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 №1

на тему

«Основы программирования на Win 32 API. Оконное приложение Win 32 с минимальной достаточной функциональностью. Обработка основных оконных сообщений»

Выполнил:

студент гр. 153504

Сивый А.А.

Проверил:

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цели работы 3](#_gjdgxs)

[2 Краткие теоретические сведения 4](#_30j0zll)

[3 Полученные результаты 5](#_1fob9te)

[Вывод 6](#_4q3p755gzys6)

[Список использованных источников 7](#_4q3p755gzys6)

[Приложение А (обязательное) Листинг кода 8](#_4q3p755gzys6)

**1 ЦЕЛИ РАБОТЫ**

1 Изучить основы программирования в Win 32 API.

2 Создать оконное приложение Win 32 с минимальной достаточной функциональностью.

3 Реализовать обработку основных оконных сообщений.

4 Разработать текстовый редактор с поддержкой множества открытых документов, реализовать функциональность сохранения и загрузки файлов.

**2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Оконное приложение Win32 API - это приложение Windows, которое использует библиотеки Win32 API для создания и управления окнами и элементами пользовательского интерфейса. Минимально достаточное оконное приложение на Win32 API обычно состоит из окна, которое может быть открыто, закрыто и перерисовано.

Для создания окна в Win32 API, программист должен зарегистрировать класс окна и создать экземпляр этого класса. Зарегистрированный класс содержит информацию о том, как окно должно выглядеть и какие обработчики событий должны быть вызваны для обработки сообщений, отправленных в окно.

Обработка основных оконных сообщений включает в себя обработку сообщений, таких как WM\_CREATE, WM\_PAINT, WM\_COMMAND, WM\_RBUTTONDOWN, WM\_LBUTTONDOWN, WM\_KEYDOWN WM\_CLOSE и WM\_DESTROY. Сообщение WM\_CREATE отправляется системой в окно при создании окна, сообщение WM\_PAINT отправляется при необходимости перерисовки окна, сообщение WM\_COMMAND отправляется при действиях с элементами управления(кнопки, меню), сообщение WM\_RBUTTONDOWN(WM\_LBUTTONDOWN) отправляется при нажатии правой(левой) кнопки мыши, сообщение WM\_KEYDOWN отправляется при нажатии клавиши на клавиатуре, сообщение WM\_CLOSE отправляется, когда пользователь закрывает окно, а сообщение WM\_DESTROY отправляется, когда окно должно быть уничтожено.

Реализация обработки этих сообщений в приложении Win32 API обычно осуществляется через обработчики сообщений оконной процедуры, которые определены программистом. Оконная процедура приложения может быть определена как статическая функция в коде приложения, которая будет вызвана при каждом получении сообщения окном.

Также возможно использование других функций Win32 API, таких как CreateWindow, ShowWindow и UpdateWindow, чтобы создавать, отображать и обновлять окна приложения.

Результатом использования Win32 API являются интуитивно понятные и функциональные приложения, которые позволяют пользователям взаимодействовать с компьютером посредством элементов управления, таких как текст, графика, кнопки и поля ввода. Тем не менее, следует учитывать, что в различных контекстах некоторые термины в документации Windows могут иметь разные значения, например, слово "служба" может относиться к вызываемой подпрограмме, драйверу устройства или к обслуживающему процессу.

**3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

Была создана программа с возможностью открывать, сохранять и редактировать документы (рисунок 3.1).

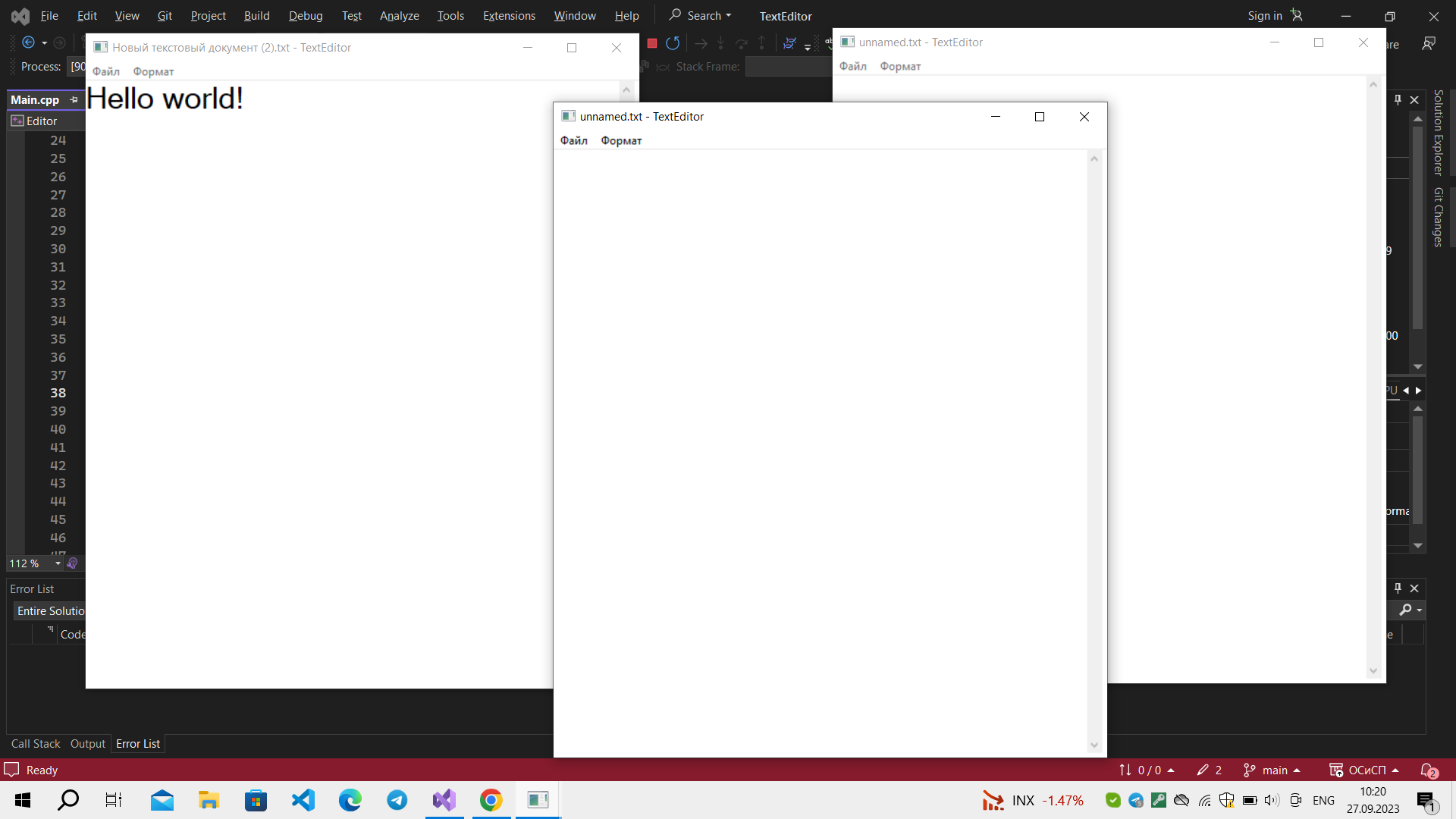


Рисунок 3.1– Результат работы программы

В меню можно найти следующие кнопки:

1 Создать – при нажатии создастся новый пустой документ.

2 Новое окно – при нажатии откроется новое окно, в котором будет создан пустой документ.

3 Открыть – будет предложено выбрать файл для открытия, после чего в рабочей области появится содержимое этого файла.

4 Сохранить –при нажатии будет предложено выбрать директорию для фала.

5 Выход – при нажатии на эту кнопку приложение завершит свою работу.

**ВЫВОД**

В ходе выполнения лабораторной работы были изучены основы программирования в Win 32 API. В результате было создано оконное приложение, которое позволяет пользователю открывать, редактировать и сохранять текст. Добавлена возможность одновременно работать сразу с несколькими документами.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

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

[2] Разработка приложений для Windows [Электронный ресурс]. – Режим доступа:  [https://learn.microsoft.com/ru-ru/windows/win32/menurc/using-menus](%20https://learn.microsoft.com/ru-ru/windows/win32/menurc/using-menus).

[3] Справочник по программированию для Win32 [Электронный ресурс]. – Режим доступа:  [https://learn.microsoft.com/ru-ru/windows/win32/api/.](%20https://learn.microsoft.com/ru-ru/windows/win32/api/.)

[4] Основы программирования для Win32 API [Электронный ресурс].

– Режим доступа: https://dims.karelia.ru/win32/.

[5] Введение в Win32 API [Электронный ресурс]. – Режим доступа: http://cppstudio.com/post/9384/.

[6] Начало работы с Win32 API [Электронный ресурс]. – Режим доступа: https://riptutorial.com/winapi.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Листинг кода**

int WINAPI WinMain(HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR lpCmdLine, int nCmdShow) {

hInstance = hInst;

WNDCLASSEX wc = { sizeof(WNDCLASSEX) };

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);

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

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

wc.hIconSm = LoadIcon(NULL, IDI\_APPLICATION);

wc.hInstance = hInstance;

wc.lpfnWndProc = WindowProc;

wc.lpszClassName = L"MyWindowClass";

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

if (!RegisterClassEx(&wc)) {

MessageBox(NULL, L"Ошибка регистрации окна!", L"Ошибка!", MB\_ICONERROR | MB\_OK);

return 1;

}

HWND hwndMain = CreateNewWindow(hInstance);

if (!IsValidWindow(hwndMain))

return 1;

HHOOK keyboardHook = SetWindowsHookEx(WH\_KEYBOARD\_LL, KeyboardHook, hInstance, 0);

ShowWindow(hwndMain, nCmdShow);

UpdateWindow(hwndMain);

MSG msg;

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

TranslateMessage(&msg);

DispatchMessage(&msg);

}

UnhookWindowsHookEx(keyboardHook);

UnregisterClass(wc.lpszClassName, hInstance);

return 0;

}

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

switch (uMsg) {

case WM\_COMMAND:

{

switch (wParam){

case OnExitClicked:

PostQuitMessage(0);

break;

case OnOpenClicked:

LoadData(hwnd);

break;

case OnSaveClicked:

SaveData(hwnd);

break;

case OnCreateClicked:

ResetWindowData(hwnd);

break;

case OnTextColorClicked:

SetTextColor(hwnd);

break;

case OnBGColorClicked:

SetBGColor(hwnd);

break;

case OnFontClicked:

SetFont(hwnd);

break;

case OnNewWindowClicked:

HWND hwndNew = CreateNewWindow(hInstance);

windowCount++;

ShowWindow(hwndNew, SW\_SHOWNORMAL);

break;

}

break;

}

case WM\_SIZE:

{

int newWidth = LOWORD(lParam) - 5;

int newHeight = HIWORD(lParam) - 5;

HWND hEdit = GetDlgItem(hwnd, IDC\_EDIT\_ID1);

MoveWindow(hEdit, 0, 0, newWidth, newHeight, TRUE);

return 0;

}

case WM\_CLOSE:

{

if (hwnd != GetConsoleWindow()) {

WindowData\* windowData = (WindowData\*)GetWindowLongPtr(hwnd, GWLP\_USERDATA);

if (windowData) {

delete windowData;

}

DestroyWindow(hwnd);

windowCount--;

}

}

case WM\_DESTROY:

{

if (windowCount == 0) {

PostQuitMessage(0);

}

break;

}

case WM\_CTLCOLOREDIT:

{

WindowData\* data = (WindowData\*)GetWindowLongPtr(hwnd, GWLP\_USERDATA);

HDC hdcEdit = (HDC)wParam;

SetTextColor(hdcEdit, data->textColor);

SetBkColor(hdcEdit, data->bgColor);

return (LRESULT)data->hEditTextBrush;

}

default:

{

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

}

}

}

HWND CreateNewWindow(HINSTANCE hInst) {

WindowData\* windowData = new WindowData;

windowData->windowText = L"unnamed.txt";

windowData->hEditBgBrush = CreateSolidBrush(windowData->bgColor);

windowData->hEditTextBrush = CreateSolidBrush(windowData->bgColor);

std::wstring title = windowData->windowText + L" - TextEditor";

HWND hwnd = CreateWindow(\_T("MyWindowClass"), title.c\_str(), WS\_OVERLAPPEDWINDOW, 100, 100, 600, 700, NULL, NULL, hInstance, NULL);

SetWindowLongPtr(hwnd, GWLP\_USERDATA, (LONG\_PTR)windowData);

HMENU RootMenu = CreateMenu();

HMENU SubMenu = CreateMenu();

HMENU SubMenu1 = CreateMenu();

AppendMenu(SubMenu, MF\_STRING, OnCreateClicked, L"Создать");

AppendMenu(SubMenu, MF\_STRING, OnNewWindowClicked, L"Новое окно");

AppendMenu(SubMenu, MF\_STRING, OnOpenClicked, L"Открыть");

AppendMenu(SubMenu, MF\_STRING, OnSaveClicked, L"Сохранить");

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

AppendMenu(SubMenu, MF\_STRING, OnExitClicked, L"Выход");

AppendMenu(SubMenu1, MF\_STRING, OnTextColorClicked, L"Цвет текста");

AppendMenu(SubMenu1, MF\_STRING, OnBGColorClicked, L"Цвет фона");

AppendMenu(SubMenu1, MF\_STRING, OnFontClicked, L"Стиль шрифта");

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

AppendMenu(RootMenu, MF\_POPUP, (UINT\_PTR)SubMenu1, L"Формат");

SetMenu(hwnd, RootMenu);

HWND edit = CreateWindowA("edit", "", WS\_VISIBLE | WS\_CHILD | ES\_MULTILINE | WS\_VSCROLL, 0, 0, 585, 700, hwnd, (HMENU)IDC\_EDIT\_ID1, NULL, NULL);

SendMessage(edit, WM\_SETFONT, (WPARAM)windowData->hFont, TRUE);

return hwnd;

}

bool IsValidWindow(HWND hwnd){

if (hwnd == NULL) {

MessageBox(NULL, \_T("Ошибка при создании окна."), \_T("Ошибка"), MB\_ICONERROR);

return false;

}

return true;

}

void SaveData(HWND hwnd) {

HWND edit = GetDlgItem(hwnd, IDC\_EDIT\_ID1);

WindowData\* data = (WindowData\*)GetWindowLongPtr(hwnd, GWLP\_USERDATA);

int textLength = GetWindowTextLength(edit);

wchar\_t\* text = new wchar\_t[textLength + 1];

GetWindowText(edit, text, textLength + 1);

OPENFILENAME ofn = { 0 };

wchar\_t filePath[MAX\_PATH];

wcscpy\_s(filePath, data->windowText.c\_str());

ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = hwnd;

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

ofn.lpstrFile = filePath;

ofn.nMaxFile = MAX\_PATH;

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

if (GetSaveFileName(&ofn)) {

std::wofstream file(ofn.lpstrFile, std::ios::out | std::ios::binary);

file.imbue(std::locale(file.getloc(), new std::codecvt\_utf8\_utf16<wchar\_t>));

if (file.is\_open()) {

file.write(text, wcslen(text));

file.close();

MessageBox(NULL, L"Файл был успешно сохранен!", L"Внимание!", MB\_OK);

}

}

delete[] text;

}

void LoadData(HWND hwnd){

HWND edit = GetDlgItem(hwnd, IDC\_EDIT\_ID1);

WindowData\* data = (WindowData\*)GetWindowLongPtr(hwnd, GWLP\_USERDATA);

OPENFILENAME ofn = { 0 };

wchar\_t filePath[MAX\_PATH] = L"";

ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = hwnd;

ofn.lpstrFilter = L"Текстовые файлы\0\*.txt\0Все файлы\0\*.\*\0";

ofn.lpstrFile = filePath;

ofn.nMaxFile = MAX\_PATH;

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

if (GetOpenFileName(&ofn)) {

std::wifstream file(ofn.lpstrFile, std::ios::in | std::ios::binary);

if (file.is\_open()) {

std::locale utf8\_locale(std::locale(), new std::codecvt\_utf8<wchar\_t>);

file.imbue(utf8\_locale);

std::wstring fileContent((std::istreambuf\_iterator<wchar\_t>(file)), std::istreambuf\_iterator<wchar\_t>());

file.close();

if (!fileContent.empty()){

SetWindowTextW(edit, fileContent.c\_str());

data->windowText = PathFindFileName(ofn.lpstrFile);

std::wstring title = data->windowText + L" - TextEditor";

SetWindowText(hwnd, title.c\_str());

MessageBox(NULL, L"Файл был успешно загружен!", L"Внимание!", MB\_OK);

}

else{

MessageBox(NULL, L"Файл пустой.", L"Предупреждение", MB\_ICONWARNING);

}

}

else {

MessageBox(NULL, L"Не удалось открыть файл.", L"Ошибка", MB\_ICONERROR);

}

}

}

void ResetWindowData(HWND hWnd) {

HWND edit = GetDlgItem(hWnd, IDC\_EDIT\_ID1);

WindowData\* data = (WindowData\*)GetWindowLongPtr(hWnd, GWLP\_USERDATA);

SetWindowTextW(edit, L"");

data->windowText = L"unnamed.txt";

std::wstring title = data->windowText + L" - TextEditor";

SetWindowText(hWnd, title.c\_str());

}