**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. 2305 |  | Макурин А. Е. |
| Преподаватель |  | Тимофеев А. В. |

Санкт-Петербург

2024

ОГЛАВЛЕНИЕ

[ЦЕЛЬ РАБОТЫ 3](#_Toc161086368)

[ЗАДАНИЯ НА ЛАБОРАТОРНУЮ РАБОТУ 4](#_Toc161086369)

[Задание 1.1. Управление дисками, каталогами и файлами. 4](#_Toc161086370)

[Задание 1.2. Копирование файла с помощью операций перекрывающегося ввода-вывода. 4](#_Toc161086371)

[ЗАДАНИЕ 1.1. УПРАВЛЕНИЕ ДИСКАМИ, КАТАЛОГАМИ И ФАЙЛАМИ 6](#_Toc161086372)

[Главное меню. 6](#_Toc161086373)

[Выполнение первой функции главного меню. 6](#_Toc161086374)

[Выполнение третьей функции главного меню. 7](#_Toc161086375)

[Удаление каталога. 7](#_Toc161086376)

[Выполнение четвёртой функции главного меню. 8](#_Toc161086377)

[Выполнение пятой функции главного меню. 8](#_Toc161086378)

[Выполнение шестой функции главного меню. 9](#_Toc161086379)

[Вывод задания 1.1. 10](#_Toc161086380)

[ЗАДАНИЕ 1.2. КОПИРОВАНИЕ ФАЙЛА С ПОМОЩЬЮ ОПЕРАЦИЙ ПЕРЕКРЫВАЮЩЕГОСЯ ВВОДА-ВЫВОДА 11](#_Toc161086381)

[Первый опыт. 11](#_Toc161086382)

[Второй опыт. 12](#_Toc161086383)

[Вывод задания 1.2. 14](#_Toc161086384)

[ВЫВОД 15](#_Toc161086385)

[ПРИЛОЖЕНИЯ 16](#_Toc161086386)

[Приложение 1 для работы с Windows API: 16](#_Toc161086387)

[Приложение 2 для работы с прерывающимися операциями ввода-вывода: 30](#_Toc161086388)

# ЦЕЛЬ РАБОТЫ

Исследование управления файловой системой с помощью Win32 API и прерывающимися операциями ввода-вывода Linux.

# ЗАДАНИЯ НА ЛАБОРАТОРНУЮ РАБОТУ

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

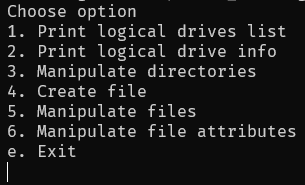
1. Создать консольное приложение с меню (каждая выполняемая функция и/или операция должна быть доступна по отдельному пункту меню), которое выполняет:
   1. Вывод списка дисков;
   2. Для одного из выбранных дисков вывод информации о диске и размер свободного пространства;
   3. Создание и удаление заданных каталогов;
   4. Создание файлов в новых каталогах;
   5. Копирование и перемещение файлов между каталогами с возможностью выявления попытки работы с файлами, имеющие совпадающие имена;
   6. Анализ и изменение атрибутов файлов;
2. Запустить приложение и проверить его работоспособность на нескольких выборах вводимых данных. Запротоколировать результаты в отчёт. Дать комментарии в отчёте относительно выполнения функций Win32 API.
3. Подготовить итоговый отчёт с развёрнутыми выводами по заданию.

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

1. Создайте консольное приложение, которое выполняет:
   1. Открытие-создание файлов;
   2. Файловый ввод-вывод;
   3. Ожидания срабатывания вызова;
   4. Измерение продолжительности выполнения операции копирования файла;
2. 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 данных наибольшая. Запротоколировать результаты в отчёт. Дать комментарии в отчёте относительно выполнения системных вызовов Linux.
3. Провести замеры времени выполнения приложения для разного числа перекрывающихся операций ввода и вывода (1, 2, 4, 8, 12, 16), не забывая проверять работоспособность приложения. По результатам измерений построить график зависимости и определите число перекрывающихся операций ввода и вывода, при котором достигается наибольшая скорость копирования файла. Запротоколировать результаты в отчёт.
4. Подготовить отчёт с развёрнутыми выводами по заданию.

# ЗАДАНИЕ 1.1. УПРАВЛЕНИЕ ДИСКАМИ, КАТАЛОГАМИ И ФАЙЛАМИ

Главное меню. Демонстрация главного меню функций для работы с файлами показана на рисунке 1.1.

Рисунок 1.1.

Выполнение первой функции главного меню. Это функция предназначена для вывода списка дисков в системе.

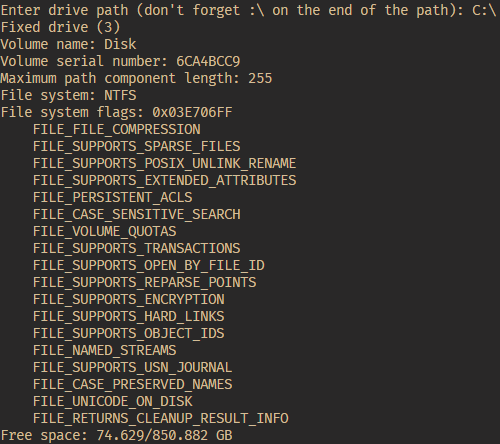
Для вывода списка дисков была использована функция GetLogicalDriveStringsW. Функция используется для получения битовой маски диска. Результат её работы изображён на рисунке 1.2.

Для получения текущей информации о логическом диске были использованы следующие функции:

* Функция GetDriveTypeW возвращает тип указанного диска;
* Функция GetVolumeInformationW возвращает информацию об имени, серийном номере, файловой системе;
* Функция GetDiskFreeSpaceEx возвращает информацию о свободном дисковом пространстве.

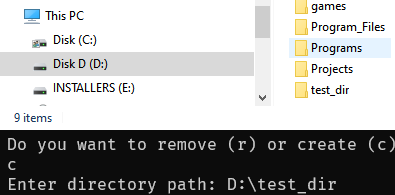
Демонстрация работы функций показаны на рисунке 1.2. Для эксперимента был выбран диск C:\\.

Рисунок 1.2.

Рисунок 1.3.

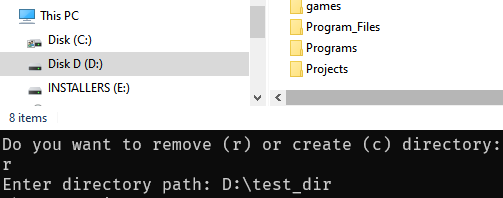
По итогу выполнения первая функция выдала практически всю информацию о диске: тип, имя, серийный номер, максимальная длина компонента пути, файловая система, флаги файловой системы, объём и оставшееся пространство.

Выполнение третьей функции главного меню. Эта функция предназначена для создания или удаления каталога по указанному пути. Это достигается путём использования функций CreateDirectoryW и RemoveDirectoryW. Результат работы функции показан на рисунке 1.4.

Рисунок 1.4.

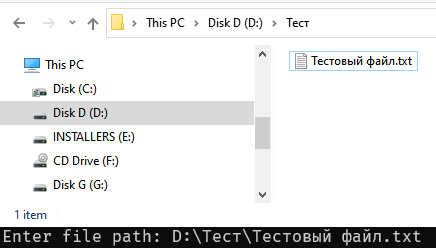
По итогу выполнения второй функции в указанной директории был создан каталог (папка).

Удаление каталога. Эта функция предназначена для удаления каталога по указанному пути. Это достигается путём использования функции RemoveDirectoryW. Результат работы показан на рисунке 1.5.

Рисунок 1.5.

По итогу выполнения третей функции в указанной директории был удалён каталог.

Выполнение четвёртой функции главного меню. Эта функция предназначена для создания файла по указанному пути. Это достигается путём использования функции CreateFileW. При создании файла, в пути, указывается его название вместе с расширением. Результат работы показан на рисунке 1.6.

Рисунок 1.6.

По итогу выполнения четвёртой функции в указанной директории был создан файл.

Выполнение пятой функции главного меню. Эта функция предназначена для копирования или перемещения файла из одного места по указанному пути в другое место по указанному пути. Это достигается путём использования функций CopyFileW и MoveFileExW. Результат показан на рисунках 1.7, 1.8 и 1.9.

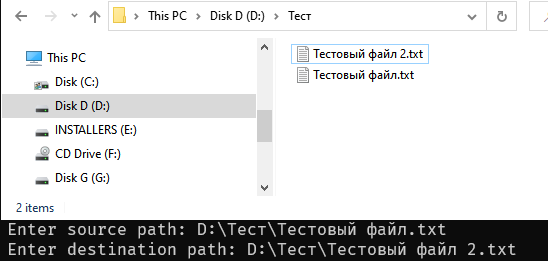
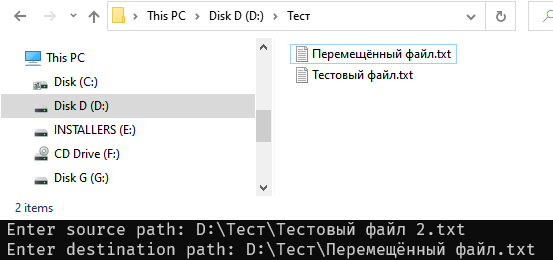
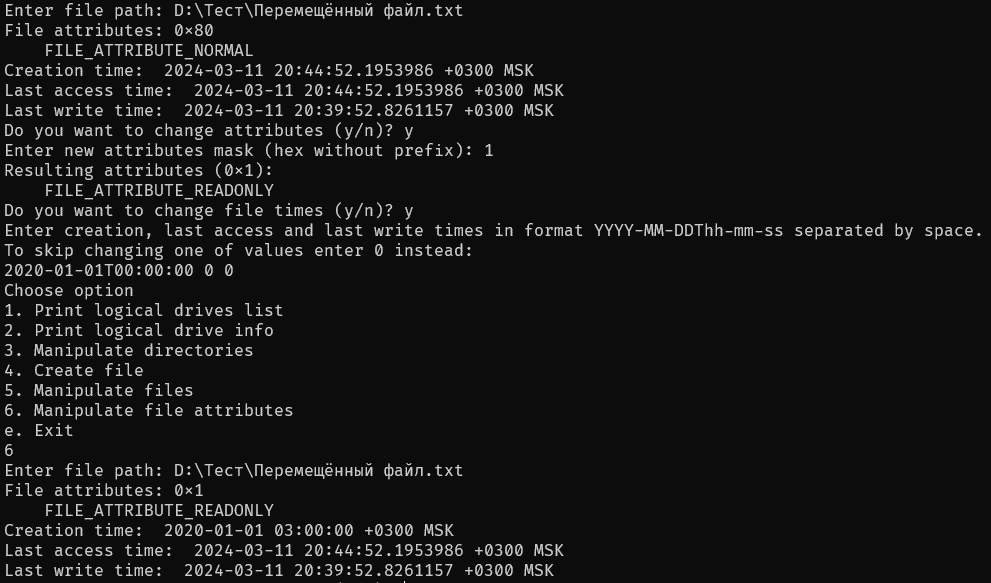
Рисунок 1.7. Копирование файла

Рисунок 1.8. Перемещение файла, если файл уже существует

Рисунок 1.9. Перемещение файла

Выполнение шестой функции главного меню. Эта функция предназначена для просмотра и изменения атрибутов и времён файла по указанному пути. Это достигается путём использования функций GetFileAttributesW, SetFileAttributesW, CreateFileW (для получения хэндла (открытия) файла), CloseHandle, GetFileTime и SetFileTime. Результат выполнения программы показан на рисунке 1.10.

Рисунок 1.10

По итогу выполнения шестой функции были изменены параметры времени и атрибуты файла. В данном примере было изменено время создания файла.

## **Вывод задания 1.1.**

В результате выполнения данного задания мы познакомились с некоторыми функциями Windows API для работы с файлами и файловой системой Windows. Данный функции имеют хороший показатель эффективности при их использовании, так как они напрямую взаимодействуют с операционной системой. Функции Windows API по своему функционалу являются очень гибкими, так как могут взаимодействовать с различными аспектами операционной системы. Также можно учесть исключительные ситуации, которые позволяют контролировать процесс выполнения функций.

Благодаря гибкости, высокой эффективности и надёжности (в плане исключительных ситуаций), которые предоставляет Windows API, позволяет разрабатывать приложения, которые будут способны эффективно работать с теми областями операционной системы, для которых это приложение и было написано.

# ЗАДАНИЕ 1.2. КОПИРОВАНИЕ ФАЙЛА С ПОМОЩЬЮ ОПЕРАЦИЙ ПЕРЕКРЫВАЮЩЕГОСЯ ВВОДА-ВЫВОДА

Для эксперимента будут взяты два файла разных размеров, а именно:

* Файл 128kb.txt величиной в 128 Кбайт или 131 072 байт;
* Файл 1G.txt величиной в 1 Гбайт или 1 073 741 824 байт;

Файлы заполнены случайным набором байт посредством копирования из /dev/urandom.

Проверим, что при копировании нет потери данных. Для этого сравним контрольные суммы исходного и итогового файлов. Как видно из результатов (Рис. 2.1) потери данных нет.

Рис 2.1.

Первый опыт. Копирование файла при использовании различного количества перекрывающихся операций ввода-вывода при различных значениях блока данных на NTFS диске, доступ к которому получаем из WSL (диск C ОС Windows).

График 2.1. изображает зависимость скорости от размера блока данных и количества операций при копировании файла 128kb.txt.

График 2.1.

Можно отметить, что наиболее эффективным размером блока данных (кластера) является 131072 — т. е. тот же размер, что и размер самого файла. При этом при количестве операций от 8 и более время выполнения не уменьшается.

График 2.2. изображает зависимость скорости от размера блока данных и количества операций при копировании файла 1G.txt.

График 2.2.

Можно отметить, что наиболее эффективным размером блока данных (кластера) является 131072 — т. е. тот же размер, что и размер самого файла. При этом значения времени при двух и более операциях почти идентичны.

Второй опыт. Копирование файла при использовании различного количества перекрывающихся операций ввода-вывода при различных значениях блока данных в домашней папке пользователя Linux.

График 2.3. изображает зависимость скорости от размера блока данных и количества операций при копировании файла 128kb.txt.

График 2.3.

График 2.4. изображает зависимость скорости от размера блока данных и количества операций при копировании файла 1G.txt.

График 2.2.

Вывод задания 1.2. Исходя из результатов первого опыта, можно отметить, что увеличение размера блока данных приводит к уменьшению времени, затрачиваемого на копирование. Для файла размером в 128 Кб самым оптимальным вариантом был размер блока данных равный 130 072 байт. В то время, как для файла размером в 1 Гб оптимальным размером блока данных был 1 073 741 824 байт. Изменение же количества операций не столь заметно уменьшило время копирования.

Исходя из этого можно отметить, для файла определённого размера необходимо подбирать значения собственные значения блока данных и операций ввода-вывода, для нахождения самого оптимального варианта, при котором копирование будет занимать меньше времени. Эти параметры зависят от количества потоков процессора и свободного объёма оперативной памяти, а также размера файла. При этом не всегда получится использовать размер блока равный размеру файла, так как от этого зависит количество потребляемой памяти.

# ВЫВОД

В результате лабораторной работы были рассмотрены функции, предоставляющие возможность взаимодействия с файловой системой операционной системы.

В первой части лабораторной работы были разобраны функции Windows API для взаимодействия с файловой системой Windows. Как было отмечено ранее, эти функции обладают высокой эффективностью, так как напрямую работают с операционной системой, хорошей гибкостью, так как подходит под различные ситуации и при возможности свойства функции могут меняться для выполнения абсолютно других задач. И исключения, которые позволяют держать приложение под контролем, обходя все исключительные ситуации для стабильной работы приложения.

Во второй части лабораторной работы были рассмотрены системные вызовы асинхронных чтения и записи операционной системы Ubuntu, дистрибутиве операционной системы Linux. Они были нужны для разработки способа копирования файлов системы с помощью прерывающихся (асинхронных) операций ввода-вывода. Исходя из проведённого опыта, мы поняли, что скорость копирования данным методом зависит от количества операций ввода-вывода, а также от размера блока данных. Однако для файлов определённого размера должны быть выставлены индивидуальные настройки, как как недобор в настройках блока данных и операциях ввода-вывода могут значительно сказаться на скорости копирования. А перебор привести к переполнению оперативной памяти и ошибке копирования.

# ПРИЛОЖЕНИЯ

## Приложение 1 для работы с Windows API:

main.go

package main

import (

    "fmt"

)

const STRING\_BUFFER\_SIZE = 1024

type VolumeInformation struct {

    volumeNameBuffer       [STRING\_BUFFER\_SIZE]uint16

    volumeNameSerialNumber uint32

    maximumComponentLength uint32

    fileSystemFlags        uint32

    fileSystemNameBuffer   [STRING\_BUFFER\_SIZE]uint16

}

func main() {

    var choice string

    for choice != "exit" && choice != "e" {

        fmt.Println("Choose option")

        fmt.Println("1. Print logical drives list")

        fmt.Println("2. Print logical drive info")

        fmt.Println("3. Manipulate directories")

        fmt.Println("4. Create file")

        fmt.Println("5. Manipulate files")

        fmt.Println("6. Manipulate file attributes")

        fmt.Println("e. Exit")

        \_, err := fmt.Scanln(&choice)

        if err != nil {

            fmt.Println(err.Error())

        }

        switch choice {

        case "1":

            DrivesList()

        case "2":

            DriveInfo()

        case "3":

            ManipulateDirectories()

        case "4":

            CreateFile()

        case "5":

            ManipulateFiles()

        case "6":

            ManipulateFileAttributes()

        }

    }

}

create\_file.go

package main

import (

    "bufio"

    "fmt"

    "os"

    "strings"

    "golang.org/x/sys/windows"

)

func CreateFile() {

    fmt.Print("Enter file path: ")

    var filePath string

    reader := bufio.NewReader(os.Stdin)

    filePath, \_ = reader.ReadString('\n')

    filePath = strings.TrimRight(filePath, "\r\n")

    utf16path, err := windows.UTF16FromString(filePath)

    if err != nil {

        fmt.Println("Error converting path to utf16:", err.Error())

        return

    }

    handle, err := windows.CreateFile(

        &utf16path[0],

        0,

        0,

        nil,

        windows.CREATE\_NEW,

        windows.FILE\_ATTRIBUTE\_NORMAL,

        windows.Handle(windows.GetShellWindow()),

    )

    if err != nil {

        fmt.Println("Error creating file:", err.Error())

        return

    }

    windows.CloseHandle(handle)

}

drive\_info.go

package main

import (

    "fmt"

    "unicode/utf16"

    "golang.org/x/sys/windows"

)

func DriveInfo() {

    var drivePath string

    fmt.Print("Enter drive path (don't forget :\\ on the end of the path): ")

    fmt.Scanln(&drivePath)

    uint16DrivePath, err := windows.UTF16PtrFromString(drivePath)

    if err != nil {

        fmt.Println("Error converting string to utf16:", err.Error())

        return

    }

    if driveType(uint16DrivePath) == 1 {

        return

    }

    volumeInfo(uint16DrivePath)

    totalNumberOfFreeBytes, totalNumberOfBytes := driveFreeSpace(

        uint16DrivePath,

    )

    fmt.Printf(

        "Free space: %.3f/%.3f GB\n",

        float64(totalNumberOfFreeBytes)/(1<<30),

        float64(totalNumberOfBytes)/(1<<30),

    )

}

func driveType(drivePath \*uint16) int {

    types := []string{

        "Unknown drive type (0)",

        "Wrong path (1)",

        "Removable drive (2)",

        "Fixed drive (3)",

        "Remote drive (4)",

        "CD-Rom (5)",

        "RAM-Disk (6)",

    }

    driveType := windows.GetDriveType(drivePath)

    if driveType >= 0 && int(driveType) <= len(types) {

        fmt.Println(types[driveType])

    } else {

        fmt.Printf("Unknown drive type (%d)", driveType)

    }

    return int(driveType)

}

func volumeInfo(drivePath \*uint16) VolumeInformation {

    var info VolumeInformation

    err := windows.GetVolumeInformation(

        drivePath,

        &info.volumeNameBuffer[0],

        uint32(STRING\_BUFFER\_SIZE),

        &info.volumeNameSerialNumber,

        &info.maximumComponentLength,

        &info.fileSystemFlags,

        &info.fileSystemNameBuffer[0],

        STRING\_BUFFER\_SIZE,

    )

    if err != nil {

        fmt.Println("Error getting drive info:", err.Error())

    }

    fmt.Println("Volume name:", string(utf16.Decode(info.volumeNameBuffer[:])))

    fmt.Printf("Volume serial number: %X\n", info.volumeNameSerialNumber)

    fmt.Println("Maximum path component length:", info.maximumComponentLength)

    fmt.Println(

        "File system:",

        string(utf16.Decode(info.fileSystemNameBuffer[:])),

    )

    fileSystemFlags := fmt.Sprintf("%X", info.fileSystemFlags)

    fileSystemFlags = "0x" + "00000000"[:8-len(fileSystemFlags)] + fileSystemFlags

    fmt.Println("File system flags:", fileSystemFlags)

    FLAGS\_MAP := map[int]string{

        0x00000001: "FILE\_CASE\_SENSITIVE\_SEARCH",

        0x00000002: "FILE\_CASE\_PRESERVED\_NAMES",

        0x00000004: "FILE\_UNICODE\_ON\_DISK",

        0x00000008: "FILE\_PERSISTENT\_ACLS",

        0x00000010: "FILE\_FILE\_COMPRESSION",

        0x00000020: "FILE\_VOLUME\_QUOTAS",

        0x00000040: "FILE\_SUPPORTS\_SPARSE\_FILES",

        0x00000080: "FILE\_SUPPORTS\_REPARSE\_POINTS",

        0x00000100: "FILE\_SUPPORTS\_REMOTE\_STORAGE",

        0x00000200: "FILE\_RETURNS\_CLEANUP\_RESULT\_INFO",

        0x00000400: "FILE\_SUPPORTS\_POSIX\_UNLINK\_RENAME",

        0x00008000: "FILE\_VOLUME\_IS\_COMPRESSED",

        0x00010000: "FILE\_SUPPORTS\_OBJECT\_IDS",

        0x00020000: "FILE\_SUPPORTS\_ENCRYPTION",

        0x00040000: "FILE\_NAMED\_STREAMS",

        0x00080000: "FILE\_READ\_ONLY\_VOLUME",

        0x00100000: "FILE\_SEQUENTIAL\_WRITE\_ONCE",

        0x00200000: "FILE\_SUPPORTS\_TRANSACTIONS",

        0x00400000: "FILE\_SUPPORTS\_HARD\_LINKS",

        0x00800000: "FILE\_SUPPORTS\_EXTENDED\_ATTRIBUTES",

        0x01000000: "FILE\_SUPPORTS\_OPEN\_BY\_FILE\_ID",

        0x02000000: "FILE\_SUPPORTS\_USN\_JOURNAL",

        0x04000000: "FILE\_SUPPORTS\_INTEGRITY\_STREAMS",

        0x08000000: "FILE\_SUPPORTS\_BLOCK\_REFCOUNTING",

        0x10000000: "FILE\_SUPPORTS\_SPARSE\_VDL",

        0x20000000: "FILE\_DAX\_VOLUME",

        0x40000000: "FILE\_SUPPORTS\_GHOSTING",

    }

    print\_flags := func(flags uint32) {

        for key, value := range FLAGS\_MAP {

            if int(flags)&key != 0 {

                fmt.Println("   ", value)

            }

        }

    }

    print\_flags(info.fileSystemFlags)

    return info

}

func driveFreeSpace(

    drivePath \*uint16,

) (totalNumberOfFreeBytes uint64, totalNumberOfBytes uint64) {

    var freeBytesAvailableToCaller uint64

    err := windows.GetDiskFreeSpaceEx(

        drivePath,

        &freeBytesAvailableToCaller,

        &totalNumberOfBytes,

        &totalNumberOfFreeBytes,

    )

    if err != nil {

        fmt.Println("Error getting drive free space:", err.Error())

    }

    return

}

drives\_list.go

package main

import (

    "fmt"

    "strings"

    "unicode/utf16"

    "golang.org/x/sys/windows"

)

func DrivesList() {

    var buffer = [STRING\_BUFFER\_SIZE]uint16{}

    \_, err := windows.GetLogicalDriveStrings(STRING\_BUFFER\_SIZE, &buffer[0])

    if err != nil {

        fmt.Println("Error getting logical drives:", err.Error())

        return

    }

    fmt.Println(

        "Logical drives list:",

        strings.Join(

            strings.FieldsFunc(

                strings.TrimRight(

                    string(utf16.Decode(buffer[:])),

                    string(rune(0)),

                ),

                func(r rune) bool { return r == 0 },

            ), " "),

    )

}

manipulate\_directories.go

package main

import (

    "bufio"

    "fmt"

    "os"

    "strings"

    "golang.org/x/sys/windows"

)

func ManipulateDirectories() {

    fmt.Println("Do you want to remove (r) or create (c) directory: ")

    var choice string

    for choice != "r" && choice != "c" {

        fmt.Scanln(&choice)

    }

    fmt.Print("Enter directory path: ")

    reader := bufio.NewReader(os.Stdin)

    var directoryPath string

    directoryPath, \_ = reader.ReadString('\n')

    directoryPath = strings.TrimRight(directoryPath, "\n\r")

    utf16Path := windows.StringToUTF16(directoryPath)

    switch choice {

    case "c":

        err := windows.CreateDirectory(&utf16Path[0], nil)

        if err != nil {

            fmt.Println("Error creating directory:", err.Error())

        }

    case "r":

        err := windows.RemoveDirectory(&utf16Path[0])

        if err != nil {

            fmt.Println("Error removing directory:", err.Error())

        }

    }

}

manipulate\_file\_attributes.go

package main

import (

    "bufio"

    "encoding/hex"

    "fmt"

    "os"

    "strings"

    "syscall"

    "time"

    "golang.org/x/sys/windows"

)

func ManipulateFileAttributes() {

    fmt.Print("Enter file path: ")

    var filePath string

    reader := bufio.NewReader(os.Stdin)

    filePath, \_ = reader.ReadString('\n')

    filePath = strings.TrimRight(filePath, "\r\n")

    utf16filePath, err := windows.UTF16FromString(filePath)

    if err != nil {

        fmt.Println("Error converting file path:", err.Error())

        return

    }

    attributes, err := windows.GetFileAttributes(&utf16filePath[0])

    if err != nil {

        fmt.Println("Error getting file attributes:", err.Error())

        return

    }

    ATTRIBUTES\_MAP := map[int]string{

        0x00000001: "FILE\_ATTRIBUTE\_READONLY",

        0x00000002: "FILE\_ATTRIBUTE\_HIDDEN",

        0x00000004: "FILE\_ATTRIBUTE\_SYSTEM",

        0x00000010: "FILE\_ATTRIBUTE\_DIRECTORY",

        0x00000020: "FILE\_ATTRIBUTE\_ARCHIVE",

        0x00000040: "FILE\_ATTRIBUTE\_DEVICE",

        0x00000080: "FILE\_ATTRIBUTE\_NORMAL",

        0x00000100: "FILE\_ATTRIBUTE\_TEMPORARY",

        0x00000200: "FILE\_ATTRIBUTE\_SPARSE\_FILE",

        0x00000400: "FILE\_ATTRIBUTE\_REPARSE\_POINT",

        0x00000800: "FILE\_ATTRIBUTE\_COMPRESSED",

        0x00001000: "FILE\_ATTRIBUTE\_OFFLINE",

        0x00002000: "FILE\_ATTRIBUTE\_NOT\_CONTENT\_INDEXED",

        0x00004000: "FILE\_ATTRIBUTE\_ENCRYPTED",

        0x00008000: "FILE\_ATTRIBUTE\_INTEGRITY\_STREAM",

        0x00010000: "FILE\_ATTRIBUTE\_VIRTUAL",

        0x00020000: "FILE\_ATTRIBUTE\_NO\_SCRUB\_DATA",

        0x00080000: "FILE\_ATTRIBUTE\_PINNED",

        0x00100000: "FILE\_ATTRIBUTE\_UNPINNED",

        0x00040000: "FILE\_ATTRIBUTE\_RECALL\_ON\_OPEN",

        0x00400000: "FILE\_ATTRIBUTE\_RECALL\_ON\_DATA\_ACCESS",

    }

    print\_attributes := func(attributes uint32) {

        for key, value := range ATTRIBUTES\_MAP {

            if int(attributes)&key != 0 {

                fmt.Println("   ", value)

            }

        }

    }

    fmt.Printf("File attributes: 0x%X\n", attributes)

    print\_attributes(attributes)

    get\_file\_handle := func(access\_rights uint32) (windows.Handle, error) {

        return windows.CreateFile(

            &utf16filePath[0],

            access\_rights,

            windows.FILE\_SHARE\_READ,

            nil,

            windows.OPEN\_EXISTING,

            windows.FILE\_FLAG\_BACKUP\_SEMANTICS,

            windows.Handle(windows.GetShellWindow()),

        )

    }

    handle, err := get\_file\_handle(windows.GENERIC\_READ)

    if err != nil {

        fmt.Println("Error opening file:", err.Error())

        return

    }

    var (

        creationTimestamp   windows.Filetime

        lastAccessTimestamp windows.Filetime

        lastWriteTimestamp  windows.Filetime

    )

    err = windows.GetFileTime(

        handle,

        &creationTimestamp,

        &lastAccessTimestamp,

        &lastWriteTimestamp,

    )

    if err != nil {

        fmt.Println("Error reading file times:", err.Error())

        return

    }

    creationTime, lastAccessTime, lastWriteTime := time.Unix(

        0,

        creationTimestamp.Nanoseconds(),

    ), time.Unix(

        0,

        lastAccessTimestamp.Nanoseconds(),

    ), time.Unix(

        0,

        lastWriteTimestamp.Nanoseconds(),

    )

    fmt.Println("Creation time: ", creationTime)

    fmt.Println("Last access time: ", lastAccessTime)

    fmt.Println("Last write time: ", lastWriteTime)

    var choice string

    windows.CloseHandle(handle)

    fmt.Print("Do you want to change attributes (y/n)? ")

    fmt.Scanln(&choice)

    if choice == "y" {

        fmt.Print("Enter new attributes mask (hex without prefix): ")

        var hex\_attributes string

        fmt.Scanln(&hex\_attributes)

        if len(hex\_attributes)%2 == 1 {

            hex\_attributes = "0" + hex\_attributes

        }

        bytes\_attr, err := hex.DecodeString(hex\_attributes)

        if err != nil {

            fmt.Println("Error! Malformed hex string:", err.Error())

            return

        }

        attributes = 0

        for i, val := range bytes\_attr {

            attributes |= uint32(val) << ((len(bytes\_attr) - i - 1) \* 8)

        }

        fmt.Printf("Resulting attributes (0x%X):\n", attributes)

        print\_attributes(attributes)

        err = windows.SetFileAttributes(&utf16filePath[0], attributes)

        if err != nil {

            fmt.Println("Error changing file attributes:", err.Error())

            return

        }

    }

    fmt.Print("Do you want to change file times (y/n)? ")

    fmt.Scanln(&choice)

    if choice == "y" {

        handle, err = get\_file\_handle(windows.FILE\_WRITE\_ATTRIBUTES)

        if err != nil {

            fmt.Println("Error opening file:", err.Error())

            return

        }

        fmt.Println(

            "Enter creation, last access and last write times in format YYYY-MM-DDThh-mm-ss separated by space.",

        )

        fmt.Println("To skip changing one of values enter 0 instead:")

        timeStrings := make([]string, 3)

        \_, err = fmt.Scanln(&timeStrings[0], &timeStrings[1], &timeStrings[2])

        if err != nil {

            fmt.Println("Error! Wrong times count:", err.Error())

            windows.CloseHandle(handle)

            return

        }

        Filetime := func(t time.Time) windows.Filetime {

            syscallFiletime := syscall.NsecToFiletime(t.UnixNano())

            return windows.Filetime{

                LowDateTime:  syscallFiletime.LowDateTime,

                HighDateTime: syscallFiletime.HighDateTime,

            }

        }

        timestamps := []\*windows.Filetime{

            &creationTimestamp,

            &lastAccessTimestamp,

            &lastWriteTimestamp,

        }

        for i, timeString := range timeStrings {

            if timeString != "0" {

                tempTime, err := time.Parse("2006-01-02T15:04:05", timeString)

                if err == nil {

                    \*timestamps[i] = Filetime(tempTime)

                } else {

                    fmt.Printf("Wrong time format: %s\n", timeString)

                }

            }

        }

        err = windows.SetFileTime(

            handle,

            &creationTimestamp,

            &lastAccessTimestamp,

            &lastWriteTimestamp,

        )

        windows.CloseHandle(handle)

        if err != nil {

            fmt.Println("Error changing file times:", err.Error())

            return

        }

    }

}

manipulate\_files.go

package main

import (

    "bufio"

    "fmt"

    "os"

    "strings"

    "syscall"

    "unsafe"

    "golang.org/x/sys/windows"

)

func ManipulateFiles() {

    fmt.Print("Do you want to copy (c) or move (m) file: ")

    var choice, sourcePath, destinationPath string

    for choice != "c" && choice != "m" {

        fmt.Scanln(&choice)

    }

    fmt.Print("Enter source path: ")

    reader := bufio.NewReader(os.Stdin)

    sourcePath, \_ = reader.ReadString('\n')

    sourcePath = strings.TrimRight(sourcePath, "\r\n")

    fmt.Print("Enter destination path: ")

    destinationPath, \_ = reader.ReadString('\n')

    destinationPath = strings.TrimRight(destinationPath, "\r\n")

    utf16sourcePath, err := windows.UTF16FromString(sourcePath)

    if err != nil {

        fmt.Println("Error encoding source file path:", err.Error())

        return

    }

    utf16destinationPath, err := windows.UTF16FromString(destinationPath)

    if err != nil {

        fmt.Println("Error encoding destination file path:", err.Error())

        return

    }

    switch choice {

    case "c":

        procCopyFile := syscall.NewLazyDLL("kernel32.dll").NewProc("CopyFileW")

        r1, \_, err := syscall.SyscallN(

            procCopyFile.Addr(),

            uintptr(unsafe.Pointer(&utf16sourcePath[0])),

            uintptr(unsafe.Pointer(&utf16destinationPath[0])),

            uintptr(1),

        )

        if r1 == 0 {

            fmt.Println("Error copying file:", err.Error())

        }

    case "m":

        err = windows.MoveFileEx(

            &utf16sourcePath[0],

            &utf16destinationPath[0],

            windows.MOVEFILE\_COPY\_ALLOWED,

        )

        if err != nil {

            fmt.Println("Error moving file:", err.Error())

        }

    }

}

## Приложение 2 для работы с прерывающимися операциями ввода-вывода:

#include <aio.h>

#include <fcntl.h>

#include <signal.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

#include <chrono>

#include <cstdlib>

#include <cstring>

#include <filesystem>

#include <fstream>

#include <iostream>

#include <string>

#include <vector>

#define PATH\_SIZE 200

#define CLUSTER\_SIZE 512

using namespace std;

using namespace chrono;

enum OperationType { WRITE, READ };

struct AIOOperation {

    aiocb \*aio;

    char \*buffer;

    OperationType type;

    bool is\_active;

    AIOOperation();

    AIOOperation(OperationType type, off\_t cluster\_size, off\_t offset);

    ~AIOOperation();

};

struct Config {

    off\_t cluster\_size, operations\_count;

    string in\_path, out\_path;

    off\_t chunk\_size();

    Config() = default;

    Config(const Config &) = delete;

    Config(int argc, char \*argv[]);

};

int in\_file\_descriptor;

int out\_file\_descriptor;

off\_t file\_size;

Config config;

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

void init\_operations(AIOOperation \*\*&operations, int cluster);

void start\_operations(AIOOperation \*\*operations);

int write\_results(string pathRead, int file\_size, int operations\_count,

                  int cluster, int duration);

int main(int argc, char \*argv[]) {

    struct stat fileStat;

    config = Config(argc, argv);

    if (config.chunk\_size() < 0) {

        cerr << "Error: too big chunk size (product of operations count and "

                "cluster "

                "size)"

             << endl;

    }

    AIOOperation \*\*operations;

    in\_file\_descriptor =

        open(config.in\_path.c\_str(), O\_RDONLY | O\_NONBLOCK, 0666);

    out\_file\_descriptor = open(config.out\_path.c\_str(),

                               O\_CREAT | O\_WRONLY | O\_TRUNC | O\_NONBLOCK, 0666);

    if (in\_file\_descriptor == -1 || out\_file\_descriptor == -1) {

        cout << "Ошибка: Невозможно открыть файл!" << endl;

        return 1;

    }

    if (fstat(in\_file\_descriptor, &fileStat) == 1) {

        cout << "Ошибка: Нет доступа к информации!" << endl;

        close(in\_file\_descriptor);

        close(out\_file\_descriptor);

        return 1;

    }

    file\_size = fileStat.st\_size;

    cout << "Размер файла - " << file\_size << endl;

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

    init\_operations(operations, config.cluster\_size);

    start\_operations(operations);

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

    auto duration = end - Start;

    auto duration\_ms = duration\_cast<milliseconds>(duration);

    auto duration\_u = duration\_cast<microseconds>(duration);

    close(in\_file\_descriptor);

    close(out\_file\_descriptor);

    for (int i = 0; i < config.operations\_count; i++) {

        delete operations[i];

    }

    delete[] operations;

    cout << "Копирование Завершено!" << endl;

    cout << "Затрачено времени: ";

    if (duration.count() < 10000) {

        cout << duration.count() << "ns" << endl;

    } else if (duration.count() < 10000000) {

        cout << duration\_u.count() << "u" << endl;

    } else {

        cout << duration\_ms.count() << "ms" << endl;

    }

    return write\_results(config.in\_path, file\_size, config.operations\_count,

                         config.cluster\_size, duration\_ms.count());

}

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

    AIOOperation &aio\_op = \*(AIOOperation \*)sigval.sival\_ptr;

    if (aio\_op.type == READ) {

        aio\_op.type = WRITE;

        aio\_op.aio->aio\_fildes = out\_file\_descriptor;

        if (aio\_op.aio->aio\_offset + config.cluster\_size > file\_size) {

            aio\_op.aio->aio\_nbytes = file\_size - aio\_op.aio->aio\_offset;

        }

        if (aio\_write(aio\_op.aio) == -1) {

            cerr << "Ошибка: асинхронная запись невозможна" << endl;

            aio\_op.is\_active = false;

            return;

        }

    } else if (aio\_op.type == WRITE) {

        aio\_op.aio->aio\_offset += config.chunk\_size();

        if (aio\_op.aio->aio\_offset >= file\_size) {

            aio\_op.is\_active = false;

        } else {

            aio\_op.type = READ;

            aio\_op.aio->aio\_fildes = in\_file\_descriptor;

            if (aio\_read(aio\_op.aio) == -1) {

                cerr << "Ошибка: асинхронное чтение невозможно" << endl;

                aio\_op.is\_active = false;

                return;

            }

        }

    } else {

        cerr << "Ошибка: неизвестный запрос" << endl;

        return;

    }

}

void init\_operations(AIOOperation \*\*&operations, int cluster) {

    operations = new AIOOperation \*[config.operations\_count]();

    for (off\_t i = 0; i < config.operations\_count; i++) {

        operations[i] = new AIOOperation(READ, cluster, cluster \* i);

    }

}

void start\_operations(AIOOperation \*\*operations) {

    for (off\_t i = 0; i < config.operations\_count; i++) {

        if (operations[i]->aio != nullptr) {

            if (operations[i]->aio->aio\_offset >= file\_size) {

                operations[i]->is\_active = false;

            } else {

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

                    operations[i]->is\_active = true;

                } else {

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

                    cerr << aio\_error(operations[i]->aio) << endl;

                    operations[i]->is\_active = false;

                    return;

                }

            }

        }

    }

    bool working;

    do {

        working = false;

        for (off\_t i = 0; i < config.operations\_count; i++) {

            if (operations[i]->is\_active) {

                working = true;

                break;

            }

        }

        usleep(100);

    } while (working);

}

int write\_results(string pathRead, int file\_size, int operations\_count,

                  int cluster, int duration) {

    ofstream results\_file;

    results\_file.open("results.csv", ios\_base::app);

    if (!results\_file.is\_open()) {

        cerr << "Error writing results" << endl;

        return 1;

    }

    results\_file << pathRead << ";" << file\_size << ";" << operations\_count

                 << ";" << cluster << ";" << duration << "\n";

    results\_file.close();

    return 0;

}

off\_t Config::chunk\_size() { return operations\_count \* cluster\_size; }

Config::Config(int argc, char \*argv[]) {

    if (argc == 5) {

        vector<string> args(&argv[0], &argv[argc]);

        in\_path = args[1];

        out\_path = args[2];

        operations\_count = atoi(args[3].c\_str());

        cluster\_size = atoi(args[4].c\_str());

    } else {

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

        cin >> in\_path;

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

        cin >> out\_path;

        cout << "Введите количество операций: ";

        cin >> operations\_count;

        cout << "Введите размер блока данных: ";

        cin >> cluster\_size;

    }

}

AIOOperation::~AIOOperation() {

    if (buffer != nullptr) {

        delete[] buffer;

    }

    if (aio != nullptr) {

        delete aio;

    }

}

AIOOperation::AIOOperation()

    : aio(nullptr), buffer(nullptr), is\_active(nullptr){};

AIOOperation::AIOOperation(OperationType type, off\_t buffer\_size, off\_t offset)

    : AIOOperation() {

    if (offset > file\_size) {

        return;

    }

    if (aio == nullptr) {

        aio = new aiocb();

    }

    this->type = type;

    aio->aio\_offset = offset;

    aio->aio\_sigevent.sigev\_notify = SIGEV\_THREAD;

    aio->aio\_sigevent.sigev\_notify\_function = aio\_completion\_handler;

    aio->aio\_sigevent.sigev\_value.sival\_ptr = this;

    aio->aio\_fildes = in\_file\_descriptor;

    aio->aio\_nbytes = buffer\_size;

    buffer = new char[aio->aio\_nbytes];

    aio->aio\_buf = buffer;

}