Минобрнауки России
Федеральное государственное автономное образовательное
Учреждение высшего образования
«Санкт-Петербургский государственный электротехнический
Университет им. В.И. Ульянова (Ленина)»
(СПбГЭТУ «ЛЭТИ»)
Факультет компьютерных технологий и информатики

Кафедра вычислительной техники

Отчет по лабораторной работе №1

на тему:

«Управление файловой системой»

по дисциплине «Операционные системы»

Выполнил студент группы 9308: Яловега Н.В.

Принял: к.т.н., доцент Тимофеев А.В.

Санкт-Петербург
2021 г.

Оглавление

1. Цель работы:

Исследовать управление файловой системой с помощью Win32 API.

2. Управление дисками, каталогами и файлами.

2.1. Указания к выполнению.

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).

Ссылка на репозиторий с исходным кодом первой части работы: https://github.com/kivyfreakt/eltech/tree/master/os/lab1/task1

2.2. Результаты работы программы.

1. Меню с пунктами, необходимыми в процессе выполнения программы:

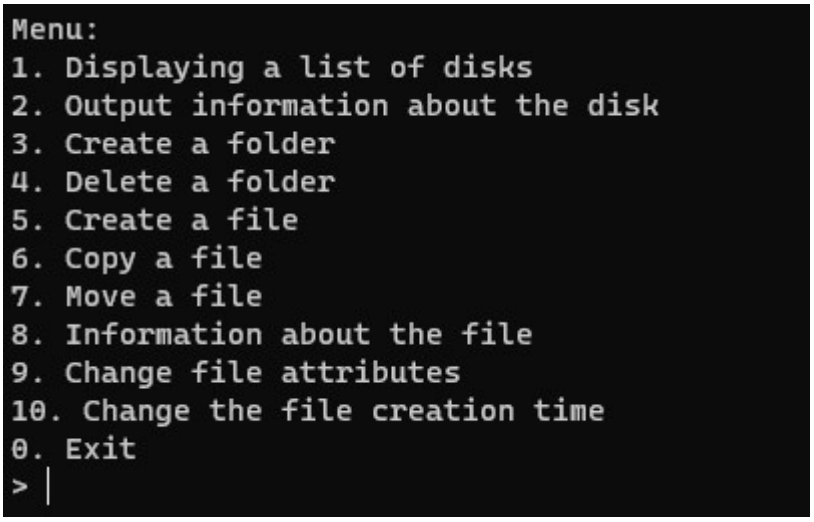

Рисунок 1: Главное меню

2. Вывод списка дисков (функция GetLogicalDrives()):

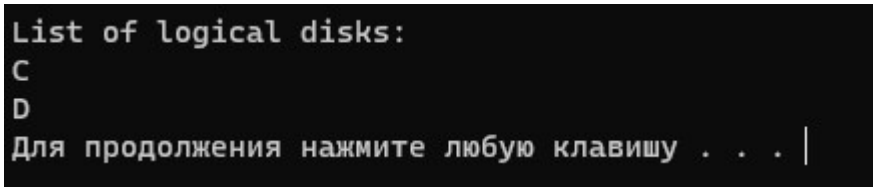

Рисунок 2: Список дисков

3. Вывод информации о дисках и размере свободного пространства (функция GetVolumeInformation()), получаемая переменная lpFileSystemFlags декодируется путём конъюнкции с определёнными атрибутами:

```
Enter the volume label (for example, C): C
Disk type: Fixed drive
Disk name:
Serial number: 907334597
File system type: NTFS
System flags:
The specified volume supports preserved case of file names when it places a name on disk.
The specified volume supports case-sensitive file names.
The specified volume supports file-based compression.
The specified volume supports named streams.
The specified volume preserves and enforces access control lists (ACL).
The specified volume supports the Encrypted File System (EFS).
The specified volume supports extended attributes.
The specified volume supports hard links.
The specified volume supports object identifiers.
The specified volume supports open by FileID.
The specified volume supports reparse points.
The specified volume supports sparse files.
The specified volume supports transactions.
The specified volume supports update sequence number (USN) journals.
The specified volume supports Unicode in file names as they appear on disk.
The specified volume supports disk quotas.
Number of sectors in a cluster: 8
The number of bytes in the sector: 512
Number of free clusters: 2857940
Total number of clusters: 15827199
Disk space (free/total): 11163 / 61824 MiB
Для продолжения нажмите любую клавишу . . .
```

Рисунок 3: Свойства диска

4. Создание каталога (функция CreateDirectory()):

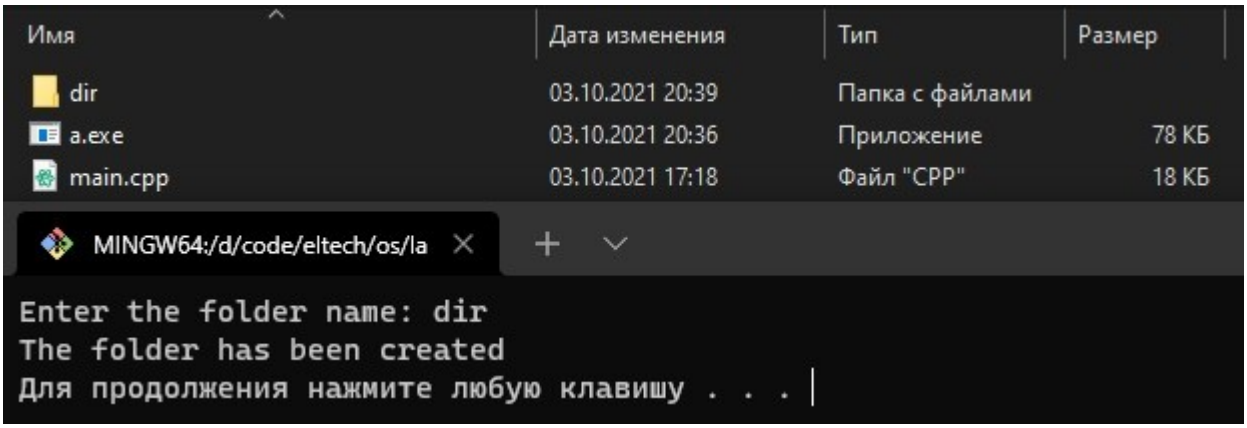

Рисунок 4: Создание каталога

5. Удаление каталога (функция RemoveDirectory()):

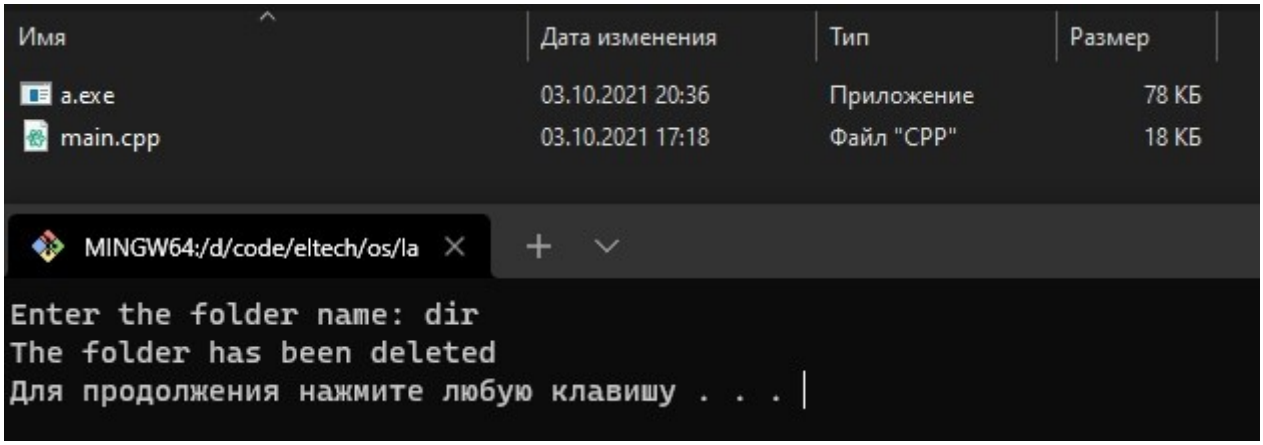

Рисунок 5: Удаление каталога

6. Создание файла (функция CreateFile()):

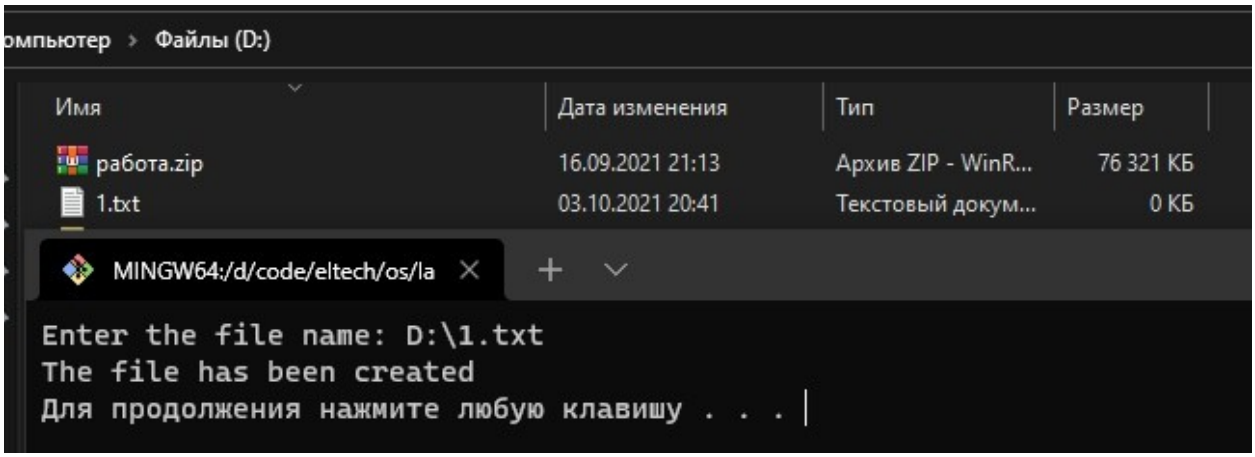

Рисунок 6: Создание файла

7. Копирование файла (функция CopyFile()):

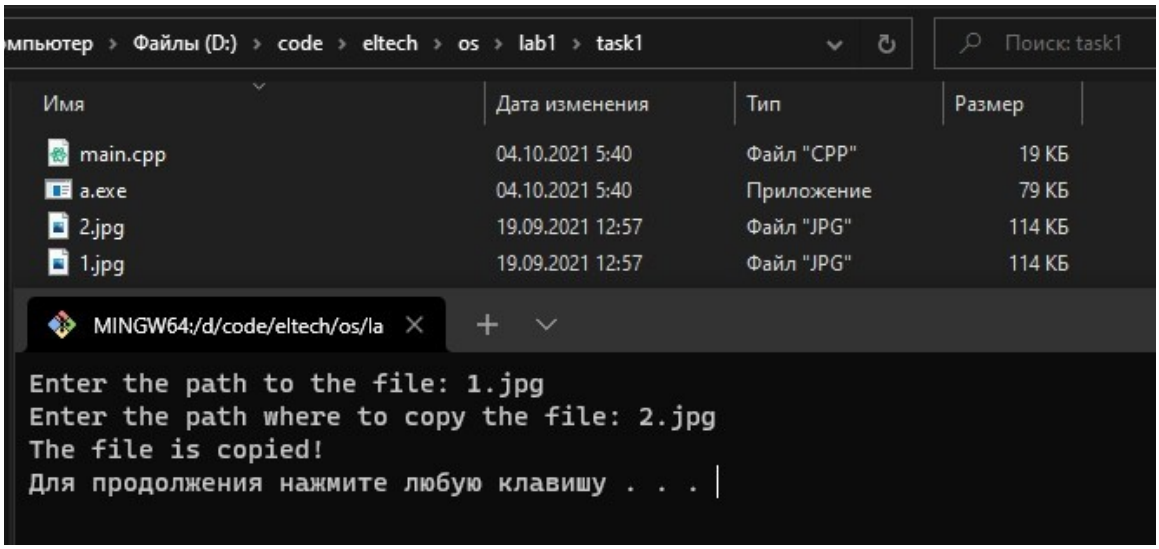

Рисунок 7: Копирование файла

8. Перемещение файла (функция MoveFile()):

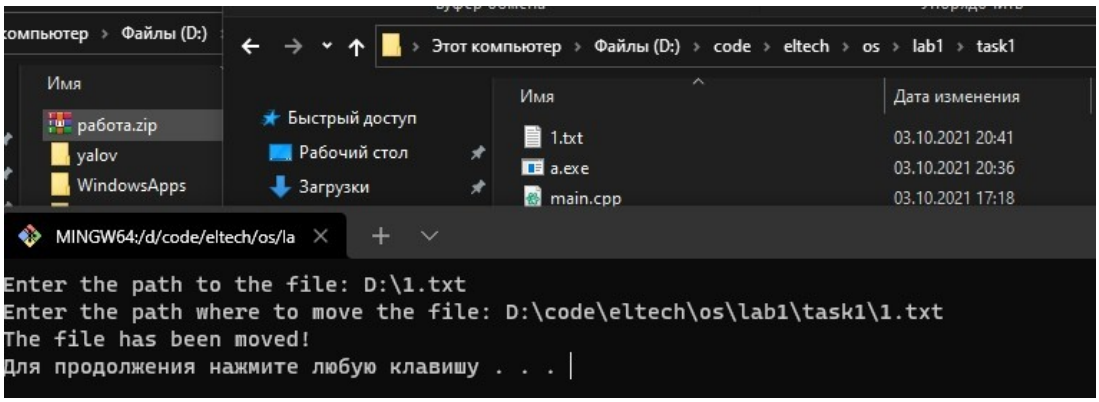

Рисунок 8: Перемещение файла

9. Получение атрибутов файла (функция GetFileAttributes()):

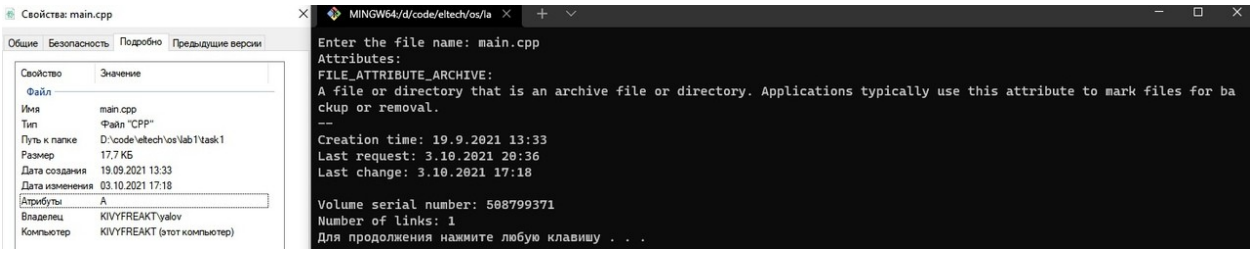

Рисунок 9: Получение аттрибутов файла

10. Установка атрибутов файла (функция SetFileAttributes()):

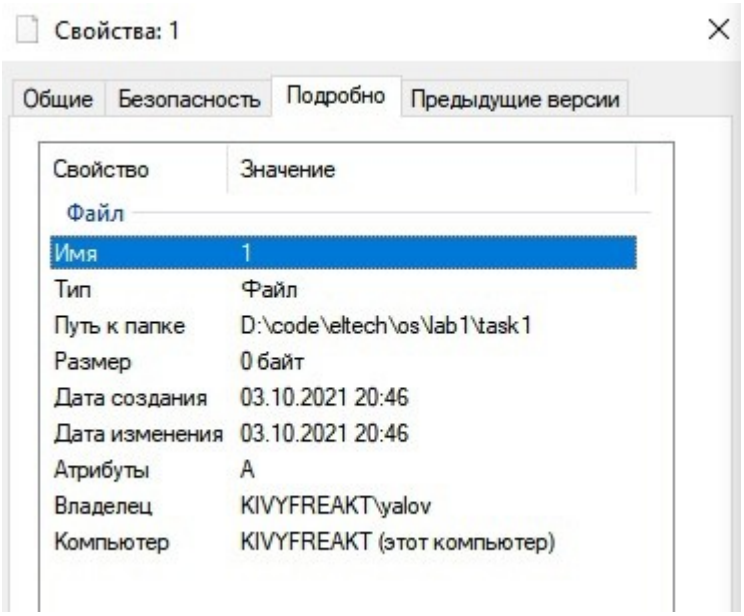

Рисунок 10: Атрибуты до

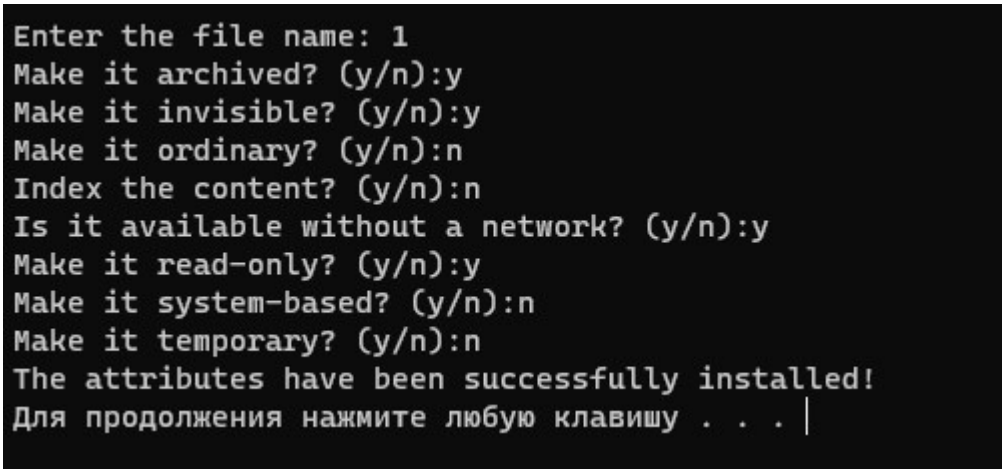

Рисунок 11: Установка атрибутов

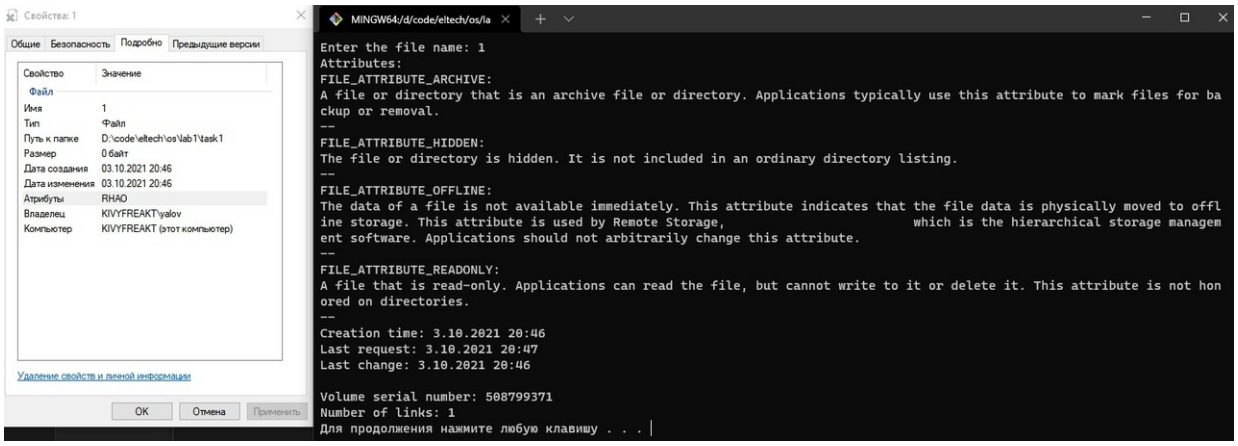

Рисунок 12: Проверка атрибутов

11. Вывод временных характеристик файла (функция GetFileTime()):

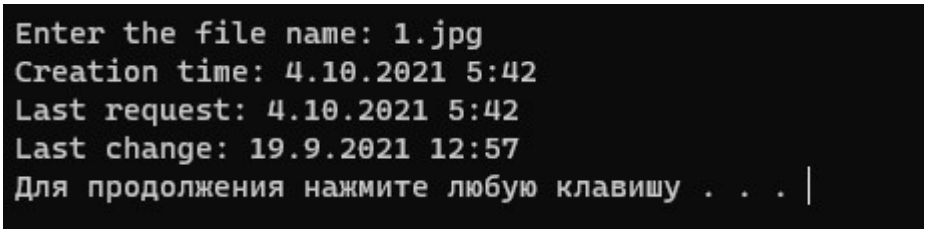

Рисунок 13: Вывод времени

12. Изменение временных характеристик файла (функция SetFileTime()), устанавливается текущее системное время:

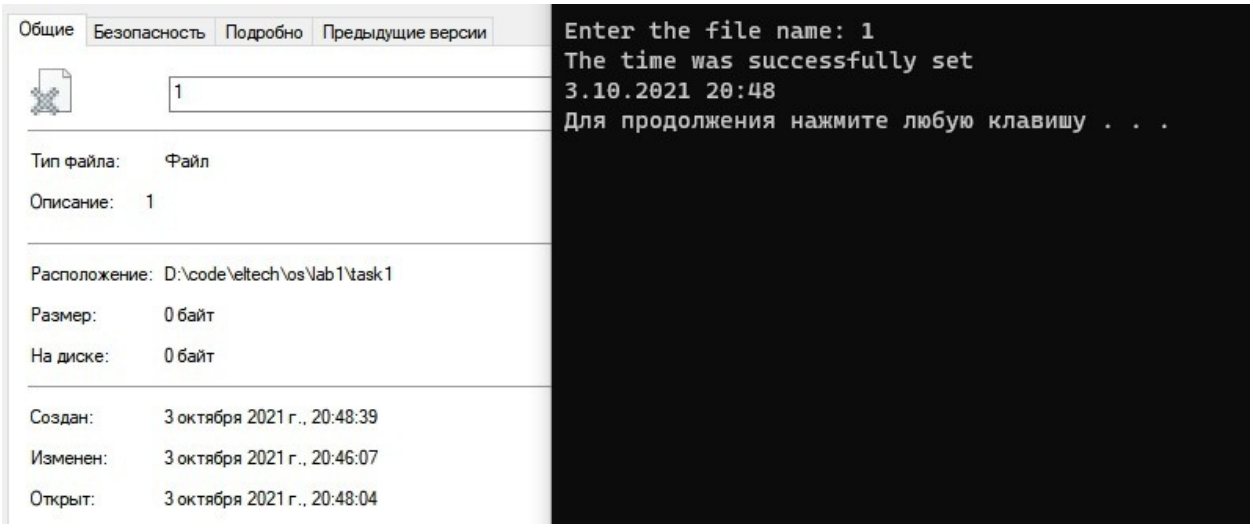

Рисунок 14: Изменение времени

2.3. Вывод

В ходе выполнения первого задания были изучены механизмы работы с файловой системой с помощью Win32 API. Использованные функции позволяют получать информацию о носителях и файлах, манипулировать файлами и директориями, а также получать доступ к атрибутам файловой системы.

3. Копирование файла с помощью перекрывающегося ввода-вывода

3.1. Указания к выполнению

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).

Ссылка на репозиторий с исходным кодом второй части работы: https://github.com/kivyfreakt/eltech/tree/master/os/lab1/task2

3.2. Результаты выполнения программы

1. Проверка работоспособности программы

Для проверки:

- был скопирован файл объёмом 5.8 ГБ;
- был выбран размер блока – 2000;
- были использованы 16 перекрывающих операций.

Результаты проверки приведены ниже:

```
$ ./a.exe
Enter size of the block:
> 4096*2000
Enter number of the operations:
> 16
Enter the directory of first file:
> stress_test
Enter the directory of second file:
> stress_test_copy
File copied successfully. Copy time: 142547 milliseconds

yalov@kivyfreakt MINGW64 /d/code/eltech/os/lab1/task2 (master)
$ Certutil -hashfile stress_test
Хэш SHA1 stress_test:
f33411e9f452bc745f59b661572e76498d8e65bd
CertUtil: -hashfile — команда успешно выполнена.

yalov@kivyfreakt MINGW64 /d/code/eltech/os/lab1/task2 (master)
$ Certutil -hashfile stress_test_copy
Хэш SHA1 stress_test_copy:
f33411e9f452bc745f59b661572e76498d8e65bd
CertUtil: -hashfile — команда успешно выполнена.
```

Рисунок 15: Проверка работоспособности

Для проверки идентичности файлов сравнили их хеши. Отличий между хешами исходного и скопированного файла нет, что означает, что программа работает корректно.

2. Анализ оптимального размера блока

Условия проведения эксперимента

Для того, чтобы найти оптимальный размер блока, была использована 1 перекрывающаяся операция, размер блока увеличивался с каждым запуском программы.

Для проверки было выбрано два файла размерами 150 МБ и 1.3ГБ

При проведении эксперимента возникают случайные погрешности, неизбежно возникающие в работающей операционной системе, которая может начать процесс обновления или оптимизации, не уведомляя пользователя. Для уменьшения влияния случайных факторов, будем проводить не один, а сразу K экспериментов (для всех экспериментов использовалось значение 5) с программой, не меняя исходные данные. Получается K замеров времени, которые в общем случае будут различными вследствие различных случайных факторов, влияющих на проводимый эксперимент. Будем принимать результирующее время как среднее арифметическое K замеров времени. Кроме того, система минимально нагружена дополнительными процессами.

Результат эксперимента

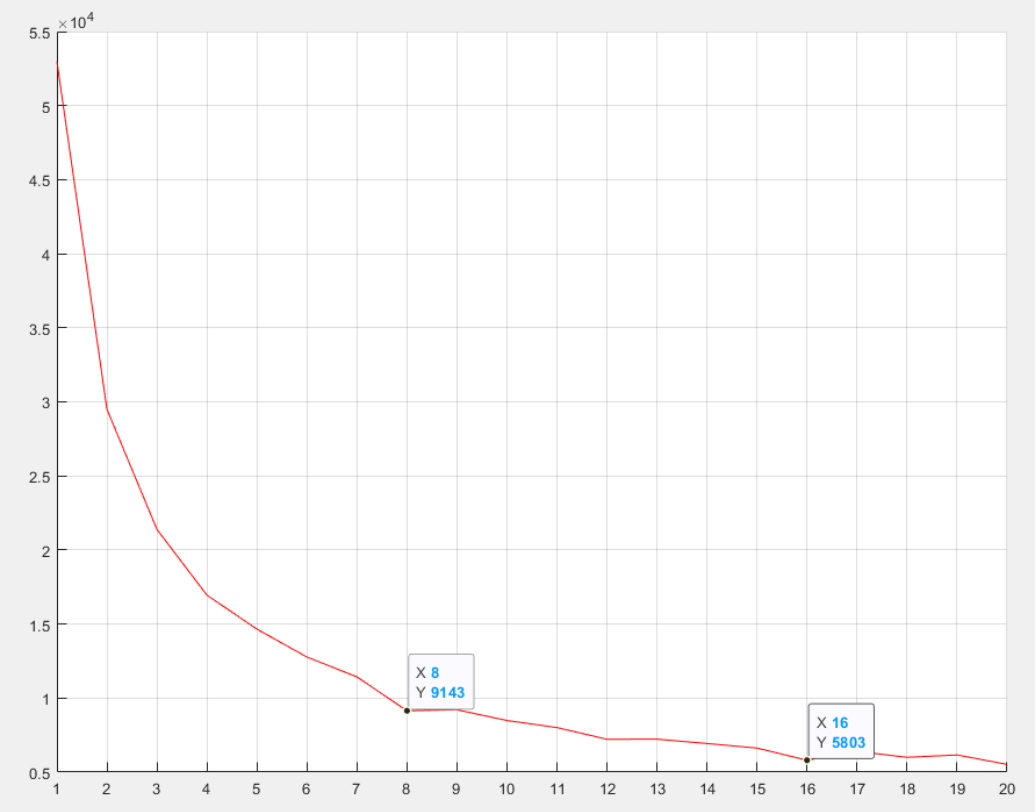

Рисунок 16: Для файла размером 150МБ

Из графика видно, что размер блока обратно пропорционален времени копирования: чем больше размер блока, тем меньше время копирования. Минимальное время было достигнуто при значении множителя 16. Также можно заметить, что после множителя 8, скорость изменения времени уменьшается. Вероятно, оптимальным будет размер блока равный размеру файла, разделенному на количество перекрывающих операций.

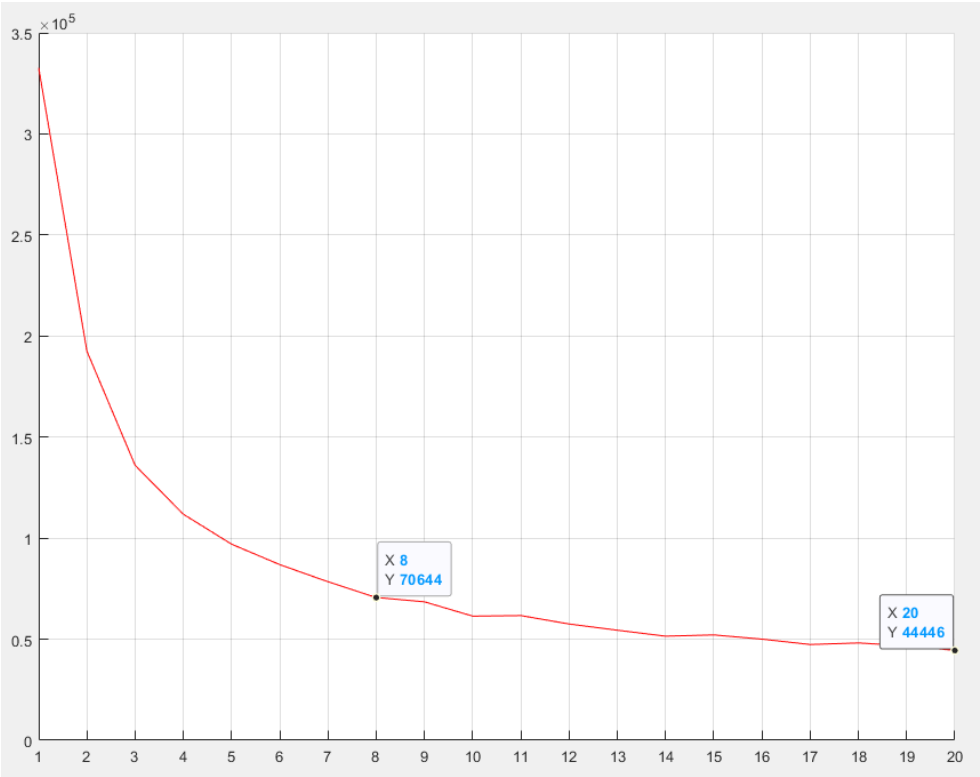

Рисунок 17: Для файла 1.3ГБ

Из графика видно, что размер блока обратно пропорционален времени копирования: чем больше размер блока, тем меньше время копирования. Зависимость логарифмическая 1/log(x). Минимальное время было достигнуто при значении множителя 20. Также можно заметить, что после множителя 8, скорость изменения времени уменьшается. Вероятно, оптимальным будет размер блока равный размеру файла, разделенному на количество перекрывающих операций.

3. Анализ оптимального количества перекрывающих операций

Условия проведения эксперимента

Для поиска оптимального количества перекрывающих операций возьмём размер блока, равный 4096 байт$*8 \approx 32$ Мбайт.

Для проверки было выбрано два файла размерами 150 МБ и 1.3ГБ

При проведении эксперимента возникают случайные погрешности, неизбежно возникающие в работающей операционной системе, которая может начать процесс обновления или оптимизации, не уведомляя пользователя. Для уменьшения влияния случайных факторов, будем проводить не один, а сразу K экспериментов (для всех экспериментов использовалось значение 5) с программой, не меняя исходные данные. Получается K замеров времени, которые в общем случае будут различными вследствие различных случайных факторов, влияющих на проводимый эксперимент. Будем принимать результирующее время как среднее арифметическое K замеров времени. Кроме того, система минимально нагружена дополнительными процессами. Минимальное значение времени при величине количества операции равной 2.

Результат эксперимента

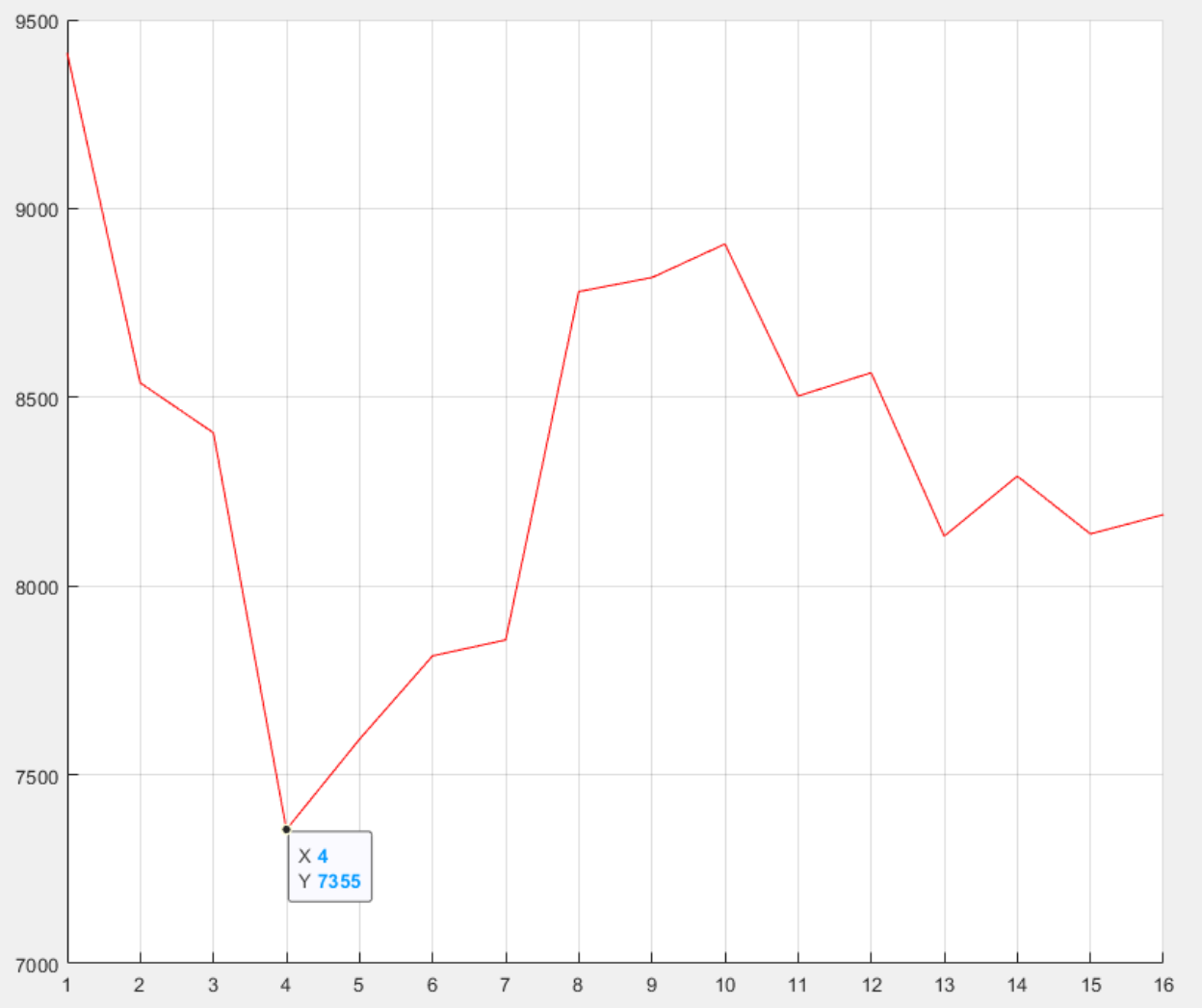

Рисунок 18: Для файла 1.5МБ

Судя по данному графику можно сказать, что прямой зависимости между количеством операций ввода/вывода нет, и оптимальное количество составляет 4 операции, далее время увеличивается. Это можно объяснить тем, что блоков для копирования оказывается меньше, чем операций, которые хотят осуществить копирование. Неким образом, они скорее мешают друг другу.

Рисунок 19: Для файла 1.3ГБ

Судя по данному графику можно сказать, что увеличение числа количества операций снижает время на копирование файла, но с большим увеличением прирост ощущается не так сильно. Оптимальное количество составляет 16 операции, далее прирост незначительный.

3.3. Вывод

В ходе выполнения данного задания был изучен механизм асинхронного копирования файла. При этом использовалось переменное количество блоков, участвующих в копировании, и переменное количество перекрывающих операций ввода/вывода.

В процессе выполнения получены результаты, исходя из которых можно сделать выводы:

- изменение параметров заметно сказывается на скорости копирования;
- при асинхронном копировании при увеличении размера копируемого блока увеличивается скорость копирования файла;
- оптимальное количество операций перекрывающего ввода/вывода зависит от файла, для маленького(150МБ) — 4, для большого(1.3 ГБ) — 16 .