0 - назад: \n";

cout << "Введите полный путь к новой директории: \n";

cin >> path;

if (path != "0")

{

if (CreateDirectory(string\_to\_wstring(path).c\_str(), NULL))

{

cout << "Директория создана\n" << endl;

Sleep(2000);

choice();

}

else

{

DWORD dwError = GetLastError();

cout << "Ошибка создания директории: " << dwError << endl;

Sleep(2000);

choice();

}

}

else

{

choice();

}

}

void delete\_folder()

{

system("cls");

string path;

cout << "0 - назад: \n";

cout << "Введите полный путь к удаляемой директории: \n";

cin >> path;

if (path != "0")

{

if (RemoveDirectory(string\_to\_wstring(path).c\_str()))

{

cout << "Директория удалена\n" << endl;

Sleep(2000);

choice();

}

else

{

DWORD dwError = GetLastError();

cout << "Ошибка удаления директории: " << dwError << endl;

Sleep(2000);

choice();

}

}

else

{

choice();

}

}

void new\_file()

{

system("cls");

string path;

cout << "0 - назад: \n";

cout << "Введите полный путь к новому файлу: \n";

cin >> path;

if (path != "0")

{

HANDLE hFile = CreateFile(

string\_to\_wstring(path).c\_str(),

GENERIC\_READ,

FILE\_SHARE\_READ,

NULL,

CREATE\_ALWAYS,

FILE\_ATTRIBUTE\_NORMAL,

NULL

);

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

CloseHandle(hFile);

cout << "Файл создан\n";

Sleep(2000);

choice();

}

else {

DWORD dwError = GetLastError();

cout << "Ошибка создания файла: " << dwError << endl;

Sleep(2000);

choice();

}

}

else

{

choice();

}

}

void copy\_file() {

system("cls");

string path, name, trim\_path;

cout << "0 - назад: \n";

if (path != "0")

{

cout << "Введите полный путь к копируемому файлу: \n";

cin >> path;

cout << "Введите название копии с расширением: \n";

cin >> name;

trim\_path = trimToLastDelimiter(path) + "\\" + name;

if (!CopyFile(string\_to\_wstring(path).c\_str(), string\_to\_wstring(trim\_path).c\_str(), true))

{

DWORD dwError = GetLastError();

cout << "Ошибка " << dwError << endl;

Sleep(2000);

choice();

}

else

{

cout << "Файл успешно копировант\n";

Sleep(2000);

choice();

}

}

else

{

choice();

}

}

void move\_file() {

system("cls");

string source\_path, destination\_path;

cout << "0 - назад: \n";

if (source\_path != "0") {

cout << "Введите полный путь к перемещаемому файлу: \n";

cin >> source\_path;

cout << "Введите новый полный путь с названием (можно выбрать новое) \n";

cin >> destination\_path;

if (PathFileExists(string\_to\_wstring(destination\_path).c\_str())) {

cout << "Файл с таким именем уже существует. Заменить? (y/n): ";

char choice;

cin >> choice;

if (choice == 'y' || choice == 'Y') {

if (!MoveFileEx(string\_to\_wstring(source\_path).c\_str(), string\_to\_wstring(destination\_path).c\_str(), MOVEFILE\_REPLACE\_EXISTING)) {

DWORD dwError = GetLastError();

cout << "Ошибка " << dwError << endl;

}

else {

cout << "Файл успешно перемещен\n";

}

}

else {

cout << "Файл не был перемещен\n";

}

}

else {

if (!MoveFile(string\_to\_wstring(source\_path).c\_str(), string\_to\_wstring(destination\_path).c\_str())) {

DWORD dwError = GetLastError();

cout << "Ошибка " << dwError << endl;

Sleep(2000);

choice();

}

else {

cout << "Файл успешно перемещен\n";

}

}

Sleep(2000);

choice();

}

else {

choice();

}

}

string path\_input()

{

string path;

system("cls");

cout << "0 - назад\nВведите полное имя файла: \n";

cin >> path;

return path;

}

void show\_file\_attr(string path) {

wstring wpath;

int flag;

if (path != "0") {

wpath = string\_to\_wstring(path);

DWORD attributes = GetFileAttributes(wpath.c\_str());

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

cout << "0 - назад\nИнформация о файле: \n";

cout << "Атрибуты файла:\n";

if (attributes & FILE\_ATTRIBUTE\_ARCHIVE)

cout << "Архивный\n";

if (attributes & FILE\_ATTRIBUTE\_COMPRESSED)

cout << "Сжатый\n";

if (attributes & FILE\_ATTRIBUTE\_DIRECTORY)

cout << "Директория\n";

if (attributes & FILE\_ATTRIBUTE\_HIDDEN)

cout << "Скрытый\n";

if (attributes & FILE\_ATTRIBUTE\_NORMAL)

cout << "Обычный\n";

if (attributes & FILE\_ATTRIBUTE\_READONLY)

cout << "Только для чтения\n";

if (attributes & FILE\_ATTRIBUTE\_SYSTEM)

cout << "Системный\n";

HANDLE hFile = CreateFile(wpath.c\_str(), GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

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

BY\_HANDLE\_FILE\_INFORMATION fileInformation;

if (GetFileInformationByHandle(hFile, &fileInformation)) {

FILETIME ftCreate, ftAccess, ftWrite;

SYSTEMTIME stUTC, stLocal;

ftCreate = fileInformation.ftCreationTime;

ftAccess = fileInformation.ftLastAccessTime;

ftWrite = fileInformation.ftLastWriteTime;

FileTimeToSystemTime(&ftCreate, &stUTC);

SystemTimeToTzSpecificLocalTime(NULL, &stUTC, &stLocal);

cout << "Время создания: " << stLocal.wYear << "/" << stLocal.wMonth << "/" << stLocal.wDay << " " << stLocal.wHour << ":" << stLocal.wMinute << ":" << stLocal.wSecond << endl;

FileTimeToSystemTime(&ftAccess, &stUTC);

SystemTimeToTzSpecificLocalTime(NULL, &stUTC, &stLocal);

cout << "Время последнего доступа: " << stLocal.wYear << "/" << stLocal.wMonth << "/" << stLocal.wDay << " " << stLocal.wHour << ":" << stLocal.wMinute << ":" << stLocal.wSecond << endl;

FileTimeToSystemTime(&ftWrite, &stUTC);

SystemTimeToTzSpecificLocalTime(NULL, &stUTC, &stLocal);

cout << "Время последнего изменения: " << stLocal.wYear << "/" << stLocal.wMonth << "/" << stLocal.wDay << " " << stLocal.wHour << ":" << stLocal.wMinute << ":" << stLocal.wSecond << endl;

}

CloseHandle(hFile);

}

else {

DWORD dwError = GetLastError();

cout << "Ошибка при открытии файла: " << dwError << endl;

Sleep(2000);

choice();

}

}

else {

DWORD dwError = GetLastError();

cout << "Ошибка " << dwError << endl;

Sleep(2000);

choice();

}

cin >> flag;

while (flag != 0)

{

cout << "Введите корректное значение (0): \n";

cin >> flag;

}

choice();

}

else {

choice();

}

}

void edit\_file\_attr()

{

system("cls");

string path = path\_input();

wstring wpath = string\_to\_wstring(path);

int flag=-1;

cout << "Какой атрибут вы хотите добавить/убрать?\n";

cout << "0 - Вернуться в меню\n";

cout << "1 - Архивный\n";

cout << "2 - Скрытый\n";

cout << "3 - Обычный\n";

cout << "4 - Сжатый\n";

cout << "5 - Директория\n";

cout << "6 - Системный\n";

cout << "7 - Только для чтения\n";

cout << "8 - Время создания\n";

cout << "9 - Время последнего изменения\n";

cout << "10 - Время последнего доступа\n";

cin >> flag;

DWORD fileAttributes = GetFileAttributes(wpath.c\_str());

switch (flag) {

case 0:

choice();

case 1:

if (fileAttributes & FILE\_ATTRIBUTE\_ARCHIVE)

fileAttributes &= ~FILE\_ATTRIBUTE\_ARCHIVE;

else

fileAttributes |= FILE\_ATTRIBUTE\_ARCHIVE;

break;

case 2:

if (fileAttributes & FILE\_ATTRIBUTE\_HIDDEN)

fileAttributes &= ~FILE\_ATTRIBUTE\_HIDDEN;

else

fileAttributes |= FILE\_ATTRIBUTE\_HIDDEN;

break;

case 3:

break;

case 4:

if (fileAttributes & FILE\_ATTRIBUTE\_COMPRESSED)

fileAttributes &= ~FILE\_ATTRIBUTE\_COMPRESSED;

else

fileAttributes |= FILE\_ATTRIBUTE\_COMPRESSED;

break;

case 5:

cout << "Директорию изменить таким способом нельзя, выполните перемещение из соответствующего пункта\n";

Sleep(2000);

choice();

case 6:

if (fileAttributes & FILE\_ATTRIBUTE\_SYSTEM)

fileAttributes &= ~FILE\_ATTRIBUTE\_SYSTEM;

else

fileAttributes |= FILE\_ATTRIBUTE\_SYSTEM;

break;

case 7:

if (fileAttributes & FILE\_ATTRIBUTE\_READONLY)

fileAttributes &= ~FILE\_ATTRIBUTE\_READONLY;

else

fileAttributes |= FILE\_ATTRIBUTE\_READONLY;

break;

case 8:

{

string newCreationDateStr, newCreationTimeStr;

cout << "Введите новую дату создания файла в формате ГГГГ-MM-ДД: ";

cin >> newCreationDateStr;

cout << "Введите новое время создания файла в формате ЧЧ:ММ:СС: ";

cin >> newCreationTimeStr;

SYSTEMTIME creationSystemTime;

sscanf\_s(newCreationDateStr.c\_str(), "%d-%d-%d", &creationSystemTime.wYear, &creationSystemTime.wMonth, &creationSystemTime.wDay);

sscanf\_s(newCreationTimeStr.c\_str(), "%d:%d:%d", &creationSystemTime.wHour, &creationSystemTime.wMinute, &creationSystemTime.wSecond);

FILETIME creationFileTime;

SystemTimeToFileTime(&creationSystemTime, &creationFileTime);

HANDLE hFile = CreateFile(

wpath.c\_str(),

FILE\_WRITE\_ATTRIBUTES,

FILE\_SHARE\_READ | FILE\_SHARE\_WRITE | FILE\_SHARE\_DELETE,

NULL,

OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL,

NULL

);

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

cout << "Ошибка при открытии файла для изменения времени создания." << endl;

Sleep(2000);

choice();

}

if (!SetFileTime(hFile, &creationFileTime, NULL, NULL)) {

cout << "Ошибка при изменении времени создания файла." << endl;

CloseHandle(hFile);

Sleep(2000);

choice();

}

else {

cout << "Время создания файла успешно изменено." << endl;

CloseHandle(hFile);

Sleep(2000);

choice();

}

break;

}

case 9:

{

string newWriteDateStr, newWriteTimeStr;

cout << "Введите новую дату изменения файла в формате ГГГГ-MM-ДД: ";

cin >> newWriteDateStr;

cout << "Введите новое время изменения файла в формате ЧЧ:MM:СС: ";

cin >> newWriteTimeStr;

SYSTEMTIME writeSystemTime;

sscanf\_s(newWriteDateStr.c\_str(), "%d-%d-%d", &writeSystemTime.wYear, &writeSystemTime.wMonth, &writeSystemTime.wDay);

sscanf\_s(newWriteTimeStr.c\_str(), "%d:%d:%d", &writeSystemTime.wHour, &writeSystemTime.wMinute, &writeSystemTime.wSecond);

FILETIME writeFileTime;

SystemTimeToFileTime(&writeSystemTime, &writeFileTime);

HANDLE hFile = CreateFile(

wpath.c\_str(),

FILE\_WRITE\_ATTRIBUTES,

FILE\_SHARE\_READ | FILE\_SHARE\_WRITE | FILE\_SHARE\_DELETE,

NULL,

OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL,

NULL

);

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

cout << "Ошибка при открытии файла для изменения времени изменения." << endl;

Sleep(2000);

choice();

}

if (!SetFileTime(hFile, NULL, NULL, &writeFileTime)) {

cout << "Ошибка при изменении времени изменения файла." << endl;

CloseHandle(hFile);

Sleep(2000);

choice();

}

else {

cout << "Время изменения файла успешно изменено." << endl;

CloseHandle(hFile);

Sleep(2000);

choice();

}

break;

}

case 10:

{

string newAccessDateStr, newAccessTimeStr;

cout << "Введите новую дату последнего доступа к файлу в формате ГГГГ-MM-ДД: ";

cin >> newAccessDateStr;

cout << "Введите новое время последнего доступа к файлу в формате ЧЧ:MM:СС: ";

cin >> newAccessTimeStr;

SYSTEMTIME accessSystemTime;

sscanf\_s(newAccessDateStr.c\_str(), "%d-%d-%d", &accessSystemTime.wYear, &accessSystemTime.wMonth, &accessSystemTime.wDay);

sscanf\_s(newAccessTimeStr.c\_str(), "%d:%d:%d", &accessSystemTime.wHour, &accessSystemTime.wMinute, &accessSystemTime.wSecond);

FILETIME accessFileTime;

SystemTimeToFileTime(&accessSystemTime, &accessFileTime);

HANDLE hFile = CreateFile(

wpath.c\_str(),

FILE\_WRITE\_ATTRIBUTES,

FILE\_SHARE\_READ | FILE\_SHARE\_WRITE | FILE\_SHARE\_DELETE,

NULL,

OPEN\_EXISTING,

FILE\_ATTRIBUTE\_NORMAL,

NULL

);

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

cout << "Ошибка при открытии файла для изменения времени последнего доступа." << endl;

Sleep(2000);

choice();

}

if (!SetFileTime(hFile, NULL, &accessFileTime, NULL)) {

cout << "Ошибка при изменении времени последнего доступа к файлу." << endl;

CloseHandle(hFile);

Sleep(2000);

choice();

}

else {

cout << "Время последнего доступа к файлу успешно изменено." << endl;

CloseHandle(hFile);

Sleep(2000);

choice();

}

break;

}

default:

cout << "Некорректный выбор атрибута." << endl;

while (flag < 1 || flag > 7)

{

cout << "Введите корректный номер выбора (0-9):\n";

cin >> flag;

}

}

if (!SetFileAttributes(wpath.c\_str(), fileAttributes)) {

cout << "Ошибка при изменении атрибутов файла." << endl;

Sleep(2000);

choice();

}

else {

cout << "Атрибуты файла успешно изменены." << endl;

Sleep(2000);

choice();

}

}

void choice()

{

system("cls");

int choice=-1;

cout << "1 - Вывести список дисков\n";

cout << "2 - Вывести информацию о диске\n";

cout << "3 - Создать каталог\n";

cout << "4 - Удалить каталог\n";

cout << "5 - Создать файл\n";

cout << "6 - Копировать файл\n";

cout << "7 - Переместить файл\n";

cout << "8 - Вывести атрибуты файла\n";

cout << "9 - Изменить атрибуты файла\n";

cout << "0 - Завершить программу\n";

cin >> choice;

while (choice < -1 || choice > 9)

{

cout << "Ошибка. Повторите ввод\n";

cin >> choice;

}

switch (choice)

{

case 0: exit(0);

case 1: show\_disks();

case 2: disk\_choice();

case 4: delete\_folder();

case 3: new\_folder();

case 5: new\_file();

case 6: copy\_file();

case 7: move\_file();

case 8: show\_file\_attr(path\_input());

case 9: edit\_file\_attr();

}

}

int main()

{

setlocale(LC\_ALL, "Russian");

choice();

return 0;

}

Задание 2 (основная программа на с++)

#include <iostream>

#include <cstring>

#include <aio.h>

#include <fcntl.h>

#include <unistd.h>

#include <errno.h>

#include <signal.h>

#include <chrono>

#include <sys/stat.h>

using namespace std;

struct aio\_operation

{

struct aiocb aio\_read;

struct aiocb aio\_write;

char \*read\_buffer;

char \*write\_buffer;

int write\_operation; // 1 - запись, 0 - чтение

};

off\_t get\_file\_size(const char\* filename) {

struct stat file\_stat;

if (stat(filename, &file\_stat) == 0)

return file\_stat.st\_size;

return -1;

}

void aio\_completion\_handler(sigval\_t sigval)

{

struct aio\_operation \*aio\_op = (struct aio\_operation \*)sigval.sival\_ptr;

if (!aio\_op->write\_operation)

{

ssize\_t bytes\_read = aio\_return(&aio\_op->aio\_read);

if (bytes\_read != -1)

{

aio\_op->write\_buffer = new char[bytes\_read];

memcpy(aio\_op->write\_buffer, aio\_op->read\_buffer, bytes\_read);

aio\_op->write\_operation = 1;

aio\_op->aio\_write.aio\_nbytes = bytes\_read;

aio\_op->aio\_write.aio\_buf = aio\_op->write\_buffer;

aio\_op->aio\_write.aio\_offset = aio\_op->aio\_read.aio\_offset;

aio\_write(&aio\_op->aio\_write);

}

}

}

void wait\_for\_all\_aio\_operations(struct aiocb \*aio\_list[], int num\_operations) {

while (true) {

int completed = 0;

for (int i = 0; i < num\_operations; ++i) {

if (aio\_error(aio\_list[i]) == EINPROGRESS) {

continue;

} else {

completed++;

}

}

if (completed == num\_operations) {

break;

}

}

}

char\* input\_source()

{

static char source\_file[256];

cout << "Введите путь к исходному файлу: ";

cin.getline(source\_file, 256);

return source\_file;

}

char\* input\_destination()

{

static char destination\_file[256];

cout << "Введите путь к новому местоположению: ";

cin.getline(destination\_file, 256);

return destination\_file;

}

int\* input\_params()

{

static int params[2];

cout << "Введите количество параллельных операций: ";

cin >> params[0];

cout << "Введите множитель блока данных (умножается на 4096): ";

cin >> params[1];

return params;

}

int main() {

char\* source\_file = input\_source();

char\* destination\_file = input\_destination();

int\* params = input\_params();

int num\_operations = params[0];

int multiplier = params[1];

const int block\_size\_multiplier = 4096;

off\_t file\_size = get\_file\_size(source\_file);

int sets = file\_size / (num\_operations \* block\_size\_multiplier);

if (file\_size % (num\_operations \* block\_size\_multiplier) > 0) {

sets++;

}

if (file\_size == -1)

{

cout << "Ошибка при получении размера исходного файла." << endl;

exit(EXIT\_FAILURE);

}

auto start\_time = chrono::steady\_clock::now();

int source\_fd = open(source\_file, O\_RDONLY | O\_NONBLOCK, 0666);

if (source\_fd == -1)

{

cout << "Ошибка при открытии исходного файла: " << strerror(errno) << endl;

exit(EXIT\_FAILURE);

}

int destination\_fd = open(destination\_file, O\_CREAT | O\_WRONLY | O\_TRUNC | O\_NONBLOCK, 0666);

if (destination\_fd == -1)

{

cerr << "Ошибка при открытии/создании целевого файла: " << strerror(errno) << endl;

exit(EXIT\_FAILURE);

}

cout << "Целевой файл успешно открыт/создан." << endl;

off\_t copied\_size = 0;

aiocb\* aio\_read\_list[num\_operations];

aiocb\* aio\_write\_list[num\_operations];

off\_t common\_offset = 0;

while (copied\_size < file\_size) {

aio\_operation\* aio\_ops = new aio\_operation[num\_operations];

for (int i = 0; i < num\_operations; ++i) {

int block\_size\_operation = (file\_size - copied\_size) < (multiplier \* block\_size\_multiplier) ?

(file\_size - copied\_size) : (multiplier \* block\_size\_multiplier);

//cout << copied\_size << endl;

aio\_ops[i].read\_buffer = new char[block\_size\_operation];

aio\_ops[i].write\_buffer = nullptr;

aio\_ops[i].write\_operation = 0;

memset(&aio\_ops[i].aio\_read, 0, sizeof(struct aiocb));

memset(&aio\_ops[i].aio\_write, 0, sizeof(struct aiocb));

aio\_ops[i].aio\_read.aio\_fildes = source\_fd;

aio\_ops[i].aio\_write.aio\_fildes = destination\_fd;

aio\_ops[i].aio\_read.aio\_buf = aio\_ops[i].read\_buffer;

aio\_ops[i].aio\_write.aio\_buf = nullptr;

aio\_ops[i].aio\_read.aio\_nbytes = block\_size\_operation;

aio\_ops[i].aio\_write.aio\_nbytes = 0;

aio\_ops[i].aio\_read.aio\_sigevent.sigev\_notify = SIGEV\_THREAD;

aio\_ops[i].aio\_read.aio\_sigevent.sigev\_notify\_function = aio\_completion\_handler;

aio\_ops[i].aio\_read.aio\_sigevent.sigev\_notify\_attributes = NULL;

aio\_ops[i].aio\_read.aio\_sigevent.sigev\_value.sival\_ptr = &aio\_ops[i];

aio\_read\_list[i] = &aio\_ops[i].aio\_read;

aio\_write\_list[i] = &aio\_ops[i].aio\_write;

}

for (int i = 0; i < num\_operations; ++i)

{

aio\_ops[i].aio\_read.aio\_offset = common\_offset;

aio\_read(&aio\_ops[i].aio\_read);

common\_offset += aio\_ops[i].aio\_read.aio\_nbytes;

}

wait\_for\_all\_aio\_operations(aio\_read\_list, num\_operations);

for (int i = 0; i < num\_operations; ++i)

{

ssize\_t bytes\_read = aio\_return(&aio\_ops[i].aio\_read);

if (bytes\_read > 0)

{

char \*write\_buffer = new char[bytes\_read];

memcpy(write\_buffer, aio\_ops[i].read\_buffer, bytes\_read);

aio\_ops[i].write\_buffer = write\_buffer;

aio\_ops[i].aio\_write.aio\_nbytes = bytes\_read;

aio\_ops[i].aio\_write.aio\_offset = aio\_ops[i].aio\_read.aio\_offset;

aio\_write(&aio\_ops[i].aio\_write);

copied\_size += bytes\_read;

//cout << "Прочитано байт: " << bytes\_read << endl;

}

}

wait\_for\_all\_aio\_operations(aio\_write\_list, num\_operations);

for (int i = 0; i < num\_operations; ++i)

{

ssize\_t bytes\_written = aio\_return(&aio\_ops[i].aio\_write);

//cout << copied\_size << endl;

//cout << "Записано байт: " << bytes\_written << endl;

delete[] aio\_ops[i].write\_buffer;

}

delete[] aio\_ops;

}

wait\_for\_all\_aio\_operations(aio\_read\_list, num\_operations);

wait\_for\_all\_aio\_operations(aio\_write\_list, num\_operations);

close(source\_fd);

close(destination\_fd);

cout << "Копирование завершено успешно!" << endl;

auto end\_time = chrono::steady\_clock::now();

auto elapsed\_time = chrono::duration\_cast<chrono::milliseconds>(end\_time - start\_time);

cout << "Время копирования: " << elapsed\_time.count() << " миллисекунд" << endl;

off\_t destination\_size = get\_file\_size(destination\_file);

if (destination\_size != -1)

{

cout << "Размер исходного файла: " << file\_size << " байт" << endl;

cout << "Размер созданной копии: " << destination\_size << " байт" << endl;

}

else

{

cout << "Ошибка при получении размера копии файла." << endl;

}

cout << "Сравнение файлов с помощью утилиты diff:" << endl;

cout << "----------------------------------------" << endl;

system(("diff " + string(source\_file) + " " + string(destination\_file)).c\_str());

return 0;

}

Задание 2 (скрипты на python для прогона разный параметров)

Разное количество операций

import subprocess

import matplotlib.pyplot as plt

num\_operations = [1, 2, 4, 8, 12, 16]

source\_file = "bp.mp4"

destination\_file = "new\_bp.mp4"

execution\_times = []

for num\_op in num\_operations:

num\_op\_str = str(num\_op)

parameters = '\n'.join([source\_file, destination\_file, num\_op\_str, '10'])

try:

result = subprocess.run(['./main'], input=parameters, text=True, check=True, capture\_output=True)

print("Программа успешно выполнена:", result.stdout)

output\_lines = result.stdout.split('\n')

time\_line = [line for line in output\_lines if 'Время копирования:' in line]

if time\_line:

time\_str = time\_line[0].split(":")[-1].strip().split()[0]

execution\_time = int(time\_str)

execution\_times.append(execution\_time)

else:

print("Время копирования не найдено в выводе программы.")

except subprocess.CalledProcessError as e:

print(f"Ошибка при выполнении тестирования: {e}")

print(f"stderr: {e.stderr}")

plt.plot(num\_operations, execution\_times, marker='o')

plt.xlabel('Число операций')

plt.ylabel('Время выполнения (мс)')

plt.title('Зависимость времени выполнения от числа операций')

plt.grid(True)

plt.show()

Разные множители для размера блока

import subprocess

import matplotlib.pyplot as plt

block\_size\_mp = [1, 5, 10, 15, 20, 25, 30, 35, 40]

source\_file = "bp.mp4"

destination\_file = "new\_bp.mp4"

execution\_times = []

for mp in block\_size\_mp:

mp\_str = str(mp)

parameters = '\n'.join([source\_file, destination\_file, "12", mp\_str])

try:

result = subprocess.run(['./main'], input=parameters, text=True, check=True, capture\_output=True)

print("Программа успешно выполнена:", result.stdout)

output\_lines = result.stdout.split('\n')

time\_line = [line for line in output\_lines if 'Время копирования:' in line]

if time\_line:

time\_str = time\_line[0].split(":")[-1].strip().split()[0]

execution\_time = int(time\_str)

execution\_times.append(execution\_time)

else:

print("Время копирования не найдено в выводе программы.")

except subprocess.CalledProcessError as e:

print(f"Ошибка при выполнении тестирования: {e}")

print(f"stderr: {e.stderr}")

plt.plot(block\_size\_mp, execution\_times, marker='o')

plt.xlabel('Число операций')

plt.ylabel('Время выполнения (мс)')

plt.title('Зависимость времени выполнения от множителя размера блока')

plt.grid(True)

plt.show()