Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Операционные системы и системное программирование (ОСиСП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему

ФАЙЛОВЫЙ МЕНЕДЖЕР

БГУИР КР 1-40 01 01 618 ПЗ

Студент гр. 851006 Мискевич П.Л.
Руководитель Жиденко А.Л.

Минск 2020

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой ПОИТ
Лапицкая Н.В.

_____
(подпись)
_____2020 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту Мискевичу Павлу Леонидовичу, 851006

1. Тема работы Файловый менеджер
2. Срок сдачи студентом законченной работы 01.12.2020 г.
3. Исходные данные к работе Документация по WinAPI
4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке)

Введение

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству;
2. Моделирование предметной области и разработка функциональных требований;
3. Проектирование программного средства;
4. Тестирование, проверка работоспособности и анализ полученных результатов;
5. Руководство по установке и использованию;

Заключение

Список использованных источников

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)
1. "Копирование файлов из одного каталога в другой. Схема алгоритма", А1, схема алгоритма, чертеж.
6. Консультант по курсовой работе
Жиденко А.Л.
7. Дата выдачи задания 05.09.2020
8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):
раздел 1 к 20.09.2020 – 15 % готовности работы;
разделы 2, 3 к 13.10.2020 – 30 % готовности работы;
разделы 4 к 02.11.2020 – 60 % готовности работы;
раздел 5 к 26.11.2020 – 90 % готовности работы;
оформление пояснительной записки и графического материала к 01.12.2020 – 100 % готовности работы.

РУКОВОДИТЕЛЬ_____Жиденко А.Л.
(подпись)

Задание принял к исполнению Мискевич П.Л. _____
(дата и подпись студента)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Ни одна операционная система на сегодняшний день не может обойтись без удобного и надежного файлового менеджера. Огромное количество нарастающих с каждым днем данных нуждается в грамотной структуризации и разделении. Все современные операционные системы, как правило, включают в свой состав, в первую очередь, именно файловый менеджер, как неотъемлемую часть самой ОС и необходимый компонент для реализации всех возможностей по доступу к файловой системе. При этом такой доступ должен удовлетворять многим, зачастую противоположным условиям, к которым относятся: возможность быстрого поиска и отображения нужной информации, полнота операций над этими данными, гарантированное исключение ошибок при этих операциях, простота и т.д.

Наличие файлового менеджера в самих ОС не останавливает пользователей в поисках программы «для себя». Основная задача – необходимый минимум в сочетании с простотой. Программа реализует в себе все способы взаимодействия с пользователем и другими программами, предоставляемые операционной системой Windows.

Данный файловый менеджер не претендует на звание самого полного по функциональности, а лишь отражает взгляд на то, каким должен быть простой, но вместе с тем функциональный проводник по файловой системе. Доступ к информации о системе полностью осуществляется через API Windows, что делает программу легко переносимой среди ОС этого класса. Программа написана в среде Microsoft Visual Studio 2019 Community Edition.

В этой пояснительной записке отображены следующие этапы написания курсовой работы:

1 Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству.

2 Анализ требований к программному средству и разработка функциональных требований.

3 Проектирование программного средства.

4 Тестирование, проверка работоспособности и анализ полученных результатов.

5 Руководство по установке и использованию.

1 АНАЛИЗ ПРОТОТИПОВ, ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ И ФОРМИРОВАНИЕ ТРЕБОВАНИЙ К ПРОЕКТИРУЕМОМУ ПРОГРАММНОМУ СРЕДСТВУ

1.1 Анализ существующих аналогов

На сегодняшний день существует многочисленное количество файловых менеджеров. При поиске и изучении аналогов было замечено, что пользователи часто пытаются найти программное средство, которое не только отвечает требованиям пользователя, но и имеет весьма приятный интерфейс.

1.1.1 Total Commander – один из наиболее распространённых файловых менеджеров. Разработчик Christian Ghisler [1].

Интерфейс файлового менеджера представлен на рисунке 1.1.

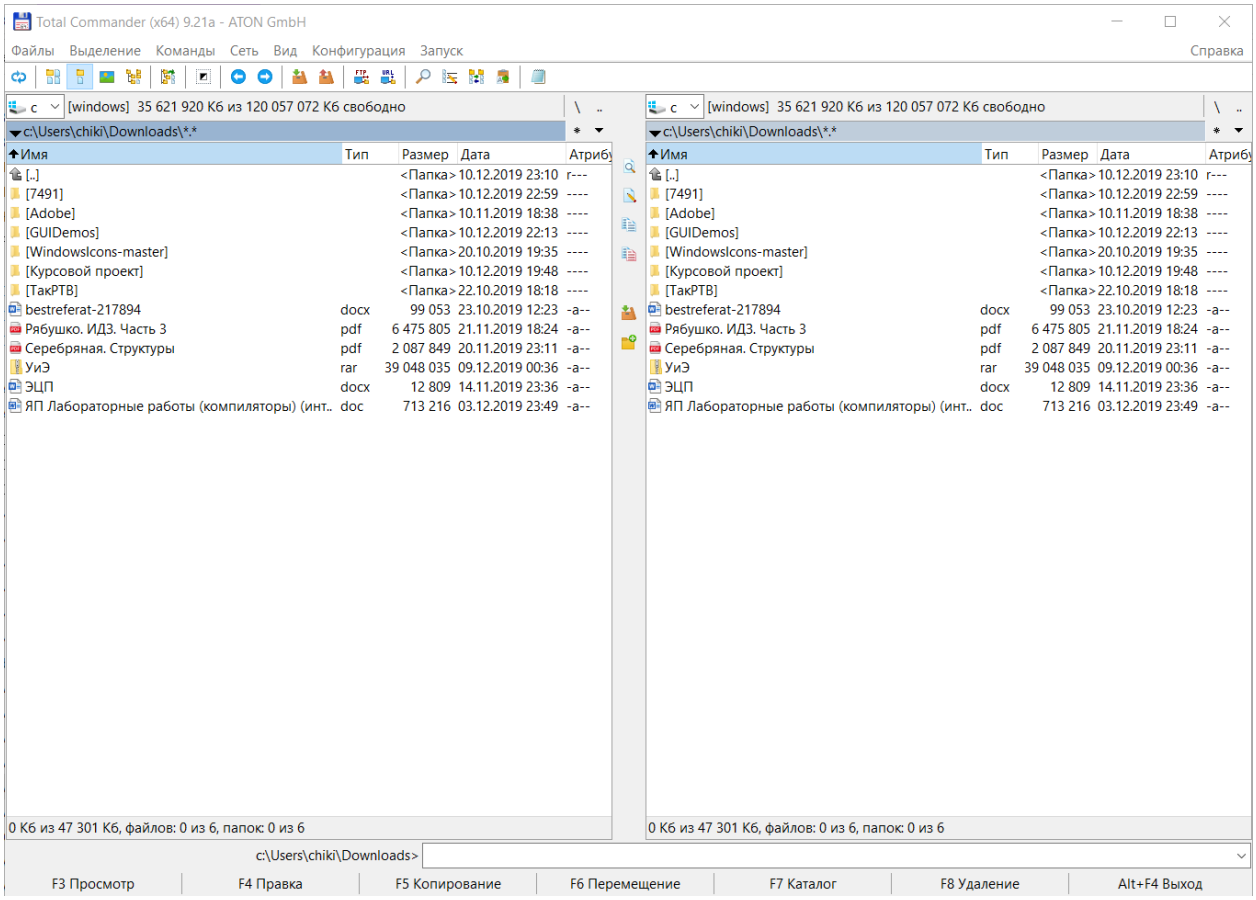

Рисунок 1.1 – Интерфейс Total Commander

Плюсы:

- удобный пользовательский интерфейс;
- кроссплатформенность;
- работа с архивами;
- есть 64-разрядная версия приложения;
- удобная настройка интерфейса;
- большое количество плагинов;
- возможность использования в качестве FTP-клиента.

Минусы:

- платная;
- невозможность управления файлами при помощи клавиатуры;
- требует время на изучение программы и адаптацию к ней.

1.1.2 Far Manager – консольный файловый менеджер для операционных систем семейства Microsoft Windows, разработанный российской компанией FAR Group [2]. Far Manager работает в текстовом режиме и позволяет просто и наглядно выполнять большинство необходимых действий: просматривать файлы и каталоги, редактировать, копировать и переименовывать файлы, а также многое другое.

Внешний вид данного аналога представлен на рисунке 1.2.

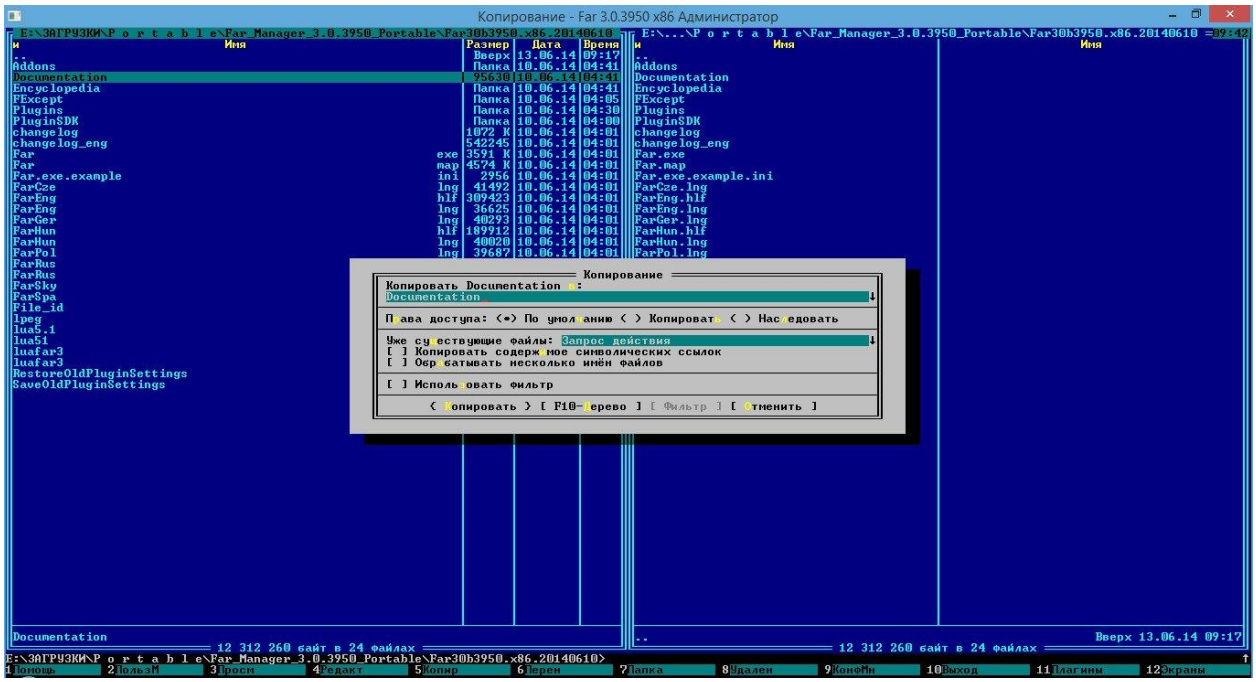

Рисунок 1.2 – Интерфейс Far Manager

Плюсы:

- бесплатный;
- открытый код;
- возможность шифрования файлов;
- большое количество плагинов;
- работа с реестром;
- полноценная поддержка командной строки;
- возможность просмотра списка процессов.

Минусы:

- редко обновляется;
- отсутствие функции «drag-and-drop»;
- старый неудобный интерфейс.

1.2 Постановка задачи

В задачу курсового проекта входит создание оконного приложения «Файловый менеджер» на языке C++ с использованием WinAPI в интегрированной среде разработки Microsoft Visual Studio 2019.

Для того, чтобы программное средство можно было считать файловым менеджером, на основе анализа популярных клиентов, в конечном приложении необходимо наличие следующих возможностей:

- вывод на экран директорий и файлов, их размер и атрибуты;
- открытие директорий и файлов;
- удаление объектов;
- создание каталогов и файлов;
- перемещение объектов;
- копирование объектов.

2 МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ И РАЗРАБОТКА ФУНКЦИОНАЛЬНЫХ ТРЕБОВАНИЙ

2.1 Основные сведения о файловой системе

2.1.1 Файлы

Файл – поименованная область на диске или другом носителе информации. В файлах могут храниться тексты программ, документы, готовые к выполнению программы и любые другие данные.

Атрибуты файла – совокупность байтов, выделяющих файл из множества других файлов. Атрибутами файла являются:

– имя файла и тип содержимого;

– дата и время создания файла;

– имя владельца файла;

– размер файла;

– права доступа к файлу;

– метод доступа к файлу.

Имя файла состоит из двух частей. Собственно имя содержит от 1 до 256 букв, цифр и других символов. Затем пишут точку, после которой пишут расширение файла. Это как фамилия или как профессия. Расширение может содержать до 8 символов, но может и отсутствовать.

Обычно расширение показывает тип файла. Например, текстовые файлы часто имеют расширения «TXT» и «DOC». Файлы с программами имеют расширения «EXE» и «COM». Рисунки имеют расширения «JPEG», «GIF», «BMP» и так далее.

Имя и расширение можно набирать вперемешку большими и маленькими буквами. Windows в данном случае не различает большие и маленькие буквы.

Над файлом можно производить такие операции, как копирование с одного диска на другой, удаление, переименование.

Несколько файлов можно объединить под одной оболочкой – папкой или каталогом.

2.1.2 Каталоги

Каталог – это специальное место на диске, в котором хранятся имена файлов, сведения о размерах файлов, времени их последнего обновления, атрибуты (свойства) файлов и т.д.

Если в каталоге хранится имя файла, то говорят, что этот файл находится в данном каталоге. В операционных системах Windows и MS-DOS папки и файлы образуют на дисках иерархическую файловую структуру. Необходимо знать, что понятия папка и каталог – это одно и тоже.

Организация файловой структуры очень проста. Файлы находятся в папках. Папки вложены в другие папки, более высокого уровня. Папка самого высокого уровня называется корневой – она одна на каждом диске. Назначение файловой структуры – обеспечить однозначное отыскание любого файла, если известно его имя и путь поиска. Путь поиска начинается с корневой папки (ее имя совпадает с обозначением диска) и далее ведет через все вложенные папки к той папке, где находится разыскиваемый файл. Создание и обслуживание файловой структуры – это одна из основных функций операционной системы.

2.1.3 Файловые системы

Файловая система – это часть операционной системы, назначение которой состоит в том, чтобы обеспечить пользователю удобный интерфейс при работе с данными, хранящимися на диске, и обеспечить совместное использование файлов несколькими пользователями и процессами.

В широком смысле понятие «файловая система» включает:

а) совокупность всех файлов на диске;

б) наборы структур данных, используемых для управления файлами, такие, например, как каталоги файлов, дескрипторы файлов, таблицы распределения свободного и занятого пространства на диске;

в) комплекс системных программных средств, реализующих управление файлами, в частности: создание, уничтожение, чтение, запись, именование, поиск и другие операции над файлами.

Существует довольно много разных файловых систем, которые отличаются друг от друга внутренним устройством, однако пользователь везде найдёт привычную структуру из вложенных каталогов и файлов. Типичная логическая структура файловой системы представлена на рисунке 2.1. Файловые системы различаются скоростью доступа, надёжностью хранения данных, степенью устойчивости при сбоях, некоторыми дополнительными возможностями. Современные операционные системы поддерживают по несколько типов файловых систем. Хотя, для каждой операционной системы обычно есть одна «традиционная» файловая система, которая предлагается по умолчанию, является универсальной и подходит абсолютному большинству пользователей.

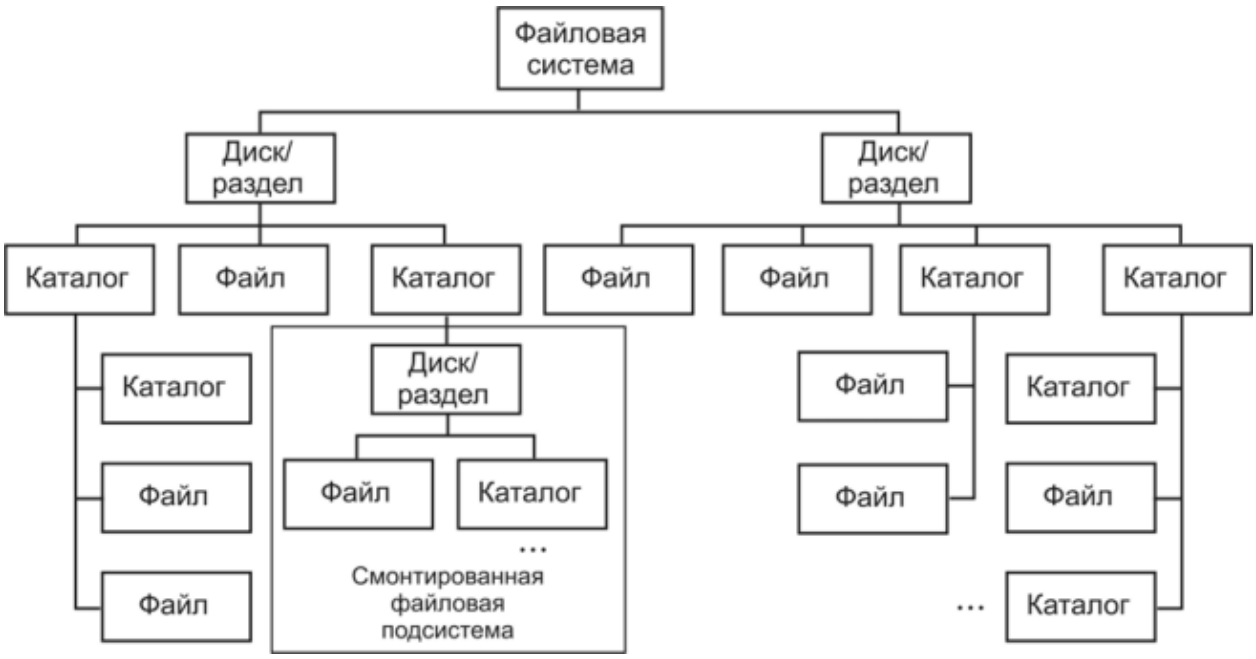

Рисунок 2.1 – Типичная логическая структура файловой системы

Типы файловых систем в ОС Windows:

1 FAT. Это простая классическая архитектура файловой системы, изначально предназначенная для небольших дисков и простых структур папок. Иными словами, файловая система FAT представляет собой групповой метод организации, в котором таблица размещения файлов выделена в отдельную логическую область и находится в начале тома. Для исключения непреднамеренных или случайных ошибок, способных повлиять на корректное отображение таблицы, система, в целях безопасности, хранит копию массива индексных указателей. Самый большой её недостаток – это максимальный объём диска, который составляет всего два гигабайта, что в современных компьютерах практически не встречается. Таким образом, если ваш диск имеет больший объём, то она перестаёт работать.

2 FAT32. 32 – это разрядность системы. Данная версия является обновленной разновидностью предыдущей файловой системы. Размер отдельных файлов на диске с файловой системой FAT32 не может превышать четыре гигабайта. Кроме того, весь раздел должен быть менее восьми терабайт. Если вы пользуетесь более ранней версией Windows, то у вас могут возникнуть некоторые проблемы при форматировании диска. Однако данная система гораздо стабильнее, чем её предшественница, а работа с файлами будет протекать намного быстрее.

3 exFAT. Данный стандарт является обновленной версией файловой системы FAT32. Основными параметрами система exFAT чрезвычайно похожа на FAT32, но главным отличием является устранение ограничений, что позволяет пользователям хранить файлы намного большего размера, чем четыре гигабайта. Также в файловой системе exFAT значительно снижено число перезаписей секторов, ответственных за непосредственное хранение информации, что особенно важно для флэш-накопителей, ввиду необратимого изнашивания ячеек после определённого количества операций записи, и улучшен механизм распределения свободного места.

4 NTFS. Данная система хранения файлов появилась сравнительно недавно и является более современной, чем две предыдущие. Однако, несмотря на огромное количество достоинств, она включает и недостатки. Большинство дисков, выпускаемых сегодня коммерческими фирмами, имеют именно такую файловую систему. Она хранит данные намного лучше, однако достаточно требовательна к ресурсам вашего компьютера. Кроме того, в случае, когда логический диск имеет полную загрузку до 90 процентов, работа файловой системы резко понижается. Также, если операционная система окажется старше, чем Windows XP, то на ней такая файловая система работать просто откажется. Засунув диск в дисковод, ваш компьютер просто не сможет распознать его или будет отмечен как неизвестный раздел. Говоря о достоинствах, можно отметить, что работа такой файловой системы с малыми файлами проходит намного быстрее и качественнее. Самый большой размер, который может иметь диск – это восемнадцать терабайт. Здесь же имеется такое понятие, как фрагментация файлов. При ней работа файловой системы не будет замедляться, а продолжит работу в обычном режиме. Также при использовании NTFS вы можете быть целиком и полностью уверены, что порча файла не произойдет. Система очень экономно расходует пространство на диске и позволяет сжимать файлы до минимального размера, совершенно не портя их. Именно благодаря данной системе стало возможно восстановление данных в случае их потери. Соответственно, если сравнивать эту систему с FAT, то все преимущества налицо. Самое главное, что она вам сможет предложить – это безопасность.

3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

3.1 Интерфейс программного средства

Интерфейс обеспечивает комфортное взаимодействие между пользователем и приложением. При его разработке следует учитывать эргономику и устройства, с какими будет взаимодействовать приложение. Комфорт и удобство пользователя должны быть главным критерием в построении интерфейса и создании дизайна.

Интерфейс данной программы построен по принципу минимализма и имеет предельно простой вид: при запуске программы видна сама сцена с двумя панелями. Двухпанельный интерфейс является одним из самых удобных для файлового менеджера.

Пример интерфейса файлового менеджера продемонстрирован на рисунке 3.1.

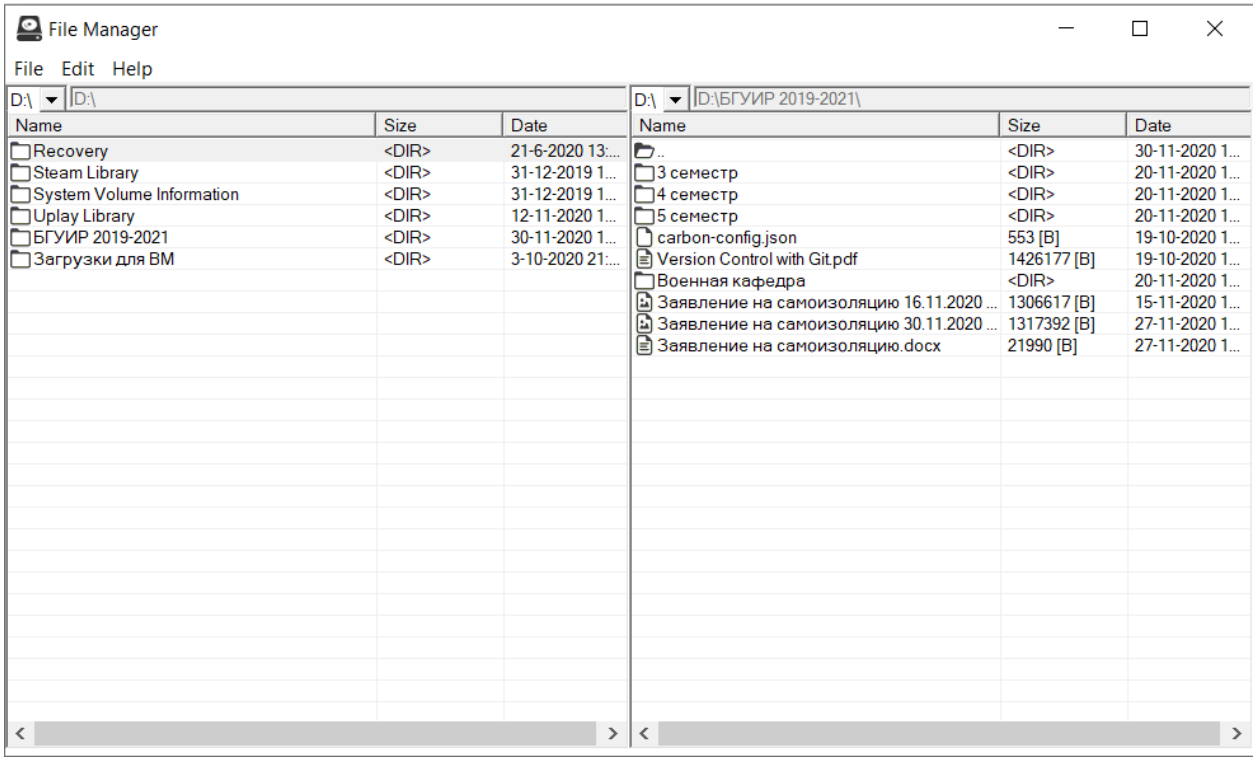

Рисунок 3.1 – Интерфейс файлового менеджера

Для более удобного взаимодействия с программой в ресурсы программы были добавлены иконки. С помощью их можно визуально определить, где в списке каталог, а где файл и какой это тип файла (например, документ или изображение).

3.2 Проектирование функционала

Для работы программы требуется использовать пять структурных блоков:

– Workspace – класс, отвечающий за инициализацию и обновление рабочего пространства;

– Procedure – модуль, отвечающий за функционал файлового менеджера;

– CustomWindow – класс, отвечающий за отображение окна;

– DiskInfo – класс, отвечающий за информацию о логических дисках;

– Conversion – вспомогательный модуль для работы с путями файлов и каталогов.

Диаграмма классов изображена на рисунке 3.2. Исходный код программы представлен в приложении А.

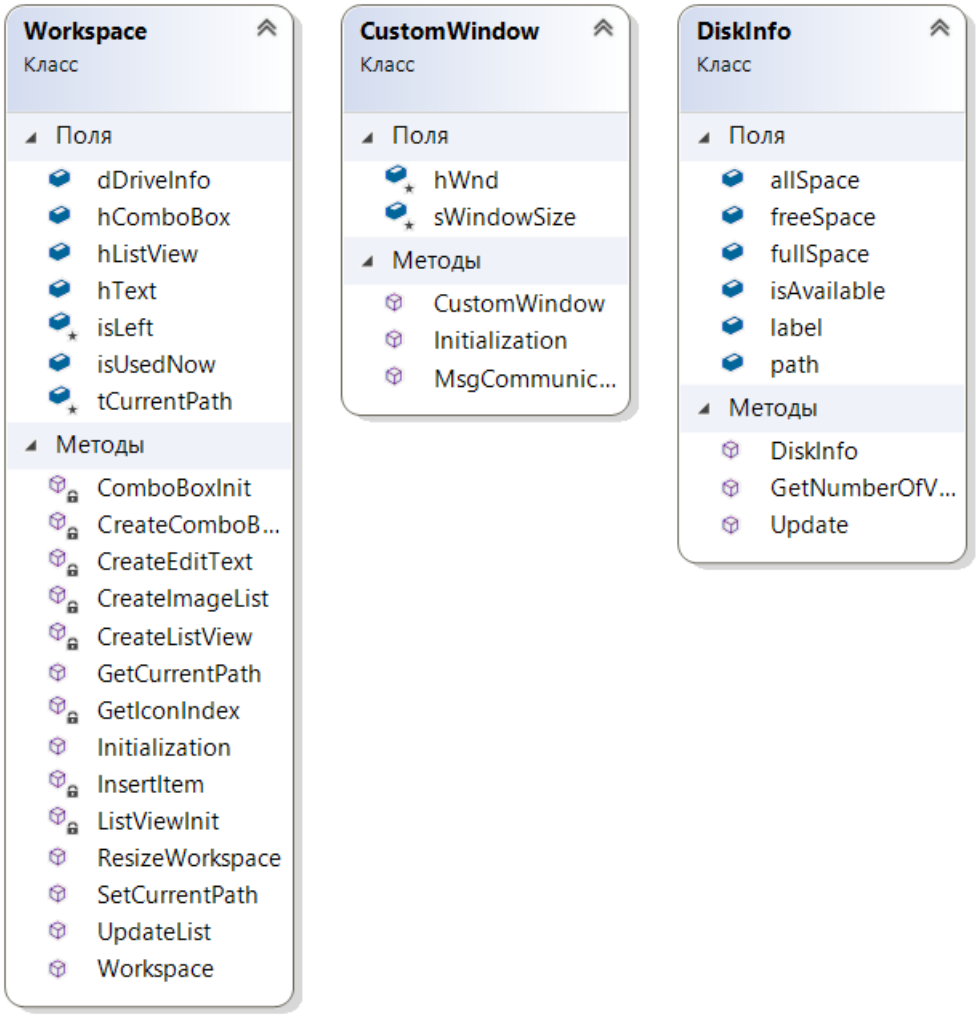

Рисунок 3.2 – Диаграмма классов

3.2.1 Класс Workspace

Функционал для взаимодействия с рабочим пространством был выделен в класс Workspace.

Методы данного класса отвечают за создание и инициализацию таких объектов, как ListView, ComboBox, EditText, ImageList. Метод UpdateList служит для обновления содержимого списка. Метод Resize необходим для изменения размера панелей.

3.2.2 Класс DiskInfo

Для анализа логических дисков был введён вспомогательный класс FileDirectoryInfo.

Данный класс содержит такие методы, как:

1 GetNumberOfVolume. Данный метод с помощью функции GetLogicalDriveStrings позволяет получить названия и количество логических дисков.

2 Update. В этом методе с помощью функции GetDiskFreeSpaceEx мы находим размер всего дискового пространства и размер свободного пространства, тем самым находя размер занятого пространства.

3.2.3 Модуль Procedure

В данном модуле хранятся функции, отвечающие за весь функционал файлового менеджера:

1 NotifyProcedure. Данная функция реагирует на нажатия на панели.

2 CommandProcedure. Данная функция выполняет команду, выбранную пользователем.

3 NewFileDialogProcedure. Данная функция создаёт диалоговое окно, где пользователь вводит название нового файла или каталога.

4 OpenFolderDialogProcedure. Данная функция создаёт диалоговое окно, где пользователь вводит путь каталога, который он хочет открыть.

5 CreateNewFileOrFolder. Данная функция создаёт введённый пользователем файл или каталог.

6 CopySelectedItemToCopyBuffer. Данная функция копирует выделенные файлы и каталоги в буфер обмена.

7 CopyOrMoveFiles. Данная функция вставляет или перемещает файлы из буфера обмена в указанное место.

Разработка приложения велась с использованием системы контроля версий GitHub, позволившая сохранять состояние программы на каждом

отдельном этапе по ходу добавления нового функционала или изменения уже существующего. Появление новых точек возврата происходит посредством группировки изменённых файлов, затем они объединяются под общим именем «коммита», в котором кратко изложена суть изменений. Также можно добавлять к каждому этапу новые файлы, или удалять устаревшие варианты. После накопления определённого количества групп изменений, их следует отправить на удалённый репозиторий, где видна вся история приложения и разница между каждым новым «коммитом».

3.3 Проектирование установщика

С помощью расширения Microsoft Visual Studio Installer Project был разработан установщик программы. Добавлены возможности выбора пути установки программы (см. рисунок 3.4), создание ярлыка на рабочем столе и папки в меню «Пуск».

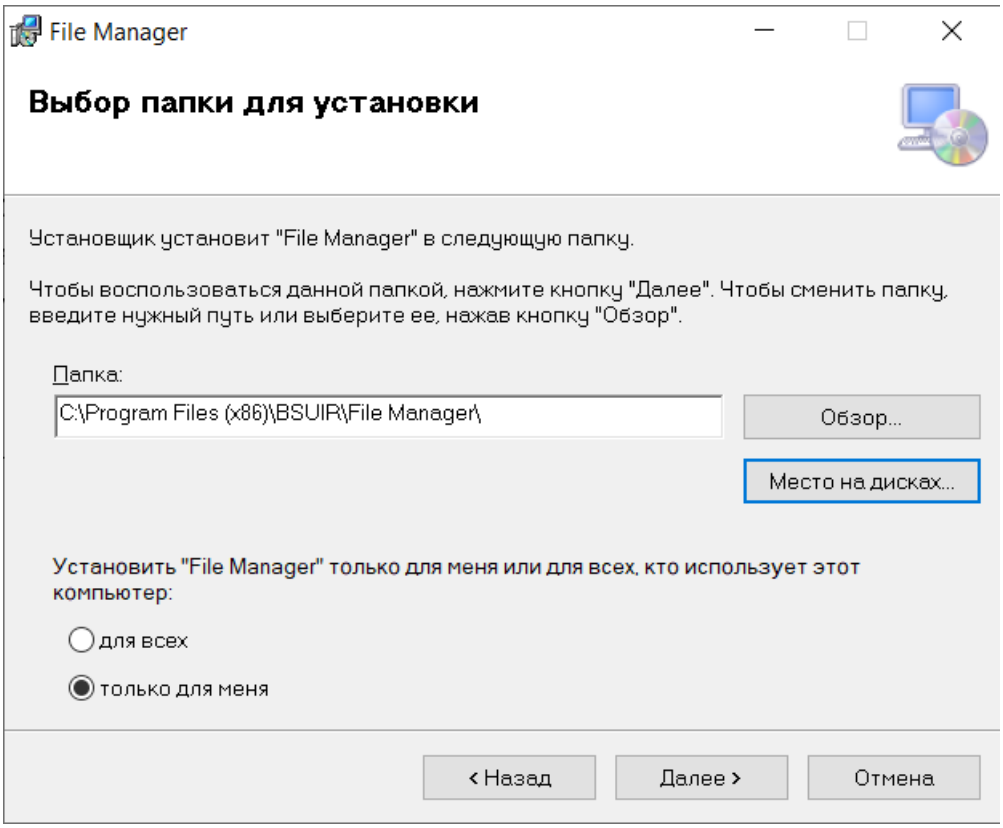

Рисунок 3.2 – Выбор папки для установки

4 ТЕСТИРОВАНИЕ, ПРОВЕРКА РАБОТОСПОСОБНОСТИ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

Проведено тестирование программного средства. Тестирование программного средства производилась на персональном компьютере с установленной операционной системой Windows 10. Дополнительно работоспособность была проверена на виртуальной машине с установленной Windows XP.

4.1 Тестирование функционала программы

Таблица 4.1 – Тестирование функционала программы

Номер теста	**Тестируемая функциональность**	**Последовательность действий**	**Ожидаемый и полученный результаты**
1	Открытие файла.	– двойным щелчком мыши открыть любой файл.	Стандартным приложением Windows открывается файл.
2	Открытие каталога.	– двойным щелчком мыши открыть любую директорию.	На панели отображается содержимое каталога.
3	Открытие каталога, который существует.	– открыть выпадающее меню пункта «File»; – открыть подпункт «Open»; – в поле для ввода ввести путь существующего каталога; – нажать кнопку «ОК».	На панели отображается содержимое каталога.
4	Выбор локального диска.	– открыть выпадающее меню; – выбрать любой из дисков.	На панели отображается содержимое выбранного диска.

Продолжение таблицы 4.1

Номер теста	**Тестируемая функциональность**	**Последовательность действий**	**Ожидаемый и полученный результаты**
5	Открытие каталога, которого не существует.	– открыть выпадающее меню пункта «File»; – открыть подпункт «Open»; – в поле для ввода ввести путь несуществующего каталога; – нажать кнопку «ОК».	Отображается сообщение о неверном пути.
6	Создание нового каталога.	– открыть выпадающее меню пункта «File»; – открыть подпункт «New»; – в поле для ввода ввести название нового каталога. – нажать кнопку «ОК».	Создан каталог. На панели отображается содержимое текущего каталога с новым каталогом.
7	Создание нового файла.	– открыть выпадающее меню пункта «File»; – открыть подпункт «New»; – в поле для ввода ввести название нового файла с расширением. – нажать кнопку «ОК».	Создан файл. На панели отображается содержимое текущего каталога с новым файлом.
8	Удаление файла.	– нажать на файл, который надо удалить; – нажать кнопку «Del».	Файл удалён. Отображается содержимое каталога без удалённого файла.

Продолжение таблицы 4.1

Номер теста	**Тестируемая функциональность**	**Последовательность действий**	**Ожидаемый и полученный результаты**
9	Удаление файла.	– нажать правой кнопкой мыши на файл, который надо удалить; – в появившемся контекстном меню выбрать пункт «Delete».	Файл удалён. Отображается содержимое текущего каталога без удалённого файла.
10	Удаление пустого каталога.	– нажать правой кнопкой мыши на каталог, который надо удалить; – в появившемся контекстном меню выбрать пункт «Delete».	Каталог удалён. Отображается содержимое текущего каталога без удалённого каталога.
11	Удаление каталога с файлами.	– нажать правой кнопкой мыши на каталог, который надо удалить; – в появившемся контекстном меню выбрать пункт «Delete».	Отображается сообщение о том, что каталог не пустой.
12	Копирование файла.	– нажать правой кнопкой мыши на файл, который надо скопировать; – в появившемся контекстном меню выбрать пункт «Copy»; – нажать правой кнопкой мыши вне элементов списка;	Файл скопирован. Отображается содержимое текущего каталога со скопированным файлом.

Продолжение таблицы 4.1

Номер теста	**Тестируемая функциональность**	**Последовательность действий**	**Ожидаемый и полученный результаты**
		– в появившемся контекстном меню выбрать пункт «Paste».	
13	Перемещение файла.	– нажать правой кнопкой мыши на файл, который надо переместить; – в появившемся контекстном меню выбрать пункт «Cut»; – нажать правой кнопкой мыши вне элементов списка; – в появившемся контекстном меню выбрать пункт «Paste».	Файл перемещён. Отображается содержимое текущего каталога с перемещённым файлом.
14	Выход из программы.	– нажать сочетание клавиш «Alt+F4»; – в диалоговом окне нажать кнопку «OK».	Программа закрыта.
15	Отмена выхода из программы.	– нажать сочетание клавиш «Alt+F4»; – в диалоговом окне нажать кнопку «Cancel».	Программа продолжает функционировать.

4.2 Вывод из прохождения тестирования

Программа успешно прошла все тесты, что показывает корректность работы программы и соответствие функциональным требованиям.

5 РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ

Для того, чтобы приступить непосредственно к использованию файлового менеджера необходимо установить приложение. Для этого требуется запустить Setup.exe и пройти шаги установки, следуя инструкциям. После установки необходимо запустить программу. Начальное окно программы изображено на рисунке 5.1.

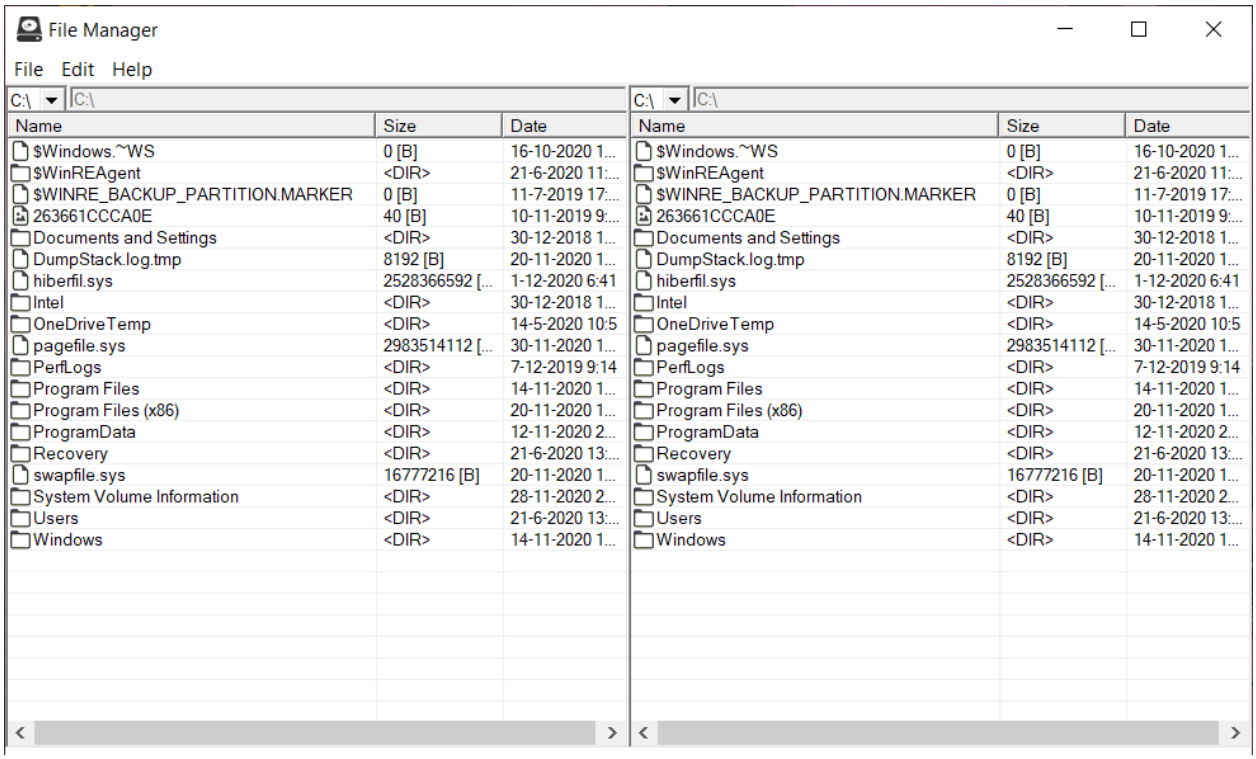

Рисунок 5.1 – Начальное окно программы

Открытие выбранного файла или директории производится двойным щелчком мыши или нажатием на клавиатуре кнопки Enter. Также открыть каталог или файл можно раскрыв пункт меню «File» и нажав на подпункт «Open». В открывшемся диалоговом окне необходимо ввести путь к файлу или каталогу и нажать «OK» (см. рисунок 5.2).

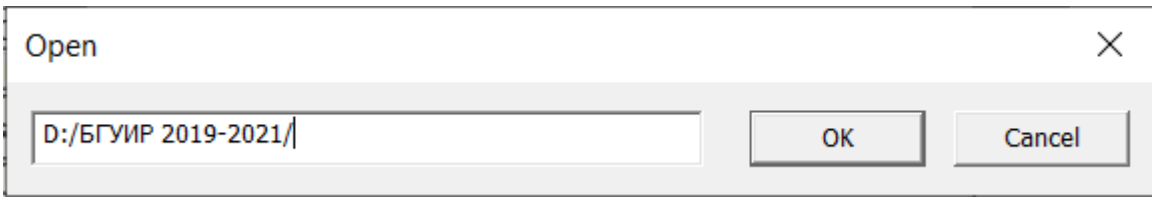

Рисунок 5.2 – Пример открытия каталога

При выборе в выпадающем меню другого диска отобразится корневая папка выбранного диска на одной из панелей. Выбор диска отображён на рисунке 5.3.

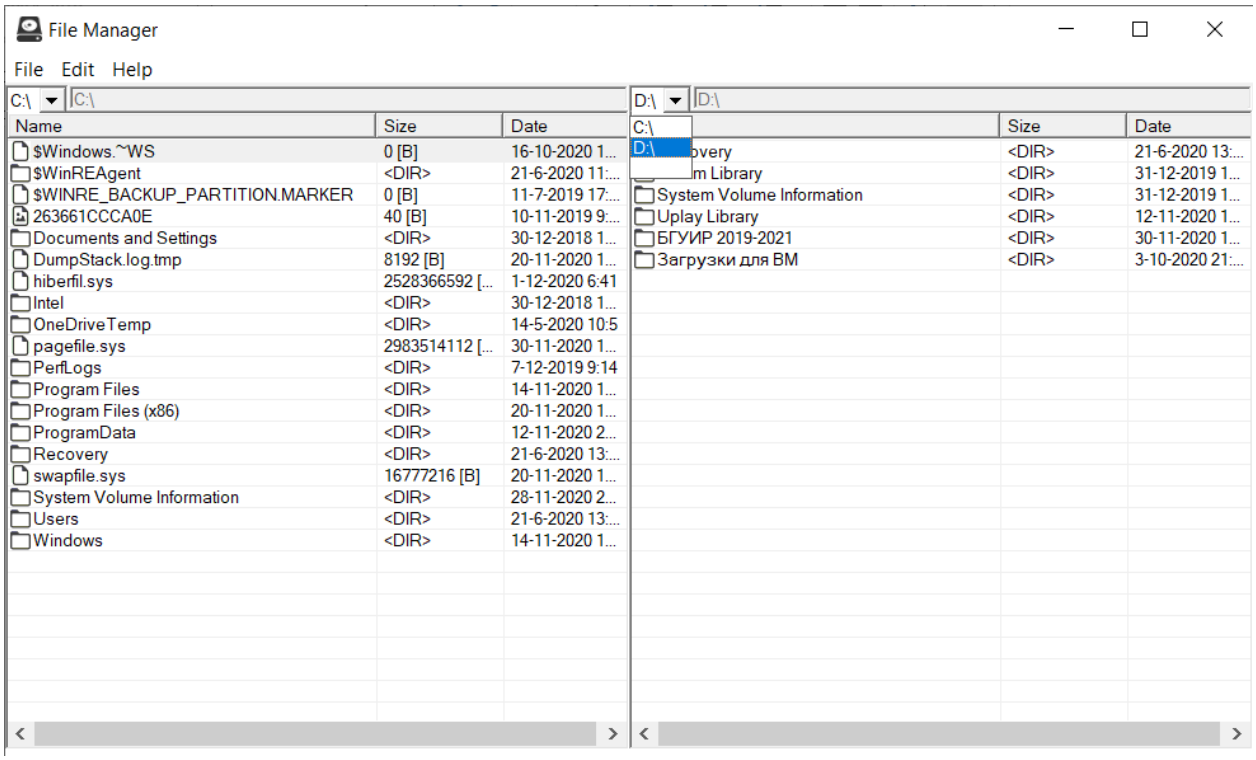

Рисунок 5.3 – Окно программы при выборе диска

Для создания файла или каталога необходимо развернуть пункт меню «Open» и нажать на подпункт меню «New». В открывшемся диалоговом окне необходимо ввести название файла или каталога (см. рисунок 5.4).

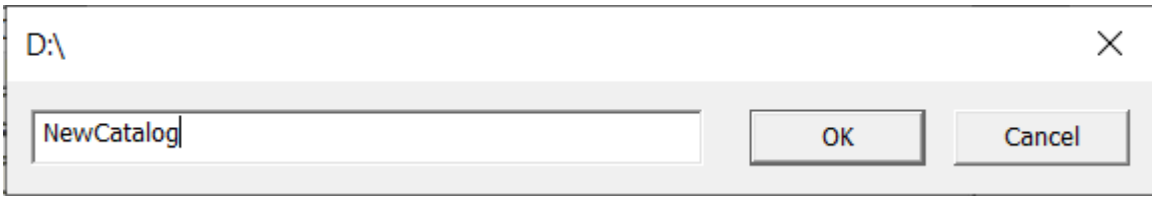

Рисунок 5.4 – Пример создания каталога

Для выполнения одной из функции необходимо:

а) раскрыть пункт меню «Edit» и выбрать необходимую функцию (см. рисунок 5.5);

б) нажать правой кнопкой мыши на выделенный объект и в контекстном меню выбрать необходимую функцию (см. рисунок 5.6);

в) нажать сочетание клавиш (например, для копирования «Ctrl+C»).

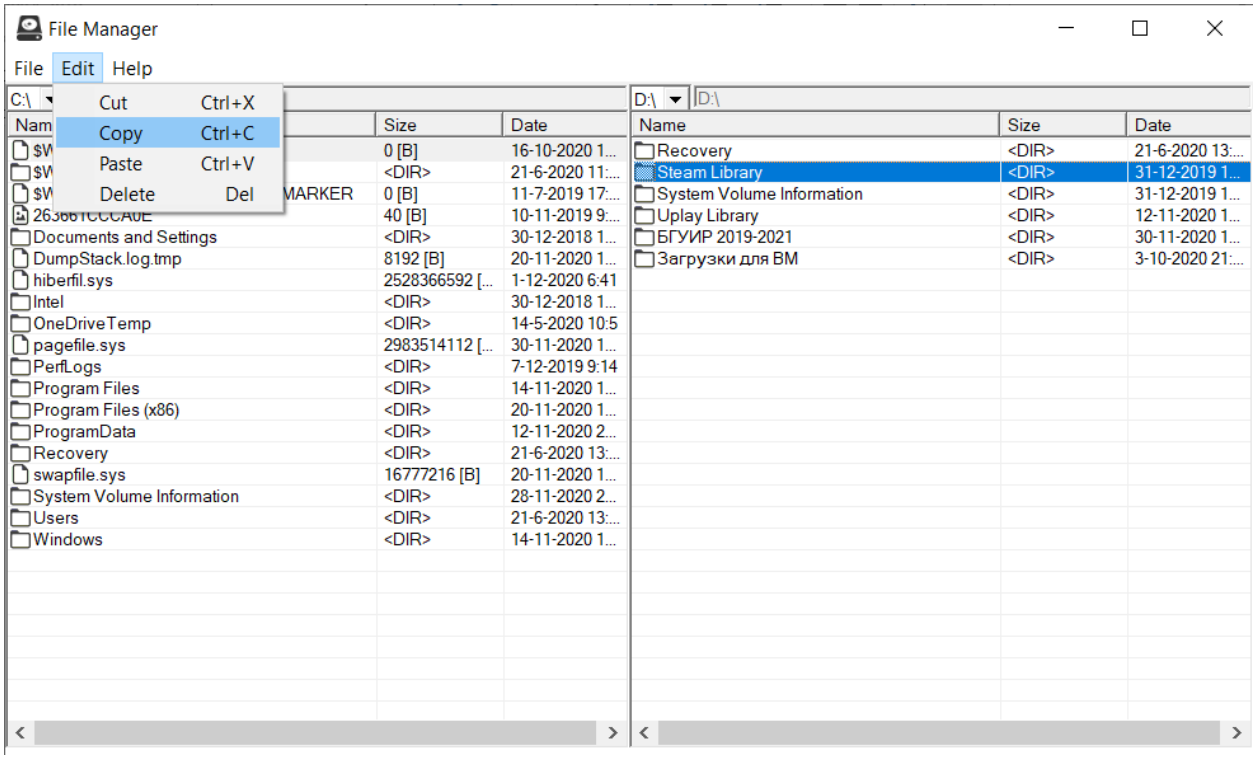

Рисунок 5.5 – Выбор функции в меню

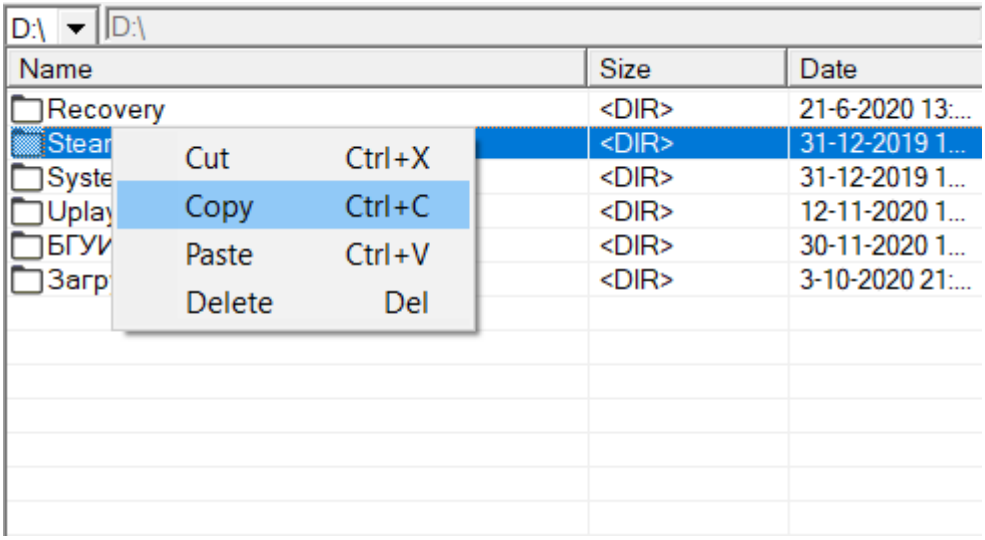

Рисунок 5.6 – Контекстное меню для элемента списка

При закрытии программы будет выведено диалоговое окно для подтверждения или отмены вашего действия.

ЗАКЛЮЧЕНИЕ

В рамках данного курсового проекта был произведен анализ предметной области и реализовано программное и информационное обеспечение "Файловый менеджер". Согласно поставленным задачам, в данном приложении были реализованы такие возможности, как удаление, копирование, перемещение объектов, создание каталогов и файлов, просмотр скрытых файлов.

Для успешного выполнения всех поставленных задач потребовалось подробно разобраться с файловой системой, изучить основы C++, ознакомиться с возможностями среды Microsoft Visual Studio.

При тестировании и отладке не было выявлено случаев некорректной работы программы, нестабильной работы, появления сторонних ошибок и т.д.

Написанный код легко модифицируется для добавления новых функций. В дальнейшем возможны улучшения и доработки, вносящие новый функционал в программу.

Спроектированная система имеет интуитивно понятный интерфейс для пользователя, проста в использовании, не требует серьезных аппаратных затрат. Разработанное программное средство можно применять в личных целях. Например, замена Проводника Windows.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Total Commander – Wikipedia [Электронный ресурс] Режим доступа: https://ru.wikipedia.org/wiki/Total_Commander — Дата доступа: 23.09.2020.
[2] FAR Manager – Wikipedia [Электронный ресурс] Режим доступа: https://ru.wikipedia.org/wiki/FAR_Manager — Дата доступа: 23.09.2020.
[3] Центр разработки для Windows [Электронный ресурс] Режим доступа: https://docs.microsoft.com — Дата доступа: 03.10.2020.
[4] Литвиненко Н. А. Технология программирования на C++. Win32 API-приложения. — СПб.: БХВ-Петербург,2010. — 288 с.
[5] Щупак Ю. А. Win32 API. Разработка приложений для Windows — СПб.: Питер, 2008. — 592 с.
[6] Безруков В. А. Win32 API. Программирование / учебное пособие — СПб.: СПбГУ ИТМО, 2009. — 90 с.

ПРИЛОЖЕНИЕ А
(обязательное)
Исходный код программы (к подразделу 3.2)

Файл Workspace.cpp

```
#include <Windows.h>
#include <CommCtrl.h>
#include <windowsx.h>
#include <Shlwapi.h>
#include <tchar.h>

#include "resource.h"
#include "Workspace.h"
#include "Formats.h"
#include "Conversion.h"

Workspace::Workspace() : hComboBox(nullptr), hListView(nullptr), hText(nullptr),
isUsedNow(FALSE)
{
	memset(tCurrentPath, '\0', MAX_PATH_LENGHT);
}

// ComboBox Methods
BOOL Workspace::CreateComboBox(HWND hWnd, UINT uId)
{
	hComboBox = CreateWindowEx(
		WS_EX_CLIENTEDGE,
		WC_COMBOBOX,
		TEXT("ComboBox"),
		WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWNLIST,
		0, 0, 0, 100,
		hWnd,
		(HMENU)uId,
		nullptr,
		nullptr);

	if (hComboBox == nullptr)
		return FALSE;
	else return TRUE;
}

void Workspace::ComboBoxInit()
```

```
{
	TCHAR drives[10][4];
	UINT uNumber = dDriveInfo.GetNumberOfVolume(drives);

	TCHAR A[40];
	memset(&A, 0, sizeof(A));

	for (size_t k = 0; k <= uNumber; k++)
	{
		wcscpy_s(A, sizeof(A) / sizeof(TCHAR), (TCHAR*)drives[k]);
		SendMessage(hComboBox, (UINT)CB_ADDSTRING, (WPARAM)0,
(LPARAM)A);
	}

	SendMessage(hComboBox, CB_SETCURSEL, (WPARAM)0, (LPARAM)0);
	size_t length = ComboBox_GetTextLength(hComboBox);
	auto string = (LPWSTR)GlobalAlloc(GPTR, length + 1);
	ComboBox_GetText(hComboBox, string, length + 1);

	SetCurrentPath(string);
	dDriveInfo.Update(string);
}

// EditTex Methods
BOOL Workspace::CreateEditText(HWND hWnd, UINT uId)
{
	hText = CreateWindowEx(
		WS_EX_CLIENTEDGE,
		WC_EDIT,
		nullptr,
		WS_CHILD | WS_VISIBLE | WS_BORDER | WS_DISABLED |
WS_EX_LEFT,
		0, 0, 0, 0,
		hWnd,
		(HMENU)uId,
		nullptr,
		nullptr);

	if (hText == nullptr)
		return FALSE;
	else return TRUE;
}

// ListView Methods
```

```
BOOL Workspace::CreateListView(HWND hWnd, UINT uId)
{
	hListView = CreateWindowEx(
		WS_EX_CLIENTEDGE,
		WC_LISTVIEW,
		TEXT("ListView"),
		WS_CHILD | WS_VISIBLE | LVS_REPORT | LVS_EDITLABELS |
LVS_SHOWSELALWAYS,
		0, 0, 0, 0,
		hWnd,
		(HMENU)uId,
		nullptr,
		nullptr);

	ListView_SetExtendedListViewStyle(hListView, LVS_EX_FULLROWSELECT |
LVS_EX_GRIDLINES);

	if (hListView == nullptr)
		return FALSE;
	else return TRUE;
}

void Workspace::ListViewInit()
{
	CreateImageList();

	LVCOLUMN lvc;
	lvc.mask = LVCF_FMT | LVCF_WIDTH | LVCF_TEXT | LVCF_SUBITEM;

	lvc.fmt = LVCFMT_LEFT;
	lvc.cx = 290;
	lvc.pszText = TEXT("Name");
	lvc.iSubItem = 0;
	lvc.iOrder = 0;
	ListView_InsertColumn(hListView, 0, &lvc);

	lvc.cx = 100;
	lvc.pszText = TEXT("Size");
	lvc.iSubItem = 1;
	ListView_InsertColumn(hListView, 1, &lvc);

	lvc.cx = 100;
	lvc.pszText = TEXT("Date");
	lvc.iSubItem = 2;
```

```
	ListView_InsertColumn(hListView, 2, &lvc);
	UpdateList(dDriveInfo.path);
	SetCurrentPath(dDriveInfo.path);
}

void Workspace::InsertItem(UINT uItem, WIN32_FIND_DATA data, HANDLE hFile)
{
	LVITEM lvi;
	lvi.mask = LVIF_TEXT | LVIF_IMAGE;

	// Name column
	lvi.pszText = data.cFileName;
	lvi.iItem = uItem;
	lvi.iSubItem = 0;
	lvi.iImage = GetIconIndex(data);
	ListView_InsertItem(hListView, &lvi);

	// Size column
	if (lvi.iImage == 1 || lvi.iImage == 2)
		lvi.pszText = TEXT("<DIR>");
	else
	{
		LPWSTR s1 = GetLPWSTR(data.nFileSizeHigh * (MAXDWORD + 1) +
data.nFileSizeLow), s2 = TEXT(" [B]");
		int lenght = lstrlen(s1) + lstrlen(s2);
		auto* tmp = new TCHAR[lenght];
		wcscpy(tmp, s1);
		wcscat(tmp, s2);

		lvi.pszText = tmp;
	}
	lvi.iSubItem = 1;
	ListView_SetItem(hListView, &lvi);

	// Date column
	lvi.pszText = GetLPWSTR(data.ftLastWriteTime);
	lvi.iSubItem = 2;
	ListView_SetItem(hListView, &lvi);
}

void Workspace::CreateImageList()
{
	HIMAGELIST hImageList = ImageList_Create(16, 16, ILC_COLOR32, 8, 0);
	HBITMAP hBmp = nullptr;
```

```
        // 0. File icon
        hBmp = LoadBitmap(GetModuleHandle(nullptr), MAKEINTRESOURCE(IDB_FILE));
        ImageList_Add(hImageList, hBmp, (HBITMAP)nullptr);
        DeleteObject(hBmp);
        // 1. Folder icon
        hBmp = LoadBitmap(GetModuleHandle(nullptr),
MAKEINTRESOURCE(IDB_FOLDER));
        ImageList_Add(hImageList, hBmp, (HBITMAP)nullptr);
        DeleteObject(hBmp);
        // 2. Folder open icon
        hBmp = LoadBitmap(GetModuleHandle(nullptr),
MAKEINTRESOURCE(IDB_FOLDER_OPEN));
        ImageList_Add(hImageList, hBmp, (HBITMAP)nullptr);
        DeleteObject(hBmp);
        // 3. File exe icon
        hBmp = LoadBitmap(GetModuleHandle(nullptr), MAKEINTRESOURCE(IDB_EXE));
        ImageList_Add(hImageList, hBmp, (HBITMAP)nullptr);
        DeleteObject(hBmp);
        // 4. File audio type icon
        hBmp = LoadBitmap(GetModuleHandle(nullptr),
MAKEINTRESOURCE(IDB_AUDIO));
        ImageList_Add(hImageList, hBmp, (HBITMAP)nullptr);
        DeleteObject(hBmp);
        // 5. File video type icon
        hBmp = LoadBitmap(GetModuleHandle(nullptr),
MAKEINTRESOURCE(IDB_VIDEO));
        ImageList_Add(hImageList, hBmp, (HBITMAP)nullptr);
        DeleteObject(hBmp);
        // 6. File Image type icon
        hBmp = LoadBitmap(GetModuleHandle(nullptr),
MAKEINTRESOURCE(IDB_IMAGE));
        ImageList_Add(hImageList, hBmp, (HBITMAP)nullptr);
        DeleteObject(hBmp);
        // 7. File document type icon
        hBmp = LoadBitmap(GetModuleHandle(nullptr),
MAKEINTRESOURCE(IDB_DOCUMENT));
        ImageList_Add(hImageList, hBmp, (HBITMAP)nullptr);
        DeleteObject(hBmp);

        ListView_SetImageList(hListView, hImageList, LVSIL_SMALL);
}

int Workspace::GetIconIndex(WIN32_FIND_DATA data)
{
```

```
	LPWSTR lpsPath = ConnectTwoString(this->tCurrentPath, (LPWSTR)data.cFileName,
TRUE);
	LPWSTR lpsExt = PathFindExtension(lpsPath);

	if ((!_tcscmp(lpsExt, TEXT("")) || !_tcscmp(data.cFileName, TEXT(".."))) &&
data.nFileSizeLow == 0)
	{
		return !_tcscmp(data.cFileName, TEXT("..")) ? 2 : 1;
	}
	else
	{
		for (int i = 0; i < MAX_COUNT_FORMATS; i++)
		{
			if (!_tcscmp(lpsExt, tAudioFormat[i])) return 4;
			if (!_tcscmp(lpsExt, tVideoFormat[i])) return 5;
			if (!_tcscmp(lpsExt, tImageFormat[i])) return 6;
			if (!_tcscmp(lpsExt, tDocumentFormat[i])) return 7;
		}
	}
	return 0;
}

void Workspace::UpdateList(TCHAR* _path)
{
	auto* tOldPath = new TCHAR[256];
	memset(tOldPath, '\0', 256);
	wcscpy(tOldPath, tCurrentPath);

	ListView_DeleteAllItems(hListView);

	int length = lstrlen(_path) + lstrlen(TEXT("*.*"));
	auto* path = new TCHAR[length];
	wcscpy(path, _path);
	Edit_SetText(hText, path);
	SetCurrentPath(path);
	wcscat(path, TEXT("*.*"));


	WIN32_FIND_DATA data;
	HANDLE hFile = FindFirstFile(path, &data);

	SendMessage(hListView, LB_RESETCONTENT, 0, 0);

	int i = 0;
```

```
	while (FindNextFile(hFile, &data))
	{
		InsertItem(i, data, hFile);
		i++;
	}

	if (i == 0 && (MessageBox(nullptr, TEXT("Access denied!"), TEXT("Error"), MB_OK |
MB_ICONERROR) == IDOK))
	{
		UpdateList(tOldPath);
	}

	FindClose(hFile);
}

void Workspace::SetCurrentPath(TCHAR* tNewPath)
{
	lstrcpyn(tCurrentPath, tNewPath, 256);
}

LPWSTR Workspace::GetCurrentPath()
{
	return tCurrentPath;
}

// Other Method
void Workspace::ResizeWorkspace(HWND hWnd, LPARAM lParam)
{
	SIZE sNewSize;
	sNewSize.cx = LOWORD(lParam) / 2;
	sNewSize.cy = HIWORD(lParam);
	if (isLeft)
	{
		MoveWindow(hComboBox, 0, 0, 50, 20, TRUE);
		MoveWindow(hText, 50, 0, sNewSize.cx - 50, 20, TRUE);
		MoveWindow(hListView, 0, 20, sNewSize.cx, sNewSize.cy - 25, TRUE);
	}
	else
	{
		MoveWindow(hComboBox, sNewSize.cx, 0, 50, 2, TRUE);
		MoveWindow(hText, sNewSize.cx + 50, 0, sNewSize.cx - 50, 20, TRUE);
		MoveWindow(hListView, sNewSize.cx, 20, sNewSize.cx, sNewSize.cy - 25,
TRUE);
	}
```

```
}

BOOL Workspace::Initialization(HWND hWnd, UINT uComboId, UINT uListId, UINT
uTextId, BOOL isLeft)
{
	this->isLeft = isLeft;

	InitCommonControls();

	if (!CreateComboBox(hWnd, uComboId))
	{
		MessageBox(nullptr, TEXT("Function CreateComboBox failure!"),
TEXT("Error"), MB_OK | MB_ICONERROR);
		return FALSE;
	}
	ComboBoxInit();

	if (!CreateEditText(hWnd, uTextId))
	{
		MessageBox(nullptr, TEXT("Function CreateEditText failure!"), TEXT("Error"),
MB_OK | MB_ICONERROR);
		return FALSE;
	}

	if (!CreateListView(hWnd, uListId))
	{
		MessageBox(nullptr, TEXT("Function CreateComboBox failure!"),
TEXT("Error"), MB_OK | MB_ICONERROR);
		return FALSE;
	}

	ListViewInit();

	auto hNormalFont = (HFONT)GetStockObject(DEFAULT_GUI_FONT);
	SendMessage(hComboBox, WM_SETFONT, (WPARAM)hNormalFont, 0);
	SendMessage(hText, WM_SETFONT, (WPARAM)hNormalFont, 0);
	SendMessage(hListView, WM_SETFONT, (WPARAM)hNormalFont, 0);

	return TRUE;
}
```

Файл Procedure.cpp

```
#include <Windows.h>
#include <windowsx.h>
#include <CommCtrl.h>
#include <wchar.h>
#include <string.h>
#include <Shlwapi.h>
#include <tchar.h>

#include "resource.h"
#include "Procedure.h"
#include "Workspace.h"
#include "Conversion.h"

Workspace wLeftWorkspace, wRightWorkspace;
HBITMAP hBmp;
TCHAR* tBuffer0 = new TCHAR[256], * tBuffer1 = new TCHAR[256], ** tCopyBuffer =
NULL;
int countCopyBuffer = 0;
BOOL cutMode;
POINT pPoint;

LRESULT CALLBACK WindowProcedure(HWND hWnd, UINT uMsg, WPARAM wParam,
LPARAM lParam)
{
	switch (uMsg)
	{
	case WM_CREATE:
		if (!wLeftWorkspace.Initialization(hWnd, ID_LEFTCOMBOBOX,
ID_LEFTLISTVIEW, ID_LEFTTEXT, TRUE))
		{
			MessageBox(nullptr, TEXT("Function wLeftWorkspace.Initialization
failure!"), TEXT("Error"), MB_OK | MB_ICONERROR);
		}
		wLeftWorkspace.isUsedNow = TRUE;
		if (!wRightWorkspace.Initialization(hWnd, ID_RIGHTCOMBOBOX,
ID_RIGHTLISTVIEW, ID_RIGHTTEXT, FALSE))
		{
			MessageBox(nullptr, TEXT("Function wRightWorkspace.Initialization
failure!"), TEXT("Error"), MB_OK | MB_ICONERROR);
		}
		break;
	case WM_COMMAND:
```

```
		CommandProcedure(hWnd, uMsg, wParam, lParam);
		break;
	case WM_NOTIFY:
		NotifyProcedure(hWnd, uMsg, wParam, lParam);
		break;
	case WM_SIZE:
		wLeftWorkspace.ResizeWorkspace(hWnd, lParam);
		wRightWorkspace.ResizeWorkspace(hWnd, lParam);
		break;
	case WM_CLOSE:
		if (MessageBox(nullptr, TEXT("Do you want close the window?"),
TEXT("Question"), MB_OKCANCEL | MB_ICONQUESTION) == IDOK)
		{
			DestroyWindow(hWnd);
		}
		break;
	case WM_DESTROY:
		PostQuitMessage(0);
	default:
		return DefWindowProc(hWnd, uMsg, wParam, lParam);
	}
}

LRESULT CommandProcedure(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{
	switch (LOWORD(wParam))
	{
	case ID_LEFTCOMBOBOX:
		wLeftWorkspace.isUsedNow = TRUE;
		wRightWorkspace.isUsedNow = FALSE;

		if (HIWORD(wParam) == CBN_SELENDOK)
		{
			ComboBox_GetCurSel(wLeftWorkspace.hComboBox);
			size_t length =
ComboBox_GetTextLength(wLeftWorkspace.hComboBox);

			auto lpsString = (LPWSTR)GlobalAlloc(GPTR, length + 1);
			ComboBox_GetText(wLeftWorkspace.hComboBox, lpsString, length +
1);

			wLeftWorkspace.dDriveInfo.Update(lpsString);
			wLeftWorkspace.UpdateList(wLeftWorkspace.dDriveInfo.path);
```

```
                GlobalFree(lpsString);
            }
            break;
        case ID_RIGHTCOMBOBOX:
            wRightWorkspace.isUsedNow = TRUE;
            wLeftWorkspace.isUsedNow = FALSE;

            if (HIWORD(wParam) == CBN_SELENDOK)
            {
                ComboBox_GetCurSel(wRightWorkspace.hComboBox);
                size_t length =
ComboBox_GetTextLength(wRightWorkspace.hComboBox);

                auto lpsString = (LPWSTR)GlobalAlloc(GPTR, length + 1);
                ComboBox_GetText(wRightWorkspace.hComboBox, lpsString, length +
1);

                wRightWorkspace.dDriveInfo.Update(lpsString);
                wRightWorkspace.UpdateList(wRightWorkspace.dDriveInfo.path);

                GlobalFree(lpsString);
            }
            break;
        case ID_FILE_NEW:
            DialogBox(NULL, MAKEINTRESOURCE(IDD_FILE_NEW_DIALOG),
hWnd, (DLGPROC)NewFileDialogProcedure);
            break;
        case ID_FILE_OPEN:
            DialogBox(NULL, MAKEINTRESOURCE(IDD_FILE_NEW_DIALOG),
hWnd, (DLGPROC)OpenFolderDialogProcedure);
            break;
        case ID_FILE_EXIT:
            SendMessage(hWnd, WM_CLOSE, NULL, NULL);
            break;
        case ID_COPY_MODE:
        case ID_EDIT_COPY:
            cutMode = FALSE;
            if (wLeftWorkspace.isUsedNow)
CopySelectedItemToCopyBuffer(wLeftWorkspace.hListView,
wLeftWorkspace.GetCurrentPath());
            else CopySelectedItemToCopyBuffer(wRightWorkspace.hListView,
wRightWorkspace.GetCurrentPath());
            break;
```

```
        case ID_CUT_MODE:
        case ID_EDIT_CUT:
                cutMode = TRUE;
                if (wLeftWorkspace.isUsedNow)
CopySelectedItemToCopyBuffer(wLeftWorkspace.hListView,
wLeftWorkspace.GetCurrentPath());
                else CopySelectedItemToCopyBuffer(wRightWorkspace.hListView,
wRightWorkspace.GetCurrentPath());
                break;
        case ID_PASTE_MODE:
        case ID_EDIT_PASTE:
                if (wLeftWorkspace.isUsedNow)
                {
                        CopyOrMoveFiles(wLeftWorkspace.hListView,
wLeftWorkspace.GetCurrentPath());
                }
                else
                {
                        CopyOrMoveFiles(wRightWorkspace.hListView,
wRightWorkspace.GetCurrentPath());
                }
                wLeftWorkspace.UpdateList(wLeftWorkspace.GetCurrentPath());
                wRightWorkspace.UpdateList(wRightWorkspace.GetCurrentPath());
                break;
        case ID_DEL_MODE:
        case ID_EDIT_DELETE:
                SendMessage(hWnd, WM_COMMAND, ID_EDIT_COPY, NULL);
                if (tCopyBuffer)
                {
                        for (int i = 0; i < countCopyBuffer; i++)
                        {
                                if (PathIsDirectory(tCopyBuffer[i]))
                                {
                                        if (!PathIsDirectoryEmpty(tCopyBuffer[i]))
                                        {
                                                MessageBox(nullptr, TEXT("Directory is not
empty! Remove content!"), TEXT("Warning"), MB_OK | MB_ICONWARNING);
                                        }
                                        RemoveDirectory(tCopyBuffer[i]);
                                }
                                else DeleteFile(tCopyBuffer[i]);

                                wLeftWorkspace.UpdateList(wLeftWorkspace.GetCurrentPath());
```

```
		wRightWorkspace.UpdateList(wRightWorkspace.GetCurrentPath());
					delete[] tCopyBuffer[i];
				}
				delete[] tCopyBuffer;
			}
			break;
		case ID_INFO_MODE:
		case ID_HELP_ABOUTFILEMANAGER:
			MessageBox(nullptr, TEXT("Copyright (c) 2020 by Pavel Miskevich"),
TEXT("About File Manager"), MB_OK | MB_ICONINFORMATION);
			break;
		default:
			return DefWindowProc(hWnd, uMsg, wParam, lParam);
		}
}

void CopySelectedItemToCopyBuffer(HWND hList, TCHAR* tPath)
{
		countCopyBuffer = ListView_GetSelectedCount(hList);
		tCopyBuffer = new TCHAR * [countCopyBuffer];
		auto position = ListView_GetNextItem(hList, -1, LVNI_SELECTED);
		for (int i = 0; i < countCopyBuffer; i++)
		{
			tCopyBuffer[i] = new TCHAR[256];
			memset(tBuffer0, '\0', 256);
			ListView_GetItemText(hList, position, 0, tBuffer0, 256);
			tCopyBuffer[i] = ConnectTwoString(tPath, tBuffer0, TRUE);

			position = ListView_GetNextItem(hList, position, LVNI_SELECTED);
		}
}

void CopyOrMoveFiles(HWND hList, TCHAR* tPath)
{
		if (tCopyBuffer)
		{
			for (int i = 0; i < countCopyBuffer; i++)
			{
				memset(tBuffer0, '\0', 256);
				tBuffer0 = PathFindFileName(tCopyBuffer[i]);

				memset(tBuffer1, '\0', 256);
				tBuffer1 = ConnectTwoString(tPath, tBuffer0, TRUE);
```

```
                        if (cutMode)
                                MoveFile(tCopyBuffer[i], tBuffer1);
                        else
                                CopyFile(tCopyBuffer[i], tBuffer1, FALSE);

                        delete[] tCopyBuffer[i];
                }
                delete[] tCopyBuffer;
                tCopyBuffer = nullptr;
        }
}

LRESULT NotifyProcedure(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{
        auto lpnmItem = (LPNMITEMACTIVATE)lParam;
        bool isBack = false;

        if ((((LPNMHDR)lParam)->hwndFrom) == GetDlgItem(hWnd, ID_LEFTLISTVIEW))
        {
                switch (((LPNMHDR)lParam)->code)
                {
                case NM_RCLICK:
                        wLeftWorkspace.isUsedNow = TRUE;
                        wRightWorkspace.isUsedNow = FALSE;

                        ListView_GetItemPosition(wLeftWorkspace.hListView, lpnmItem->iItem,
&pPoint);
                        ClientToScreen(wLeftWorkspace.hListView, &pPoint);

                        TrackPopupMenu(
                                GetSubMenu(LoadMenu(NULL,
MAKEINTRESOURCE(IDR_CUSTOM_MENU)), 1),
                                0,
                                pPoint.x + 50, pPoint.y,
                                0,
                                hWnd,
                                NULL);

                        break;
                case NM_CLICK:
                        wLeftWorkspace.isUsedNow = TRUE;
                        wRightWorkspace.isUsedNow = FALSE;
```

```
                    break;
                case NM_DBLCLK:
                    memset(tBuffer0, '\0', 256);
                    memset(tBuffer1, '\0', 256);
                    isBack = false;
                    ListView_GetItemText(wLeftWorkspace.hListView, lpnmItem->iItem, 0,
tBuffer0, 256);
                    if (!_tcscmp(tBuffer0, TEXT("..")))
                        isBack = true;

                    if (isBack)
                    {
                        tBuffer1 = GetPastPath(wLeftWorkspace.GetCurrentPath());
                    }
                    else
                    {
                        tBuffer1 = ConnectTwoString(wLeftWorkspace.GetCurrentPath(),
tBuffer0, FALSE);
                    }

                    memset(tBuffer0, '\0', 256);
                    ListView_GetItemText(wLeftWorkspace.hListView, lpnmItem->iItem, 1,
tBuffer0, 256);

                    if (!_tcscmp(tBuffer0, TEXT("<DIR>")))
                    {
                        wLeftWorkspace.UpdateList(tBuffer1);
                    }
                    else
                    {
                        ShellExecute(nullptr, TEXT("open"), tBuffer1, nullptr, nullptr,
SW_SHOWNORMAL);
                    }
                    break;
                default:
                    return DefWindowProc(hWnd, uMsg, wParam, lParam);
                }
        }
        else if ((((LPNMHDR)lParam)->hwndFrom) == GetDlgItem(hWnd,
ID_RIGHTLISTVIEW))
        {
            switch (((LPNMHDR)lParam)->code)
            {
            case NM_RCLICK:
```

```
                    wRightWorkspace.isUsedNow = TRUE;
                    wLeftWorkspace.isUsedNow = FALSE;

                    ListView_GetItemPosition(wRightWorkspace.hListView, lpnmItem-
>iItem, &pPoint);
                    ClientToScreen(wRightWorkspace.hListView, &pPoint);

                    TrackPopupMenu(
                            GetSubMenu(LoadMenu(NULL,
MAKEINTRESOURCE(IDR_CUSTOM_MENU)), 1),
                            0,
                            pPoint.x + 50, pPoint.y,
                            0,
                            hWnd,
                            NULL);

                    break;
            case NM_CLICK:
                    wRightWorkspace.isUsedNow = TRUE;
                    wLeftWorkspace.isUsedNow = FALSE;
                    break;

            case NM_DBLCLK:
                    memset(tBuffer0, '\0', 256);
                    memset(tBuffer1, '\0', 256);
                    isBack = false;

                    ListView_GetItemText(wRightWorkspace.hListView, lpnmItem->iItem,
0, tBuffer0, 256);
                    if (!_tcscmp(tBuffer0, TEXT("..")))
                            isBack = true;

                    if (isBack)
                    {
                            tBuffer1 = GetPastPath(wRightWorkspace.GetCurrentPath());
                    }
                    else
                    {
                            tBuffer1 =
ConnectTwoString(wRightWorkspace.GetCurrentPath(), tBuffer0, FALSE);
                    }

                    memset(tBuffer0, '\0', 256);
```

```
                        ListView_GetItemText(wRightWorkspace.hListView, lpnmItem->iItem,
1, tBuffer0, 256);

                        if (!_tcscmp(tBuffer0, TEXT("<DIR>")))
                        {
                                wRightWorkspace.UpdateList(tBuffer1);
                        }
                        else
                        {
                                ShellExecute(nullptr, TEXT("open"), tBuffer1, nullptr, nullptr,
SW_SHOWNORMAL);
                        }
                        break;
                default:
                        return DefWindowProc(hWnd, uMsg, wParam, lParam);
                }
        }
}

BOOL CALLBACK NewFileDialogProcedure(HWND hWnd, UINT uMsg, WPARAM
wParam, LPARAM lParam)
{
        HWND hTextField = GetDlgItem(hWnd, IDC_EDIT1);

        switch (uMsg)
        {
        case WM_INITDIALOG:
                if (wLeftWorkspace.isUsedNow == TRUE) SetWindowText(hWnd,
wLeftWorkspace.GetCurrentPath());
                else SetWindowText(hWnd, wRightWorkspace.GetCurrentPath());
                break;
        case WM_COMMAND:
                memset(tBuffer0, '\0', 256);
                memset(tBuffer1, '\0', 256);

                switch (LOWORD(wParam))
                {
                case IDOK:
                        Edit_GetText(hTextField, tBuffer0, 256);
                        if (wLeftWorkspace.isUsedNow)
                        {
                                CreateNewFileOrFolder(wLeftWorkspace.GetCurrentPath());
                        }
                        else
```

```
                        {
                                CreateNewFileOrFolder(wRightWorkspace.GetCurrentPath());
                        }
                        wLeftWorkspace.UpdateList(wLeftWorkspace.GetCurrentPath());
                        wRightWorkspace.UpdateList(wRightWorkspace.GetCurrentPath());
                case IDCANCEL:
                        EndDialog(hWnd, NULL);
                        break;
                default:
                        break;
                }
                break;
        default:
                return FALSE;
        }
        return TRUE;
}

void CreateNewFileOrFolder(TCHAR* tPath)
{
        tBuffer1 = ConnectTwoString(tPath, tBuffer0, TRUE);
        memset(tBuffer0, '\0', 256);
        tBuffer0 = PathFindExtension(tBuffer1);

        if (!_tcscmp(tBuffer0, TEXT("")))
        {
                wcscat(tBuffer1, TEXT("\\"));
                CreateDirectory(tBuffer1, nullptr);
        }
        else
        {
                HANDLE hAppend = CreateFile(tBuffer1,
                        GENERIC_READ | GENERIC_WRITE,
                        FILE_SHARE_READ | FILE_SHARE_WRITE,
                        nullptr,
                        CREATE_NEW,
                        FILE_ATTRIBUTE_NORMAL,
                        nullptr);
                CloseHandle(hAppend);
        }
}

BOOL CALLBACK OpenFolderDialogProcedure(HWND hWnd, UINT uMsg, WPARAM
wParam, LPARAM lParam)
```

```
{
	HWND hTextField = GetDlgItem(hWnd, IDC_EDIT1);
	switch (uMsg)
	{
	case WM_INITDIALOG:
		SetWindowText(hWnd, TEXT("Open"));
		break;
	case WM_COMMAND:
		memset(tBuffer0, '\0', 10);
		memset(tBuffer1, '\0', 256);

		switch (LOWORD(wParam))
		{
		case IDOK:
			Edit_GetText(hTextField, tBuffer0, 256);

			tBuffer1 = PathFindExtension(tBuffer0);

			if (!_tcscmp(tBuffer1, TEXT("")))
			{
				if (wLeftWorkspace.isUsedNow)
					wLeftWorkspace.UpdateList(ConnectTwoString(tBuffer0,
TEXT("\\"), TRUE));
				else
					wRightWorkspace.UpdateList(ConnectTwoString(tBuffer0,
TEXT("\\"), TRUE));
			}
			else
				ShellExecute(nullptr, TEXT("open"), tBuffer0, nullptr, nullptr,
SW_SHOWNORMAL);

		case IDCANCEL:
			EndDialog(hWnd, NULL);
			break;
		default:
			break;
		}
		break;
	default:
		return FALSE;
	}
	return TRUE;
}
```

Обозначение	Наименование	Дополнительные сведения
	Текстовые документы	
БГУИР КР 1–40 01 01 618 ПЗ	Пояснительная записка	44 с.
	Графические документы	
ГУИР 851006-01 СА	"Копирование файлов из одного каталога в другой. Схема алгоритма", А1, схема алгоритма, чертеж	Формат А1

					БГУИР КР 1-40 01 01 618 Д1			
Изм.	Л.	№ докум.	Подп.	Дата	Файловый менеджер Ведомость курсовой работы		Лист	Листов
Разраб.		Мискевич П.Л.				Т	45	45
Пров.		Жиденко А.Л.				Кафедра ПОИТ гр. 851006		