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

ОТЧЁТ

к лабораторной работе №3

на тему

Управление памятью и вводом-выводом, расширенные возможности

ввода-вывода *Windows*

Выполнил: студент группы 153503

Жечко Иван Сергеевич

Проверил: Гриценко Никита Юрьевич

Минск 2023

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146728177)

[2 Краткие теоретические сведения 4](#_Toc146728178)

[3 Результаты выполнения лабораторной работы 5](#_Toc146728179)

[Выводы 8](#_Toc146728180)

[Список использованных источников 9](#_Toc146728181)

[Приложение А (обязательное) Листинг кода 10](#_Toc146728182)

## 1 ЦЕЛЬ РАБОТЫ

Целью выполнения лабораторной работы является создание оконного приложения на *Win32 API* реализующее управление памятью и вводом-выводом, работу с файлами, в том числе асинхронную. Для выполнения цели была реализована программа файловый менеджер.

## 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Для асинхронной работы с файлами были использованы функции *CreateFile* и *WriteFile*. Для реализации неблокирующей асинхронности используется флаг *FILE\_FLAG\_OVERLAPPED*. Неблокирующая асинхронность – это подход к организации асинхронных вычислений, при котором приложение может продолжать работу, не дожидаясь завершения операции ввода-вывода. Это достигается путем использования неблокирующих операций ввода-вывода, которые не блокируют поток исполнения, а возвращают управление приложению, позволяя ему продолжать работу.

Для пользовательского ввода используются форма *Edit* и контекстное меню *PopupMenu*.

## 3 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы был разработан файловый менеджер. Менеджер представляет собой набор кнопок, отображающих файлы и директории по пути, введенному в поле для пути. Начальный экран изображен на рисунке 3.1.

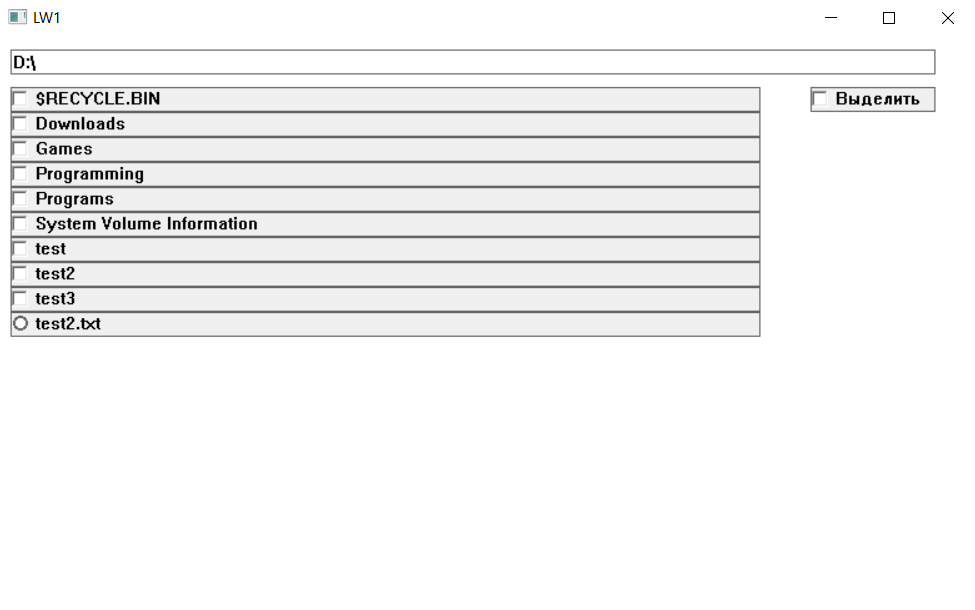


Рисунок 3.1 – Начальный экран

Для файлов по нажатию на правую кнопку мыши доступно контекстное меню. Пример контекстного меню для файла изображен на рисунке 3.2.

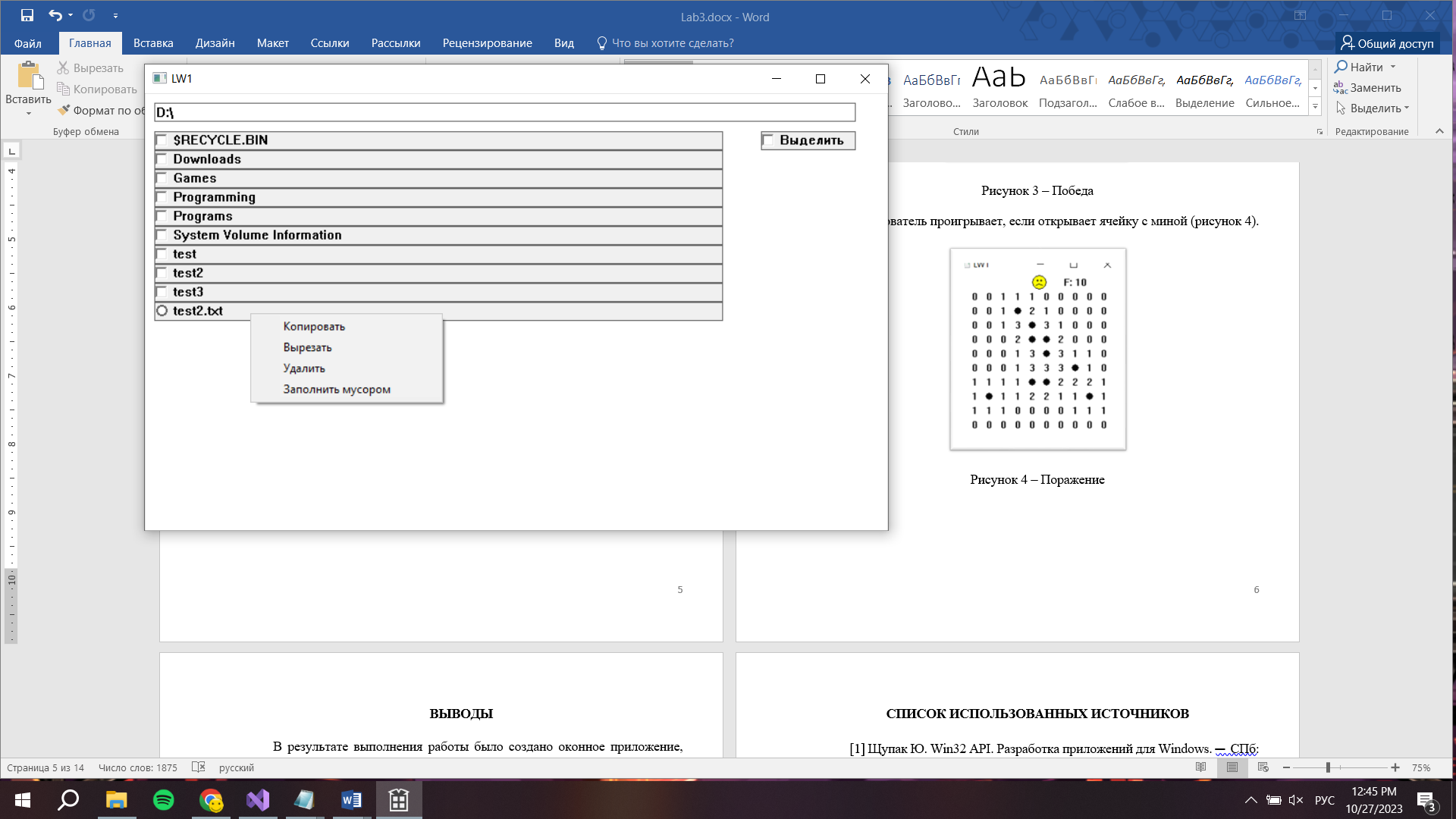


Рисунок 3.2 ─ Контекстное меню функций файла

При выборе функции “Выделить” (при нажатии на кнопку “Выделить”) пользователь может выбрать файлы нажатием на них для применения функций сразу над несколькими файлами. Пример выделения файлов изображен на рисунке 3.3.

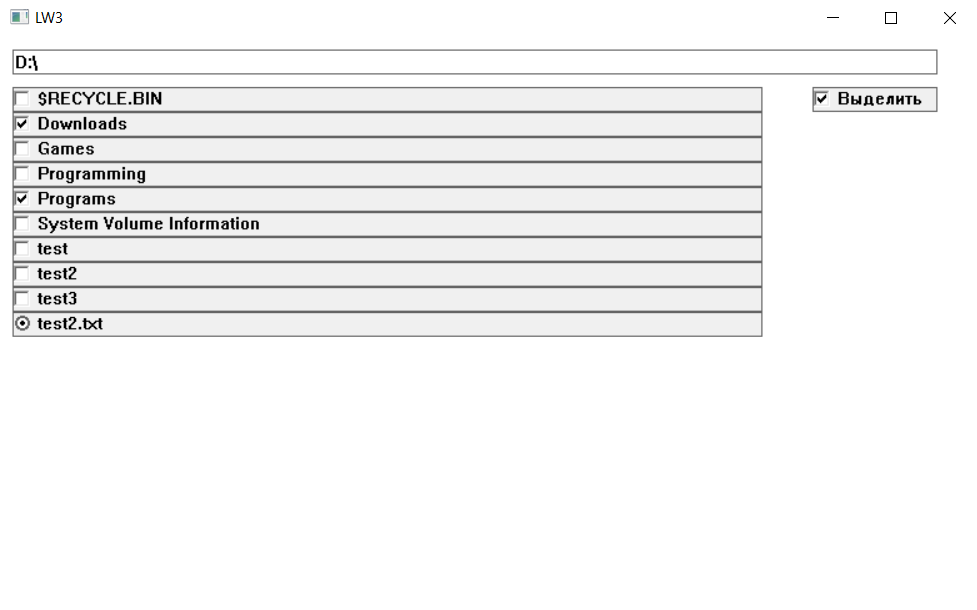


Рисунок 3.3 – Выделение файлов

После копирования или вырезания файла или группы файлов и перехода в нужную директорию пользователь может вставить данные в текущую выбранную директорию через контекстное меню. Контекстное меню вставки в директорию изображено на рисунке 3.4.

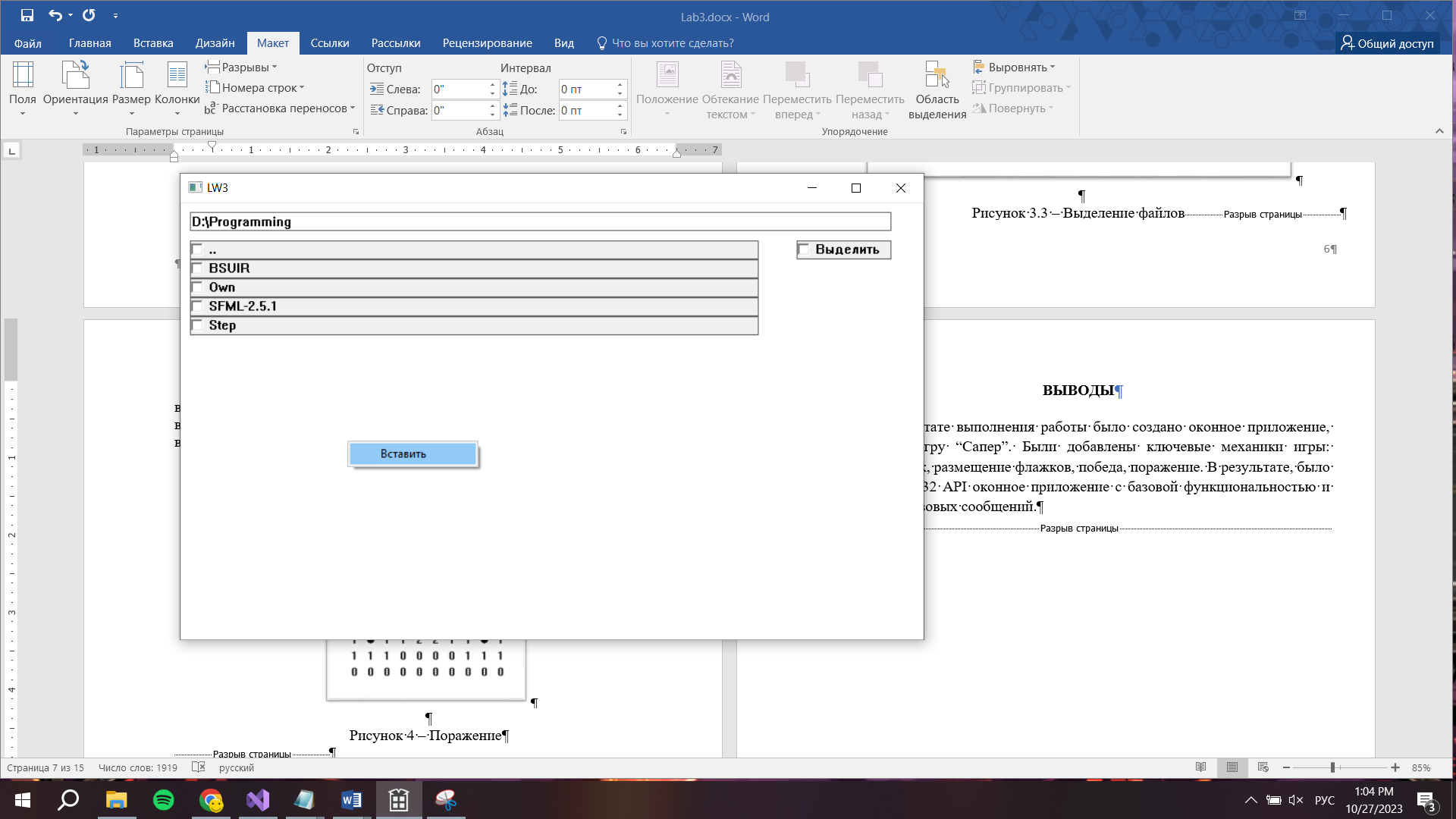


Рисунок 4 – Меню вставки файлов

## ВЫВОДЫ

В результате выполнения работы было создано оконное приложение файловый менеджер. Была реализована работа с файлами, в том числе асинхронная, управление памятью и вводом-выводом.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Programming reference for the Win32 API [Электронный ресурс]. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32/api/ – Дата доступа: 27.10.2023.

## ПРИЛОЖЕНИЕ А

## (обязательное)

## Листинг кода

#include <windows.h>

#include <tchar.h>

#include <time.h>

#include <string>

#include <vector>

using namespace std;

#define ID\_COPY 1001

#define ID\_CUTOUT 1002

#define ID\_DELETE 1003

#define ID\_PASTE 1004

#define ID\_FILL 1005

#define SELECT\_ID 1006

HWND mainWindow;

HWND pathEdit;

HWND selectButton;

HHOOK hHook;

wstring filePath; //путь файла у которого открыто контекстное меню

LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam, LPARAM lParam);

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

WNDPROC editWndProc;

bool selectActive = false;

wstring currentPath; //текущая директрия

std::vector<wstring> selectedGroup; //буфер для всех путей к файлам

LPVOID buffer;

int cntSelected = 0; //кол-во выделенных файлов (применяется в удалении файлов)

bool cutOperation = false; //выделенные файлы необходимо при вставке перемещать

char\* bigbuffer = new char[2000000000]; //для работы с файлами

bool IsDirectory(wstring path)

{

DWORD attributes = GetFileAttributes(path.c\_str());

return (attributes != INVALID\_FILE\_ATTRIBUTES && (attributes & FILE\_ATTRIBUTE\_DIRECTORY));

}

void change\_folder(wstring path) {

if (path.back() == '\\') {

path.pop\_back();

}

EnumChildWindows(mainWindow, [](HWND hWndChild, LPARAM lParam) -> BOOL { //удаление кнопок

wchar\_t className[256];

GetClassName(hWndChild, className, 256);

if (hWndChild == selectButton) { return TRUE; }

if (wcscmp(className, L"Button") == 0) {

DestroyWindow(hWndChild);

}

return TRUE;

}, NULL);

auto result = path;

if (path.back() != '\*') {

result = result + L"\\\*";

}

WIN32\_FIND\_DATA FindFileData;

HANDLE hFind;

hFind = FindFirstFile(result.c\_str(), &FindFileData);

int filecnt = 0;

if (hFind == INVALID\_HANDLE\_VALUE)

{

return;

}

do

{

if (FindFileData.cFileName != wstring(L".")) {

if (FindFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY) { //папка

HWND hWnd = CreateWindow(L"BUTTON", FindFileData.cFileName, WS\_VISIBLE | WS\_CHILD | BS\_LEFT | BS\_CHECKBOX | WS\_BORDER, 10, 40 + filecnt \* 20, 600, 20, mainWindow, NULL, (HINSTANCE)GetWindowLongPtr(mainWindow, GWLP\_HINSTANCE), NULL);

filecnt++;

}

}

} while (FindNextFile(hFind, &FindFileData));

hFind = FindFirstFile(result.c\_str(), &FindFileData);

if (hFind == INVALID\_HANDLE\_VALUE)

{

return;

}

do

{

if (FindFileData.cFileName != wstring(L".")) {

if (!(FindFileData.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)) { // файл

HWND hWnd = CreateWindow(L"BUTTON", FindFileData.cFileName, WS\_VISIBLE | WS\_CHILD | BS\_LEFT | BS\_FLAT | BS\_RADIOBUTTON | WS\_BORDER, 10, 40 + filecnt \* 20, 600, 20, mainWindow, NULL, (HINSTANCE)GetWindowLongPtr(mainWindow, GWLP\_HINSTANCE), NULL);

filecnt++;

}

}

} while (FindNextFile(hFind, &FindFileData));

currentPath = path;

FindClose(hFind);

}

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR pCmdLine, int nCmdShow)

{

// Window class.

const wchar\_t CLASS\_NAME[] = L"Window";

WNDCLASS wc = { };

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

const int window\_x = 800;

const int window\_y = 500;

// Window creation

HWND hwnd = CreateWindowEx(

0,

CLASS\_NAME,

L"LW3",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, window\_x, window\_y,

NULL,

NULL,

hInstance,

NULL

);

if (hwnd == NULL)

{

return 0;

}

SetWindowLongPtr(hwnd, GWL\_STYLE, GetWindowLongPtr(hwnd, GWL\_STYLE) | WS\_EX\_ACCEPTFILES); //устанвка кодировки UTF-8

ShowWindow(hwnd, nCmdShow);

// Message loop.

MSG msg = {};

while (GetMessage(&msg, NULL, 0, 0) > 0)

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

LRESULT CALLBACK editProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

case WM\_KEYDOWN: {

if (wParam == VK\_RETURN)

{

wchar\_t buffer[2048];

GetWindowTextW((HWND)pathEdit, buffer, sizeof(buffer));

wstring path = wstring(buffer);

change\_folder(path);

return 0;

}

}

default:

return CallWindowProc(editWndProc,hwnd, uMsg, wParam, lParam); //вызывает предыдущий обработчик для корректной отрисовки

}

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

}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

switch (uMsg)

{

case WM\_DESTROY: {

PostQuitMessage(0);

return 0;

}

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hwnd, &ps);

HBRUSH brush = CreateSolidBrush(

RGB(255, 255, 255)

);

FillRect(hdc, &ps.rcPaint, brush);

EndPaint(hwnd, &ps);

return 0;

}

case WM\_CREATE:

{

mainWindow = hwnd;

HWND hEdit = CreateWindowEx(0,

L"EDIT",

NULL, WS\_CHILD | WS\_VISIBLE | WS\_BORDER | ES\_LEFT,

10, 10, 740, 20,

hwnd,

NULL,

(HINSTANCE)GetWindowLongPtr(hwnd, GWLP\_HINSTANCE),

NULL);

editWndProc = (WNDPROC) SetWindowLongPtr(hEdit, GWLP\_WNDPROC, (LONG\_PTR)editProc);

SetWindowText(hEdit, L"D:\\");

pathEdit = hEdit;

selectButton = CreateWindow(L"Button", L"Выделить", WS\_VISIBLE | WS\_CHILD | BS\_LEFT | BS\_CHECKBOX | WS\_BORDER, 650, 40, 100, 20, mainWindow, (HMENU)SELECT\_ID, (HINSTANCE)GetWindowLongPtr(mainWindow, GWLP\_HINSTANCE), NULL);

memset(bigbuffer, 'A', sizeof(2000000000)); // буфер для мусора в файлах

change\_folder(L"D:\\");

return 0;

}

case WM\_COMMAND: {

if ((HWND)lParam == selectButton) {

if(IsDlgButtonChecked(mainWindow, SELECT\_ID)){

SendMessage(selectButton, BM\_SETCHECK, BST\_UNCHECKED, 0);

selectActive = false;

}

else {

SendMessage(selectButton, BM\_SETCHECK, BST\_CHECKED, 0);

selectActive = true;

}

return 0;

}

if (wParam == ID\_COPY) { //контекстное меню:копирование

cutOperation = false;

selectedGroup.clear();

EnumChildWindows(mainWindow, [](HWND hWndChild, LPARAM lParam) -> BOOL { //запись в буфер

wchar\_t className[256];

GetClassName(hWndChild, className, 256);

if (hWndChild == selectButton) { return TRUE; }

if (wcscmp(className, L"Button") == 0) {

LRESULT checked = SendMessage(hWndChild, BM\_GETCHECK, 0, 0);

if (checked == BST\_CHECKED) {

wchar\_t buffer[257]; // буфер для текста окна

GetWindowText(hWndChild, buffer, sizeof(buffer));

selectedGroup.push\_back(currentPath+L"\\"+buffer);

}

}

return TRUE;

}, NULL);

if(selectedGroup.size() == 0) { //текущий выбранный файл в буфер, если нет выделенных

selectedGroup.push\_back(filePath);

}

}

if (wParam == ID\_DELETE) { //контекстное меню:удаление

cntSelected = 0;

EnumChildWindows(mainWindow, [](HWND hWndChild, LPARAM lParam) -> BOOL {

wchar\_t className[256];

GetClassName(hWndChild, className, 256);

if (hWndChild == selectButton) { return TRUE; }

if (wcscmp(className, L"Button") == 0) {

LRESULT checked = SendMessage(hWndChild, BM\_GETCHECK, 0, 0);

if (checked == BST\_CHECKED) {

cntSelected++;

wchar\_t buffer[256];

GetWindowText(hWndChild, buffer, sizeof(buffer));

wstring filePath = currentPath + L"\\" + buffer;

if (!IsDirectory(filePath)) {

DeleteFile(filePath.c\_str());

}

else { //синтаксис удаления непустых папок

wchar\_t\* pathWithNull = new wchar\_t[filePath.length() + 2]; //строка пути должна иметь 2 нультерминирующих символа

wcscpy\_s(pathWithNull, filePath.length() + 1, filePath.c\_str());

pathWithNull[filePath.length()] = L'\0';

pathWithNull[filePath.length() + 1] = L'\0';

SHFILEOPSTRUCT file\_op = { NULL, FO\_DELETE, pathWithNull , L"", FOF\_NOCONFIRMATION | FOF\_NOERRORUI | FOF\_SILENT, false, 0, L"" }; //удаление без предупреждений

SHFileOperation(&file\_op);

}

}

}

return TRUE;

}, NULL);

if (cntSelected == 0) {

if (!IsDirectory(filePath)) {

DeleteFile(filePath.c\_str());

}

else { //синтаксис удаления непустых папок

wchar\_t\* pathWithNull = new wchar\_t[filePath.length() + 2]; //строка пути должна иметь 2 нультерминирующих символа

wcscpy\_s(pathWithNull, filePath.length() + 1, filePath.c\_str());

pathWithNull[filePath.length()] = L'\0';

pathWithNull[filePath.length() + 1] = L'\0';

SHFILEOPSTRUCT file\_op = { NULL, FO\_DELETE, pathWithNull , L"", FOF\_NOCONFIRMATION | FOF\_NOERRORUI | FOF\_SILENT, false, 0, L"" }; //удаление без предупреждений

SHFileOperation(&file\_op);

}

}

change\_folder(currentPath);

}

if (wParam == ID\_CUTOUT) { //контекстное меню:вырезка

cutOperation = true;

selectedGroup.clear();

EnumChildWindows(mainWindow, [](HWND hWndChild, LPARAM lParam) -> BOOL { //запись в буфер

wchar\_t className[256];

GetClassName(hWndChild, className, 256);

if (hWndChild == selectButton) { return TRUE; }

if (wcscmp(className, L"Button") == 0) {

LRESULT checked = SendMessage(hWndChild, BM\_GETCHECK, 0, 0);

if (checked == BST\_CHECKED) {

wchar\_t buffer[257]; // буфер для текста окна

GetWindowText(hWndChild, buffer, sizeof(buffer));

selectedGroup.push\_back(currentPath + L"\\" + buffer);

}

}

return TRUE;

}, NULL);

if (selectedGroup.size() == 0) { //текущий выбранный файл в буфер, если нет выделенных

selectedGroup.push\_back(filePath);

}

}

if (wParam == ID\_PASTE) { //контекстное меню:вставка

for (auto filePath : selectedGroup) {

size\_t pos = filePath.find\_last\_of(L"\\");

auto filename = filePath.substr(pos + 1);

wstring nextPath = currentPath + L"\\" + filename;

if (!IsDirectory(filePath)) {

int num = 2;

if (GetFileAttributes(nextPath.c\_str()) != INVALID\_FILE\_ATTRIBUTES) {

nextPath = currentPath + L"\\" + to\_wstring(1) + filename;

}

while (GetFileAttributes(nextPath.c\_str()) != INVALID\_FILE\_ATTRIBUTES) {

nextPath = currentPath + L"\\" + to\_wstring(num) + filename;

num++;

}

CopyFile(filePath.c\_str(), nextPath.c\_str(), TRUE); //true на перезапись файлов при копировании, не имеет значения

}

else { //синтаксис копирования непустых папок

int num = 2;

if (GetFileAttributes(nextPath.c\_str()) != INVALID\_FILE\_ATTRIBUTES) {

nextPath = currentPath + L"\\" + to\_wstring(1) + filename;

}

while (GetFileAttributes(nextPath.c\_str()) != INVALID\_FILE\_ATTRIBUTES) {

nextPath = currentPath + L"\\" + to\_wstring(num) + filename;

num++;

}

wchar\_t\* fromPathWithNull = new wchar\_t[filePath.length() + 2]; //строка пути должна иметь 2 нультерминирующих символа

wcscpy\_s(fromPathWithNull, filePath.length() + 1, filePath.c\_str());

fromPathWithNull[filePath.length()] = L'\0';

fromPathWithNull[filePath.length() + 1] = L'\0';

wchar\_t\* toPathWithNull = new wchar\_t[nextPath.length() + 2]; //строка пути должна иметь 2 нультерминирующих символа

wcscpy\_s(toPathWithNull, nextPath.length() + 1, nextPath.c\_str());

toPathWithNull[nextPath.length()] = L'\0';

toPathWithNull[nextPath.length() + 1] = L'\0';

SHFILEOPSTRUCT file\_op = { NULL, FO\_COPY, fromPathWithNull , toPathWithNull, FOF\_NOCONFIRMATION | FOF\_NOCONFIRMMKDIR | FOF\_SILENT, false, 0, L"" };

SHFileOperation(&file\_op);

}

if (cutOperation) { //удаление

if (!IsDirectory(filePath)) {

DeleteFile(filePath.c\_str());

}

else { //синтаксис удаления непустых папок

wchar\_t\* pathWithNull = new wchar\_t[filePath.length() + 2]; //строка пути должна иметь 2 нультерминирующих символа

wcscpy\_s(pathWithNull, filePath.length() + 1, filePath.c\_str());

pathWithNull[filePath.length()] = L'\0';

pathWithNull[filePath.length() + 1] = L'\0';

SHFILEOPSTRUCT file\_op = { NULL, FO\_DELETE, pathWithNull , L"", FOF\_NOCONFIRMATION | FOF\_NOERRORUI | FOF\_SILENT, false, 0, L"" }; //удаление без предупреждений

SHFileOperation(&file\_op);

}

}

}

if (cutOperation) {

selectedGroup.clear();

}

change\_folder(currentPath);

}

if (wParam == ID\_FILL) { //контекстное меню: асинхронное заполнение мусором

auto currentFile = CreateFile(filePath.c\_str(), GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_FLAG\_OVERLAPPED, NULL); //FILE\_FLAG\_OVERLAPPED отвечает за асинхронность (перекрывающий ввод)

OVERLAPPED overlapped = { 0 };

overlapped.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

WriteFileEx(currentFile, bigbuffer, 2000000000, &overlapped,NULL);

CloseHandle(overlapped.hEvent);

CloseHandle(currentFile);

}

if (HIWORD(wParam) == BN\_CLICKED) //нажатие на кнопку

{

if (!selectActive) { //переход по директориям

wchar\_t buffer[2048];

wchar\_t folder[512];

GetWindowTextW((HWND)lParam, folder, sizeof(folder));

GetWindowTextW((HWND)pathEdit, buffer, sizeof(buffer));

wstring buf\_str = wstring(buffer);

wstring fold\_str = wstring(folder);

wstring result;

if (buf\_str.back() == '\\') {

buf\_str.pop\_back();

}

if (fold\_str == L".") {

return 0;

}

else if (fold\_str == L"..") {

size\_t pos = buf\_str.find\_last\_of('\\');

result = buf\_str.substr(0, pos);

}

else {

result = buf\_str + L"\\" + fold\_str;

}

if (!IsDirectory(result)) {

return 0;

}

SetWindowText((HWND)pathEdit, (LPWSTR)result.c\_str());

change\_folder(result);

}

else {//выделение

wchar\_t buffer[256];

GetWindowText((HWND)lParam, buffer, sizeof(buffer));

if (wcscmp(buffer, L"..") == 0)

{

return 0;

}

LRESULT checked = SendMessage((HWND)lParam, BM\_GETCHECK, 0, 0);

if (checked == BST\_CHECKED) {

SendMessage((HWND)lParam, BM\_SETCHECK, BST\_UNCHECKED, 0);

}

else {

SendMessage((HWND)lParam, BM\_SETCHECK, BST\_CHECKED, 0);

}

}

}

return 0;

}

case WM\_CONTEXTMENU:

{

POINT pt;

pt.x = LOWORD(lParam);

pt.y = HIWORD(lParam);

HWND menuTarget = WindowFromPoint(pt);

if (menuTarget != mainWindow) {

wchar\_t filename[256];

GetWindowText(menuTarget, filename, 256);

filePath = currentPath + L"\\" + filename;

}

wchar\_t buffer1[256];

wchar\_t buffer2[256];

GetClassName(menuTarget, buffer1, sizeof(buffer1));

GetWindowText(menuTarget, buffer2, sizeof(buffer2));

if (wcscmp(buffer1, L"Button") == 0 && wcscmp(buffer2, L"..") == 0)

{

return 0;

}

HMENU hMenu = CreatePopupMenu();

if (menuTarget == mainWindow) {

if(selectedGroup.size() !=0)InsertMenu(hMenu, 0, MF\_BYPOSITION | MF\_STRING, ID\_PASTE, L"Вставить");

else InsertMenu(hMenu, 0, MF\_BYPOSITION | MF\_STRING | MF\_GRAYED, ID\_PASTE, L"Вставить");

}

else {

InsertMenu(hMenu, 0, MF\_BYPOSITION | MF\_STRING, ID\_COPY, L"Копировать");

InsertMenu(hMenu, 1, MF\_BYPOSITION | MF\_STRING, ID\_CUTOUT, L"Вырезать");

InsertMenu(hMenu, 2, MF\_BYPOSITION | MF\_STRING, ID\_DELETE, L"Удалить");

if(!IsDirectory(filePath))InsertMenu(hMenu, 3, MF\_BYPOSITION | MF\_STRING, ID\_FILL, L"Заполнить мусором");

}

TrackPopupMenu(hMenu, TPM\_LEFTALIGN | TPM\_TOPALIGN, LOWORD(lParam), HIWORD(lParam), 0, hwnd, NULL);

DestroyMenu(hMenu);

return 0;

}

}

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

}