Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей Кафедра информатики

Дисциплина: Операционные среды и системное программирование

**ОТЧЕТ**

к лабораторной работе № 3

на тему «Управление памятью и вводом-выводом, расширенные возможности ввода-вывода Windows. Функции API подсистемы памяти Win 32. Организация и контроль асинхронных операций ввода-вывода. Отображение файлов в память»

Выполнил:

студент гр. 153504

Хрищанович А.К.

Проверил:

Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

[1 Постановка задачи 3](#_Toc146631498)

[2 Краткие теоретические сведения 4](#_Toc146631499)

[3 Результаты выполнения лабораторной работы 6](#_Toc146631500)

[Выводы 8](#_Toc146631501)

[Список использованных источников 9](#_Toc146631502)

[Приложение А (обязательное) Листинг кода 10](#_Toc146631503)

## 1 ПОСТАНОВКА ЗАДАЧИ

Целью выполнения данной лабораторной работы является создание многозадачного файлового менеджера с минимальным функционалом, позволяющего пользователю удалять, копировать и перемещать одновременно несколько файлов между директориями.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Win32 API (Windows API) представляет собой набор функций и интерфейсов, предоставляемых операционной системой Windows для разработки приложений. Этот мощный набор инструментов обеспечивает доступ к различным функциональным возможностям Windows, включая создание и управление окнами, обработку сообщений, работу с файлами и реестром, а также многие другие операции. Win32 API играет ключевую роль в разработке приложений для Windows и обеспечивает высокую степень контроля над поведением приложений.

Win32 API предоставляет разработчикам возможность эффективно управлять памятью и вводом-выводом в приложениях под операционной системой Windows. Одним из основных элементов управления памятью является функция new(), позволяющая выделять динамическую память, и функция delete[], предназначенная для освобождения выделенных ресурсов. Эти функции позволяют эффективно управлять памятью и избегать утечек памяти в приложениях.

В контексте ввода-вывода, Win32 API предоставляет множество функций для работы с файлами, устройствами и сетевыми ресурсами. Функции, такие как CreateFile(), CopyFile(), DeleteFile(), FindFirstFile(), FindNextFile() обеспечивают возможность открытия, чтения и записи файлов. Кроме того, API предоставляет функции для асинхронного ввода-вывода, что позволяет выполнять операции ввода-вывода параллельно и повышает производительность приложений. Win32 API также поддерживает отображение файлов в память с использованием функции CreateFileMapping(), что обеспечивает быстрый доступ к данным в файлах без необходимости частого чтения с диска.

В приложении "FileManager" была реализована асинхронная обработка операций копирования, перемещения и удаления файлов. Это позволяет улучшить отзывчивость приложения и предоставить пользователю возможность продолжать взаимодействие с интерфейсом приложения, не блокируя его на время выполнения долгих операций ввода-вывода. Win32 API также предоставляет механизмы контроля над асинхронными операциями, позволяя приложению отслеживать состояние и результаты операций ввода-вывода.

Для выполнения данной лабораторной работы, были использованы следующие теоретические сведения и концепции:

1. Оконное приложение: программа создает графическое оконное приложение с помощью Win32 API. Главное окно приложения имеет элементы управления, такие как панель инструментов, два списковых представления (ListView), два выпадающих списка (ComboBox), а также две метки (Label) для отображения текущих путей к директориям.

2. Использование Common Controls: для создания элементов управления, таких как панель инструментов и списковые представления, в коде используется библиотека Common Controls (comctl32.lib), которая предоставляет стандартные элементы пользовательского интерфейса.

3. Панель инструментов (Toolbar): создается панель инструментов с помощью функции CreateSimpleToolbar(). Панель инструментов содержит кнопки для операций копирования, удаления, перемещения и создания файлов.

4. Списковые представления (ListView): для отображения файлов и директорий используются два списковых представления (hListView\_1 и hListView\_2). Функция InitListViewImageLists() инициализирует изображения для элементов в списках. Функции FindFile() и Viev\_List() позволяют загружать информацию о файлах и директориях в списки.

5. Выпадающие списки (ComboBox): два выпадающих списка (hComboBox\_1 и hComboBox\_2) используются для выбора дисковых устройств и переключения между директориями.

6. Обработка событий мыши и клавиш клавиатуры: в коде реализованы обработчики событий, такие как двойной щелчок на элементе списка, клик на кнопке панели инструментов и выбор элементов в выпадающих списках. В зависимости от события выполняются соответствующие операции, такие как открытие файла или директории, копирование, удаление и перемещение файлов.

7. Работа с файлами и директориями: функции CopyFile(), DeleteFile(), CreateFile(), FindFirstFile(), FindNextFile() используются для копирования, удаления и создания файлов, а также для поиска файлов и директорий.

8. Управление путями к директориям: текущие директории для обоих списковых представлений (dir и dir1) обновляются в соответствии с выбором в выпадающих списках.

9. Отображение файлов и директорий: файлы и директории отображаются в списках вместе с соответствующими иконками.

10. Интерфейс пользователя: пользователь может выполнять действия, такие как копирование, удаление и создание файлов, а также перемещение между директориями с использованием элементов управления в графическом интерфейсе приложения.

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы был реализован файловый менеджер с минимальной функциональностью, позволяющей пользователю копировать, перемещать и удалять несколько файлов одновременно, а также организована многозадачность, которая подразумевает под собой наличие двух списков файлов у пользователя, что позволяет работать с несколькими директориями одновременно. Результат работы программы показан на рисунке 3.1.

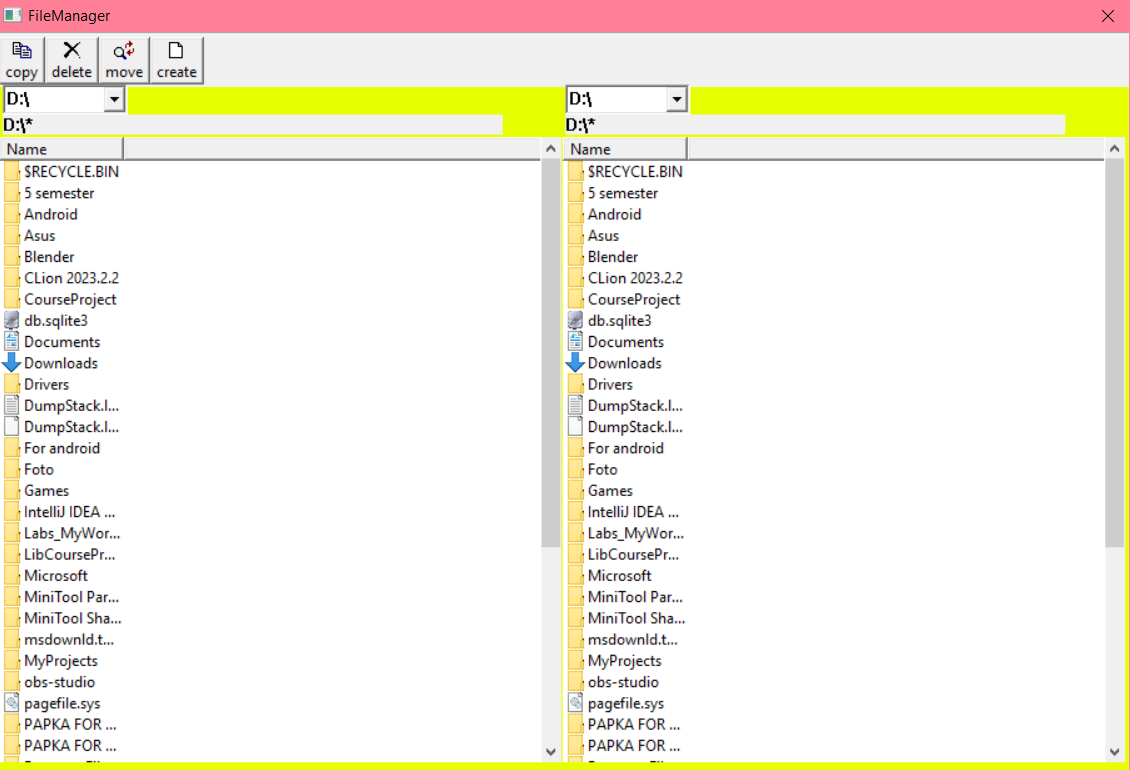


Рисунок 3.1 – Результат работы программы

Пользователь имеет возможность переходить в разные категории и производить нужные операции из минимального функционала файлового менеджера. Возможности перемещения файлов будут наглядно показаны на рисунках 3.3 и 3.4.

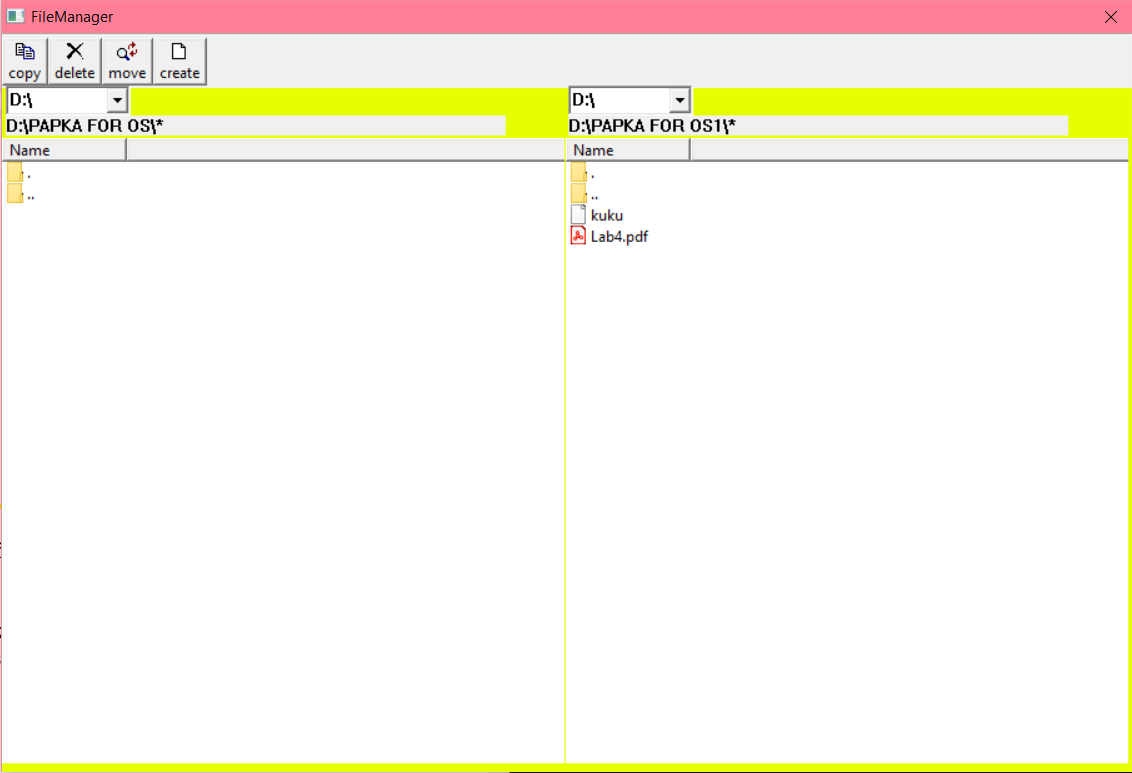


Рисунок 3.3 – Результат открытия различных директорий

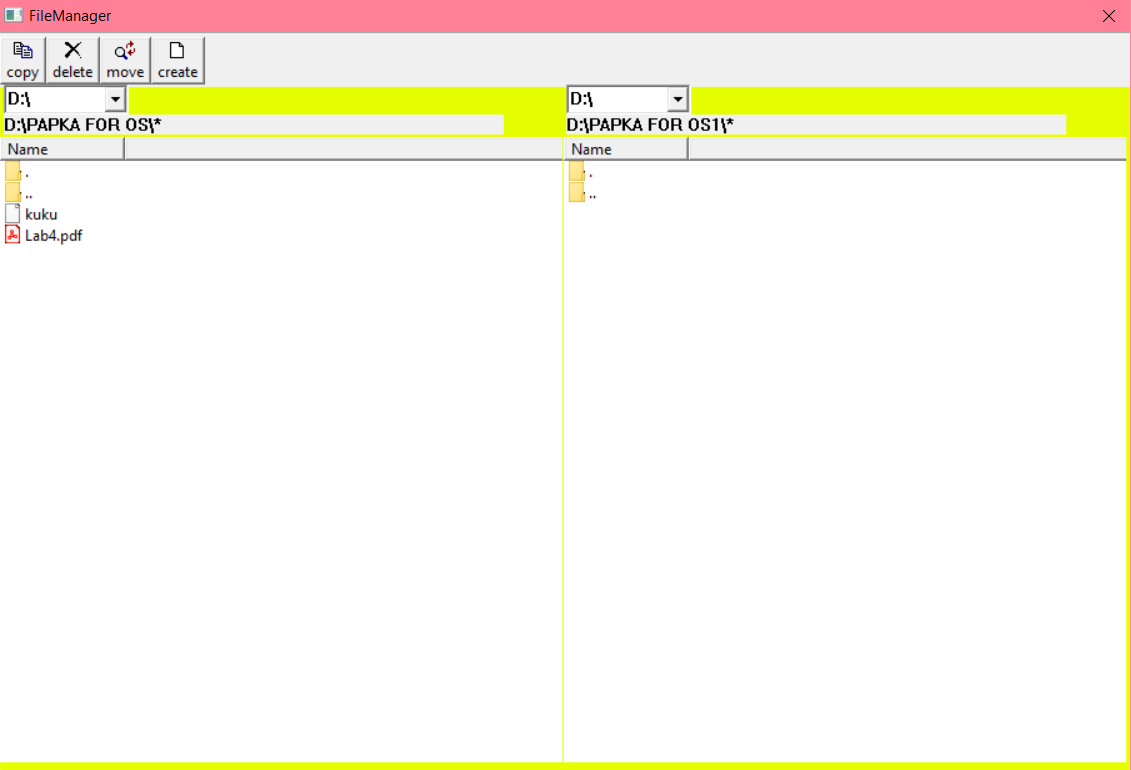


Рисунок 3.4 – Результат перемещения файлов

## ВЫВОДЫ

В ходе выполнения данной лабораторной работы был создан простой файловый менеджер с минимальной функциональностью. Менеджер позволяет пользователю просматривать содержимое директорий, копировать, перемещать и удалять файлы. Пользователь может выбрать путь к директории и просмотреть список файлов и поддиректорий в этой директории. Список выводится в консоль с использованием иконок для различения файлов и директорий.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Щупак Ю. Win32 API. Разработка приложений для Windows. – СПб: Питер, 2008. – 592 с.: ип.

[2] [Электронный ресурс]. – Режим доступа: [https://learn.microsoft.com/ru-ru/windows/win32/api – Дата доступа 03.10.2023](https://learn.microsoft.com/ru-ru/windows/win32/api%20–%20Дата%20доступа%2020.09.2023)

[3] [Электронный ресурс]. – Режим доступа:  [https://tproger.ru/articles/asynchronous-programming – Дата доступа 05.10.2023](https://stackoverflow.com/questions/17187265/how-to-group-radio-box-buttons-using-win32-api%20–%20Дата%20доступа%2023.09.2023)

[4] [Электронный ресурс]. – Режим доступа: https://www.softmagazin.ru/blog/total\_commander\_chto\_eto\_za\_programma\_i\_kak\_polzovatsya/ – Дата доступа 03.10.2023

[5] [Электронный ресурс]. – Режим доступа:  [https://ru.stackoverflow.com/questions/449291/ – Дата доступа 03.10.2023](https://learn.microsoft.com/en-us/windows/win32/gdi/windows-gdi%20–%20Дата%20доступа%2024.09.2023)

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

**Task.cpp**

#pragma comment(lib, "comctl32.lib") // for Common Controls

#pragma comment(lib, "comdlg32.lib") // for Common Dialogs

#include <Windows.h>

#include <commctrl.h>

#include <cstring>

#include <string>

#include <shlwapi.h>

#include <thread>

using namespace std;

#define BTNS\_AUTOSIZE 16

#define TB\_SETIMAGELIST (WM\_USER+48) // для установки изображения в панель инструментов

#define TB\_LOADIMAGES (WM\_USER+50) // для загрузки изображений в панель инструментов

#define IDM\_COPY 1001

#define IDM\_DEL 1003

#define IDM\_MOVE 1004

#define IDM\_CREATE 1005

#define ID\_LISTVIEW\_1 3001

#define ID\_LISTVIEW\_2 3002

#define ID\_COMBOBOX\_1 4001

#define ID\_COMBOBOX\_2 4002

#define ID\_LABEL\_3 5003

#define ID\_LABEL\_4 5004

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM); // Обработчик соозений для главного окна приложения

char nameMC[] = "FileManager";

char dir[MAX\_PATH] = "D:\\\*"; // Начальные пути к директориям

char dir1[MAX\_PATH] = "D:\\\*"; // -//-

char copy\_buf1[MAX\_PATH];

// Элементы хранения

HINSTANCE hInstance; // экземпляр текущего окна

HIMAGELIST g\_hImageList = NULL; // изображения

// Храним уникальные идентификаторы

HWND hWndChild = NULL; // оконный

static HWND hListView\_1, hListView\_2;

static HWND hComboBox\_1, hComboBox\_2;

static HWND hLabel\_1, hLabel\_2, hLabel\_3, hLabel\_4, hToolBar;

// Создание панели инстурментов

const int ImageListID = 0;

const int numButtons = 4;

const DWORD buttonStyles = BTNS\_AUTOSIZE;

const int bitmapSize = 16;

OPENFILENAME ofn;

char szFileName[MAX\_PATH];

TBBUTTON tbButtons[numButtons] =

{

{MAKELONG(STD\_COPY, ImageListID), IDM\_COPY, TBSTATE\_ENABLED,

BTNS\_AUTOSIZE, {0}, 0, (INT\_PTR) "copy"},

{MAKELONG(STD\_DELETE, ImageListID), IDM\_DEL, TBSTATE\_ENABLED,

buttonStyles, {0}, 0, (INT\_PTR) "delete"},

{MAKELONG(STD\_REPLACE, ImageListID), IDM\_MOVE, TBSTATE\_ENABLED,

buttonStyles, {0}, 0, (INT\_PTR) "move"},

{MAKELONG(STD\_FILENEW, ImageListID), IDM\_CREATE, TBSTATE\_ENABLED,

buttonStyles, {0}, 0, (INT\_PTR) "create"},

};

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,

LPSTR lpszCmdLine, int nCmdShow) {

MSG Msg;

HWND hWnd;

WNDCLASS WndClass;

WndClass.style = CS\_VREDRAW | CS\_HREDRAW;

WndClass.cbClsExtra = 0;

WndClass.cbWndExtra = 0;

WndClass.lpfnWndProc = WndProc;

WndClass.hInstance = hInstance;

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

WndClass.hbrBackground = (HBRUSH) CreateSolidBrush(RGB(230, 255, 0));

WndClass.hIcon = LoadIcon(NULL, IDI\_APPLICATION);

WndClass.lpszMenuName = NULL;

WndClass.lpszClassName = nameMC;

if (!RegisterClass(&WndClass))

return FALSE;

hWnd = CreateWindow(nameMC,

"FileManager", WS\_BORDER | WS\_SYSMENU | WS\_VISIBLE,

100, 100, 910, 620,

NULL, NULL, hInstance, NULL);

InitCommonControls();

while (GetMessage(&Msg, NULL, 0, 0)) {

DispatchMessage(&Msg);

TranslateMessage(&Msg);

}

return Msg.wParam;

}

HWND CreateSimpleToolbar(HWND hWndParent) {

HWND hWndToolbar = CreateWindowEx(0, TOOLBARCLASSNAME, NULL,

WS\_CHILD | TBSTYLE\_WRAPABLE,

0, 0, 0, 0,

hWndParent, NULL, hInstance, NULL);

if (hWndToolbar == NULL) { return NULL; }

g\_hImageList = ImageList\_Create(

bitmapSize, bitmapSize,

ILC\_COLOR16 | ILC\_MASK,

numButtons, 0);

SendMessage(hWndToolbar, TB\_SETIMAGELIST, (WPARAM) ImageListID,

(LPARAM) g\_hImageList);

SendMessage(hWndToolbar, TB\_LOADIMAGES, (WPARAM) IDB\_STD\_SMALL\_COLOR,

(LPARAM) HINST\_COMMCTRL);

SendMessage(hWndToolbar, TB\_BUTTONSTRUCTSIZE, (WPARAM) sizeof(TBBUTTON), 0);

SendMessage(hWndToolbar, TB\_ADDBUTTONS, (WPARAM) numButtons, (LPARAM) &tbButtons);

SendMessage(hWndToolbar, TB\_AUTOSIZE, 0, 0);

ShowWindow(hWndToolbar, TRUE);

return hWndToolbar;

}

void AddColToListView(const char \*st, int sub, int size) {

LVCOLUMN lvc;

lvc.mask = LVCF\_FMT | LVCF\_WIDTH | LVCF\_TEXT | LVCF\_SUBITEM;

lvc.iSubItem = sub;

lvc.pszText = const\_cast<LPSTR>(st);

lvc.cx = size;

lvc.fmt = LVCFMT\_LEFT;

ListView\_InsertColumn(hListView\_1, sub, &lvc);

ListView\_InsertColumn(hListView\_2, sub, &lvc);

}

int i;

BOOL InitListViewImageLists(HWND hWndListView, int size, char c\_dir[MAX\_PATH]) {

HIMAGELIST hSmall;

SHFILEINFO lp;

hSmall = ImageList\_Create(GetSystemMetrics(SM\_CXSMICON), GetSystemMetrics(SM\_CYSMICON), ILC\_MASK | ILC\_COLOR32,

size + 2, 1);

WIN32\_FIND\_DATA FindFileData;

HANDLE hFind;

hFind = FindFirstFile(c\_dir, &FindFileData);

if (hFind == INVALID\_HANDLE\_VALUE) {

MessageBox(0, "Error", "Not found", MB\_OK | MB\_ICONWARNING);

} else {

do {

if (strcmp(FindFileData.cFileName, ".") == 0) {

char buf1[MAX\_PATH];

strcpy(buf1, c\_dir);

strcat(buf1, FindFileData.cFileName);

SHGetFileInfo("", FILE\_ATTRIBUTE\_DEVICE, &lp, sizeof(lp),

SHGFI\_ICONLOCATION | SHGFI\_ICON | SHGFI\_SMALLICON);

ImageList\_AddIcon(hSmall, lp.hIcon);

DestroyIcon(lp.hIcon);

}

if (strcmp(FindFileData.cFileName, "..") == 0) {//если фаилы,папки

char buf1[MAX\_PATH];

strcpy(buf1, c\_dir);

strcat(buf1, FindFileData.cFileName);

SHGetFileInfo("", FILE\_ATTRIBUTE\_DIRECTORY, &lp, sizeof(lp),

SHGFI\_ICONLOCATION | SHGFI\_ICON | SHGFI\_SMALLICON);

ImageList\_AddIcon(hSmall, lp.hIcon);

DestroyIcon(lp.hIcon);

}

char buf1[MAX\_PATH];

strcpy(buf1, c\_dir);

buf1[strlen(buf1) - 1] = 0;

strcat(buf1, FindFileData.cFileName);

DWORD num = GetFileAttributes(buf1);

SHGetFileInfo(buf1, num, &lp, sizeof(lp), SHGFI\_ICONLOCATION | SHGFI\_ICON | SHGFI\_SMALLICON);

ImageList\_AddIcon(hSmall, lp.hIcon);

DestroyIcon(lp.hIcon);

} while (FindNextFile(hFind, &FindFileData) != 0);

FindClose(hFind);

}

ListView\_SetImageList(hWndListView, hSmall, LVSIL\_SMALL);

return TRUE;

}

void ListView\_GetSelectedItems(HWND hWndListView, int \*selectedItems) {

int index = -1;

int count = 0;

while ((index = ListView\_GetNextItem(hWndListView, index, LVNI\_SELECTED)) != -1) {

selectedItems[count] = index;

count++;

}

}

void Viev\_List(const char \*buf, HWND hList, int i, int j) {

LVITEM lvItem;

lvItem.mask = LVIF\_IMAGE | LVIF\_TEXT;

lvItem.state = 0;

lvItem.stateMask = 0;

lvItem.iItem = i;

lvItem.iImage = i;

lvItem.iSubItem = j;

lvItem.pszText = const\_cast<LPSTR>(buf);

lvItem.cchTextMax = sizeof(buf);

ListView\_InsertItem(hList, &lvItem);

}

void FindFile(HWND hList, char c\_dir[MAX\_PATH]) {

SendMessage(hList, LVM\_DELETEALLITEMS, (WPARAM) 0, (LPARAM) 0);

i = 0;

WIN32\_FIND\_DATA FindFileData;

HANDLE hFind;

hFind = FindFirstFile(c\_dir, &FindFileData);

if (hFind == INVALID\_HANDLE\_VALUE) {

MessageBox(0, "Error", "Not found", MB\_OK | MB\_ICONWARNING);

} else {

do {

if ((FILE\_ATTRIBUTE\_DIRECTORY & GetFileAttributes(FindFileData.cFileName)) != FILE\_ATTRIBUTE\_DIRECTORY)

Viev\_List("File", hList, i, 1);

else Viev\_List("Directory", hList, i, 1);

Viev\_List(FindFileData.cFileName, hList, i,

0);

++i;

} while (FindNextFile(hFind, &FindFileData) != 0);

FindClose(hFind);

InitListViewImageLists(hList, i, c\_dir);

}

}

bool ShowCreateFileDialog(HWND hWnd) {

ZeroMemory(&ofn, sizeof(OPENFILENAME));

ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = hWnd;

ofn.lpstrFilter = "Text Files (\*.txt)\0\*.txt\0All Files (\*.\*)\0\*.\*\0";

ofn.lpstrFile = szFileName;

ofn.nMaxFile = MAX\_PATH;

ofn.lpstrTitle = "Create Text File";

ofn.Flags = OFN\_EXPLORER | OFN\_PATHMUSTEXIST | OFN\_HIDEREADONLY | OFN\_OVERWRITEPROMPT;

return GetSaveFileName(&ofn);

}

void CopyFileAsync(const char \*source, const char \*destination) {

std::thread([=] {

// Ваш код для операции копирования файла

CopyFile(source, destination, FALSE);

}).detach(); // Отсоединяем поток, чтобы он работал асинхронно

}

void MoveFileAsync(const char \*source, const char \*destination) {

std::thread([=] {

if (MoveFile(source, destination)) {

// Перемещение файла успешно, теперь можно удалить исходный файл

DeleteFile(source);

}

}).detach(); // Отсоединяем поток, чтобы он работал асинхронно

}

void DelFileAsync(const char \*source) {

std::thread([=] {

DeleteFile(source);

}).detach(); // Отсоединяем поток, чтобы он работал асинхронно

}

LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam) {

DWORD Drivers;

int i;

char buf[] = " ";

char \*ls;

string s;

int y = 9;

char buf1[MAX\_PATH];

LPNMHDR lpnmHdr;

lpnmHdr = (LPNMHDR) lParam;

LPNMLISTVIEW pnmLV = (LPNMLISTVIEW) lParam;

GetLogicalDrives();

GetLogicalDriveStrings(128, (LPTSTR) buf);

switch (msg) {

case WM\_COMMAND: {

switch (LOWORD(wParam)) {

case IDM\_CREATE: {

if (ShowCreateFileDialog(hwnd)) {

if (PathFindExtension(szFileName) == NULL ||

\_stricmp(PathFindExtension(szFileName), ".txt") != 0) {

strcat(szFileName, ".txt");

}

HANDLE hFile = CreateFile(szFileName, GENERIC\_WRITE, 0, NULL, CREATE\_NEW, FILE\_ATTRIBUTE\_NORMAL,

NULL);

if (hFile != INVALID\_HANDLE\_VALUE) {

CloseHandle(hFile);

HWND activeListView = (GetFocus() == hListView\_1) ? hListView\_1 : hListView\_2;

char \*targetDir = (activeListView == hListView\_1) ? dir : dir1;

FindFile(activeListView, targetDir);

} else {

MessageBox(hwnd, "Failed to create the file.", "Error", MB\_ICONERROR | MB\_OK);

}

}

return 0;

}

case IDM\_COPY: {

HWND activeListView = (GetFocus() == hListView\_1) ? hListView\_1 : hListView\_2;

int itemCount = ListView\_GetSelectedCount(activeListView);

if (itemCount > 0) {

int \*selectedItems = new int[itemCount];

ListView\_GetSelectedItems(activeListView, selectedItems);

char \*targetDir = (activeListView == hListView\_1) ? dir : dir1;

for (int i = 0; i < itemCount; i++) {

LVITEM lvItem;

lvItem.iItem = selectedItems[i];

lvItem.iSubItem = 0;

lvItem.mask = LVIF\_TEXT;

lvItem.pszText = copy\_buf1;

lvItem.cchTextMax = sizeof(copy\_buf1);

if (ListView\_GetItem(activeListView, &lvItem)) {

char cm\_dir\_from[MAX\_PATH];

char cm\_dir\_to[MAX\_PATH], cm\_dir\_to\_[MAX\_PATH];

strcpy(cm\_dir\_from, targetDir);

cm\_dir\_from[strlen(cm\_dir\_from) - 1] = 0;

strcat(cm\_dir\_from, copy\_buf1);

char \*otherTargetDir = (activeListView == hListView\_1) ? dir1 : dir;

strcpy(cm\_dir\_to, otherTargetDir);

strcpy(cm\_dir\_to\_, otherTargetDir);

cm\_dir\_to[strlen(cm\_dir\_to) - 1] = 0;

strcat(cm\_dir\_to, copy\_buf1);

// Создайте динамическую копию имен файла

char \*sourceFileName = new char[MAX\_PATH];

char \*destFileName = new char[MAX\_PATH];

strcpy(sourceFileName, cm\_dir\_from);

strcpy(destFileName, cm\_dir\_to);

// Запускаем операцию копирования в отдельном потоке с использованием копий имен файлов

CopyFileAsync(sourceFileName, destFileName);

// Освобождаем память после завершения операции

delete[] sourceFileName;

delete[] destFileName;

}

}

delete[] selectedItems;

// Необходимо перерисовать ListView после завершения операции

FindFile(activeListView, targetDir);

HWND otherListView = (activeListView == hListView\_1) ? hListView\_2 : hListView\_1;

char \*otherTargetDir = (activeListView == hListView\_1) ? dir1 : dir;

FindFile(otherListView, otherTargetDir);

}

return 0;

}

case IDM\_DEL: {

HWND activeListView = (GetFocus() == hListView\_1) ? hListView\_1 : hListView\_2;

int itemCount = ListView\_GetSelectedCount(activeListView);

if (itemCount > 0) {

int \*selectedItems = new int[itemCount];

ListView\_GetSelectedItems(activeListView, selectedItems);

char \*targetDir = (activeListView == hListView\_1) ? dir : dir1;

for (int i = 0; i < itemCount; i++) {

LVITEM lvItem;

lvItem.iItem = selectedItems[i];

lvItem.iSubItem = 0;

lvItem.mask = LVIF\_TEXT;

lvItem.pszText = copy\_buf1;

lvItem.cchTextMax = sizeof(copy\_buf1);

if (ListView\_GetItem(activeListView, &lvItem)) {

char cm\_dir\_from[MAX\_PATH];

char cm\_dir\_to[MAX\_PATH], cm\_dir\_to\_[MAX\_PATH];

strcpy(cm\_dir\_from, targetDir);

cm\_dir\_from[strlen(cm\_dir\_from) - 1] = 0;

strcat(cm\_dir\_from, copy\_buf1);

char \*otherTargetDir = (activeListView == hListView\_1) ? dir1 : dir;

strcpy(cm\_dir\_to, otherTargetDir);

strcpy(cm\_dir\_to\_, otherTargetDir);

cm\_dir\_to[strlen(cm\_dir\_to) - 1] = 0;

strcat(cm\_dir\_to, copy\_buf1);

// Создаем динамическую копию имен файла

char \*sourceFileName = new char[MAX\_PATH];

char \*destFileName = new char[MAX\_PATH];

strcpy(sourceFileName, cm\_dir\_from);

strcpy(destFileName, cm\_dir\_to);

// Запускаем операцию удаления в отдельном потоке с использованием копий имен файлов

DelFileAsync(sourceFileName);

// Освобождаем память после завершения операции

delete[] sourceFileName;

delete[] destFileName;

}

}

delete[] selectedItems;

// Необходимо перерисовать ListView после завершения операции

FindFile(activeListView, targetDir);

HWND otherListView = (activeListView == hListView\_1) ? hListView\_2 : hListView\_1;

char \*otherTargetDir = (activeListView == hListView\_1) ? dir1 : dir;

FindFile(otherListView, otherTargetDir);

}

return 0;

}

case IDM\_MOVE: {

HWND activeListView = (GetFocus() == hListView\_1) ? hListView\_1 : hListView\_2;

int itemCount = ListView\_GetSelectedCount(activeListView);

if (itemCount > 0) {

int \*selectedItems = new int[itemCount];

ListView\_GetSelectedItems(activeListView, selectedItems);

char \*targetDir = (activeListView == hListView\_1) ? dir : dir1;

for (int i = 0; i < itemCount; i++) {

LVITEM lvItem;

lvItem.iItem = selectedItems[i];

lvItem.iSubItem = 0;

lvItem.mask = LVIF\_TEXT;

lvItem.pszText = copy\_buf1;

lvItem.cchTextMax = sizeof(copy\_buf1);

if (ListView\_GetItem(activeListView, &lvItem)) {

char cm\_dir\_from[MAX\_PATH];

char cm\_dir\_to[MAX\_PATH], cm\_dir\_to\_[MAX\_PATH];

strcpy(cm\_dir\_from, targetDir);

cm\_dir\_from[strlen(cm\_dir\_from) - 1] = 0;

strcat(cm\_dir\_from, copy\_buf1);

char \*otherTargetDir = (activeListView == hListView\_1) ? dir1 : dir;

strcpy(cm\_dir\_to, otherTargetDir);

strcpy(cm\_dir\_to\_, otherTargetDir);

cm\_dir\_to[strlen(cm\_dir\_to) - 1] = 0;

strcat(cm\_dir\_to, copy\_buf1);

// Создайте динамическую копию имен файла

char \*sourceFileName = new char[MAX\_PATH];

char \*destFileName = new char[MAX\_PATH];

strcpy(sourceFileName, cm\_dir\_from);

strcpy(destFileName, cm\_dir\_to);

// Запускаем операцию перемещения в отдельном потоке с использованием копий имен файлов

MoveFileAsync(sourceFileName, destFileName);

// Освобождаем память после завершения операции

delete[] sourceFileName;

delete[] destFileName;

}

}

delete[] selectedItems;

// Необходимо перерисовать ListView после завершения операции

FindFile(activeListView, targetDir);

HWND otherListView = (activeListView == hListView\_1) ? hListView\_2 : hListView\_1;

char \*otherTargetDir = (activeListView == hListView\_1) ? dir1 : dir;

FindFile(otherListView, otherTargetDir);

}

return 0;

}

case ID\_COMBOBOX\_1: {

switch (HIWORD(wParam)) {

case CBN\_SELENDOK: {

char path[80];

int sel;

strcpy(path, "");

sel = SendMessage(hComboBox\_1, CB\_GETCURSEL, 0, 0);

SendMessage(hComboBox\_1, CB\_GETLBTEXT, sel, (LPARAM) path);

strcat(path, "\\*");

strcpy(dir, path);

SetWindowText(hLabel\_3, dir);

FindFile(hListView\_1, dir);

return 0;

}

default:

return 0;

}

}

case ID\_COMBOBOX\_2: {

switch (HIWORD(wParam)) {

case CBN\_SELENDOK: {

char path[80];

int sel;

strcpy(path, "");

sel = SendMessage(hComboBox\_2, CB\_GETCURSEL, 0, 0);

SendMessage(hComboBox\_2, CB\_GETLBTEXT, sel, (LPARAM) path);

strcat(path, "\\*");

strcpy(dir1, path);

SetWindowText(hLabel\_4, dir1);

FindFile(hListView\_2, dir1);

return 0;

}

default:

return 0;

}

}

return 0;

}

}

case WM\_NOTIFY: {

if ((lpnmHdr->idFrom == ID\_LISTVIEW\_1) &&

(lpnmHdr->code == NM\_CLICK)) {

ListView\_GetItemText(lpnmHdr->hwndFrom, pnmLV->iItem, pnmLV->iSubItem, buf1, MAX\_PATH);

strcpy(copy\_buf1, buf1);

char \_dir[MAX\_PATH];

strcpy(\_dir, dir);

\_dir[strlen(\_dir) - 1] = 0;

strcat(\_dir, buf1);

}

if (((lpnmHdr->idFrom == ID\_LISTVIEW\_1) || (lpnmHdr->idFrom == ID\_LISTVIEW\_2)) &&

(lpnmHdr->code == NM\_DBLCLK)) {

ListView\_GetItemText(lpnmHdr->hwndFrom, pnmLV->iItem, pnmLV->iSubItem, buf1, MAX\_PATH);

if (lpnmHdr->idFrom == ID\_LISTVIEW\_1) {

char \_dir[MAX\_PATH];

int k = 0;

strcpy(\_dir, dir);

\_dir[strlen(\_dir) - 1] = 0;

strcat(\_dir, buf1);

for (int i = 0; i < strlen(buf1); i++) {

string s;

s = buf1[i];

if (s == ".") k = i;

}

if ((k != 0) && (k != 1)) {

ShellExecute(0, "open", \_dir, NULL, NULL, SW\_SHOWNORMAL);

} else {

if (strcmp(buf1, "..") == 0) {

int k;

dir[strlen(dir) - 2] = 0;

for (i = 0; i < strlen(dir); i++) {

string s;

s = dir[i];

if (s == "\\") k = i;

}

dir[k + 1] = 0;

strcat(dir, "\*");

} else if (strcmp(buf1, ".") == 0) {

dir[3] = 0;

strcat(dir, "\*");

} else {

strcat(buf1, "\\\*");

dir[strlen(dir) - 1] = 0;

strcat(dir, buf1);

}

SetWindowText(hLabel\_3, dir);

FindFile(hListView\_1, dir);

}

} else if (lpnmHdr->idFrom == ID\_LISTVIEW\_2) {

char \_dir1[MAX\_PATH];

int k = 0;

strcpy(\_dir1, dir1);

\_dir1[strlen(\_dir1) - 1] = 0;

strcat(\_dir1, buf1);

for (int i = 0; i < strlen(buf1); i++) {

string s;

s = buf1[i];

if (s == ".") k = i;

}

if ((k != 0) && (k != 1)) {

ShellExecute(0, "open", \_dir1, NULL, NULL, SW\_SHOWNORMAL);

} else {

if (strcmp(buf1, "..") == 0) {

int k;

dir1[strlen(dir1) - 2] = 0;

for (i = 0; i < strlen(dir1); i++) {

string s;

s = dir1[i];

if (s == "\\") k = i;

}

dir1[k + 1] = 0;

strcat(dir1, "\*");

} else if (strcmp(buf1, ".") == 0) {

dir1[3] = 0;

strcat(dir1, "\*");

} else {

strcat(buf1, "\\\*");

dir1[strlen(dir1) - 1] = 0;

strcat(dir1, buf1);

}

SetWindowText(hLabel\_4, dir1);

FindFile(hListView\_2, dir1);

}

}

}

break;

}

case WM\_CREATE: {

hToolBar = CreateSimpleToolbar(hwnd);

hLabel\_3 = CreateWindow("static", "path\_left",

WS\_CHILD | WS\_VISIBLE | WS\_TABSTOP,

3, 57 + y, 400, 16,

hwnd, (HMENU) ID\_LABEL\_3, hInstance, NULL);

hLabel\_4 = CreateWindow("static", "path\_right",

WS\_CHILD | WS\_VISIBLE | WS\_TABSTOP,

453, 57 + y, 400, 16,

hwnd, (HMENU) ID\_LABEL\_4, hInstance, NULL);

hComboBox\_1 = CreateWindow("ComboBox", NULL,

WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | CBS\_DROPDOWN | CBS\_SORT,

3, 33 + y, 100, 110, hwnd,

(HMENU) ID\_COMBOBOX\_1, hInstance, NULL);

hComboBox\_2 = CreateWindow("ComboBox", NULL,

WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | CBS\_DROPDOWN | CBS\_SORT,

453, 33 + y, 100, 110, hwnd,

(HMENU) ID\_COMBOBOX\_2, hInstance, NULL);

hListView\_1 = CreateWindow(WC\_LISTVIEW, NULL,

LVS\_REPORT | WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | LVS\_AUTOARRANGE,

0, 60 + y + 15, 450, 500, hwnd,

(HMENU) ID\_LISTVIEW\_1, hInstance, NULL);

hListView\_2 = CreateWindow(WC\_LISTVIEW, NULL,

LVS\_REPORT | WS\_CHILD | WS\_VISIBLE | WS\_CLIPSIBLINGS | LVS\_AUTOARRANGE,

451, 60 + y + 15, 450, 500, hwnd,

(HMENU) ID\_LISTVIEW\_2, hInstance, NULL);

ls = buf;

while (\*ls) {

SendMessage(hComboBox\_1, CB\_ADDSTRING, 0, (LPARAM) ls);

SendMessage(hComboBox\_2, CB\_ADDSTRING, 0, (LPARAM) ls);

ls += strlen(ls) + 1;

}

SendMessage(hComboBox\_1, CB\_SETCURSEL, 1, 0);

SendMessage(hComboBox\_2, CB\_SETCURSEL, 1, 0);

AddColToListView("Name", 1, 100);

SetWindowText(hLabel\_3, dir);

FindFile(hListView\_1, dir);

SetWindowText(hLabel\_3, dir);

FindFile(hListView\_2, dir);

SetWindowText(hLabel\_4, dir1);

return 0;

}

case WM\_DESTROY: {

DestroyWindow(hListView\_1);

DestroyWindow(hListView\_2);

DestroyWindow(hComboBox\_1);

DestroyWindow(hComboBox\_2);

DestroyWindow(hLabel\_1);

DestroyWindow(hLabel\_2);

DestroyWindow(hLabel\_3);

DestroyWindow(hLabel\_4);

DestroyWindow(hToolBar);

PostQuitMessage(0);

return 0;

}

}

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

}