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 №1

на тему

ОСНОВЫ ПРОГРАММИРОВАНИЯ В WIN 32 API. ОКОННОЕ ПРИЛОЖЕНИЕ WIN 32 С МИНИМАЛЬНОЙ ДОСТАТОЧНОЙ ФУНКЦИОНАЛЬНОСТЬЮ. ОБРАБОТКА ОСНОВНЫХ ОКОННЫХ СООБЩЕНИЙ.

Выполнил студент гр.153502 Толстой Д.В.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

[1 Формулировка задачи 3](#_Toc146294202)

[2 Теоретические сведения 4](#_Toc146294203)

[3 Описание функций программы 6](#_Toc146294204)

[3.1 Открытие файла 6](#_Toc146294205)

[3.2 Редактирование файла 6](#_Toc146294206)

[3.3 Сохранение текстового файла 6](#_Toc146294207)

[Список использованных источников 9](#_Toc146294208)

[Приложение А 10](#_Toc146294209)

1 ФОРМУЛИРОВКА ЗАДАЧИ

Целью выполнения лабораторной работы является создание оконного приложения на Win32 API, обладающее минимальным функционалом, позволяющим отработать базовые навыки написания программы на Win32 API, таких как обработка оконных сообщений.

В качестве задачи необходимо Создать текстовый редактор с поддержкой множества открытых документов. Реализовать функциональность сохранения и загрузки файлов.

# **2** ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Win32 API (также называемый Windows API) — это исходная платформа для собственных Windows-приложений на языке C/C++, которым требуется прямой доступ к Windows и оборудованию. Он обеспечивает первоклассный опыт разработки без зависимости от управляемой среды выполнения, такой как .NET и WinRT (для приложений UWP для Windows 10). Благодаря этому API Win32 стает оптимальной платформой для приложений, которым требуется самый высокий уровень производительности и прямой доступ к системному оборудованию.

API Win32 существуют для многих функций и технологий в Windows 10, включая основные интерфейсы пользовательского интерфейса и API окон, аудио и графики, а также сети.

Windows API спроектирован для использования в языке Си для написания прикладных программ, предназначенных для работы под управлением операционной системы MS Windows. Работа через Windows API — это наиболее близкий к операционной системе способ взаимодействия с ней из прикладных программ.

Для создания текстового редактора с использованием Windows API (WinAPI) нам потребуются следующие теоретические сведения.

Окна (Windows): WinAPI базируется на концепции окон. Каждое окно имеет уникальный идентификатор (HWND). Для создания основного окна вашего текстового редактора используйте функцию CreateWindowEx.

Обработка сообщений (Message Handling): Основной цикл обработки сообщений выполняется функцией GetMessage, TranslateMessage.

Меню и команды. Добавление меню в наш текстовый редактор с помощью функций CreateMenu, AppendMenu и SetMenu. Обработка команд меню выполняется через WM\_COMMAND.

Работа с файлами. Для открытия, сохранения и редактирования файлов используются функции, такие как CreateFile, ReadFile, WriteFile и CloseHandle.

Текстовое поле (Edit Control). Для создания области ввода текста используется элемент управления "Edit" с помощью функции CreateWindowEx с классом EDIT.

Управление полосами прокрутки (Scroll Bars). Для обеспечения прокрутки текста в редакторе, можно использовать элементы управления "Scroll Bar" с классом SCROLLBAR.

Работа с шрифтами и цветами. Для настройки шрифтов и цветов текста и фона используются функции CreateFont и SetTextColor.

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы:

– Открытие текстовых файлов;

– Сохранение текстовых файлов;

– Вызов диалогового окна при выходе без сохранения файла;

– Отслеживание изменения файла;

– Копирование текста в буфер обмена;

– Вырезание текста в буфер обмена;

– Вставка текста из буфера обмена в редактор;

– Изменение названия окна с учетом открытых файлов.

3.1 Открытие файла

Для открытия файла, выберите в меню Open, далее выберите файл с расширением .txt (Рисунок 1). Если не возникнет ошибка, о которой будет доложено, файл откроется.

3.2 Редактирование файла

В режиме редактирования текста вы можете изменять текст без последствий. В названии файла появится значок “\*”, который оповещает пользователя о том, что файл изменен. Если сохранить файл, он исчезнет.

3.3 Сохранение текстового файла

Когда вы закончите с файлом, вы можете его сохранить указав путь и его название (Рисунок 2). Если же файл был открыт или сохранен под существующем именем, вы можете сохранить быстро с помощью функции Save. Файл сохранится под текущим именем по тому же пути. Стоит заметить, что при открытии файла и последующим сохранении текст будет сохраняться в открытый ранее документ, но вы также можете выбрать куда созранить именно с помощью функции “Save as”.

3.4 Работа с буфером обмена

Чтобы работать с буфером обмена, вы можете выбрать данные опции из меню “File”. Там доступны три опции: копирование текста из редактора, вырезка текста из редактора, и вставка текста в редактор из буфера обмена (Рисунок 3).

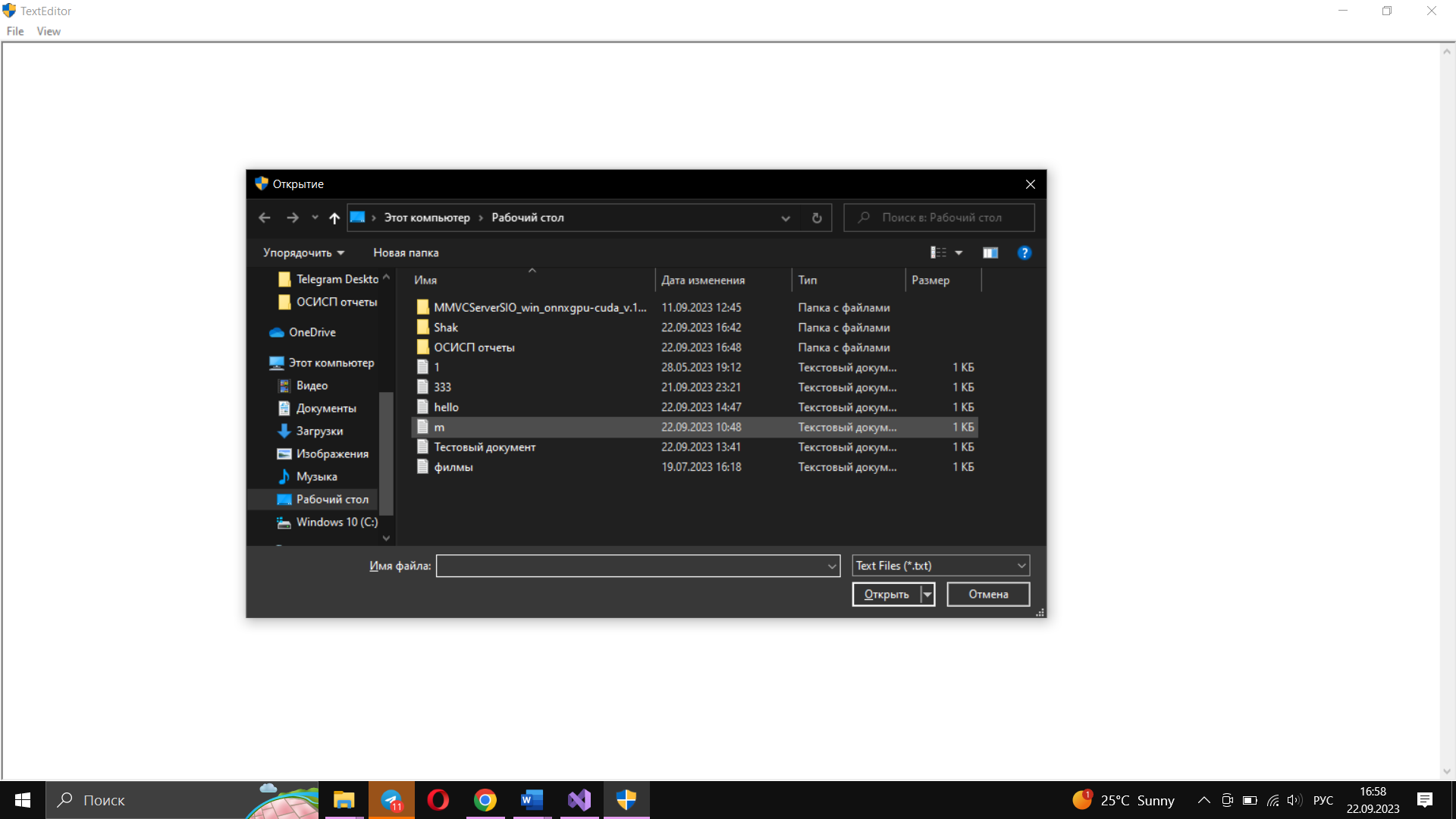


Рисунок 1 – Открытие файла

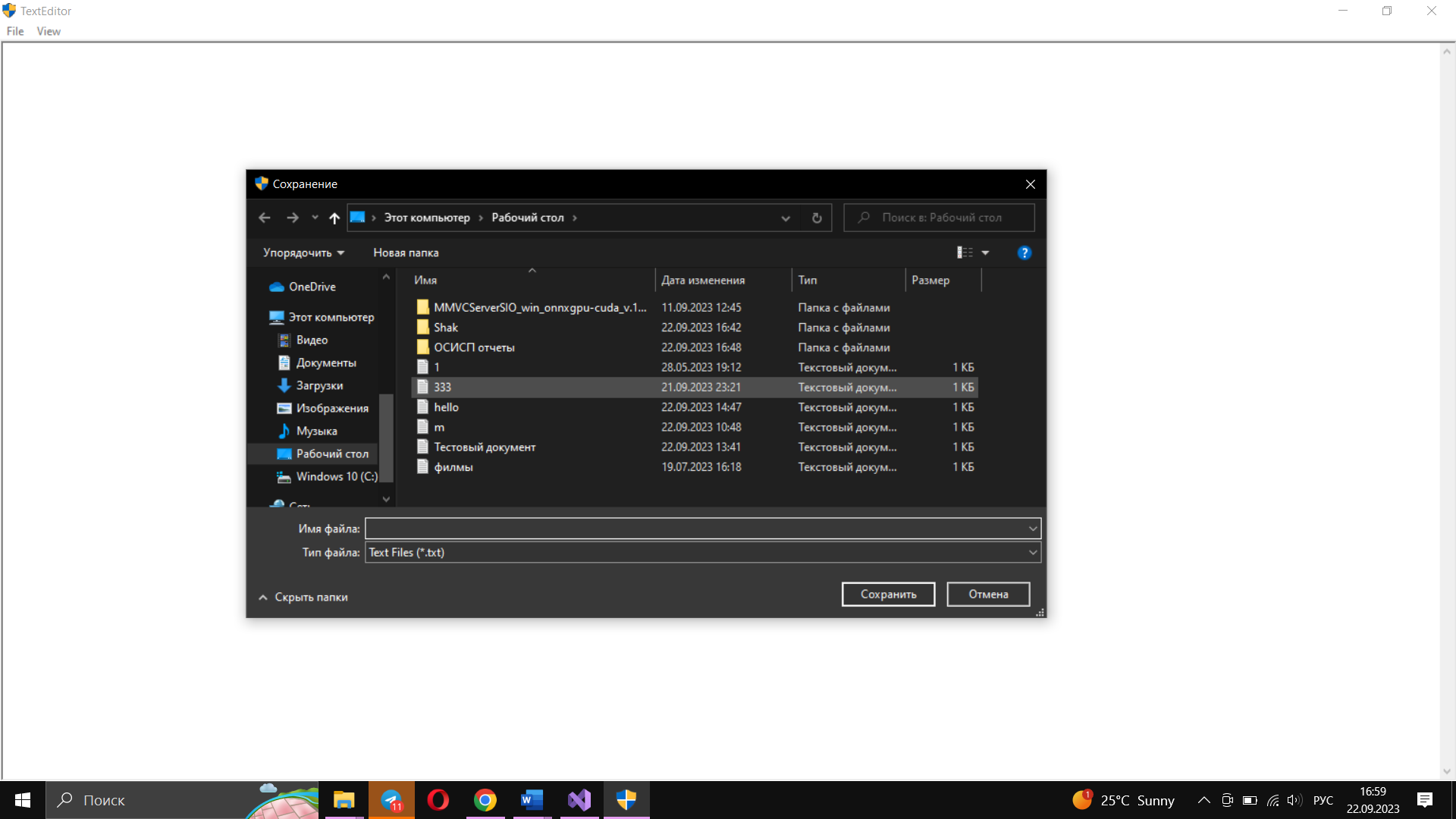


Рисунок 2 – Сохранение файла

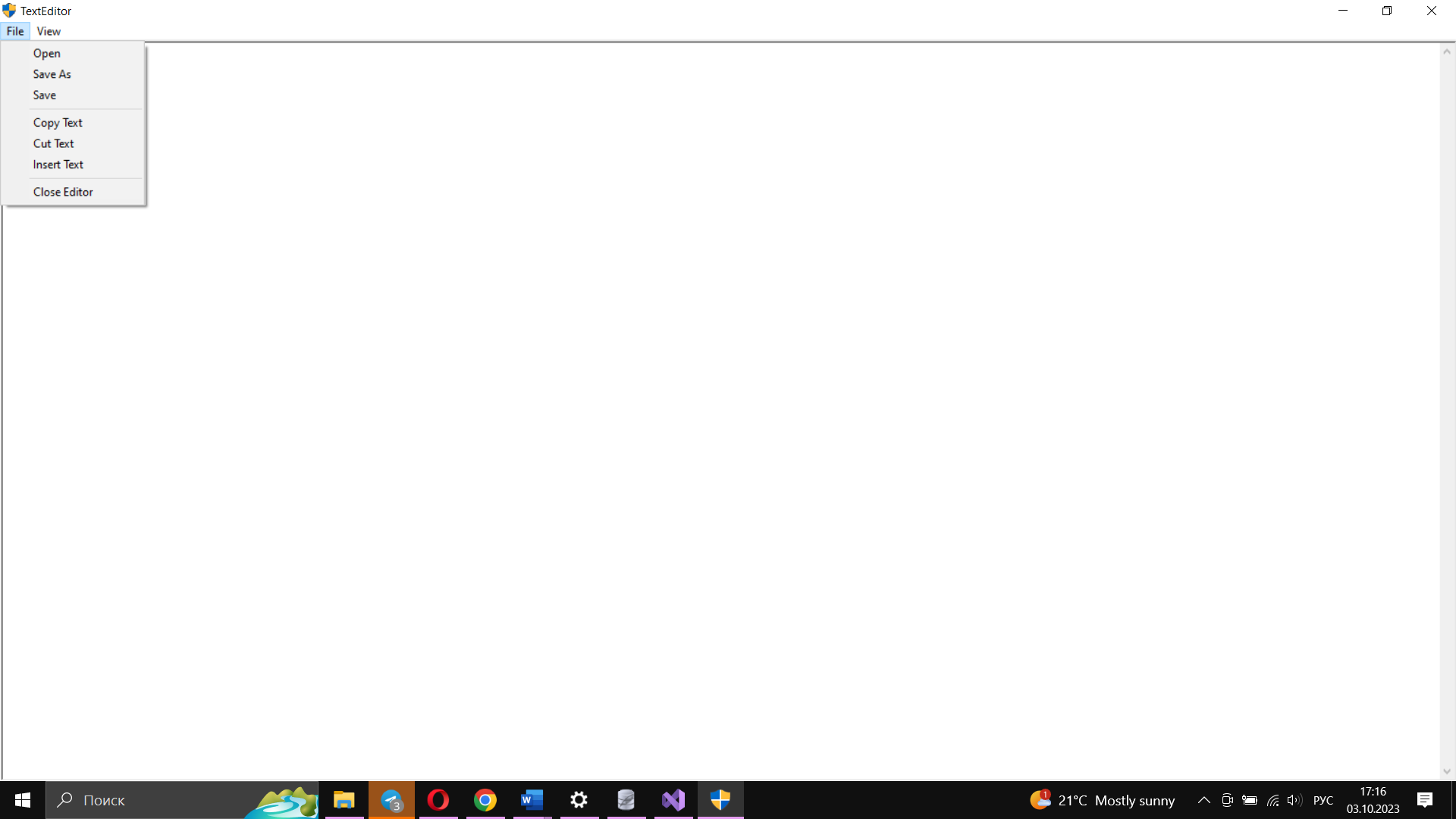


Рисунок 3 – Опции работы с буфером обмена

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Build desktop Windows apps using the Win32 API [Электронный ресурс]. – Электронные данные. – Режим доступа: https://learn.microsoft.com/en-us/windows/win32

[2] C++ WINDOWS API – Электронные данные. – Режим доступа: https://www.youtube.com/@egordeveloper3794

ПРИЛОЖЕНИЕ А

Листинг кода

Файл SowtwareDefinitions.h

#pragma once

#include <string>

#include <windows.h>

#define OnMenuClicked 1

#define OnOpenFileClicked 2

#define OnSaveFileClicked 3

#define OnSaveAsFileClicked 4

#define OnCloseEditorClicked 5

#define IDC\_EDIT 6

#define HOTKEY\_CTRL\_S 7

#define HOTKEY\_CTRL\_O 8

HWND hEditControl;

std::wstring currFilePath;

bool SAVED = true;

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

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

void MainWindAddMenus(HWND hwnd);

void MainWindAddWidgets(HWND hwnd);

std::wstring GetFilenameFromPath(std::wstring path);

void SaveTextToFile(const HWND& hwnd);

bool AskForExit();

void DeleteStarFromName(const HWND& hwnd);

Файл TextEditor.cpp

#ifndef UNICODE

#define UNICODE

#endif

#include <windows.h>

#include <wchar.h>

#include <fstream>

#include "SoftwareDefinitions.h"

#include <string>

#include <atlstr.h>

#include <vector>

#include <iostream>

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

{

// Register the window class.

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

WNDCLASS wc = { };

wc.hIcon = LoadIcon(NULL, IDI\_SHIELD);

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

// Create the window

HWND hwnd = CreateWindowExW(0, CLASS\_NAME, L"TextEditor", WS\_OVERLAPPEDWINDOW, CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT, NULL, NULL, hInstance, NULL);

if (hwnd == NULL)

{

return 0;

}

ShowWindow(hwnd, nCmdShow);

// Run the message loop.

MSG msg = { };

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

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

{

wchar\_t msg[32];

switch (uMsg)

{

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

if (wmId == IDC\_EDIT)

{

if (HIWORD(wParam) == EN\_CHANGE)

{

if (SAVED)

{

SAVED = false;

wchar\_t oldName[100];

GetWindowTextW(hwnd, oldName, 100);

std::wstring newName(oldName);

SetWindowTextW(hwnd, newName.append(L"\*").c\_str());

}

}

}

switch (wParam)

{

case OnOpenFileClicked:

{

wchar\_t buff[100];

OPENFILENAME ofn = { 0 };

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hwnd;

ofn.lpstrFile = buff;

ofn.nMaxFile = sizeof(buff);

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

ofn.nFilterIndex = 1;

ofn.lpstrDefExt = L"txt";

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

if (GetOpenFileName(&ofn))

{

wchar\_t textBuffer[1024];

currFilePath = buff;

auto name = GetFilenameFromPath(currFilePath);

auto newNameOfWnd = std::wstring(L"Text Editor - ").append(name);

std::wifstream outFile((LPWSTR)currFilePath.c\_str());

if (outFile.is\_open()) {

outFile >> textBuffer;

outFile.close();

SetWindowTextW(hEditControl, textBuffer);

SetWindowTextW(hwnd, newNameOfWnd.c\_str());

SAVED = true;

}

else {

MessageBox(hwnd, L"Failed to open file!", L"Error", MB\_OK | MB\_ICONERROR);

}

}

break;

}

case OnSaveAsFileClicked:

{

SaveTextToFile(hwnd);

SAVED = true;

DeleteStarFromName(hwnd);

break;

}

case OnSaveFileClicked:

{

if (currFilePath.empty())

{

SaveTextToFile(hwnd);

}

else

{

wchar\_t textBuffer[1024];

GetWindowTextW(hEditControl, textBuffer, sizeof(textBuffer) / sizeof(textBuffer[0]));

std::wofstream outFile(currFilePath);

if (outFile.is\_open()) {

outFile << textBuffer;

outFile.close();

MessageBox(hwnd, L"File saved successfully!", L"Success", MB\_OK | MB\_ICONINFORMATION);

SAVED = true;

DeleteStarFromName(hwnd);

}

else {

MessageBox(hwnd, L"Failed to save file!", L"Error", MB\_OK | MB\_ICONERROR);

}

}

break;

}

case OnCloseEditorClicked:

{

if (AskForExit())

{

return 0;

}

break;

}

}

break;

}

case WM\_SIZE:

{

int newWidth = LOWORD(lParam);

int newHeight = HIWORD(lParam);

// Установите новый размер окна

SetWindowPos(hEditControl, NULL, 1, 0, newWidth, newHeight, SWP\_NOMOVE | SWP\_NOZORDER);

break;

}

case WM\_CREATE:

{

MainWindAddMenus(hwnd);

MainWindAddWidgets(hwnd);

break;

}

case WM\_CLOSE:

{

if (AskForExit())

{

return 0;

}

break;

}

case WM\_HOTKEY:

{

int id = wParam;

// Проверяем, была ли нажата клавиша Ctrl

switch (id)

{

// Проверяем, была ли нажата клавиша "S"

case HOTKEY\_CTRL\_S:

{

// Обработка нажатия Ctrl + S

if (currFilePath.empty())

{

SaveTextToFile(hwnd);

}

else

{

wchar\_t textBuffer[1024];

GetWindowTextW(hEditControl, textBuffer, sizeof(textBuffer) / sizeof(textBuffer[0]));

std::wofstream outFile(currFilePath);

if (outFile.is\_open()) {

outFile << textBuffer;

outFile.close();

MessageBox(hwnd, L"File saved successfully!", L"Success", MB\_OK | MB\_ICONINFORMATION);

SAVED = true;

DeleteStarFromName(hwnd);

}

else {

MessageBox(hwnd, L"Failed to save file!", L"Error", MB\_OK | MB\_ICONERROR);

}

}

break;

}

case HOTKEY\_CTRL\_O:

{

// Обработка нажатия Ctrl + O

wchar\_t buff[100];

OPENFILENAME ofn = { 0 };

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hwnd;

ofn.lpstrFile = buff;

ofn.nMaxFile = sizeof(buff);

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

ofn.nFilterIndex = 1;

ofn.lpstrDefExt = L"txt";

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

if (GetOpenFileName(&ofn))

{

wchar\_t textBuffer[1024];

currFilePath = buff;

auto name = GetFilenameFromPath(currFilePath);

auto newNameOfWnd = std::wstring(L"Text Editor - ").append(name);

std::wifstream outFile((LPWSTR)currFilePath.c\_str());

if (outFile.is\_open()) {

outFile >> textBuffer;

outFile.close();

SetWindowTextW(hEditControl, textBuffer);

SetWindowTextW(hwnd, newNameOfWnd.c\_str());

SAVED = true;

}

else {

MessageBox(hwnd, L"Failed to open file!", L"Error", MB\_OK | MB\_ICONERROR);

}

}

break;

}

}

break;

}

default:

{

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

}

}

}

void MainWindAddMenus(HWND hwnd)

{

HMENU RootMenu = CreateMenu();

HMENU SubMenu = CreateMenu();

AppendMenuW(SubMenu, MF\_STRING, OnOpenFileClicked, L"Open");

AppendMenuW(SubMenu, MF\_STRING, OnSaveAsFileClicked, L"Save As");

AppendMenuW(SubMenu, MF\_STRING, OnSaveFileClicked, L"Save");

AppendMenuW(SubMenu, MF\_SEPARATOR, NULL, NULL);

AppendMenuW(SubMenu, MF\_STRING, OnCloseEditorClicked, L"Close Editor");

AppendMenu(RootMenu, MF\_POPUP, (UINT\_PTR)SubMenu, L"File");

RegisterHotKey(hwnd, HOTKEY\_CTRL\_S, MOD\_CONTROL, 'S');

RegisterHotKey(hwnd, HOTKEY\_CTRL\_O, MOD\_CONTROL, 'O');

SetMenu(hwnd, RootMenu);

}

void MainWindAddWidgets(HWND hwnd)

{

// Получите размер и позицию окна в экранных координатах

RECT windowRect;

GetWindowRect(hwnd, &windowRect);

int windowWidth = windowRect.right - windowRect.left;

int windowHeight = windowRect.bottom - windowRect.top;

// windowWidth и windowHeight содержат ширину и высоту окна

// windowRect.left и windowRect.top содержат координаты верхнего левого угла

hEditControl = CreateWindowEx(WS\_EX\_CLIENTEDGE, L"Edit", L"", WS\_CHILD | WS\_VISIBLE | ES\_LEFT |

WS\_VSCROLL | ES\_MULTILINE | ES\_AUTOVSCROLL,

1, 0, windowWidth - 20, windowHeight - 20, hwnd, (HMENU)IDC\_EDIT, NULL, NULL);

}

std::wstring GetFilenameFromPath(std::wstring path)

{

// Найти последний символ разделителя пути

size\_t found = path.find\_last\_of(L"\\/");

if (found != std::wstring::npos) {

// Извлечь имя файла после последнего разделителя пути

std::wstring fileName = path.substr(found + 1);

// вернуть имя файла

return fileName;

}

else {

// Если разделитель не найден, это может быть имя файла без пути

// Вывести всю строку как имя файла

return path;

}

}

void SaveTextToFile(const HWND& hwnd)

{

currFilePath.clear();

wchar\_t buff[100];

OPENFILENAME ofn = { 0 };

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hwnd;

ofn.lpstrFile = buff;

ofn.nMaxFile = sizeof(buff);

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

ofn.nFilterIndex = 1;

ofn.lpstrDefExt = L"txt";

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

if (GetSaveFileName(&ofn))

{

wchar\_t textBuffer[1024];

currFilePath = buff;

auto name = GetFilenameFromPath(currFilePath);

auto newNameOfWnd = std::wstring(L"Text Editor - ").append(name);

GetWindowTextW(hEditControl, textBuffer, sizeof(textBuffer) / sizeof(textBuffer[0]));

std::wofstream outFile(buff);

if (outFile.is\_open()) {

outFile << textBuffer;

outFile.close();

MessageBox(hwnd, L"File saved successfully!", L"Success", MB\_OK | MB\_ICONINFORMATION);

SetWindowTextW(hwnd, newNameOfWnd.c\_str());

SAVED = true;

DeleteStarFromName(hwnd);

}

else {

MessageBox(hwnd, L"Failed to save file!", L"Error", MB\_OK | MB\_ICONERROR);

}

}

}

bool AskForExit()

{

if (!SAVED)

{

int msgboxID = MessageBox(

NULL,

L"Do you really want to exit? You have unsaved changes",

L"Exit",

MB\_ICONEXCLAMATION | MB\_YESNO);

if (msgboxID == IDYES)

{

PostQuitMessage(0);

return true;

}

return false;

}

else

{

PostQuitMessage(0);

return true;

}

}

void DeleteStarFromName(const HWND& hwnd)

{

wchar\_t oldName[100];

GetWindowTextW(hwnd, oldName, 100);

std::wstring newName(oldName);

// Поиск и удаление символа "\*"

size\_t pos = newName.find(L"\*");

while (pos != std::wstring::npos) {

newName.erase(pos, 1); // Удаление 1 символа, начиная с позиции pos

pos = newName.find(L"\*", pos); // Поиск следующего символа "\*"

}

SetWindowTextW(hwnd, newName.c\_str());

}