Министерство образования Республики Беларусь

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**КУРСОВОЙ ПРОЕКТ**

по дисциплине **«Операционные системы и системное программирование»**

на тему **«Атрибуты файлов в NTFS ОС Windows»**

Выполнил:

студент группы 10701116 Багиров Р.А.

Руководитель: Иванченко В.В.

Минск 2018

Министерство образования Республики Беларусь

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовому проекту**

по дисциплине **«Операционные системы и системное программирование»**

на тему **«Атрибуты файлов в NTFS ОС Windows»**

Выполнил:

студент группы 10701116 Багиров Р.А.

Руководитель: Иванченко В.В.

Минск 2018

Введение 4

1. Аналитический обзор состояния вопроса 6

1.1. Базовые атрибуты файлов 6

1.2. Атрибуты сжатия и шифрования (compressed/encrypted) 7

1.3. Информация о файле 7

1.4. Тип файла 8

2. Техническое задание на предмет разработки 9

3. Моделирование и проектирование программного обеспечения 10

3.1. Графический интерфейс 10

3.2. Серверная часть 10

3.3. Клиентская часть 13

4. Реализация программного обеспечения 15

4.1. Руководство для системного администратора 15

4.2. Руководство для пользователя (клиентская часть) 16

5. Развертывание и тестирование программного обеспечения 20

Заключение 26

Список использованных источников 27

ПРИЛОЖЕНИЕ А 28

Графическая часть 28

ПРИЛОЖЕНИЕ Б 31

Исходный код приложения 31

# Введение

Тема данного курсового проекта – атрибуты файлов в NTFS ОС Windows. Данная тема предусматривает разработку простейшего приложения для семейства операционных систем Windows с использованием функций Win32 API.

API — это аббревиатура названия Application Programming Interface (интерфейс прикладного программирования). API представляет собой совокупность функций и инструментов, позволяющих программисту создавать приложения (программы), работающие в некоторой среде.

Win32 API — это набор функций для создания программ, работающих под управлением Microsoft Windows. Все функции этого набора являются 32-битными, что отражено в названии интерфейса.

Интерфейс программирования приложений Win32 API ориентирован на язык программирования С или, в более широком смысле, на процедурные языки программирования. Поэтому в этом интерфейсе, не используются такие возможности языка программирования С++, как классы, ссылки и механизм обработки исключений. Чтобы сделать интерфейс Win32 API более независимым от конкретного языка программирования или, может быть, более соответствующим аппаратному обеспечению компьютера, разработчики этого интерфейса определили новые простые типы данных. Эти типы данных используются в прототипах функций интерфейса Win32 API. Новые простые типы данных определены как синонимы простых типов данных языка программирования С. Чтобы отличать эти типы от других типов, их имена определены прописными буквами.

При написании прикладных программ для Windows программистам приходится пользоваться огромным объемом документации, который изложен в справочной системе MSDN, содержащей очень мало примеров и представленной только на английском языке. Часто приходится тратить много времени, чтобы понять, как работает та или иная функция и какова область ее применения.

NTFS (аббревиатура от англ. *new technology file system* — «файловая система новой технологии») — стандартная файловая система для семейства операционных систем Windows NT фирмы Microsoft.

NTFS обеспечивает комбинацию эффективности, надежности и совместимости, отсутствующую в FAT или HPFS. Она разработана для быстрого выполнения стандартных файловых операций типа чтения, записи и поиска, а также улучшенных операций типа восстановления файловой системы на очень больших жестких дисках.

NTFS также включает возможности безопасности, требуемые для файловых серверов и высококачественных персональных компьютеров в корпоративной среде. NTFS поддерживает управление доступом к данным и привилегии владельца, что является важным для целостности корпоративных данных. В то время как каталогам, разделяемым при помощи Windows NT Server, назначаются специфические разрешения, файлам и каталогам NTFS могут назначаться разрешения вне зависимости, разделены они или нет. NTFS — единственная файловая система в Windows NT, которая позволяет назначить разрешения для отдельных файлов.

NTFS является простой, но очень мощной разработкой. Для этой перспективной файловой системы вся информация на томе NTFS является файлом или частью файла. Каждый распределенный на томе NTFS сектор принадлежит некоторому файлу. Даже метаданные (metadata) файловой системы (информация, которая описывает непосредственно файловую систему) являются частью файла.

Эта основанная на атрибутах файловая система поддерживает объектно-ориентированные приложения, обрабатывая все файлы как объекты, которые имеют определяемые пользователем и системой атрибуты.

Данный курсовой проект посвящен разработке приложения, работающего с атрибутами файлов и основной информацией о файле.

В Разделе 1 кратко рассмотрены основы работы с атрибутами файла с использованием Win32 API.

В Разделе 2 поставлена основная задача курсового проекта.

В Разделе 3 описаны графический интерфейс(схематично) и модульная архитектура проекта.

В Разделе 4 рассмотрены реализация важных методов и набор функций Win32 API, которые были использованы в данных методах.

Пятый раздел посвящен вопросам развертывания и тестирования разработанного приложения.

# Аналитический обзор состояния вопроса

## Базовые атрибуты файлов

Для получения атрибутов выбранного файла используется функция Win32 API DWORD GetFileAttributes(LPCSTR lpFileName). Данная функция возвращает целое число – атрибуты файла. На вход функция принимает путь к файлу (lpFileName).

В Win32 API описаны целочисленные константы, соответствующие своему атрибуту. В данном списке перечислены те константы, которые необходимо использовать при реализации нашего приложения:

* FILE\_ATTRIBUTE\_ARCHIVE (“Archive”) – атрибут, означающий, что файл готов для архивирования
* FILE\_ATTRIBUTE\_READONLY (“Read-only”) – атрибут, означающий, что нельзя изменять содержимое файла
* FILE\_ATTRIBUTE\_HIDDEN (“Hidden”) – файл с таким атрибутом считается скрытым
* FILE\_ATTRIBUTE\_NOT\_CONTENT\_INDEXED (“Not indexed”) – атрибут, означающий, что индексирование содержимого данного файла не разрешено
* FILE\_ATTRIBUTE\_COMPRESSED (“Compressed”) – атрибут, означающий, что Windows должен хранить данный файл на диске в сжатом виде
* FILE\_ATTRIBUTE\_ENCRYPTED (“Encrypted”) - атрибут, означающий, что Windows должен хранить данный файл на диске в зашифрованном виде.

Примечание:

Атрибут “Encrypted”, как правило, недоступен в домашних редакциях MS Windows.

Для проверки состояния определенного атрибута файла необходимо получить атрибуты файла и выполнить побитовое «И» с константой, обозначающей интересующий нас атрибут, и сравнить результат с данной константой. Если результат равен константе, то атрибут установлен, в ином случае – не установлен.

Данный пример проверяет, установлен ли атрибут “Read-only”:

DWORD attributes = GetFileAttributes(filename);

return (attributes & FILE\_ATTRIBUTE\_READONLY) == FILE\_ATTRIBUTE\_READONLY;

Для изменения состояния атрибутов “Read-only”, “Hidden”, “Archive” и “Not indexed” из списка выше можно использовать функцию BOOL SetFileAttributes(LPCSTR lpFileName, DWORD dwFileAttributes). На вход данная функция принимает имя файла (lpFileName) и атрибуты файла (dwFileAttributes). Если выполнение функции провалено – она вернет ноль.

## Атрибуты сжатия и шифрования (compressed/encrypted)

В предыдущем пункте был описан способ изменения базовых атрибутов файла с использованием функции SetFileAttributes. Однако в документации к данной функции в msdn имеется список атрибутов, состояние которых нельзя изменить при помощи данной функции. К таким атрибутам относятся и атрибуты “Compressed” и “Encrypted”. К счастью, в документации описаны способы изменения состояния тех атрибутов, которые нельзя изменить с помощью SetFileAttributes.

Прежде всего следует заметить, что активным среди атрибутов “Compressed” и “Encrypted” может быть только один. Иначе говоря, файл может быть либо только сжатым, либо только зашифрованным.

Для изменения состояния атрибута “Compressed” следует использовать функцию BOOL DeviceIoControl(HANDLE hDevice, DWORD dwIoControlCode, LPVOID lpInBuffer, DWORD nInBufferSize, LPVOID lpOutBuffer, DWORD nOutBufferSize, LPDWORD lpBytesReturned, LPOVERLAPPED lpOverlapped), где в качестве hDevice следует использовать дескриптор файла, а в качестве dwIoControlCode поставить константу FSCTL\_SET\_COMPRESSION.

Для того, чтобы изменить состояние атрибута “Encrypted”, нужно использовать функцию EncryptFile(LPCSTR lpFileName) для шифрования файла и функцию DecryptFile(LPCSTR lpFileName, DWORD dwReserved) для расшифрования файла (в параметр dwReserved поставить ноль).

Следует иметь в виду, что как правило, в домашних редакциях Windows шифрование файлов таким способом недоступно.

## Информация о файле

Получить информацию о файле можно с помощью функции BOOL GetFileInformationByHandle(HANDLE hFile, LPBY\_HANDLE\_FILE\_INFORMATION lpFileInformation), где hFile – дескриптор файла, а lpFileInformation – ссылка на экземпляр структуры BY\_HANDLE\_FILE\_INFORMATION. С помощью данной структуры можно получить:

* Атрибуты файла (поле dwFileAttributes)
* Время создания файла (поле ftCreationTime)
* Время последнего открытия файла (поле ftAccessTime)
* Время последнего изменения файла (поле ftWriteTime)
* Размер файла (поля nFileSizeLow и nFileSizeHigh)
* И так далее

## Тип файла

Получить тип файла и дескриптор иконки файла можно при помощи функции для работы с оболочкой Windows DWORD\_PTR SHGetFileInfo(LPCSTR pszPath, DWORD dwFileAttributes, SHFILEINFOA \*psfi, UINT cbFileInfo, UINT uFlags). Для того, чтобы получить тип файла и дескриптор иконки, необходимо вызвать данную функцию с параметрами:

* pszPath – путь к файлу
* dwFileAttributes – в данном случае можно проигнорировать, ставим 0
* psfi – Передаем ссылку на экземпляр структуры SHFILEINFO
* cbFileInfo – размер структуры SHFILEINFO (sizeof(SHFILEINFO))
* uFlags – для получения иконки и типа файла ставим SHGFI\_ICON | SHGFI\_TYPENAME

# Техническое задание на предмет разработки

Требуется разработать простейшее приложение с графическим интерфейсом для ОС MS Windows на языке программирования С(или С++) с использованием Win32 API, которое будет работать с атрибутами файла (в том числе и с атрибутами сжатия и шифрования).

Приложение должно предоставлять пользователю возможность выбора файла и просмотра и изменения атрибутов выбранного файла. Атрибуты файлов пользователь сможет определить по так называемым «чекбоксам». Изменение выбранного атрибута файла должно производиться по нажатию на «чекбокс».

Также в приложении должна быть предусмотрена возможность просмотра некоторой информации о файле. После выбора файла пользователем, приложение должно вывести дату и время создания файла, последнего изменения и открытия, путь к файлу и размер файла.

После выбора файла пользователем также должна быть выведена информация о типе файла: тип файла и иконка.

Интерфейс приложения должен быть дружелюбным и интуитивно понятным, поэтому приложение требуется разработать с графическим интерфейсом. Оконный интерфейс необходимо реализовать с помощью набора функций Win32 API.

# Моделирование и проектирование программного обеспечения

## Графический интерфейс

Данное приложение использует графический интерфейс для взаимодействия с пользователем.

Схематический вид окна представлена на рис. 3.1.1. Также схематический вид представлен в разделе ПРИЛОЖЕНИЕ А.

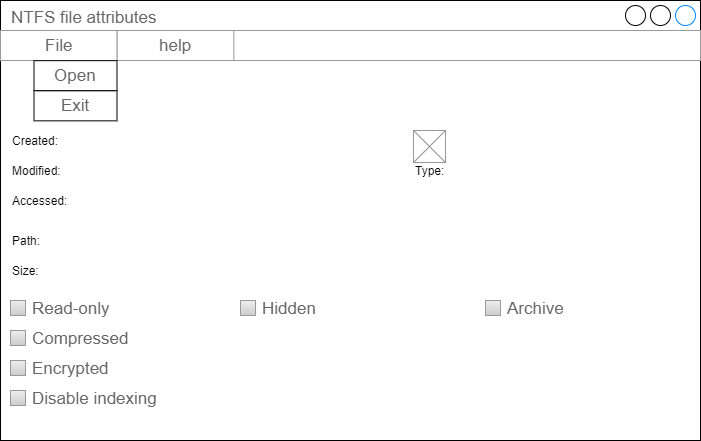


Рис. 3.1.1. – Схематический вид оконного интерфейс

В окне имеется главное меню с пунктами “File” и “help”.

В меню “File” есть два пункта выбора: выбрать файл (“Open”) и выйти (“Exit”).

Для вывода информации о файле используются статические элементы (на рис. 3.1.1 это надписи “Created:”, “Modified:”, “Accessed:”, “Type:”, “Path:”, “Size” и зачеркнутый квадрат – поле для вывода иконки).

В нижней части окна располагаются «чекбоксы», которые отображают состояние атрибутов файла и позволяют изменять его.

## Модульная архитектура проекта

Точкой входа в приложение является файл “start.cpp”. В данном файле создается контроллер с формой и запускается метод контроллера “Controller::run”, в который передается ссылка на обработчик.

Если рассматривать общую структуру, то проект состоит из контроллера, формы (класс, содержащий элементы оконного интерфейса) и классов для взаимодействия с файловой системой.

Основным классом в проекте является класс Controller, UML-диаграмма которого представлена на рис. 3.2.1. Данный класс обрабатывает запросы, которые поступают от пользователя, и отправляет на форму.

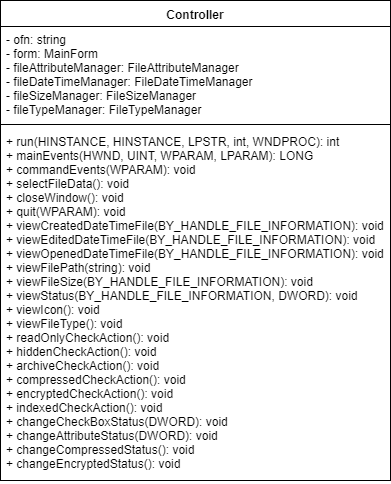


Рис. 3.2.1. – UML-диаграмма класса Controller

Класс MainForm содержит в себе окно и элементы графического интерфейса. На рис. 3.2.2 изображена UML-диаграмма класса MainForm.

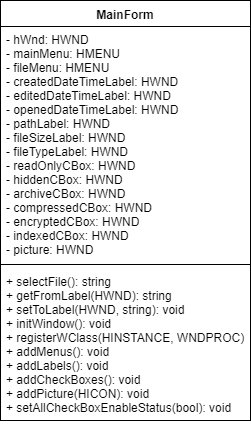


Рис 3.2.2. – UML-диаграмма класса MainForm

За работу с атрибутами файла отвечает класс FileAttributeManager. С помощью данного класса можно работать с атрибутами выбранного файла. Его UML-диаграмма изображена на рис. 3.2.3.

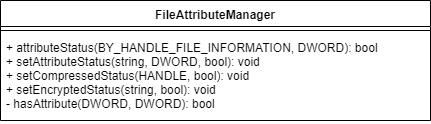


Рис. 3.2.3. – UML-диаграмма класса FileAttributeManager

Для получения даты и времени создания, последнего изменения и открытия файла используется класс FileDateTimeManager. UML-диаграмма данного класса изображена на рис. 3.2.4.



Рис. 3.2.4. – UML-диаграмма класса FileDateTimeManager

За получение размера файла ответственен небольшой вспомогательный класс FileSizeManager, UML-диаграмма которого изображена на рис. 3.2.5.

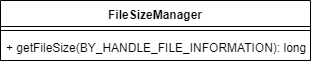


Рис. 3.2.5. – UML-диаграмма класса FileSizeManager

Есть еще один вспомогательный класс FileTypeManager, задача которого предоставлять информацию о типе файла. Его UML-диаграмма изображена на рис. 3.2.6.

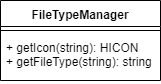


Рис. 3.2.6. – UML-диаграмма класса FileTypeManager

Также имеется утилитный класс FileUtil. Этот класс предоставляет методы для получения дескриптора файла по имени и для получения структуры информации файла. Его UML-диаграмма изображена на рис. 3.2.7.

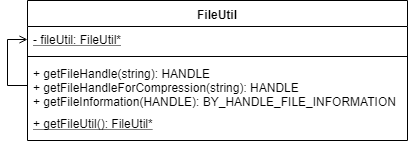


Рис. 3.2.7. – UML-диаграмма класса FileUtil

# Реализация программного обеспечения

## Точка доступа

Код точки доступа:

#include "Controller.h";

LONG CALLBACK WinEvents(HWND hwnd, UINT winEvent, WPARAM wParam,

LPARAM lParam);

MainForm &mainForm = \*(new MainForm());

Controller &controller = \*(new Controller(mainForm));

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,

LPSTR lpCmdLine, int nShowCmd)

{

return controller.run(hInstance, hPrevInstance, lpCmdLine, nShowCmd,

WinEvents);

}

LONG CALLBACK WinEvents(HWND hwnd, UINT winEvent, WPARAM wParam,

LPARAM lParam)

{

return controller.mainEvents(hwnd, winEvent, wParam, lParam);

}

WinEvents – обработчик событий.

## Руководство для пользователя (клиентская часть)

# Развертывание и тестирование программного обеспечения

Для проведения развертывания данного приложения на компьютере должна быть установлена ОС Windows с установленной на ней MS Visual C++ 2017. В этом случае запуск программы крайне простой – нужно всего лишь запустить CourseProject.exe.

Тестирование данного приложения производилось на виртуальной машине VirtualBox – ОС Windows 10 ltsc, выделено оперативной памяти – 4096 МБ. Процессор: AMD A9 9400 2.4..3.2 GHz

При запуске программы открывается главное окно, как показано на рисунке 5.1:

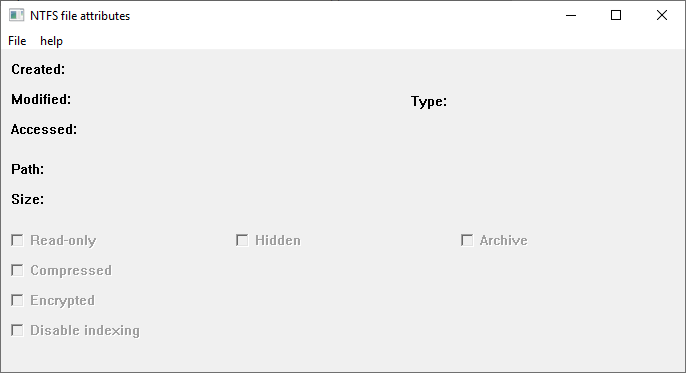


Рис. 5.1. – Главное окно приложения

Проверим, какие данные выведет приложение о файле и сравним эти данные с тем, что указано в свойствах файла. Для начала выберем файл (рис 5.2):

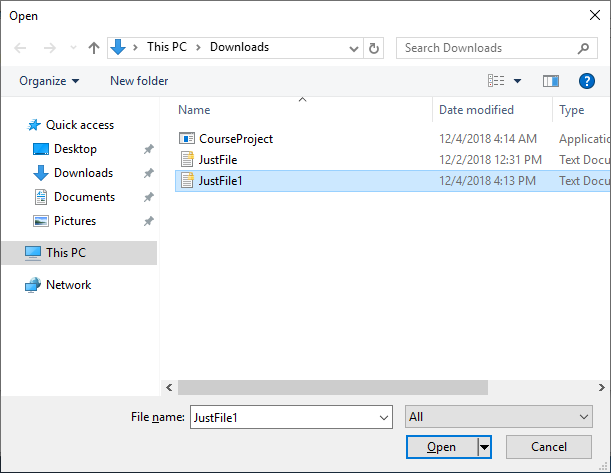


Рис. 5.2. – Выбор файла

Окно приложения после выбора файла показано на рисунке 5.3:

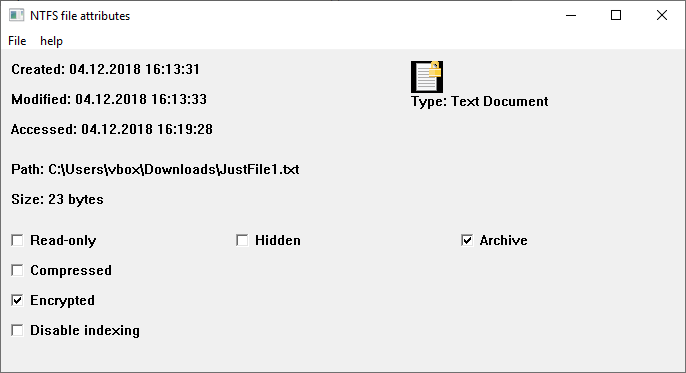


Рис. 5.3. – Окно приложения после выбора файла

Свойства выбранного файла указаны на рисунке 5.4 и рисунке 5.5:

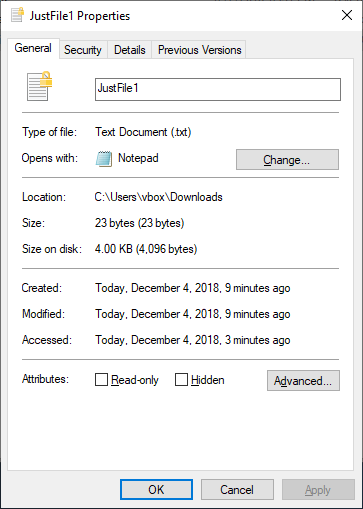


Рис. 5.4. – Свойства выбранного файла

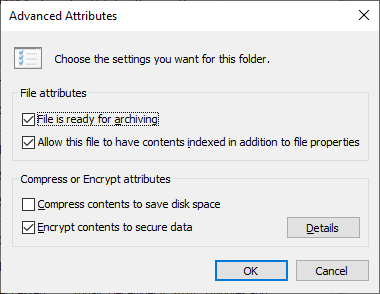


Рис. 5.5. – Дополнительные атрибуты выбранного файла

Свойства файла совпадают с теми данными, которые предоставило приложение, значит пока что поведение приложения отвечает нашим ожиданиям.

Проверим смену атрибутов файла. Для этого поставим атрибут “Compressed” включенным. При этом атрибут “Encrypted” должен отключиться. Окно приложения приведено на рисунке 5.6:

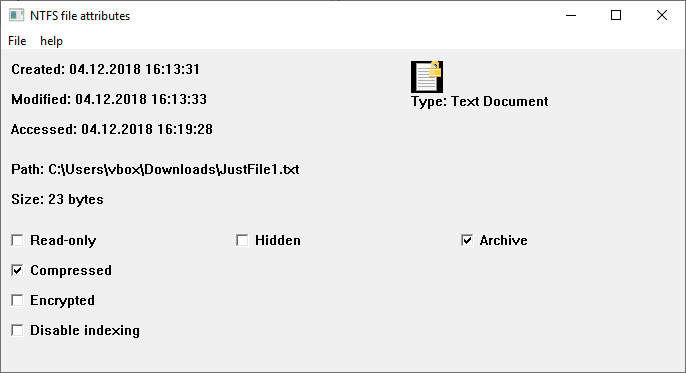


Рис. 5.6. – Ставим атрибут “Compressed”

Проверяем свойства файла (рис. 5.7):

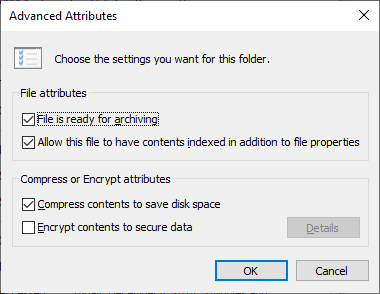


Рис. 5.7. – Проверка дополнительных атрибутов

Как видно, атрибуты в приложении и свойствах совпали, значит смена атрибутов “Encrypted” и “Compressed” работает.

Теперь протестируем базовые атрибуты. Запретим индексирование и сделаем файл скрытым (рис. 5.8):

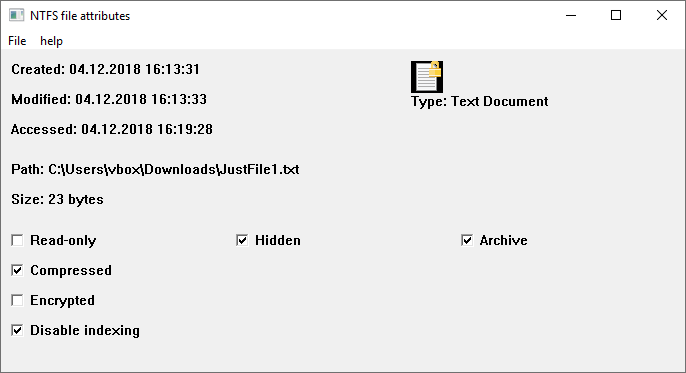


Рис. 5.8. – Изменение атрибутов файла

Проверяем свойства файла, данные свойства представлены на рисунке 5.9 и 5.10:

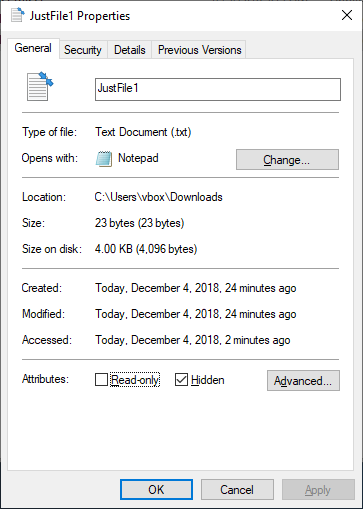


Рис. 5.9. – Свойства файла

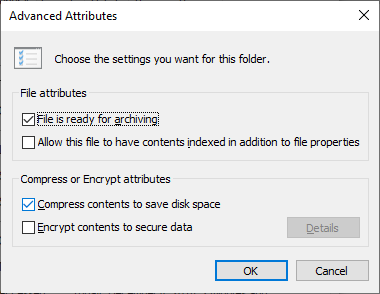


Рис. 5.10. – Дополнительные атрибуты файла

Атрибуты файла в свойствах совпали с теми, что отображаются в приложении, значит при изменении базовых атрибутов поведение приложения соответствует нашим ожиданиям.

# Заключение

При выполнении данного курсового проекта было спроектировано и реализовано простое оконное приложение для управления основными атрибутами файла в файловой системе NTFS. Были изучены некоторые функции из Win32 API для работы с файлами, атрибутами файлов и так далее. Также были закреплены на практике умения разрабатывать приложения с использованием Win32 API, использующие графический интерфейс. Также я узнал больше об атрибутах файла и как с ними работать.

Результатом выполнения данного курсового проекта стала реализация на практике умений разрабатывать приложения, использующие Win32 API как для реализации графического интерфейса, так и для реализации взаимодействия с файловой системой.

# Список использованных источников

1. Литвиненко Н.А., Технология программирования на С++. Win32 API-приложения – СПб.: БХВ-Петербург, 2010. – 288с.: ил. – (Учебное пособие)
2. Using Common Dialog Boxes. – Режим доступа: <https://docs.microsoft.com/en-us/windows/desktop/dlgbox/using-common-dialog-boxes>
3. Retrieving and Changing File Attributes. – Режим доступа: <https://docs.microsoft.com/en-us/windows/desktop/FileIO/retrieving-and-changing-file-attributes>
4. SetFileAttributesA function. – Режим доступа: <https://docs.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-setfileattributesa>
5. GetFileInformationByHandle function. – Режим доступа: <https://docs.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-getfileinformationbyhandle>
6. FileTimeToLocalFileTime function. – Режим доступа: <https://docs.microsoft.com/en-us/windows/desktop/api/fileapi/nf-fileapi-filetimetolocalfiletime>
7. SHGetFileInfoA function. – Режим доступа: <https://docs.microsoft.com/en-us/windows/desktop/api/shellapi/nf-shellapi-shgetfileinfoa>

# ПРИЛОЖЕНИЕ А

## Графическая часть

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | | | | | |
|  |  |  | |  | |  | | КП—10701116 03–ДО–201 | | | | | | | | |
|  |  |  | |  | |  | |
| Изм. | Лист | № документа | | Подпись | | Дата | |
| Разраб. | | Багиров Р.А. | |  | |  | | Use-case диаграмма | | Лит | | | | Лист | | Листов |
| Руковод. | | Иванченко В.В. | |  | |  | |  | Д | |  | **1** | | **3** |
| Консульт. | | Иванченко В.В. | |  | |  | | 1 40 01 01 БНТУ  г.Минск | | | | | | |
| Н.контр. | | Иванченко В.В. | |  | |  | |
| Зав.каф. | |  | |  | |  | |
|  | | | | | | | | | | | | | | | | | |
|  | | | Фамилия | | Подпись | | Дата | | *Диаграмма классов приложения* | | | Лист | | | Листов | | |
| Студент | | | Багиров Р.А. | |  | |  | |
| Руководитель | | | Иванченко В.В. | |  | |  | | *2* | | | *3* | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |
|  | Фамилия | Подпись | Дата | *Интерфейс пользователя* | Лист | Листов |
| Студент | Багиров Р.А. |  |  |
| Руководитель | Иванченко В.В. |  |  | *3* | *3* |

# ПРИЛОЖЕНИЕ Б

## Исходный код приложения

**Файлы заголовков:**

Controller.h

#pragma once

#include <map>

#include "MainForm.h";

#include "FileAttributeManager.h";

#include "FileDateTimeManager.h";

#include "FileSizeManager.h";

#include "FileUtil.h";

#include "FileTypeManager.h";

class Controller

{

public:

Controller();

Controller(MainForm form);

~Controller();

int run(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine,

int nShowCmd, WNDPROC wndProc);

LONG mainEvents(HWND hwnd, UINT winEvent, WPARAM wParam, LPARAM lParam);

void commandEvents(WPARAM wParam);

void selectFileData();

void closeWindow();

void quit(WPARAM wParam);

void viewCreatedDateTimeFile(BY\_HANDLE\_FILE\_INFORMATION file);

void viewEditedDateTimeFile(BY\_HANDLE\_FILE\_INFORMATION file);

void viewOpenedDateTimeFile(BY\_HANDLE\_FILE\_INFORMATION file);

void viewFilePath(std::string path);

void viewFileSize(BY\_HANDLE\_FILE\_INFORMATION file);

void viewStatus(BY\_HANDLE\_FILE\_INFORMATION file, DWORD attribute);

void viewIcon();

void viewFileType();

void readOnlyCheckAction();

void hiddenCheckAction();

void archiveCheckAction();

void compressedCheckAction();

void encryptedCheckAction();

void indexedCheckAction();

void changeCheckBoxStatus(DWORD attributeAction);

void changeAttributeStatus(DWORD attribute);

void changeCompressedStatus();

void changeEncryptedStatus();

std::string GetOfn();

MainForm getForm();

FileAttributeManager getFileAttributeManager();

FileDateTimeManager getDateTimeManager();

FileSizeManager getFileSizeManager();

FileTypeManager getFileTypeManager();

void setOfn(std::string ofn);

void setForm(MainForm form);

void setFileAttributeManager(FileAttributeManager fileAttributeManager);

void setDateTimeManager(FileDateTimeManager dateTimeManager);

void setFileSizeManager(FileSizeManager fileSizeManager);

void setFileTypeManager(FileTypeManager fileTypeManager);

private:

std::string ofn;

MainForm form;

FileAttributeManager fileAttributeManager;

FileDateTimeManager dateTimeManager;

FileSizeManager fileSizeManager;

FileTypeManager fileTypeManager;

};

typedef void(Controller::\*events)(WPARAM);

typedef void(Controller::\*commEvents)();

MainForm.h

#pragma once

#include <Windows.h>

#include <string>

#define DEFAULT\_X\_POS 10

#define DEFAULT\_X\_LAYOUT 15

#define DEFAULT\_Y\_LAYOUT 10

#define TOP\_Y\_POS 10

#define WCLASS\_NAME "myWindow"

#define TITLE "NTFS file attributes"

#define MAIN\_X\_POS 100

#define MAIN\_Y\_POS 100

#define MAIN\_WIDTH 700

#define MAIN\_HEIGHT 380

#define PIC\_X\_POS 410

#define PIC\_SIZE 32

#define LABELS\_WIDHT 380

#define LABELS\_HEIGHT 20

#define LABELS\_Y\_LAYOUT LABELS\_HEIGHT + DEFAULT\_Y\_LAYOUT

#define CREATED\_DATETIME\_LABEL "Created: "

#define EDITED\_DATETIME\_LABEL "Modified: "

#define OPENED\_DATETIME\_LABEL "Accessed: "

#define PATH\_LABEL "Path: "

#define FILE\_SIZE\_LABEL "Size: "

#define FILE\_SIZE\_LABEL\_SUFFIX " bytes"

#define FILE\_TYPE\_LABEL "Type: "

#define CREATED\_DATETIME\_LABEL\_Y\_POS TOP\_Y\_POS

#define EDITED\_DATETIME\_LABEL\_Y\_POS CREATED\_DATETIME\_LABEL\_Y\_POS + LABELS\_Y\_LAYOUT

#define OPENED\_DATETIME\_LABEL\_Y\_POS EDITED\_DATETIME\_LABEL\_Y\_POS + LABELS\_Y\_LAYOUT

#define PATH\_LABEL\_Y\_POS OPENED\_DATETIME\_LABEL\_Y\_POS + LABELS\_Y\_LAYOUT + DEFAULT\_Y\_LAYOUT

#define FILE\_SIZE\_LABEL\_Y\_POS PATH\_LABEL\_Y\_POS + LABELS\_Y\_LAYOUT

#define FILE\_TYPE\_LABEL\_X\_POS PIC\_X\_POS

#define FILE\_TYPE\_LABEL\_Y\_POS DEFAULT\_Y\_LAYOUT + PIC\_SIZE

#define CHECKBOX\_WIDTH 210

#define CHECKBOX\_HEIGHT 20

#define CHECKBOX\_X\_LAYOUT CHECKBOX\_WIDTH + DEFAULT\_X\_LAYOUT

#define CHECKBOX\_Y\_LAYOUT CHECKBOX\_HEIGHT + DEFAULT\_Y\_LAYOUT

#define READONLY\_CBOX "Read-only"

#define READONLY\_CBOX\_Y\_POS FILE\_SIZE\_LABEL\_Y\_POS + CHECKBOX\_Y\_LAYOUT + DEFAULT\_Y\_LAYOUT

#define HIDDEN\_CBOX "Hidden"

#define HIDDEN\_CBOX\_X\_POS DEFAULT\_X\_POS + CHECKBOX\_X\_LAYOUT

#define HIDDEN\_CBOX\_Y\_POS READONLY\_CBOX\_Y\_POS

#define ARCHIVE\_CBOX "Archive"

#define ARCHIVE\_CBOX\_X\_POS HIDDEN\_CBOX\_X\_POS + CHECKBOX\_X\_LAYOUT

#define ARCHIVE\_CBOX\_Y\_POS HIDDEN\_CBOX\_Y\_POS

#define COMPRESSED\_CBOX "Compressed"

#define COMPRESSED\_CBOX\_Y\_POS READONLY\_CBOX\_Y\_POS + CHECKBOX\_Y\_LAYOUT

#define ENCRYPTED\_CBOX "Encrypted"

#define ENCRYPTED\_CBOX\_Y\_POS COMPRESSED\_CBOX\_Y\_POS + CHECKBOX\_Y\_LAYOUT

#define INDEXED\_CBOX "Disable indexing"

#define INDEXED\_CBOX\_Y\_POS ENCRYPTED\_CBOX\_Y\_POS + CHECKBOX\_Y\_LAYOUT

#define HELP "help"

#define FILE\_MENU "File"

#define FILE\_MENU\_OPEN "Open"

#define FILE\_MENU\_EXIT "Exit"

#define FILE\_MENU\_OPEN\_ACTION 1

#define FILE\_MENU\_EXIT\_ACTION 3

#define READONLY\_CHECK\_ACTION 5

#define HIDDEN\_CHECK\_ACTION 7

#define ARCHIVE\_CHECK\_ACTION 9

#define COMPRESSED\_CHECK\_ACTION 11

#define ENCRYPTED\_CHECK\_ACTION 13

#define INDEXED\_CHECK\_ACTION 15

#define DEFAULT\_STR\_SIZE 260

class MainForm

{

public:

MainForm();

~MainForm();

std::string selectFile();

std::string getFromLabel(HWND label);

void setToLabel(HWND label, std::string str);

void initWindow();

void registerWClass(HINSTANCE hInstance, WNDPROC wndProc);

void addMenus();

void addLabels();

void addCheckBoxes();

void addPicture(HICON bitmap);

void setAllCheckBoxEnableStatus(bool status);

HWND getHWnd();

HMENU getMainMenu();

HMENU getFileMenu();

HWND getCreatedDateTimeLabel();

HWND getEditedDateTimeLabel();

HWND getOpenedDateTimeLabel();

HWND getPathLabel();

HWND getFileSizeLabel();

HWND getFileTypeLabel();

HWND getReadOnlyCBox();

HWND getHiddenCBox();

HWND getArchiveCBox();

HWND getCompressedCBox();

HWND getEncryptedCBox();

HWND getIndexedCBox();

HWND getPicture();

void setHWnd(HWND hWnd);

void setMainMenu(HMENU mainMenu);

void setFileMenu(HMENU fileMenu);

void setCreatedDateTimeLabel(HWND createdDateTimeLabel);

void setEditedDateTimeLabel(HWND editedDateTimeLabel);

void setOpenedDateTimeLabel(HWND openedDateTimeLabel);

void setPathLabel(HWND pathLabel);

void setFileSizeLabel(HWND fileSizeLabel);

void setFileTypeLabel(HWND fileTypeLabel);

void setReadOnlyCBox(HWND readOnlyCBox);

void setHiddenCBox(HWND hiddenCBox);

void setArchiveCBox(HWND archiveCBox);

void setCompressedCBox(HWND compressedCBox);

void setEncryptedCBox(HWND encryptedCBox);

void setIndexedCBox(HWND indexedCBox);

void setPicture(HWND picture);

private:

HWND hWnd;

HMENU mainMenu;

HMENU fileMenu;

HWND createdDateTimeLabel;

HWND editedDateTimeLabel;

HWND openedDateTimeLabel;

HWND pathLabel;

HWND fileSizeLabel;

HWND fileTypeLabel;

HWND readOnlyCBox;

HWND hiddenCBox;

HWND archiveCBox;

HWND compressedCBox;

HWND encryptedCBox;

HWND indexedCBox;

HWND picture;

};

FileAttributeManager.h

#pragma once

#include <Windows.h>

#include <string>

class FileAttributeManager

{

public:

FileAttributeManager();

~FileAttributeManager();

bool attributeStatus(BY\_HANDLE\_FILE\_INFORMATION file, DWORD attribute);

void setAttributeStatus(std::string fileName, DWORD attribute,

bool status);

void setCompressedStatus(HANDLE file, bool status);

void setEncryptedStatus(std::string fileName, bool status);

private:

bool hasAttribute(DWORD attributes, DWORD attribute);

};

FileDateTimeManager.h

#pragma once

#include <Windows.h>

#include <string>

#include <cstdio>

class FileDateTimeManager

{

public:

FileDateTimeManager();

~FileDateTimeManager();

std::string getCreatedDateTimeFile(BY\_HANDLE\_FILE\_INFORMATION file);

std::string getEditedDateTimeFile(BY\_HANDLE\_FILE\_INFORMATION file);

std::string getOpenedDateTimeFile(BY\_HANDLE\_FILE\_INFORMATION file);

};

FileSizeManager.h

#pragma once

#include <Windows.h>

class FileSizeManager

{

public:

FileSizeManager();

~FileSizeManager();

long getFileSize(BY\_HANDLE\_FILE\_INFORMATION file);

};

FileTypeManager.h

#pragma once

#include <Windows.h>

#include <string>

class FileTypeManager

{

public:

FileTypeManager();

~FileTypeManager();

HICON getIcon(std::string fileName);

std::string getFileType(std::string fileName);

};

FileUtil.h

#pragma once

#include <Windows.h>

#include <string>

class FileUtil

{

public:

HANDLE getFileHandle(std::string fileName);

HANDLE getFileHandleForCompression(std::string fileName);

BY\_HANDLE\_FILE\_INFORMATION getFileInformation(HANDLE file);

static FileUtil\* getFileUtil();

private:

FileUtil();

~FileUtil();

static FileUtil\* fileUtil;

};

**Исходные файлы:**

start.cpp

#include "Controller.h";

LONG CALLBACK WinEvents(HWND hwnd, UINT winEvent, WPARAM wParam,

LPARAM lParam);

MainForm &mainForm = \*(new MainForm());

Controller &controller = \*(new Controller(mainForm));

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,

LPSTR lpCmdLine, int nShowCmd)

{

return controller.run(hInstance, hPrevInstance, lpCmdLine, nShowCmd,

WinEvents);

}

LONG CALLBACK WinEvents(HWND hwnd, UINT winEvent, WPARAM wParam,

LPARAM lParam)

{

return controller.mainEvents(hwnd, winEvent, wParam, lParam);

}

Controller.cpp

#include "Controller.h"

std::map<DWORD, DWORD> attributesActionMap = { { FILE\_ATTRIBUTE\_READONLY, READONLY\_CHECK\_ACTION },

{ FILE\_ATTRIBUTE\_HIDDEN, HIDDEN\_CHECK\_ACTION },

{ FILE\_ATTRIBUTE\_ARCHIVE, ARCHIVE\_CHECK\_ACTION },

{ FILE\_ATTRIBUTE\_COMPRESSED, COMPRESSED\_CHECK\_ACTION },

{ FILE\_ATTRIBUTE\_ENCRYPTED, ENCRYPTED\_CHECK\_ACTION},

{ FILE\_ATTRIBUTE\_NOT\_CONTENT\_INDEXED, INDEXED\_CHECK\_ACTION } };

std::map<UINT, events> mainEventMap = { { WM\_DESTROY, &Controller::quit },

{ WM\_COMMAND, &Controller::commandEvents } };

std::map<UINT, commEvents> commandEventMap = { { FILE\_MENU\_OPEN\_ACTION, &Controller::selectFileData },

{ FILE\_MENU\_EXIT\_ACTION, &Controller::closeWindow },

{ READONLY\_CHECK\_ACTION, &Controller::readOnlyCheckAction },

{ HIDDEN\_CHECK\_ACTION, &Controller::hiddenCheckAction },

{ ARCHIVE\_CHECK\_ACTION, &Controller::archiveCheckAction },

{ COMPRESSED\_CHECK\_ACTION, &Controller::compressedCheckAction },

{ ENCRYPTED\_CHECK\_ACTION, &Controller::encryptedCheckAction },

{ INDEXED\_CHECK\_ACTION, &Controller::indexedCheckAction } };

Controller::Controller()

{

}

Controller::Controller(MainForm form)

{

this->form = form;

}

Controller::~Controller()

{

}

int Controller::run(HINSTANCE hInstance, HINSTANCE hPrevInstance,

LPSTR lpCmdLine, int nShowCmd, WNDPROC wndProc)

{

MSG msg;

form.registerWClass(hInstance, wndProc);

HWND hWnd = CreateWindow(WCLASS\_NAME, TITLE, WS\_OVERLAPPEDWINDOW, MAIN\_X\_POS,

MAIN\_Y\_POS, MAIN\_WIDTH, MAIN\_HEIGHT, HWND\_DESKTOP, NULL, hInstance,

NULL);

ShowWindow(hWnd, nShowCmd);

UpdateWindow(hWnd);

form.setHWnd(hWnd);

form.initWindow();

while (GetMessage(&msg, NULL, NULL, NULL))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return msg.wParam;

}

LONG Controller::mainEvents(HWND hwnd, UINT winEvent, WPARAM wParam, LPARAM lParam)

{

if (mainEventMap.find(winEvent) != mainEventMap.end())

{

(this->\*mainEventMap[winEvent])(wParam);

}

else

{

return DefWindowProc(hwnd, winEvent, wParam, lParam);

}

return 0;

}

void Controller::commandEvents(WPARAM wParam)

{

UINT eventCode = LOWORD(wParam);

if (commandEventMap.find(eventCode) != commandEventMap.end())

{

(this->\*commandEventMap[eventCode])();

}

}

void Controller::closeWindow()

{

DestroyWindow(form.getHWnd());

}

void Controller::quit(WPARAM wParam)

{

PostQuitMessage(0);

}

void Controller::selectFileData()

{

std::string fileName = form.selectFile();

if (fileName != "")

{

ofn = fileName;

HANDLE file = FileUtil::getFileUtil()->getFileHandle(ofn);

BY\_HANDLE\_FILE\_INFORMATION fileInf

= FileUtil::getFileUtil()->getFileInformation(file);

CloseHandle(file);

viewCreatedDateTimeFile(fileInf);

viewEditedDateTimeFile(fileInf);

viewOpenedDateTimeFile(fileInf);

viewFilePath(ofn);

viewFileSize(fileInf);

viewStatus(fileInf, FILE\_ATTRIBUTE\_READONLY);

viewStatus(fileInf, FILE\_ATTRIBUTE\_HIDDEN);

viewStatus(fileInf, FILE\_ATTRIBUTE\_ARCHIVE);

viewStatus(fileInf, FILE\_ATTRIBUTE\_COMPRESSED);

viewStatus(fileInf, FILE\_ATTRIBUTE\_ENCRYPTED);

viewStatus(fileInf, FILE\_ATTRIBUTE\_NOT\_CONTENT\_INDEXED);

form.setAllCheckBoxEnableStatus(true);

viewIcon();

viewFileType();

}

}

void Controller::viewCreatedDateTimeFile(BY\_HANDLE\_FILE\_INFORMATION file)

{

form.setToLabel(form.getCreatedDateTimeLabel(),

CREATED\_DATETIME\_LABEL + dateTimeManager.getCreatedDateTimeFile(file));

}

void Controller::viewEditedDateTimeFile(BY\_HANDLE\_FILE\_INFORMATION file)

{

form.setToLabel(form.getEditedDateTimeLabel(),

EDITED\_DATETIME\_LABEL + dateTimeManager.getEditedDateTimeFile(file));

}

void Controller::viewOpenedDateTimeFile(BY\_HANDLE\_FILE\_INFORMATION file)

{

form.setToLabel(form.getOpenedDateTimeLabel(),

OPENED\_DATETIME\_LABEL + dateTimeManager.getOpenedDateTimeFile(file));

}

void Controller::viewFilePath(std::string path)

{

form.setToLabel(form.getPathLabel(), PATH\_LABEL + path);

}

void Controller::viewFileSize(BY\_HANDLE\_FILE\_INFORMATION file)

{

std::string sizeStr = std::to\_string(fileSizeManager.getFileSize(file));

form.setToLabel(form.getFileSizeLabel(), FILE\_SIZE\_LABEL + sizeStr

+ FILE\_SIZE\_LABEL\_SUFFIX);

}

void Controller::viewStatus(BY\_HANDLE\_FILE\_INFORMATION file, DWORD attribute)

{

CheckDlgButton(form.getHWnd(), attributesActionMap[attribute],

(fileAttributeManager.attributeStatus(file, attribute)

? BST\_CHECKED : BST\_UNCHECKED));

}

void Controller::viewIcon()

{

form.addPicture(fileTypeManager.getIcon(ofn));

}

void Controller::viewFileType()

{

form.setToLabel(form.getFileTypeLabel(),

FILE\_TYPE\_LABEL + fileTypeManager.getFileType(ofn));

}

void Controller::readOnlyCheckAction()

{

changeCheckBoxStatus(attributesActionMap[FILE\_ATTRIBUTE\_READONLY]);

changeAttributeStatus(FILE\_ATTRIBUTE\_READONLY);

}

void Controller::hiddenCheckAction()

{

changeCheckBoxStatus(attributesActionMap[FILE\_ATTRIBUTE\_HIDDEN]);

changeAttributeStatus(FILE\_ATTRIBUTE\_HIDDEN);

}

void Controller::archiveCheckAction()

{

changeCheckBoxStatus(attributesActionMap[FILE\_ATTRIBUTE\_ARCHIVE]);

changeAttributeStatus(FILE\_ATTRIBUTE\_ARCHIVE);

}

void Controller::compressedCheckAction()

{

changeCompressedStatus();

changeCheckBoxStatus(attributesActionMap[FILE\_ATTRIBUTE\_COMPRESSED]);

}

void Controller::encryptedCheckAction()

{

changeEncryptedStatus();

changeCheckBoxStatus(attributesActionMap[FILE\_ATTRIBUTE\_ENCRYPTED]);

}

void Controller::indexedCheckAction()

{

changeCheckBoxStatus(attributesActionMap[FILE\_ATTRIBUTE\_NOT\_CONTENT\_INDEXED]);

changeAttributeStatus(FILE\_ATTRIBUTE\_NOT\_CONTENT\_INDEXED);

}

void Controller::changeCheckBoxStatus(DWORD attributeAction)

{

if (IsDlgButtonChecked(form.getHWnd(), attributeAction))

{

CheckDlgButton(form.getHWnd(), attributeAction, BST\_UNCHECKED);

}

else

{

CheckDlgButton(form.getHWnd(), attributeAction, BST\_CHECKED);

}

}

void Controller::changeAttributeStatus(DWORD attribute)

{

if (ofn != "")

{

fileAttributeManager.setAttributeStatus(ofn, attribute,

IsDlgButtonChecked(form.getHWnd(), attributesActionMap[attribute]));

}

}

void Controller::changeCompressedStatus()

{

if (ofn != "")

{

if (!IsDlgButtonChecked(form.getHWnd(), attributesActionMap[FILE\_ATTRIBUTE\_COMPRESSED])

&& IsDlgButtonChecked(form.getHWnd(), attributesActionMap[FILE\_ATTRIBUTE\_ENCRYPTED]))

{

encryptedCheckAction();

}

HANDLE file = FileUtil::getFileUtil()->getFileHandleForCompression(ofn);

fileAttributeManager.setCompressedStatus(file,

!IsDlgButtonChecked(form.getHWnd(), COMPRESSED\_CHECK\_ACTION));

CloseHandle(file);

}

}

void Controller::changeEncryptedStatus()

{

if (ofn != "")

{

if (!IsDlgButtonChecked(form.getHWnd(), attributesActionMap[FILE\_ATTRIBUTE\_ENCRYPTED])

&& IsDlgButtonChecked(form.getHWnd(), attributesActionMap[FILE\_ATTRIBUTE\_COMPRESSED]))

{

compressedCheckAction();

}

fileAttributeManager.setEncryptedStatus(ofn,

!IsDlgButtonChecked(form.getHWnd(), ENCRYPTED\_CHECK\_ACTION));

}

}

std::string Controller::GetOfn()

{

return ofn;

}

MainForm Controller::getForm()

{

return form;

}

FileAttributeManager Controller::getFileAttributeManager()

{

return fileAttributeManager;

}

FileDateTimeManager Controller::getDateTimeManager()

{

return dateTimeManager;

}

FileSizeManager Controller::getFileSizeManager()

{

return fileSizeManager;

}

FileTypeManager Controller::getFileTypeManager()

{

return fileTypeManager;

}

void Controller::setOfn(std::string ofn)

{

this->ofn = ofn;

}

void Controller::setForm(MainForm form)

{

this->form = form;

}

void Controller::setFileAttributeManager(FileAttributeManager fileAttributeManager)

{

this->fileAttributeManager = fileAttributeManager;

}

void Controller::setDateTimeManager(FileDateTimeManager dateTimeManager)

{

this->dateTimeManager = dateTimeManager;

}

void Controller::setFileSizeManager(FileSizeManager fileSizeManager)

{

this->fileSizeManager = fileSizeManager;

}

void Controller::setFileTypeManager(FileTypeManager fileTypeManager)

{

this->fileTypeManager = fileTypeManager;

}

MainForm.cpp

#include "MainForm.h"

MainForm::MainForm()

{

}

MainForm::~MainForm()

{

}

void MainForm::registerWClass(HINSTANCE hInstance, WNDPROC wndProc)

{

WNDCLASS wClass;

memset(&wClass, 0, sizeof(WNDCLASS));

wClass.hbrBackground = (HBRUSH)COLOR\_WINDOW;

wClass.hCursor = LoadCursor(NULL, IDC\_ARROW);

wClass.hInstance = hInstance;

wClass.lpszClassName = WCLASS\_NAME;

wClass.lpfnWndProc = wndProc;

wClass.cbWndExtra = 0;

wClass.cbClsExtra = 0;

if (!RegisterClass(&wClass))

{

MessageBox(NULL, "Error: custom class not register", "Error", MB\_OK);

}

}

std::string MainForm::selectFile()

{

OPENFILENAME ofn;

char szFile[DEFAULT\_STR\_SIZE];

HANDLE hf;

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hWnd;

ofn.lpstrFile = szFile;

ofn.lpstrFile[0] = '\0';

ofn.nMaxFile = sizeof(szFile);

ofn.lpstrFilter = "All\0\*.\*\0";

ofn.nFilterIndex = 1;

ofn.lpstrFileTitle = NULL;

ofn.nMaxFileTitle = 0;

ofn.lpstrInitialDir = NULL;

ofn.Flags = OFN\_PATHMUSTEXIST | OFN\_FILEMUSTEXIST;

GetOpenFileName(&ofn);

return ofn.lpstrFile;

}

void MainForm::initWindow()

{

addMenus();

addLabels();

addCheckBoxes();

}

void MainForm::addMenus()

{

mainMenu = CreateMenu();

fileMenu = CreateMenu();

AppendMenu(fileMenu, MF\_STRING, FILE\_MENU\_OPEN\_ACTION, FILE\_MENU\_OPEN);

AppendMenu(fileMenu, MF\_SEPARATOR, NULL, NULL);

AppendMenu(fileMenu, MF\_STRING, FILE\_MENU\_EXIT\_ACTION, FILE\_MENU\_EXIT);

AppendMenu(mainMenu, MF\_POPUP, (UINT\_PTR)fileMenu, FILE\_MENU);

AppendMenu(mainMenu, MF\_STRING, NULL, HELP);

SetMenu(hWnd, mainMenu);

}

void MainForm::addLabels()

{

createdDateTimeLabel = CreateWindow("static", CREATED\_DATETIME\_LABEL,

WS\_VISIBLE | WS\_CHILD, DEFAULT\_X\_POS, CREATED\_DATETIME\_LABEL\_Y\_POS,

LABELS\_WIDHT, LABELS\_HEIGHT, hWnd, NULL, NULL, NULL);

editedDateTimeLabel = CreateWindow("static", EDITED\_DATETIME\_LABEL,

WS\_VISIBLE | WS\_CHILD, DEFAULT\_X\_POS, EDITED\_DATETIME\_LABEL\_Y\_POS,

LABELS\_WIDHT, LABELS\_HEIGHT, hWnd, NULL, NULL, NULL);

openedDateTimeLabel = CreateWindow("static", OPENED\_DATETIME\_LABEL,

WS\_VISIBLE | WS\_CHILD, DEFAULT\_X\_POS, OPENED\_DATETIME\_LABEL\_Y\_POS,

LABELS\_WIDHT, LABELS\_HEIGHT, hWnd, NULL, NULL, NULL);

pathLabel = CreateWindow("static", PATH\_LABEL,

WS\_VISIBLE | WS\_CHILD, DEFAULT\_X\_POS, PATH\_LABEL\_Y\_POS,

LABELS\_WIDHT, LABELS\_HEIGHT, hWnd, NULL, NULL, NULL);

fileSizeLabel = CreateWindow("static", FILE\_SIZE\_LABEL,

WS\_VISIBLE | WS\_CHILD, DEFAULT\_X\_POS, FILE\_SIZE\_LABEL\_Y\_POS,

LABELS\_WIDHT, LABELS\_HEIGHT, hWnd, NULL, NULL, NULL);

fileTypeLabel = CreateWindow("static", FILE\_TYPE\_LABEL,

WS\_VISIBLE | WS\_CHILD, FILE\_TYPE\_LABEL\_X\_POS, FILE\_TYPE\_LABEL\_Y\_POS,

LABELS\_WIDHT, LABELS\_HEIGHT, hWnd, NULL, NULL, NULL);

}

void MainForm::addCheckBoxes()

{

readOnlyCBox = CreateWindow("button", READONLY\_CBOX, WS\_VISIBLE | WS\_CHILD | BS\_CHECKBOX,

DEFAULT\_X\_POS, READONLY\_CBOX\_Y\_POS, CHECKBOX\_WIDTH,

CHECKBOX\_HEIGHT, hWnd, (HMENU)READONLY\_CHECK\_ACTION, NULL, NULL);

hiddenCBox = CreateWindow("button", HIDDEN\_CBOX, WS\_VISIBLE | WS\_CHILD | BS\_CHECKBOX,

HIDDEN\_CBOX\_X\_POS, HIDDEN\_CBOX\_Y\_POS, CHECKBOX\_WIDTH,

CHECKBOX\_HEIGHT, hWnd, (HMENU)HIDDEN\_CHECK\_ACTION, NULL, NULL);

archiveCBox = CreateWindow("button", ARCHIVE\_CBOX, WS\_VISIBLE | WS\_CHILD | BS\_CHECKBOX,

ARCHIVE\_CBOX\_X\_POS, ARCHIVE\_CBOX\_Y\_POS, CHECKBOX\_WIDTH,

CHECKBOX\_HEIGHT, hWnd, (HMENU)ARCHIVE\_CHECK\_ACTION, NULL, NULL);

compressedCBox = CreateWindow("button", COMPRESSED\_CBOX, WS\_VISIBLE | WS\_CHILD | BS\_CHECKBOX,

DEFAULT\_X\_POS, COMPRESSED\_CBOX\_Y\_POS, CHECKBOX\_WIDTH,

CHECKBOX\_HEIGHT, hWnd, (HMENU)COMPRESSED\_CHECK\_ACTION, NULL, NULL);

encryptedCBox = CreateWindow("button", ENCRYPTED\_CBOX, WS\_VISIBLE | WS\_CHILD | BS\_CHECKBOX,

DEFAULT\_X\_POS, ENCRYPTED\_CBOX\_Y\_POS, CHECKBOX\_WIDTH,

CHECKBOX\_HEIGHT, hWnd, (HMENU)ENCRYPTED\_CHECK\_ACTION, NULL, NULL);

indexedCBox = CreateWindow("button", INDEXED\_CBOX, WS\_VISIBLE | WS\_CHILD | BS\_CHECKBOX,

DEFAULT\_X\_POS, INDEXED\_CBOX\_Y\_POS, CHECKBOX\_WIDTH,

CHECKBOX\_HEIGHT, hWnd, (HMENU)INDEXED\_CHECK\_ACTION, NULL, NULL);

setAllCheckBoxEnableStatus(false);

}

void MainForm::setAllCheckBoxEnableStatus(bool status)

{

EnableWindow(readOnlyCBox, status);

EnableWindow(hiddenCBox, status);

EnableWindow(archiveCBox, status);

EnableWindow(compressedCBox, status);

EnableWindow(encryptedCBox, status);

EnableWindow(indexedCBox, status);

}

void MainForm::addPicture(HICON bitmap1)

{

PAINTSTRUCT ps;

ICONINFO inf;

GetIconInfo(bitmap1, &inf);

BITMAP bm;

HBITMAP bitmap = inf.hbmColor;

GetObject(bitmap, sizeof(bm), &bm);

picture = CreateWindow("static", INDEXED\_CBOX, WS\_VISIBLE | WS\_CHILD | SS\_BITMAP,

PIC\_X\_POS, DEFAULT\_Y\_LAYOUT, bm.bmWidth,

bm.bmHeight, hWnd, NULL, NULL, NULL);

HDC hdc = GetDC(picture);

HDC memBit = CreateCompatibleDC(hdc);

SelectObject(memBit, bitmap);

ReleaseDC(picture, hdc);

hdc = BeginPaint(picture, &ps);

BitBlt(hdc, 0, 0, bm.bmWidth, bm.bmHeight, memBit, 0, 0, SRCCOPY);

EndPaint(picture, &ps);

}

std::string MainForm::getFromLabel(HWND label)

{

LPSTR str = new char[DEFAULT\_STR\_SIZE];

GetWindowText(label, str, DEFAULT\_STR\_SIZE);

return str;

}

void MainForm::setToLabel(HWND label, std::string str)

{

SetWindowText(label, (LPCSTR)str.c\_str());

}

HWND MainForm::getHWnd()

{

return hWnd;

}

HMENU MainForm::getMainMenu()

{

return mainMenu;

}

HMENU MainForm::getFileMenu()

{

return fileMenu;

}

HWND MainForm::getCreatedDateTimeLabel()

{

return createdDateTimeLabel;

}

HWND MainForm::getEditedDateTimeLabel()

{

return editedDateTimeLabel;

}

HWND MainForm::getOpenedDateTimeLabel()

{

return openedDateTimeLabel;

}

HWND MainForm::getPathLabel()

{

return pathLabel;

}

HWND MainForm::getFileSizeLabel()

{

return fileSizeLabel;

}

HWND MainForm::getFileTypeLabel()

{

return fileTypeLabel;

}

HWND MainForm::getReadOnlyCBox()

{

return readOnlyCBox;

}

HWND MainForm::getHiddenCBox()

{

return hiddenCBox;

}

HWND MainForm::getArchiveCBox()

{

return archiveCBox;

}

HWND MainForm::getCompressedCBox()

{

return compressedCBox;

}

HWND MainForm::getEncryptedCBox()

{

return encryptedCBox;

}

HWND MainForm::getIndexedCBox()

{

return indexedCBox;

}

HWND MainForm::getPicture()

{

return picture;

}

void MainForm::setHWnd(HWND hWnd)

{

this->hWnd = hWnd;

}

void MainForm::setMainMenu(HMENU mainMenu)

{

this->mainMenu = mainMenu;

}

void MainForm::setFileMenu(HMENU fileMenu)

{

this->fileMenu = fileMenu;

}

void MainForm::setCreatedDateTimeLabel(HWND createdDateTimeLabel)

{

this->createdDateTimeLabel = createdDateTimeLabel;

}

void MainForm::setEditedDateTimeLabel(HWND editedDateTimeLabel)

{

this->editedDateTimeLabel = editedDateTimeLabel;

}

void MainForm::setOpenedDateTimeLabel(HWND openedDateTimeLabel)

{

this->openedDateTimeLabel = openedDateTimeLabel;

}

void MainForm::setPathLabel(HWND pathLabel)

{

this->pathLabel = pathLabel;

}

void MainForm::setFileSizeLabel(HWND fileSizeLabel)

{

this->fileSizeLabel = fileSizeLabel;

}

void MainForm::setFileTypeLabel(HWND fileTypeLabel)

{

this->fileTypeLabel = fileTypeLabel;

}

void MainForm::setReadOnlyCBox(HWND readOnlyCBox)

{

this->readOnlyCBox = readOnlyCBox;

}

void MainForm::setHiddenCBox(HWND hiddenCBox)

{

this->hiddenCBox = hiddenCBox;

}

void MainForm::setArchiveCBox(HWND archiveCBox)

{

this->archiveCBox = archiveCBox;

}

void MainForm::setCompressedCBox(HWND compressedCBox)

{

this->compressedCBox = compressedCBox;

}

void MainForm::setEncryptedCBox(HWND encryptedCBox)

{

this->encryptedCBox = encryptedCBox;

}

void MainForm::setIndexedCBox(HWND indexedCBox)

{

this->indexedCBox = indexedCBox;

}

void MainForm::setPicture(HWND picture)

{

this->picture = picture;

}

FileAttributeManager.cpp

#include "FileAttributeManager.h"

FileAttributeManager::FileAttributeManager()

{

}

FileAttributeManager::~FileAttributeManager()

{

}

bool FileAttributeManager::attributeStatus(BY\_HANDLE\_FILE\_INFORMATION file,

DWORD attribute)

{

return hasAttribute(file.dwFileAttributes, attribute);

}

void FileAttributeManager::setAttributeStatus(std::string fileName, DWORD attribute,

bool status)

{

DWORD attributes = GetFileAttributes((LPCSTR)fileName.c\_str());

if (hasAttribute(attributes, attribute) != status)

{

SetFileAttributes((LPCSTR)fileName.c\_str(),

(status) ? (attributes + attribute) : (attributes - attribute));

}

}

void FileAttributeManager::setCompressedStatus(HANDLE file, bool status)

{

DWORD dummy;

USHORT compression = (status) ? COMPRESSION\_FORMAT\_DEFAULT : COMPRESSION\_FORMAT\_NONE;

DeviceIoControl(file, FSCTL\_SET\_COMPRESSION,

&compression, sizeof(USHORT), NULL, 0, (LPDWORD)&dummy, NULL);

}

void FileAttributeManager::setEncryptedStatus(std::string fileName, bool status)

{

(status) ? EncryptFile(fileName.c\_str()) : DecryptFile(fileName.c\_str(), 0);

}

bool FileAttributeManager::hasAttribute(DWORD attributes, DWORD attribute)

{

return (attributes & attribute) == attribute;

}

FileDateTimeManager.cpp

#include "FileDateTimeManager.h"

FileDateTimeManager::FileDateTimeManager()

{

}

FileDateTimeManager::~FileDateTimeManager()

{

}

std::string FileDateTimeManager::getCreatedDateTimeFile(BY\_HANDLE\_FILE\_INFORMATION file)

{

SYSTEMTIME createdTime;

FileTimeToLocalFileTime(&(file.ftCreationTime), &(file.ftCreationTime));

FileTimeToSystemTime(&(file.ftCreationTime), &createdTime);

char buff[260];

sprintf\_s(buff, "%02d.%02d.%d %02d:%02d:%02d", createdTime.wDay, createdTime.wMonth,

createdTime.wYear, createdTime.wHour, createdTime.wMinute, createdTime.wSecond);

return buff;

}

std::string FileDateTimeManager::getEditedDateTimeFile(BY\_HANDLE\_FILE\_INFORMATION file)

{

SYSTEMTIME editedTime;

FileTimeToLocalFileTime(&(file.ftLastWriteTime), &(file.ftLastWriteTime));

FileTimeToSystemTime(&(file.ftLastWriteTime), &editedTime);

char buff[260];

sprintf\_s(buff, "%02d.%02d.%d %02d:%02d:%02d", editedTime.wDay, editedTime.wMonth,

editedTime.wYear, editedTime.wHour, editedTime.wMinute, editedTime.wSecond);

return buff;

}

std::string FileDateTimeManager::getOpenedDateTimeFile(BY\_HANDLE\_FILE\_INFORMATION file)

{

SYSTEMTIME openedTime;

FileTimeToLocalFileTime(&(file.ftLastAccessTime), &(file.ftLastAccessTime));

FileTimeToSystemTime(&(file.ftLastAccessTime), &openedTime);

char buff[260];

sprintf\_s(buff, "%02d.%02d.%d %02d:%02d:%02d", openedTime.wDay, openedTime.wMonth,

openedTime.wYear, openedTime.wHour, openedTime.wMinute, openedTime.wSecond);

return buff;

}

FileSizeManager.cpp

#include "FileSizeManager.h"

FileSizeManager::FileSizeManager()

{

}

FileSizeManager::~FileSizeManager()

{

}

long FileSizeManager::getFileSize(BY\_HANDLE\_FILE\_INFORMATION file)

{

return file.nFileSizeLow;

}

FileTypeManager.cpp

#include "FileTypeManager.h"

FileTypeManager::FileTypeManager()

{

}

FileTypeManager::~FileTypeManager()

{

}

HICON FileTypeManager::getIcon(std::string fileName)

{

SHFILEINFO flInfo;

SHGetFileInfo(fileName.c\_str(), 0, &flInfo, sizeof(flInfo), SHGFI\_ICON);

return flInfo.hIcon;

}

std::string FileTypeManager::getFileType(std::string fileName)

{

SHFILEINFO flInfo;

SHGetFileInfo(fileName.c\_str(), 0, &flInfo, sizeof(flInfo), SHGFI\_TYPENAME);

return flInfo.szTypeName;

}

FileUtil.cpp

#include "FileUtil.h"

FileUtil\* FileUtil::fileUtil = nullptr;

FileUtil::FileUtil()

{

}

FileUtil::~FileUtil()

{

}

HANDLE FileUtil::getFileHandle(std::string fileName)

{

return CreateFile((LPCSTR)fileName.c\_str(), GENERIC\_READ, FILE\_SHARE\_READ,

NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

}

BY\_HANDLE\_FILE\_INFORMATION FileUtil::getFileInformation(HANDLE file)

{

BY\_HANDLE\_FILE\_INFORMATION fileInformation;

GetFileInformationByHandle(file, &fileInformation);

return fileInformation;

}

HANDLE FileUtil::getFileHandleForCompression(std::string fileName)

{

return CreateFile((LPCSTR)fileName.c\_str(), GENERIC\_READ | GENERIC\_WRITE, NULL,

NULL, OPEN\_EXISTING, NULL, NULL);

}

FileUtil\* FileUtil::getFileUtil()

{

if (fileUtil == nullptr)

{

fileUtil = new FileUtil();

}

return fileUtil;

}