**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»**

Тема: **«Управление файловой системой»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3311 | Баймухамедов Р. Р. |  |
| Преподаватель | Тимофеев А. В. |  |

Санкт-Петербург

2024

**Цель работы**

Исследовать управление файловой системой спомощью Win32 API

**Задание**

Управление дисками, каталогами и файлами.

**Постановка задачи и описание решения**

Для выполнения данной лабораторной работы необходимо разработать 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 / GetLogicalDrivesStrings)
* 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)

**Примеры выполнения программы**

Примеры работоспособности консольного приложения продемонстрированы на рис. 1-4

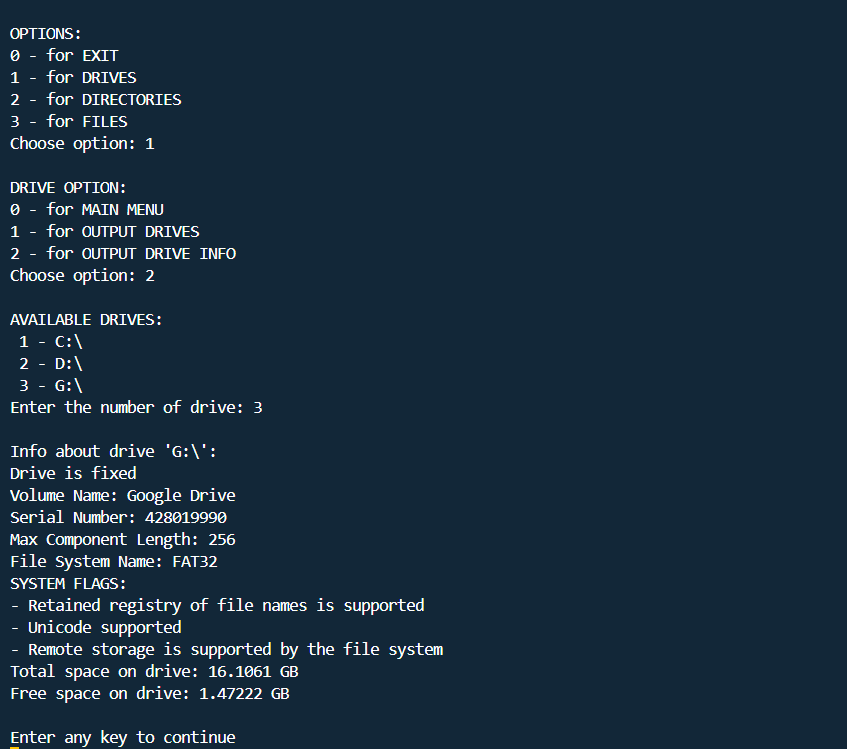


Рисунок (Вывод информации о диске)

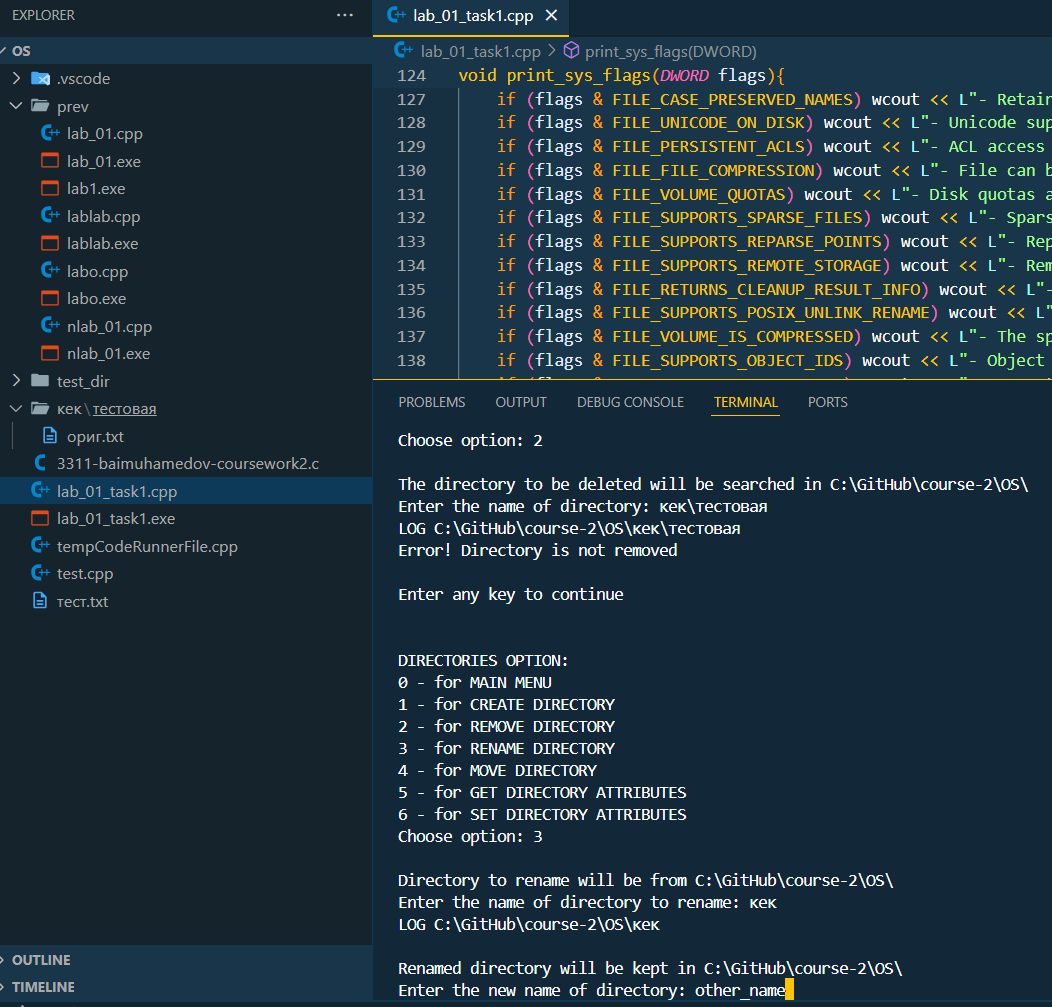


Рисунок (до изменения названия папки)

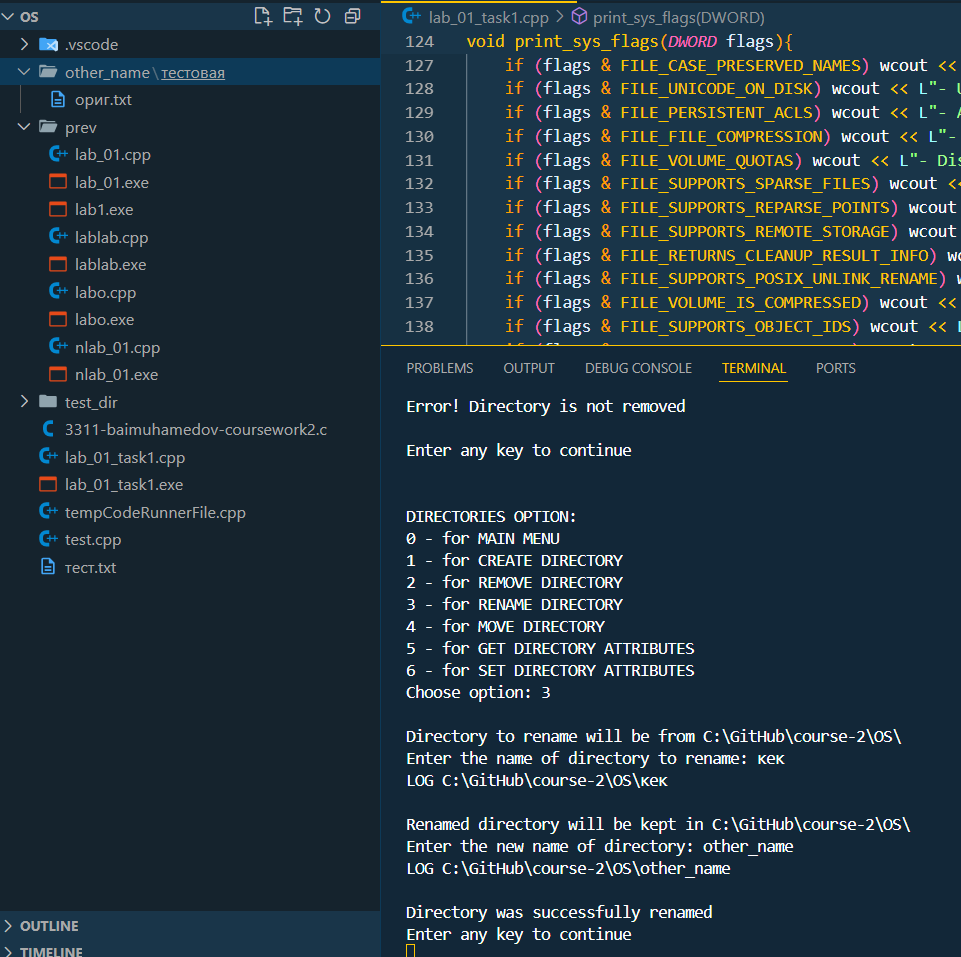
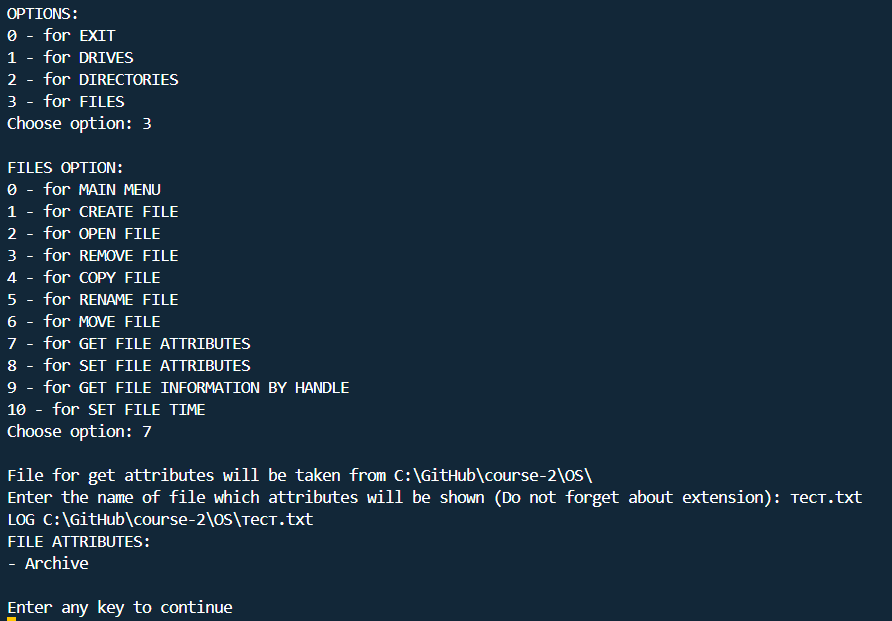


Рисунок (после изменения названия папки)



Рисунок

Win32 API – это низкоуровневый интерфейс, позволяющий точно управлять ресурсами операционной системы. В отличие от стандартных библиотек высокого уровня, он дает полный контроль над памятью, вводом-выводом, синхронизацией потоков и другими аспектами программирования.

Функции Win32 API (Windows Application Programming Interface) представляют собой фундаментальный набор инструментов, обеспечивающий взаимодействие программного обеспечения с операционной системой Windows. Этот интерфейс предоставляет разработчику доступ к управлению файлами, процессами, потоками, графической системой, сетевыми соединениями и множеством других аспектов работы системы.

Одним из значительных преимуществ Win32 API является наличие детальной документации от Microsoft. Практически каждая функция подробно описана на официальном сайте MSDN, где приведены примеры использования, возможные ошибки и рекомендации по применению.

Работа с Win32 API требует внимательности и понимания принципов функционирования операционной системы. Здесь важно учитывать такие нюансы, как работа с дескрипторами, использование правильных кодировок строк (WCHAR для широких строк), обработка многопоточности и корректное освобождение ресурсов. Ошибки при работе с API могут приводить к утечкам памяти, неожиданным сбоям и нарушению работы приложения.

Несмотря на наличие современных библиотек и фреймворков, Win32 API продолжает оставаться востребованным инструментом в системном программировании, разработке драйверов, высокопроизводительных приложений и даже в некоторых игровых движках, где требуется полный контроль над ресурсами.

Таким образом, использование Win32 API позволяет глубже понять внутренние механизмы операционной системы и дает возможность разрабатывать эффективные и производительные приложения.

Для упрощенного взаимодействия с приложением было введено “меню”, содержащее в себе 3 пункта: диски, папки, файлы. В каждом из пунктов добавлены соответствующие по заданию варианты действий.

Консольное приложение поддерживает работу с латиницей и кириллицей. Также в работе с приложением в некоторых действиях применяется относительный путь. Для относительного пути существует последовательность ../ обозначающая переход на один уровень вверх по каталогу, позволяя обращаться к родительским директориям относительно текущего расположения файла или исполняемой программы.

**Заключение**

В ходе лабораторной работы было исследовано управление файловой системой с помощью Win32 API. Win32 API предоставляет широкий набор функций для взаимодействия с операционной системой Windows на низком уровне. Оно охватывает работу с файлами, процессами, потоками, памятью, графикой и сетевыми операциями, обеспечивая точный контроль над ресурсами системы. Одним из ключевых преимуществ Win32 API является его мощность и гибкость, однако работа с ним требует внимательности: необходимо учитывать управление дескрипторами, корректное освобождение ресурсов и обработку ошибок. Благодаря обширной документации Microsoft и широкому спектру возможностей, Win32 API остается важным инструментом для системного программирования, разработки высокопроизводительных приложений и низкоуровневого управления операционной системой.

**Текст программы**

#include <windows.h>

#include <iostream>

#include <string>

#include <locale>

#include <fcntl.h>

#include <io.h>

using namespace std;

*// Helpful functions ---------------------------*

int enter\_integer(*const* *wstring&* message, int a, int b) {

*wstring* input;

    int number;

    while (true) {

        wcout << message;

        getline(wcin, input);

        try {

            number = stoi(input);

            if (number >= a && number <= b) break;

            else wcout << L"Entered value is not in the range [" << a << L", " << b << L"]. Please try again!" << endl;

        } catch (...) {

            wcout << L"Error! Please try again" << endl;

        }

        wcin.clear();

    }

    return number;

}

void clear\_screen() {

#if defined(\_WIN32) || defined(\_WIN64)

    system("cls");

#else

    system("clear");

#endif

}

int main\_menu() {

    wcout << L"\nOPTIONS:" << endl;

    wcout << L"0 - for EXIT" << endl;

    wcout << L"1 - for DRIVES" << endl;

    wcout << L"2 - for DIRECTORIES" << endl;

    wcout << L"3 - for FILES" << endl;

    return enter\_integer(L"Choose option: ", 0, 3);

}

int drive\_menu() {

    wcout << L"\nDRIVE OPTION:" << endl;

    wcout << L"0 - for MAIN MENU" << endl;

    wcout << L"1 - for OUTPUT DRIVES" << endl;

    wcout << L"2 - for OUTPUT DRIVE INFO" << endl;

    return enter\_integer(L"Choose option: ", 0, 2);

}

int dir\_menu() {

    wcout << L"\nDIRECTORIES OPTION:" << endl;

    wcout << L"0 - for MAIN MENU" << endl;

    wcout << L"1 - for CREATE DIRECTORY" << endl;

    wcout << L"2 - for REMOVE DIRECTORY" << endl;

    wcout << L"3 - for RENAME DIRECTORY" << endl;

    wcout << L"4 - for MOVE DIRECTORY" << endl;

    wcout << L"5 - for GET DIRECTORY ATTRIBUTES" << endl;

    wcout << L"6 - for SET DIRECTORY ATTRIBUTES" << endl;

    return enter\_integer(L"Choose option: ", 0, 6);

}

int file\_menu() {

    wcout << L"\nFILES OPTION:" << endl;

    wcout << L"0 - for MAIN MENU" << endl;

    wcout << L"1 - for CREATE FILE" << endl;

    wcout << L"2 - for OPEN FILE" << endl;

    wcout << L"3 - for REMOVE FILE" << endl;

    wcout << L"4 - for COPY FILE" << endl;

    wcout << L"5 - for RENAME FILE" << endl;

    wcout << L"6 - for MOVE FILE" << endl;

    wcout << L"7 - for GET FILE ATTRIBUTES" << endl;

    wcout << L"8 - for SET FILE ATTRIBUTES" << endl;

    wcout << L"9 - for GET FILE INFORMATION BY HANDLE" << endl;

    wcout << L"10 - for SET FILE TIME" << endl;

    return enter\_integer(L"Choose option: ", 0, 10);

}

void wait\_reaction(){

    wcout << L"\nEnter any key to continue" << endl;

    wcin.ignore();

*// getwchar();*

}

*wstring* get\_current\_dir(){

    wchar\_t buffer[MAX\_PATH];

*DWORD* len = GetCurrentDirectoryW(MAX\_PATH, buffer);

    if (len==0) return L"";

    else return *wstring*(buffer);

}

*wstring* function\_in\_path(*const* *wstring&* msg\_to\_path = L"Function will be executed in that path: ", *const* *wstring&* msg\_to\_enter = L"Enter the name: ") {

*wstring* path = get\_current\_dir();

*wstring* name;

    wcout << L"\n" << msg\_to\_path << path << L"\\" << endl;

    wcout << msg\_to\_enter;

    getline(wcin, name);

*wstring* wide\_path = path + L"\\" + name;

    wcout << L"LOG " << wide\_path << endl;

    return wide\_path;

}

*// Drives Functions ---------------------------*

*DWORD* get\_logical\_drives(){

*DWORD* drives\_bm = GetLogicalDrives();

    int count=1;

    wcout << L"\nAVAILABLE DRIVES:\n";

    for (int i=0; i<sizeof(*DWORD*)\*8; i++){

        if (drives\_bm & (1<<i)){

            wchar\_t drive = L'A'+i;

            wcout << L" " << count << L" - " << drive << L":\\\n";

            count++;

        }

    }

    return drives\_bm;

}

void print\_sys\_flags(*DWORD* flags){

    wcout << L"SYSTEM FLAGS:" << endl;

    if (flags & FILE\_CASE\_SENSITIVE\_SEARCH) wcout << L"- File sensitive to uppercase and lowercase" << endl;

    if (flags & FILE\_CASE\_PRESERVED\_NAMES) wcout << L"- Retained registry of file names is supported" << endl;

    if (flags & FILE\_UNICODE\_ON\_DISK) wcout << L"- Unicode supported" << endl;

    if (flags & FILE\_PERSISTENT\_ACLS) wcout << L"- ACL access supported" << endl;

    if (flags & FILE\_FILE\_COMPRESSION) wcout << L"- File can be compressed" << endl;

    if (flags & FILE\_VOLUME\_QUOTAS) wcout << L"- Disk quotas are supported on the specified volume" << endl;

    if (flags & FILE\_SUPPORTS\_SPARSE\_FILES) wcout << L"- Sparse files are supported on the volume" << endl;

    if (flags & FILE\_SUPPORTS\_REPARSE\_POINTS) wcout << L"- Reparse points are supported on the specified volume" << endl;

    if (flags & FILE\_SUPPORTS\_REMOTE\_STORAGE) wcout << L"- Remote storage is supported by the file system" << endl;

    if (flags & FILE\_RETURNS\_CLEANUP\_RESULT\_INFO) wcout << L"- File system returns cleanup result info on successful cleanup" << endl;

    if (flags & FILE\_SUPPORTS\_POSIX\_UNLINK\_RENAME) wcout << L"- POSIX-style unlink and rename operations are supported" << endl;

    if (flags & FILE\_VOLUME\_IS\_COMPRESSED) wcout << L"- The specified volume is a compressed volume" << endl;

    if (flags & FILE\_SUPPORTS\_OBJECT\_IDS) wcout << L"- Object identifiers are supported on the specified volume" << endl;

    if (flags & FILE\_SUPPORTS\_ENCRYPTION) wcout << L"- Encrypted file system (EFS) is supported on the specified volume" << endl;

    if (flags & FILE\_NAMED\_STREAMS) wcout << L"- Named streams are supported on the specified volume" << endl;

    if (flags & FILE\_READ\_ONLY\_VOLUME) wcout << L"- The specified volume is read-only" << endl;

    if (flags & FILE\_SEQUENTIAL\_WRITE\_ONCE) wcout << L"- Sequential write-once is supported on the specified volume" << endl;

    if (flags & FILE\_SUPPORTS\_TRANSACTIONS) wcout << L"- Transactions are supported on the specified volume" << endl;

    if (flags & FILE\_SUPPORTS\_HARD\_LINKS) wcout << L"- Hard links are supported on the specified volume" << endl;

    if (flags & FILE\_SUPPORTS\_EXTENDED\_ATTRIBUTES) wcout << L"- Extended attributes are supported on the specified volume" << endl;

    if (flags & FILE\_SUPPORTS\_OPEN\_BY\_FILE\_ID) wcout << L"- Opening by FileID is supported by the file system" << endl;

    if (flags & FILE\_SUPPORTS\_USN\_JOURNAL) wcout << L"- Update Sequence Number (USN) journaling is supported on the specified volume" << endl;

    if (flags & FILE\_SUPPORTS\_INTEGRITY\_STREAMS) wcout << L"- Integrity streams are supported by the file system" << endl;

    if (flags & FILE\_SUPPORTS\_BLOCK\_REFCOUNTING) wcout << L"- Logical cluster sharing between files on the same volume is supported" << endl;

    if (flags & FILE\_SUPPORTS\_SPARSE\_VDL) wcout << L"- Sparse valid data length (VDL) tracking is supported by the file system" << endl;

    if (flags & FILE\_DAX\_VOLUME) wcout << L"- The specified volume is a Direct Access (DAX) volume" << endl;

    if (flags & FILE\_SUPPORTS\_GHOSTING) wcout << L"- Ghosting is supported by the file system" << endl;

}

void output\_drive\_info(){

*// Drive select*

    wchar\_t disk;

    wchar\_t path[4];

    int option, counter=0;

*DWORD* options = get\_logical\_drives();

    for (int i=0; i<sizeof(*DWORD*)\*8; i++){

        if (options & 1<<i) counter++;

    }

    option = enter\_integer(L"Enter the number of drive: ", 1, counter);

    counter = 1;

    for (int i=0; i<sizeof(*DWORD*)\*8; i++){

        if (options & 1<<i){

            if (counter==option){

                disk = L'A'+i;

                break;

            } else counter++;

        }

    }

    wcout << L"\nInfo about drive '" << (wchar\_t)disk << L":\\\':" << endl;

    swprintf(path, 4, L"%c:\\", disk);

*// Drive info*

    int drive\_type = GetDriveTypeW(path);

    switch (drive\_type){

    case 0: wcout << L"Unknown type of drive" << endl; break;

    case 1: wcout << L"Incorrect root" << endl; break;

    case 2: wcout << L"Drive is removable" << endl; break;

    case 3: wcout << L"Drive is fixed" << endl; break;

    case 4: wcout << L"Drive is remotable" << endl; break;

    case 5: wcout << L"Drive is CD-ROM" << endl; break;

    case 6: wcout << L"Drive is RAM disk" << endl; break;

    }

    wchar\_t volume\_name[MAX\_PATH], file\_system\_name[MAX\_PATH];

*DWORD* serial\_number, max\_length, sys\_flags;

    if (GetVolumeInformationW(path, volume\_name,sizeof(volume\_name), &serial\_number, &max\_length, &sys\_flags, file\_system\_name, sizeof(file\_system\_name))){

        wcout << L"Volume Name: " << volume\_name << endl;

        wcout << L"Serial Number: " << serial\_number << endl;

        wcout << L"Max Component Length: " << max\_length << endl;

        wcout << L"File System Name: " << file\_system\_name << endl;

        print\_sys\_flags(sys\_flags);

    } else wcout << L"Error! Please try again";

*DWORD* sector\_per\_cluster, bytes\_per\_sector, numb\_free\_cluster, numb\_total\_cluster;

    GetDiskFreeSpaceW(path, &sector\_per\_cluster, &bytes\_per\_sector, &numb\_free\_cluster, &numb\_total\_cluster);

*ULONGLONG* total = (*ULONGLONG*)numb\_total\_cluster\*sector\_per\_cluster\*bytes\_per\_sector;

*ULONGLONG* free = (*ULONGLONG*)numb\_free\_cluster\*sector\_per\_cluster\*bytes\_per\_sector;

    wcout << L"Total space on drive: " << total/1e9 << L" GB" << endl;

    wcout << L"Free space on drive: " << free/1e9 << L" GB"<<endl;

}

*// Directory Functions --------------------------*

void create\_dir(){

*wstring* wide\_path = function\_in\_path(L"Directory will be created in ", L"Enter the name of directory: ");

    if (CreateDirectoryW(wide\_path.c\_str(), NULL)){

        wcout << L"Directory successfully created" << endl;

    } else wcout << L"Error! Directory is not created" << endl;

}

void remove\_dir(){

*wstring* wide\_path = function\_in\_path(L"The directory to be deleted will be searched in ", L"Enter the name of directory: ");

    if (RemoveDirectoryW(wide\_path.c\_str())){

        wcout << L"Directory is successfully removed" << endl;

    } else wcout << L"Error! Directory is not removed" << endl;

}

*// File Functions --------------------------------*

void create\_file(int action\_mode\_choise){ *// 0 - to create file, 1 - to open file*

    int access\_mode\_choise, share\_mode\_choise;

*DWORD* access\_mode, share\_mode, action\_mode;

*wstring* wide\_path;

    if (action\_mode\_choise==0) wide\_path = function\_in\_path(L"File will be created in ", L"Enter the name of file (Do not forget about extension): ");

    else {

        wide\_path = function\_in\_path(L"File will be opened in ", L"Enter the name of file (Do not forget about extension): ");

    }

    wcout << L"\nACCESS MODE\n1 - only for READ\n2 - only for WRITE\n3 - for READ and WRITE\n";

    access\_mode\_choise = enter\_integer(L"Choose access mode: ", 1, 3);

    switch(access\_mode\_choise){

        case 1: access\_mode = GENERIC\_READ; break;

        case 2: access\_mode = GENERIC\_WRITE; break;

        case 3: access\_mode = GENERIC\_READ | GENERIC\_WRITE; break;

    }

    wcout << L"\nSHARE MODE\n1 - for NOT ALLOWED\n2 - for READ\n3 - for WRITE\n4 - for DELETE\n";

    share\_mode\_choise = enter\_integer(L"Choose share mode: ", 1, 4);

    switch(share\_mode\_choise){

        case 1: share\_mode= 0; break;

        case 2: share\_mode= FILE\_SHARE\_READ; break;

        case 3: share\_mode= FILE\_SHARE\_WRITE; break;

        case 4: share\_mode= FILE\_SHARE\_DELETE; break;

    }

    if (action\_mode\_choise==0) action\_mode = CREATE\_NEW;

    else action\_mode = OPEN\_EXISTING;

*HANDLE* new\_file = CreateFileW(wide\_path.c\_str(), access\_mode, share\_mode, NULL, action\_mode, FILE\_ATTRIBUTE\_NORMAL, NULL);

    if (new\_file == INVALID\_HANDLE\_VALUE) {

*DWORD* error = GetLastError();

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

            wcout << L"\nFile with equal name exists. Do you want to rewrite?\n1 - YES\n2 - NO\nYour choice: ";

            int rewrite\_choise;

            wcin >> rewrite\_choise;

            if (rewrite\_choise == 1){

                if (action\_mode\_choise==0) action\_mode = CREATE\_ALWAYS;

                else action\_mode = OPEN\_ALWAYS;

                new\_file = CreateFileW(wide\_path.c\_str(), access\_mode, share\_mode, NULL, action\_mode, FILE\_ATTRIBUTE\_NORMAL, NULL);

                if (new\_file == INVALID\_HANDLE\_VALUE) {

                    wcout << L"\nError! Failed to rewrite file" << endl;

                } else {

                    wcout << L"\nFile successfully rewritten" << endl;

                    CloseHandle(new\_file);

                }

            } else wcout << L"File not rewritten." << endl;

        } else {

            if (error!=0){

                wcout << L"Error! Please try again";

            }

        }

    } else {

        if (action\_mode\_choise==0){

            wcout << L"\nFile successfully created!" << endl;

        } else wcout << L"\nFile successfully opened!" << endl;

        CloseHandle(new\_file);

    }

}

void remove\_file(){

*wstring* wide\_path = function\_in\_path(L"File will be deleted in ", L"Enter the name of file (Do not forget about extension): ");

    if(DeleteFileW(wide\_path.c\_str())){

        wcout << L"\nFile was successfully removed\n";

    } else wcout << L"Error! File was not removed";

}

void copy\_file(){

*wstring* wide\_path = function\_in\_path(L"File will be copied from ", L"Enter the name of file to copy (Do not forget about extension): ");

*wstring* wide\_repath = function\_in\_path(L"File will be pasted in ", L"Enter the new name of copied file: ");

    if(CopyFileW(wide\_path.c\_str(), wide\_repath.c\_str(), TRUE)){

        wcout << L"\nFile was successfully copied\n";

    } else {

        wcout << L"\nFile with equal name exists. Do you want to rewrite?\n1 - YES\n2 - NO\nYour choice: ";

        int rewrite\_choise;

        wcin >> rewrite\_choise;

        if (rewrite\_choise == 1){

            if(CopyFileW(wide\_path.c\_str(), wide\_repath.c\_str(), FALSE)){

                wcout << L"File was successfully copied";

            }

            else wcout << L"Error! Please try again";

        }

    }

}

void rename(int is\_dir){

*wstring* wide\_path, wide\_repath;

    if(is\_dir==0){

        wide\_path = function\_in\_path(L"File to rename will be from ", L"Enter the name of file to rename (Do not forget about extension): ");

        wide\_repath = function\_in\_path(L"Renamed file will be kept in ", L"Enter the new name of file: ");

    } else {

        wide\_path = function\_in\_path(L"Directory to rename will be from ", L"Enter the name of directory to rename: ");

        wide\_repath = function\_in\_path(L"Renamed directory will be kept in ", L"Enter the new name of directory: ");

    }

    if(MoveFileW(wide\_path.c\_str(), wide\_repath.c\_str())){

        if (is\_dir==0) wcout << L"\nFile was successfully renamed";

        else wcout << L"\nDirectory was successfully renamed";

    } else {

        if (is\_dir==0) wcout << L"\nFile was not renamed";

        else wcout << L"\nDirectory was not renamed";

    }

}

void move(int is\_dir){

*wstring* path = get\_current\_dir();

*wstring* name, repath;

    if (is\_dir==0) {

        wcout << L"\nFile will be moved according to this path: " << path << L"\\" << endl;

        wcout << L"Enter the name of file that will be moved (Do not forget about extension): ";

    }

    else {

        wcout << L"\nDirectory will be moved according to this path: " << path << L"\\" << endl;

        wcout << L"Enter the name of directory that will be moved: ";

    }

    getline(wcin, name);

*wstring* wide\_path = path + L"\\" + name;

    wcout << L"Specify a relative path to move: ";

    getline(wcin, repath);

*wstring* wide\_repath = path + L"\\" + repath + L"\\" + name;

    if(MoveFileW(wide\_path.c\_str(), wide\_repath.c\_str())){

        wcout << L"File was successfully moved";

    } else {

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

            int replace = enter\_integer(L"File with equal name exists. Do you want to replace it? (0 - for NO / 1 - for YES):",0, 1);

            if (replace==0) wcout << L"Operation of moving is canceled";

            else MoveFileExW(wide\_path.c\_str(), wide\_repath.c\_str(), 0x01);

        }

    }

}

void print\_file\_attr(*DWORD* attributes) {

    wcout << L"FILE ATTRIBUTES:" << endl;

    if (attributes & FILE\_ATTRIBUTE\_READONLY) wcout << L"- Only for reading" << endl;

    if (attributes & FILE\_ATTRIBUTE\_HIDDEN) wcout << L"- Hidden" << endl;

    if (attributes & FILE\_ATTRIBUTE\_SYSTEM) wcout << L"- System file" << endl;

    if (attributes & FILE\_ATTRIBUTE\_DIRECTORY) wcout << L"- Directory" << endl;

    if (attributes & FILE\_ATTRIBUTE\_ARCHIVE) wcout << L"- Archive" << endl;

    if (attributes & FILE\_ATTRIBUTE\_NORMAL) wcout << L"- Default file" << endl;

    if (attributes & FILE\_ATTRIBUTE\_TEMPORARY) wcout << L"- Temporary file" << endl;

    if (attributes & FILE\_ATTRIBUTE\_COMPRESSED) wcout << L"- Compressed" << endl;

    if (attributes & FILE\_ATTRIBUTE\_ENCRYPTED) wcout << L"- Encrypted" << endl;

    if (attributes & FILE\_ATTRIBUTE\_VIRTUAL) wcout << L"- Virtual file" << endl;

}

void get\_file\_attr(int is\_dir){

*wstring* wide\_path;

    if(is\_dir==0){

        wide\_path = function\_in\_path(L"File for get attributes will be taken from ", L"Enter the name of file which attributes will be shown (Do not forget about extension): ");

    } else {

        wide\_path = function\_in\_path(L"Directory for get attributes will be taken from ", L"Enter the name of directory which attributes will be shown: ");

    }

*DWORD* attributes = GetFileAttributesW(wide\_path.c\_str());

    if (attributes == INVALID\_FILE\_ATTRIBUTES) wcout << L"Error! Please try again";

    else print\_file\_attr(attributes);

}

void set\_file\_attr(int is\_dir){

*wstring* wide\_path;

    if (is\_dir==0){

        wide\_path = function\_in\_path(L"File for set attributes will be taken from ", L"Enter the name of file which attributes will be set (Do not forget about extension): ");

    } else {

        wide\_path = function\_in\_path(L"Directory for set attributes will be taken from ", L"Enter the name of directory which attributes will be set: ");

    }

*DWORD* attributes = GetFileAttributesW(wide\_path.c\_str());

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

        wcout << L"\nError! Please try again\n";

        return;

    }

    wcout << L"\nFor each attribute enter 0 (No) or 1 (Yes) to set it\n";

    int choice;

    wcout << L"\nOnly for reading? (1 - Yes, 0 - No): ";

    wcin >> choice;

    if (choice == 1) attributes |= FILE\_ATTRIBUTE\_READONLY;

    else attributes &= ~FILE\_ATTRIBUTE\_READONLY;

    wcout << L"Hidden? (1 - Yes, 0 - No): ";

    wcin >> choice;

    if (choice == 1) attributes |= FILE\_ATTRIBUTE\_HIDDEN;

    else attributes &= ~FILE\_ATTRIBUTE\_HIDDEN;

    wcout << L"System file? (1 - Yes, 0 - No): ";

    wcin >> choice;

    if (choice == 1) attributes |= FILE\_ATTRIBUTE\_SYSTEM;

    else attributes &= ~FILE\_ATTRIBUTE\_SYSTEM;

    wcout << L"Archived? (1 - Yes, 0 - No): ";

    wcin >> choice;

    if (choice == 1) attributes |= FILE\_ATTRIBUTE\_ARCHIVE;

    else attributes &= ~FILE\_ATTRIBUTE\_ARCHIVE;

    wcout << L"Temporary file? (1 - Yes, 0 - No): ";

    wcin >> choice;

    if (choice == 1) attributes |= FILE\_ATTRIBUTE\_TEMPORARY;

    else attributes &= ~FILE\_ATTRIBUTE\_TEMPORARY;

    wcout << L"Compressed file? (1 - Yes, 0 - No): ";

    wcin >> choice;

    if (choice == 1) attributes |= FILE\_ATTRIBUTE\_COMPRESSED;

    else attributes &= ~FILE\_ATTRIBUTE\_COMPRESSED;

    wcout << L"Encrypted file? (1 - Yes, 0 - No): ";

    wcin >> choice;

    if (choice == 1) attributes |= FILE\_ATTRIBUTE\_ENCRYPTED;

    else attributes &= ~FILE\_ATTRIBUTE\_ENCRYPTED;

    if (SetFileAttributesW(wide\_path.c\_str(), attributes)) {

        wcout << L"\nAttributes were successfully set\n";

    } else wcout << L"Error! Attributes were not set. Please try again";

}

void print\_filetime(*const* *FILETIME&* ft) {

*SYSTEMTIME* st;

    FileTimeToSystemTime(&ft, &st);

    wcout << st.wDay << L"." << st.wMonth << L"." << st.wYear << L" ";

    wcout << st.wHour << L":" << st.wMinute << L":" << st.wSecond << endl;

}

*FILETIME* systime\_to\_filetime(*const* *SYSTEMTIME&* st) {

*FILETIME* ft;

    SystemTimeToFileTime(&st, &ft);

    return ft;

}

void get\_file\_info\_by\_handle(){

*wstring* wide\_path = function\_in\_path(L"File for get file information by handle will be taken from ", L"Enter the name of file which attributes will be shown: ");

*HANDLE* handle\_file = CreateFileW(wide\_path.c\_str(), GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

    if (handle\_file == INVALID\_HANDLE\_VALUE) {

*DWORD* error = GetLastError();

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

            wcout << L"Error! File to get info is not exist" << endl;

        }

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

            wcout << L"Access denied. You do not have permission to open this file" << endl;

        } else if (error == ERROR\_SHARING\_VIOLATION) {

            wcout << L"File is being used by another process and cannot be opened" << endl;

        } else {

            wcout << L"Failed to open file" << endl;

        }

        return;

    }

*BY\_HANDLE\_FILE\_INFORMATION* file\_info;

*FILETIME* creation\_time, laccess\_time, lwrite\_time;

    if (GetFileInformationByHandle(handle\_file, &file\_info)){

        print\_file\_attr(file\_info.dwFileAttributes);

        if (GetFileTime(handle\_file, &creation\_time, &laccess\_time, &lwrite\_time)){

            wcout << L"Creation time: ";

            print\_filetime(creation\_time);

            wcout << L"Last access time: ";

            print\_filetime(laccess\_time);

            wcout << L"Last write time: ";

            print\_filetime(lwrite\_time);

        } else {

            wcout << L"Error! Cant get file time" << endl;

        }

        wcout << L"Serial number of volume that contains that file: " << file\_info.dwVolumeSerialNumber << endl;

        wcout << L"Number of links on that file: " << file\_info.nNumberOfLinks << endl;

    }

    CloseHandle(handle\_file);

}

void set\_file\_time(){

*wstring* wide\_path = function\_in\_path(L"File for set file time will be taken from ", L"Enter the name of file which file time will be set (Do not forget about extension): ");

*HANDLE* handle\_file = CreateFileW(wide\_path.c\_str(), FILE\_WRITE\_ATTRIBUTES, FILE\_SHARE\_WRITE, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

    if (handle\_file == INVALID\_HANDLE\_VALUE) {

        wcout << L"Error! Please try again" << endl;

        return;

    }

*SYSTEMTIME* st;

*FILETIME* ftCreation, ftLastAccess, ftLastWrite;

    int choice = enter\_integer(L"Set new creation time? (1 - yes, 0 - no): ",0, 1);

    if (choice) {

        wcout << L"Enter creation date (YYYY MM DD HH MM SS): ";

        wcin >> st.wYear >> st.wMonth >> st.wDay >> st.wHour >> st.wMinute >> st.wSecond;

        ftCreation = systime\_to\_filetime(st);

    } else {

        ftCreation.dwLowDateTime = 0;

        ftCreation.dwHighDateTime = 0;

    }

    wcout << L"Set new last access time? (1 - yes, 0 - no): ";

    wcin >> choice;

    if (choice) {

        wcout << L"Enter last access date (YYYY MM DD HH MM SS): ";

        wcin >> st.wYear >> st.wMonth >> st.wDay >> st.wHour >> st.wMinute >> st.wSecond;

        ftLastAccess = systime\_to\_filetime(st);

    } else {

        ftLastAccess.dwLowDateTime = 0;

        ftLastAccess.dwHighDateTime = 0;

    }

    wcout << L"Set new last write time? (1 - yes, 0 - no): ";

    wcin >> choice;

    if (choice) {

        wcout << L"Enter last write date (YYYY MM DD HH MM SS): ";

        wcin >> st.wYear >> st.wMonth >> st.wDay >> st.wHour >> st.wMinute >> st.wSecond;

        ftLastWrite = systime\_to\_filetime(st);

    } else {

        ftLastWrite.dwLowDateTime = 0;

        ftLastWrite.dwHighDateTime = 0;

    }

    if (SetFileTime(handle\_file, (ftCreation.dwLowDateTime == 0 && ftCreation.dwHighDateTime == 0) ? NULL : &ftCreation, (ftLastAccess.dwLowDateTime == 0 && ftLastAccess.dwHighDateTime == 0) ? NULL : &ftLastAccess, (ftLastWrite.dwLowDateTime == 0 && ftLastWrite.dwHighDateTime == 0) ? NULL : &ftLastWrite)) {

        wcout << L"File time successfully updated" << endl;

    } else wcout << L"Error! File time was not updated" << endl;

    CloseHandle(handle\_file);

}

*// main -----------------------------------------*

int main() {

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

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

    int option;

    do {

        option = main\_menu();

        switch (option) {

            case 1: {

                int drive\_option;

                do {

                    drive\_option = drive\_menu();

                    switch(drive\_option){

                    case 0: break;

                    case 1: get\_logical\_drives(); wait\_reaction(); break;

                    case 2: output\_drive\_info(); wait\_reaction(); break;

                    }

                } while (drive\_option != 0);

                break;

            }

            case 2: {

                int dir\_option;

                do {

                    dir\_option = dir\_menu();

                    switch(dir\_option){

                    case 0: break;

                    case 1: create\_dir(); wait\_reaction(); break;

                    case 2: remove\_dir(); wait\_reaction(); break;

                    case 3: rename(1); wait\_reaction(); break;

                    case 4: move(1); wait\_reaction(); break;

                    case 5: get\_file\_attr(1); wait\_reaction(); break;

                    case 6: set\_file\_attr(1); wait\_reaction(); break;

                    }

                } while (dir\_option != 0);

                break;

            }

            case 3: {

                int file\_option;

                do {

                    file\_option = file\_menu();

                    switch (file\_option){

                    case 0: break;

                    case 1: create\_file(0); wait\_reaction(); break;

                    case 2: create\_file(1); wait\_reaction(); break;

                    case 3: remove\_file(); wait\_reaction(); break;

                    case 4: copy\_file(); wait\_reaction(); break;

                    case 5: rename(0); wait\_reaction(); break;

                    case 6: move(0); wait\_reaction(); break;

                    case 7: get\_file\_attr(0); wait\_reaction(); break;

                    case 8: set\_file\_attr(0); wait\_reaction(); break;

                    case 9: get\_file\_info\_by\_handle(); wait\_reaction(); break;

                    case 10: set\_file\_time(); wait\_reaction(); break;

                    }

                } while (file\_option != 0);

                break;

            }

        }

    } while (option != 0);

    wcout << L"\nGoodbye!\n" << endl;

    return 0;

}