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.153504 Сацюк С.В.

Проверил ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

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 построить приложение для чтения и редактирования текстовых документов с возможностью выделения и копирования текста в буфер обмена.

2 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

Согласно формулировке задачи, были спроектированы следующие функции программы:

– Открытие текстового файла;
– Создание нового текстового файла;
– Редактирование текстового файла;
– Копирование выделенного текста в буфер обмена;
– Сохранение файла.

1. Открытие файла

Для открытия текстового файла необходимо в меню нажать File-Open и в открывшемся диалоговом окне выбрать нужный файл.

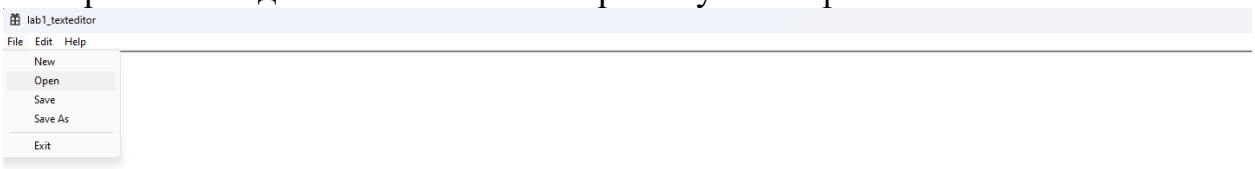

Рисунок 1 – Открытие файла

2. Редактирование текста и копирование выделенного текста в буфер обмена

После открытия файла в окне редактирования появляется текст из него, который можно отредактировать. Для копирования выделенного текста в буфер обмена можно воспользоваться меню: File-Copy.

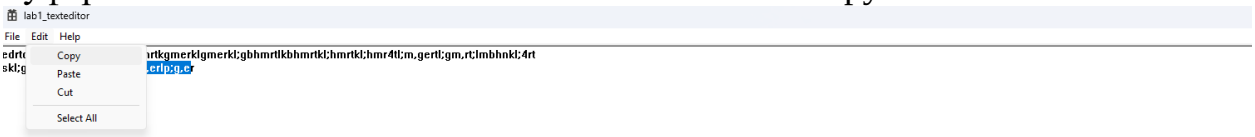

Рисунок 2 – Копирование выделенного текста в буфер обмена

Рисунок 3 – Результат вставки текста после копирования

3. Сохранение отредактированного текста в файл

Для сохранения отредактированного текста в файл необходимо воспользоваться меню: File-Save. Сохранение произойдет в файл, который был открыт вначале.

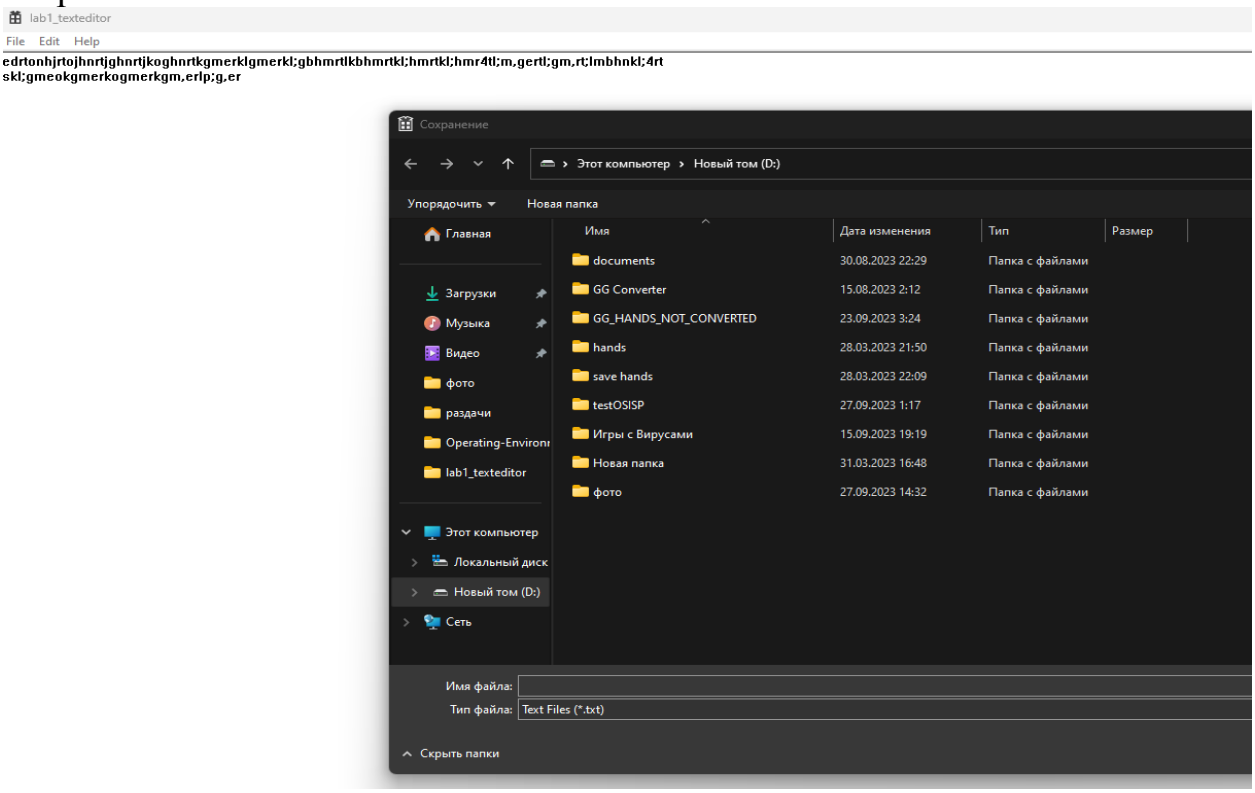

Рисунок 4 – Сохранение файла

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] RegisterHotKey function (winuser.h) - Win32 apps – Электронные данные. – Режим доступа: https://learn.microsoft.com/en us/windows/win32/api/winuser/nf-winuser-registerhotkey/

[3] CreateWindowExA function (winuser.h) - Win32 apps – Электронные данные. – Режим доступа: https://learn.microsoft.com/en us/windows/win32/api/winuser/nf-winuser-createwindowexa

ПРИЛОЖЕНИЕ А

Листинг кода

Файл lab1_texteditor.cpp

```
#include "framework.h"
#include "lab1_texteditor.h"
#include "resource.h"
#include <windows.h>
#include <commdlg.h>

#define MAX_LOADSTRING 100

HINSTANCE hInst;                                // текущий экземпляр
WCHAR szTitle[MAX_LOADSTRING];                  // Текст строки заголовка
WCHAR szWindowClass[MAX_LOADSTRING];            // имя класса главного окна

ATOM MyRegisterClass(HINSTANCE hInstance);
BOOL InitInstance(HINSTANCE, int);
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
void CreateMainMenu(HWND hWnd);
HWND hEdit;


int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
    _In_opt_ HINSTANCE hPrevInstance,
    _In_ LPWSTR    lpCmdLine,
    _In_ int       nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_LAB1TEXTEDITOR, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    if (!InitInstance(hInstance, nCmdShow))
    {
        return FALSE;
    }

    MSG msg;

    while (GetMessage(&msg, nullptr, 0, 0))
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg); // Отправка сообщения на обработку функции окна
(WndProc)
    }

    return (int)msg.wParam;
}

ATOM MyRegisterClass(HINSTANCE hInstance) //регистрация класса окна с системой
{
    WNDCLASSEXW wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style = CS_HREDRAW | CS_VREDRAW; //обеспечивает перерисовку содержимого
окна при изменении его размера
    wcex.lpfnWndProc = WndProc; //обработчик
    wcex.cbClsExtra = 0; //выделения дополнительной памяти для класса = 0
    wcex.cbWndExtra = 0;
    wcex.hInstance = hInstance; //Указывает на текущий экземпляр приложения
```

```
    wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_LAB1TEXTEDITOR)); //иконка
    wcex.hCursor = LoadCursor(nullptr, IDC_ARROW);
    wcex.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wcex.lpszMenuName = nullptr;
    wcex.lpszClassName = szWindowClass;
    wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

    return RegisterClassExW(&wcex);
}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow) //создание главного окна
{
    hInst = hInstance; //сохранение дескриптора

    HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
        CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

    if (!hWnd)
    {
        return FALSE;
    }

    ShowWindow(hWnd, nCmdShow);
    UpdateWindow(hWnd);

    // Создание меню
    CreateMainMenu(hWnd);

    RegisterHotKey(hWnd, HOTKEY_CTRL_S, MOD_CONTROL, 'S');
    RegisterHotKey(hWnd, HOTKEY_CTRL_O, MOD_CONTROL, 'O');
    RegisterHotKey(hWnd, HOTKEY_CTRL_SHIFT_S, MOD_CONTROL | MOD_SHIFT, 'S');

    return TRUE;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
//функция обратного вызова, которая обрабатывает сообщения для вашего главного окна
{
    static HWND hEdit; // Статическая переменная для хранения дескриптора элемента
управления EDIT
    switch (message)
    {
    case WM_HOTKEY:
    {
        int id = wParam;
        switch (id)
        {
        case HOTKEY_CTRL_O:
            SendMessage(hWnd, WM_COMMAND, IDM_OPEN, 0);
            break;
        case HOTKEY_CTRL_S:
            SendMessage(hWnd, WM_COMMAND, IDM_SAVE, 0);
            break;
        case HOTKEY_CTRL_SHIFT_S:
            SendMessage(hWnd, WM_COMMAND, IDM_SAVEAS, 0);
            break;
        }
    }
    break;
    case WM_CREATE:
    {
        hEdit = CreateWindowEx(
            WS_EX_CLIENTEDGE,
            L"EDIT",
            NULL,
```

```
            WS_CHILD | WS_VISIBLE | WS_VSCROLL | WS_HSCROLL | ES_MULTILINE |
ES_AUTOVSCROLL | ES_AUTOHSCROLL,
            0, 0, 0, 0,
            hWnd,
            (HMENU)IDC_TEXT_EDIT,
            hInst,
            NULL);

        if (hEdit == NULL)
        {
            MessageBox(NULL, L"Cannot create edit control.", L"Error", MB_OK |
MB_ICONERROR);
            return -1;
        }
    }
    break;

    case WM_SIZE:
    {
        int newWidth = LOWORD(lParam);
        int newHeight = HIWORD(lParam);

        SetWindowPos(hEdit, NULL, 0, 0, newWidth, newHeight, SWP_NOZORDER);
    }
    break;
    case WM_CLOSE:
    {
        wchar_t szEditText[4096];
        GetWindowText(hEdit, szEditText, sizeof(szEditText) /
sizeof(szEditText[0]));

        if (IsWindowVisible(hEdit) && wcscmp(szEditText, L"") != 0)
        {
            int result = MessageBox(hWnd, L"Do you want to save the changes?",
L"Save Changes", MB_YESNOCANCEL | MB_ICONQUESTION);

            if (result == IDYES)
            {
                SendMessage(hWnd, WM_COMMAND, IDM_SAVE, 0);
            }
            else if (result == IDCANCEL)
            {
                return 0;
            }
        }

        UnregisterHotKey(hWnd, HOTKEY_CTRL_O);
        UnregisterHotKey(hWnd, HOTKEY_CTRL_S);
        UnregisterHotKey(hWnd, HOTKEY_CTRL_SHIFT_S);
        DestroyWindow(hWnd);
    }
    break;
    case WM_COMMAND:
    {
        int wmId = LOWORD(wParam);
        switch (wmId)
        {
        case IDM_NEW:
            SetWindowText(hEdit, L"");
            break;
        case IDM_OPEN:
        {
            OPENFILENAME ofn;
            wchar_t szFileName[MAX_PATH] = L"";
```

```
            ZeroMemory(&ofn, sizeof(ofn));
            ofn.lStructSize = sizeof(ofn);
            ofn.hwndOwner = hWnd;
            ofn.lpstrFilter = L"Text Files (*.txt)\0*.txt\0All Files (*.*)\0*.*\0";
            ofn.lpstrFile = szFileName;
            ofn.nMaxFile = sizeof(szFileName);
            ofn.Flags = OFN_FILEMUSTEXIST;

            if (GetOpenFileName(&ofn))
            {
                HANDLE hFile = CreateFile(ofn.lpstrFile, GENERIC_READ,
FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
                if (hFile != INVALID_HANDLE_VALUE)
                {
                    DWORD dwFileSize = GetFileSize(hFile, NULL);
                    if (dwFileSize != INVALID_FILE_SIZE)
                    {
                        wchar_t* szFileContent = new wchar_t[dwFileSize /
sizeof(wchar_t) + 1];
                        if (szFileContent)
                        {
                            DWORD dwBytesRead;
                            if (ReadFile(hFile, szFileContent, dwFileSize,
&dwBytesRead, NULL))
                            {
                                szFileContent[dwFileSize / sizeof(wchar_t)] = L'\0';
                                SetWindowText(hEdit, szFileContent);
                            }
                            delete[] szFileContent;
                        }
                    }
                    CloseHandle(hFile);
                }
            }
        }
        break;
        case IDM_SAVE:
        {
            OPENFILENAME ofn;
            wchar_t szFileName[MAX_PATH] = L"";

            ZeroMemory(&ofn, sizeof(ofn));
            ofn.lStructSize = sizeof(ofn);
            ofn.hwndOwner = hWnd;
            ofn.lpstrFilter = L"Text Files (*.txt)\0*.txt\0All Files (*.*)\0*.*\0";
            ofn.lpstrFile = szFileName;
            ofn.nMaxFile = sizeof(szFileName);
            ofn.Flags = OFN_OVERWRITEPROMPT;

            if (GetSaveFileName(&ofn))
            {
                HANDLE hFile = CreateFile(ofn.lpstrFile, GENERIC_WRITE, 0, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
                if (hFile != INVALID_HANDLE_VALUE)
                {
                    wchar_t szText[4096];
                    GetWindowText(hEdit, szText, sizeof(szText) /
sizeof(szText[0]));

                    DWORD dwBytesWritten;
                    WriteFile(hFile, szText, lstrlen(szText) * sizeof(wchar_t),
&dwBytesWritten, NULL);

                    CloseHandle(hFile);
                }
```

```
            }
        }
        break;
        case IDM_SAVEAS:
        {
            OPENFILENAME ofn;
            wchar_t szFileName[MAX_PATH] = L"";

            ZeroMemory(&ofn, sizeof(ofn));
            ofn.lStructSize = sizeof(ofn);
            ofn.hwndOwner = hWnd;
            ofn.lpstrFilter = L"Text Files (*.txt)\0*.txt\0All Files (*.*)\0*.*\0";
            ofn.lpstrFile = szFileName;
            ofn.nMaxFile = sizeof(szFileName);
            ofn.Flags = OFN_OVERWRITEPROMPT;

            if (GetSaveFileName(&ofn))
            {
                HANDLE hFile = CreateFile(ofn.lpstrFile, GENERIC_WRITE, 0, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
                if (hFile != INVALID_HANDLE_VALUE)
                {
                    wchar_t szText[4096];
                    GetWindowText(hEdit, szText, sizeof(szText) /
sizeof(szText[0]));

                    DWORD dwBytesWritten;
                    WriteFile(hFile, szText, lstrlen(szText) * sizeof(wchar_t),
&dwBytesWritten, NULL);

                    CloseHandle(hFile);
                }
            }
        }
        break;
        case IDM_COPY:
            SendMessage(hEdit, WM_COPY, 0, 0);
            break;
        case IDM_PASTE:
            SendMessage(hEdit, WM_PASTE, 0, 0);
            break;
        case IDM_CUT:
            SendMessage(hEdit, WM_CUT, 0, 0);
            break;
        case IDM_SELECTALL:
            SendMessage(hEdit, EM_SETSEL, 0, -1);
            break;

        default:
            return DefWindowProc(hWnd, message, wParam, lParam);
        }
    }
    break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}

void CreateMainMenu(HWND hWnd)
{
    HMENU hMenu = CreateMenu();
```

```
    // Меню "File"
    HMENU hFileMenu = CreateMenu();
    AppendMenu(hFileMenu, MF_STRING, IDM_NEW, L"New");
    AppendMenu(hFileMenu, MF_STRING, IDM_OPEN, L"Open");
    AppendMenu(hFileMenu, MF_STRING, IDM_SAVE, L"Save");
    AppendMenu(hFileMenu, MF_STRING, IDM_SAVEAS, L"Save As");
    AppendMenu(hFileMenu, MF_SEPARATOR, 0, nullptr);
    AppendMenu(hFileMenu, MF_STRING, IDM_EXIT, L"Exit");
    AppendMenu(hMenu, MF_POPUP, (UINT_PTR)hFileMenu, L"File");

    // Меню "Edit"
    HMENU hEditMenu = CreateMenu();
    AppendMenu(hEditMenu, MF_STRING, IDM_COPY, L"Copy");
    AppendMenu(hEditMenu, MF_STRING, IDM_PASTE, L"Paste");
    AppendMenu(hEditMenu, MF_STRING, IDM_CUT, L"Cut");
    AppendMenu(hEditMenu, MF_SEPARATOR, 0, nullptr);
    AppendMenu(hEditMenu, MF_STRING, IDM_SELECTALL, L"Select All");
    AppendMenu(hMenu, MF_POPUP, (UINT_PTR)hEditMenu, L"Edit");

    // Меню "Help"
    HMENU hHelpMenu = CreateMenu();
    AppendMenu(hHelpMenu, MF_STRING, IDM_ABOUT, L"About");
    AppendMenu(hMenu, MF_POPUP, (UINT_PTR)hHelpMenu, L"Help");

    SetMenu(hWnd, hMenu);
}
```