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 № 4

на тему

**УПРАВЛЕНИЕ ПРОЦЕССАМИ И ПОТОКАМИ (*WINDOWS*). ПОРОЖДЕНИЕ, ЗАВЕРШЕНИЕ, ИЗМЕНЕНИЕ ПРИОРИТЕТОВ ПРОЦЕССОВ И ПОТОКОВ**

Выполнил:

студент гр. 153503

Степанов В.Н.

Проверил:

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc146836467)

[2 Теоретические сведения 4](#_Toc146836468)

[3 Полученные результаты 6](#_Toc146836469)

[Выводы 7](#_Toc146836470)

[Список использованных источников 8](#_Toc146836471)

[Приложение А (обязательное) листинг кода 9](#_Toc146836472)

# **1 ЦЕЛЬ РАБОТЫ**

Изучить управление процессами и потоками в *Windows.* Освоить возможности порождения, завершения, изменения приоритета процессов и потоков. Создать оконное приложение, выполняющее чтение, запись данных в файл, отрисовку окна загрузки в разных потоках.

**2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

*Win32 API* (*Application Programming Interface*) является мощным инструментом, предоставляемым разработчикам для управления процессами и потоками в операционной системе *Windows*. Процесс в этом контексте представляет собой самостоятельную программу или задачу, каждая из которых обладает собственным адресным пространством и ресурсами, позволяя им функционировать независимо от остальных процессов. Важно отметить, что работа одного процесса не влияет на другие процессы, так что завершение или возникновение ошибок в одном процессе не затрагивает остальные. Операционная система *Windows* эффективно управляет выделением ресурсов для каждого процесса, таких как процессорное время и оперативная память, обеспечивая равномерное распределение ресурсов между активными процессами.

Поток, с другой стороны, является базовой единицей выполнения внутри процесса и играет ключевую роль в многозадачности, позволяя процессам выполнять различные задачи параллельно или асинхронно. Потоки принадлежат конкретному процессу, и каждый процесс как минимум имеет один главный поток, создаваемый при запуске процесса. Дополнительные потоки могут быть созданы внутри процесса для одновременного выполнения различных задач.

Многопоточность (multithreading) в операционной системе *Windows* подразумевает использование нескольких потоков внутри одного процесса для одновременного выполнения разных задач. Многопоточные приложения способствуют увеличению производительности и отзывчивости приложения, позволяя, например, одному потоку обрабатывать пользовательский интерфейс, а другому заниматься фоновыми вычислениями.

Имея в виду *Win32 API*, стоит отметить, что операции ввода и вывода (I/O) играют ключевую роль в разработке приложений, позволяя им взаимодействовать с файлами, устройствами и сетевыми ресурсами. Эти операции I/O предоставляют обширный набор функций и механизмов для эффективной и гибкой работы с данными.

В ходе выполнения данной лабораторной работы использовались следующие инструменты и концепции:

* *CreateThread* предназначена для создания новых потоков внутри процесса и позволяет разработчикам реализовать параллельное выполнение задач в приложениях под операционной системой *Windows*.
* *WaitForSingleObject* позволяет потоку ожидать завершения выполнения другого потока или объекта синхронизации. Она блокирует выполнение текущего потока, пока целевой поток или объект синхронизации не завершат свою работу или не перейдут в состояние, которое удовлетворяет заданным условиям ожидания.

# **3 ПОЛУЧЕННЫЕ РЕЗУЛЬТАТЫ**

Была добавлена функциональность подсчета слов из *Rich Edit* в отдельном потоке. Для начала или окончания работы этого потока, который пересчитывал слова нужно нажать на соответствующие кнопки.

Добавленный интерфейс (рисунок 1).



Рисунок 1 – Интерфейс меню

Добавленный интерфейс для отображения числа слов(рисунок 2).

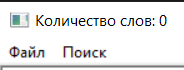


Рисунок 2 – Интерфейс количества слов

# 

# **ВЫВОДЫ**

В результате выполнения лабораторной работы было изучено управление процессами и потоками в *Windows,* освоены возможности порождения, завершения, изменения приоритета процессов и потоков. Был добавлен функционал по отображению количества слов и подсчету этих слов в отдельном потоке.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Управление потоками в *WinAPI* [Электронный ресурс]. – Режим доступа: https://ppt-online.org/799671.
2. Потоки в *Win32* [Электронный ресурс]. – Режим доступа: https://club.shelek.ru/viewart.php?id=71.

# **ПРИЛОЖЕНИЕ А (обязательное) Листинг кода**

Листинг 1 **–** Файл *source.cpp*:

#include <windows.h>

#include <richedit.h>

#include <string>

#include <sstream>

#include <iomanip>

#include <vector>

#include <tchar.h>

#include "Richedit.h"

#include "commctrl.h"

#include <iostream>

#define RICHEDIT\_CLASS L"RICHEDIT50W"

#define IDM\_OPEN 1001

#define IDM\_SAVE 1002

#define IDM\_EXIT 1003

#define IDM\_NEW 1004

#define IDC\_CLOSE\_BUTTON 1005

#define IDM\_SEARCH 1006

#define IDM\_SEARCH\_IN\_SEARCH\_WINDOW 1008

#define IDM\_CHANGE\_THEME 1009

#define IDM\_CHANGE\_FONT 1010

#define IDM\_OPEN\_BINARY 1011

#define IDM\_SAVE\_BINARY 1012

#define IDC\_START\_COUNT 2

#define IDC\_WORD\_LIST 3

#define WM\_UPDATE\_WORD\_COUNT 6

#define IDC\_STOP\_COUNT 5

#define HOTKEY\_ID 1

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

LRESULT CALLBACK SearchWindowProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);

static int wordsCount = 0;

static HWND hEdit;

static HWND hSearchEdit = nullptr;

static int searchStartPosition = 0;

static HWND hSearchWindow = nullptr;

HWND g\_hwnd;

HANDLE g\_hFileMapping;

LPVOID g\_pFileData;

DWORD g\_fileSize;

bool g\_bIsWordCountThreadRunning = true;

HANDLE hThread = nullptr;

UINT\_PTR timerId;

void OpenSearchWindow(HWND hWnd)

{

hSearchWindow = CreateWindow(

L"SearchWindowClass",

L"Search Window",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, 400, 200,

nullptr,

nullptr,

GetModuleHandle(nullptr),

nullptr

);

if (hSearchWindow != nullptr)

{

hSearchEdit = CreateWindowEx(

WS\_EX\_CLIENTEDGE,

L"EDIT",

L"",

WS\_CHILD | WS\_VISIBLE | ES\_AUTOHSCROLL,

10, 10, 150, 30,

hSearchWindow,

nullptr,

GetModuleHandle(nullptr),

nullptr

);

HWND hSearchButton = CreateWindow(

L"BUTTON",

L"Search",

WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

170, 10, 80, 30,

hSearchWindow,

reinterpret\_cast<HMENU>(IDM\_SEARCH\_IN\_SEARCH\_WINDOW),

GetModuleHandle(nullptr),

nullptr

);

if (hSearchEdit == nullptr || hSearchButton == nullptr)

{

MessageBox(hWnd, L"Не удалось создать элементы для поиска.", L"Ошибка", MB\_OK | MB\_ICONERROR);

DestroyWindow(hSearchWindow);

}

ShowWindow(hSearchWindow, SW\_SHOWNORMAL);

}

else

{

MessageBox(hWnd, L"Не удалось создать окно поиска.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

}

DWORD WINAPI WordCountThread(LPVOID lpParam) {

while (g\_bIsWordCountThreadRunning) {

int textLength = GetWindowTextLength(hEdit);

if (textLength > 0) {

std::wstring text;

text.resize(textLength + 1);

GetWindowText(hEdit, &text[0], textLength + 1);

int wordCount = 0;

bool inWord = false;

for (wchar\_t c : text) {

if (iswspace(c)) {

inWord = false;

}

else {

if (!inWord) {

wordCount++;

inWord = true;

}

}

}

wordsCount = wordCount;

}

Sleep(1000);

}

return 0;

}

void OnHotkey(HWND hWnd, WPARAM wParam, LPARAM lParam)

{

if (wParam == HOTKEY\_ID)

{

OpenSearchWindow(hWnd);

}

}

void OpenColorDialog()

{

CHOOSECOLOR cc = { sizeof(CHOOSECOLOR) };

static COLORREF custColors[16] = { 0 };

cc.rgbResult = RGB(255, 255, 255);

cc.lpCustColors = custColors;

cc.Flags = CC\_FULLOPEN | CC\_RGBINIT;

if (ChooseColor(&cc))

{

SendMessage(hEdit, EM\_SETBKGNDCOLOR, FALSE, cc.rgbResult);

}

}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PSTR lpCmdLine, int nCmdShow)

{

if (!RegisterHotKey(NULL, HOTKEY\_ID, MOD\_CONTROL, 'F'))

{

MessageBox(NULL, L"Не удалось зарегистрировать горячую клавишу!", L"Error", MB\_ICONERROR | MB\_OK);

return 1;

}

HWND hWnd{};

MSG msg{};

WNDCLASS wc{ sizeof(WNDCLASS) };

wc.lpfnWndProc = WndProc;

wc.hInstance = hInstance;

wc.cbClsExtra = 0;

wc.cbWndExtra = 0;

wc.hbrBackground = reinterpret\_cast<HBRUSH>(GetStockObject(LTGRAY\_BRUSH));

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

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

wc.hInstance = hInstance;

wc.lpszClassName = L"LAB1!";

wc.lpszMenuName = nullptr;

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

WNDCLASS wcSearchWindow{ sizeof(WNDCLASS) };

wcSearchWindow.lpfnWndProc = SearchWindowProc;

wcSearchWindow.hInstance = hInstance;

wcSearchWindow.hbrBackground = reinterpret\_cast<HBRUSH>(GetStockObject(LTGRAY\_BRUSH));

wcSearchWindow.lpszClassName = L"SearchWindowClass";

if (!RegisterClass(&wcSearchWindow) || !RegisterClassW(&wc)) {

return EXIT\_FAILURE;

}

if (hWnd = CreateWindow(wc.lpszClassName, L"LAB1", WS\_OVERLAPPEDWINDOW | WS\_CLIPCHILDREN, CW\_USEDEFAULT, CW\_USEDEFAULT, 800, 600, nullptr, nullptr, wc.hInstance, nullptr)) {

HMENU hMenu = CreateMenu();

HMENU hFileMenu = CreateMenu();

AppendMenu(hMenu, MF\_POPUP, (UINT\_PTR)hFileMenu, L"Файл");

AppendMenu(hFileMenu, MF\_STRING, IDM\_OPEN, L"Открыть");

AppendMenu(hFileMenu, MF\_STRING, IDM\_SAVE, L"Сохранить");

AppendMenu(hFileMenu, MF\_STRING, IDM\_OPEN\_BINARY, L"Открыть бинарный файл");

AppendMenu(hFileMenu, MF\_STRING, IDM\_SAVE\_BINARY, L"Сохранить бинарный файл");

AppendMenu(hFileMenu, MF\_STRING, IDC\_START\_COUNT, L"Начать счетчик");

AppendMenu(hFileMenu, MF\_STRING, IDC\_STOP\_COUNT, L"Закончить счетчик");

AppendMenu(hFileMenu, MF\_SEPARATOR, 0, NULL);

AppendMenu(hFileMenu, MF\_STRING, IDM\_EXIT, L"Выход");

AppendMenu(hFileMenu, MF\_SEPARATOR, 0, NULL);

AppendMenu(hFileMenu, MF\_STRING, IDM\_NEW, L"Новый");

AppendMenu(hFileMenu, MF\_SEPARATOR, 0, NULL);

AppendMenu(hFileMenu, MF\_STRING, IDM\_CHANGE\_THEME, L"Поменять тему");

AppendMenu(hFileMenu, MF\_STRING, IDM\_CHANGE\_FONT, L"Изменить шрифт");

AppendMenu(hMenu, MF\_STRING, IDM\_SEARCH, L"Поиск");

SetMenu(hWnd, hMenu);

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

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

TranslateMessage(&msg);

DispatchMessage(&msg);

if (msg.message == WM\_HOTKEY)

{

OnHotkey(msg.hwnd, msg.wParam, msg.lParam);

}

}

}

return EXIT\_SUCCESS;

}

LRESULT CALLBACK SearchWindowProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

switch (message)

{

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case IDM\_SEARCH\_IN\_SEARCH\_WINDOW:

{

int searchQueryLength = GetWindowTextLength(hSearchEdit);

if (searchQueryLength == 0)

{

MessageBox(hWnd, L"Введите текст.", L"Error", MB\_OK | MB\_ICONERROR);

return 0;

}

wchar\_t\* searchQuery = new wchar\_t[searchQueryLength + 1];

GetWindowText(hSearchEdit, searchQuery, searchQueryLength + 1);

int mainEditLength = GetWindowTextLength(hEdit);

wchar\_t\* mainEditText = new wchar\_t[mainEditLength + 1];

GetWindowText(hEdit, mainEditText, mainEditLength + 1);

wchar\_t\* found = wcsstr(mainEditText + searchStartPosition, searchQuery);

if (found != nullptr)

{

int foundPos = found - mainEditText;

SendMessage(hEdit, EM\_SETSEL, foundPos, foundPos + searchQueryLength);

CHARFORMAT2 cf;

cf.cbSize = sizeof(cf);

cf.dwMask = CFM\_BACKCOLOR;

cf.crBackColor = RGB(255, 255, 0);

SendMessage(hEdit, EM\_SETCHARFORMAT, SCF\_SELECTION, (LPARAM)&cf);

searchStartPosition = foundPos + searchQueryLength;

}

else

{

searchStartPosition = 0;

MessageBox(hWnd, L"No more matches found.", L"Search Complete", MB\_OK | MB\_ICONINFORMATION);

}

delete[] searchQuery;

delete[] mainEditText;

}

break;

case WM\_CLOSE:

if (hWnd == hSearchWindow)

DestroyWindow(hWnd);

break;

}

break;

case WM\_DESTROY:

if (hWnd == hSearchWindow)

hSearchWindow = nullptr;

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

void ResizeEditControl(HWND hWnd, HWND hEdit)

{

RECT clientRect;

GetClientRect(hWnd, &clientRect);

SetWindowPos(hEdit, nullptr, 0, 0, clientRect.right, clientRect.bottom, SWP\_NOZORDER);

}

void OpenBinaryFile(LPCTSTR filePath) {

if (g\_pFileData != NULL) {

UnmapViewOfFile(g\_pFileData);

g\_pFileData = NULL;

}

if (g\_hFileMapping != NULL) {

CloseHandle(g\_hFileMapping);

g\_hFileMapping = NULL;

}

HANDLE hFile = CreateFile(filePath, GENERIC\_READ | GENERIC\_WRITE, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

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

MessageBox(g\_hwnd, \_T("Failed to open the file"), \_T("Error"), MB\_ICONERROR);

return;

}

g\_fileSize = GetFileSize(hFile, NULL);

g\_hFileMapping = CreateFileMapping(hFile, NULL, PAGE\_READWRITE, 0, g\_fileSize, NULL);

if (g\_hFileMapping == NULL) {

MessageBox(g\_hwnd, \_T("Failed to create file mapping"), \_T("Error"), MB\_ICONERROR);

CloseHandle(hFile);

return;

}

g\_pFileData = MapViewOfFile(g\_hFileMapping, FILE\_MAP\_READ | FILE\_MAP\_WRITE, 0, 0, g\_fileSize);

if (g\_pFileData == NULL) {

MessageBox(g\_hwnd, \_T("Failed to map the file into memory"), \_T("Error"), MB\_ICONERROR);

CloseHandle(g\_hFileMapping);

g\_hFileMapping = NULL;

CloseHandle(hFile);

return;

}

}

LPCWSTR AsciiToHex(const WCHAR\* asciiString) {

std::wstringstream resultStream;

for (size\_t i = 0; i < wcslen(asciiString); ++i) {

int asciiValue = static\_cast<int>(asciiString[i]);

resultStream << std::hex << std::setw(2) << std::setfill(L'0') << asciiValue;

if (i < wcslen(asciiString) - 1) {

resultStream << L' ';

}

}

std::wstring result = resultStream.str();

LPCWSTR resultStr = \_wcsdup(result.c\_str());

return resultStr;

}

void StopWordCountThread() {

g\_bIsWordCountThreadRunning = false;

if (hThread != nullptr) {

WaitForSingleObject(hThread, INFINITE);

CloseHandle(hThread);

hThread = nullptr;

}

}

LPCWSTR HexToAsciiString(const WCHAR\* hexString) {

std::wstringstream resultStream;

std::wistringstream hexStream(hexString);

std::wstring hexByte;

while (hexStream >> hexByte) {

int number = 0;

std::wstringstream hexConverter;

hexConverter << std::hex << hexByte;

hexConverter >> number;

WCHAR asciiChar = static\_cast<WCHAR>(number);

resultStream << asciiChar;

}

std::wstring result = resultStream.str();

LPCWSTR resultStr = \_wcsdup(result.c\_str());

return resultStr;

}

void SaveHexToFile(LPCTSTR filePath, const std::wstring& hexString) {

if (g\_hFileMapping != NULL) {

CloseHandle(g\_hFileMapping);

g\_hFileMapping = NULL;

HANDLE hFile = CreateFile(filePath, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

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

DWORD bytesWritten;

WriteFile(hFile, hexString.c\_str(), static\_cast<DWORD>(hexString.size() \* sizeof(WCHAR)), &bytesWritten, NULL);

CloseHandle(hFile);

}

}

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

static wchar\_t currentFileName[MAX\_PATH] = L"";

HWND hButtonContainer = nullptr;

HWND hStartWordCountButton = nullptr;

switch (message)

{

case WM\_CREATE:

{

LoadLibrary(TEXT("Msftedit.dll"));

hEdit = CreateWindowEx(

WS\_EX\_CLIENTEDGE,

RICHEDIT\_CLASS,

L"",

WS\_CHILD | WS\_VISIBLE | WS\_VSCROLL | WS\_HSCROLL | ES\_MULTILINE | ES\_AUTOVSCROLL | ES\_AUTOHSCROLL | ES\_NOHIDESEL,

0, 0, 800, 500,

hWnd,

nullptr,

GetModuleHandle(nullptr),

nullptr

);

if (hEdit == nullptr)

{

MessageBox(hWnd, L"Не удалось создать элемент управления Edit.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

timerId = SetTimer(hWnd, 1, 1000, nullptr);

break;

}

case WM\_TIMER:

if (wParam == 1) {

std::wstring titleText = L"Количество слов: " + std::to\_wstring(wordsCount);

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

}

break;

case WM\_COMMAND:

switch (LOWORD(wParam))

{

case IDM\_CHANGE\_THEME:

OpenColorDialog();

break;

case IDM\_CHANGE\_FONT:

{

CHOOSEFONT cf = { sizeof(CHOOSEFONT) };

LOGFONT lf = { 0 };

cf.hwndOwner = hWnd;

cf.lpLogFont = &lf;

cf.Flags = CF\_SCREENFONTS | CF\_EFFECTS | CF\_INITTOLOGFONTSTRUCT;

if (ChooseFont(&cf))

{

HFONT hFont = CreateFontIndirect(&lf);

SendMessage(hEdit, WM\_SETFONT, (WPARAM)hFont, TRUE);

}

break;

}

case IDM\_OPEN:

{

OPENFILENAME ofn;

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

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hWnd;

ofn.lpstrFilter = L"C++ файлы (\*.cpp;\*.txt)\0\*.cpp;\*.txt\0Все файлы (\*.\*)\0\*.\*\0";

ofn.lpstrFile = szFile;

ofn.nMaxFile = MAX\_PATH;

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

if (GetOpenFileName(&ofn))

{

wcscpy\_s(currentFileName, MAX\_PATH, ofn.lpstrFile);

HANDLE hFile = CreateFile(ofn.lpstrFile, GENERIC\_READ, FILE\_SHARE\_READ, NULL, OPEN\_EXISTING, 0, NULL);

if (hFile != INVALID\_HANDLE\_VALUE)

{

DWORD dwFileSize = GetFileSize(hFile, NULL);

wchar\_t\* fileContent = (wchar\_t\*)malloc((dwFileSize + 1) \* sizeof(wchar\_t));

if (fileContent)

{

DWORD bytesRead;

ReadFile(hFile, fileContent, dwFileSize \* sizeof(wchar\_t), &bytesRead, NULL);

fileContent[dwFileSize / sizeof(wchar\_t)] = L'\0';

SetWindowText(hEdit, fileContent);

free(fileContent);

}

else

{

MessageBox(hWnd, L"Не удалось выделить память для содержимого файла.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

CloseHandle(hFile);

}

else

{

MessageBox(hWnd, L"Не удалось открыть выбранный файл.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

}

break;

}

case IDM\_SAVE:

{

HANDLE hFile = INVALID\_HANDLE\_VALUE;

if (currentFileName[0] != L'\0')

{

hFile = CreateFile(currentFileName, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

}

if (hFile == INVALID\_HANDLE\_VALUE)

{

OPENFILENAME ofn;

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

ZeroMemory(&ofn, sizeof(ofn));

ofn.lStructSize = sizeof(ofn);

ofn.hwndOwner = hWnd;

ofn.lpstrFilter = L"C++ files (\*.cpp;\*.txt)\0\*.cpp;\*.txt\0All files (\*.\*)\0\*.\*\0";

ofn.lpstrFile = szFile;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_OVERWRITEPROMPT;

if (GetSaveFileName(&ofn))

{

hFile = CreateFile(ofn.lpstrFile, GENERIC\_WRITE, 0, NULL, CREATE\_ALWAYS, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile != INVALID\_HANDLE\_VALUE)

{

wcscpy\_s(currentFileName, MAX\_PATH, ofn.lpstrFile);

}

else

{

MessageBox(hWnd, L"Не удалось создать выбранный файл.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

}

}

if (hFile != INVALID\_HANDLE\_VALUE)

{

int textLength = GetWindowTextLength(hEdit);

wchar\_t\* textBuffer = (wchar\_t\*)malloc((textLength + 1) \* sizeof(wchar\_t));

if (textBuffer)

{

GetWindowText(hEdit, textBuffer, textLength + 1);

DWORD bytesWritten;

WriteFile(hFile, textBuffer, textLength \* sizeof(wchar\_t), &bytesWritten, NULL);

free(textBuffer);

CloseHandle(hFile);

}

else

{

MessageBox(hWnd, L"Не удалось выделить память для текста.", L"Ошибка", MB\_OK | MB\_ICONERROR);

CloseHandle(hFile);

}

}

break;

}

case IDM\_NEW:

{

if (MessageBox(hWnd, L"Вы уверены, что хотите создать новый документ? Все несохраненные изменения будут потеряны.", L"Подтвердите действие", MB\_YESNO | MB\_ICONQUESTION) == IDYES)

{

SetWindowText(hEdit, L"");

currentFileName[0] = L'\0';

}

break;

}

case IDM\_OPEN\_BINARY: {

OPENFILENAME ofn = { 0 };

TCHAR filePath[MAX\_PATH] = { 0 };

ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = hWnd;

ofn.lpstrFile = filePath;

ofn.nMaxFile = MAX\_PATH;

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

if (GetOpenFileName(&ofn)) {

OpenBinaryFile(filePath);

int requiredBufferSize = MultiByteToWideChar(CP\_UTF8, 0, (LPCCH)g\_pFileData, -1, NULL, 0);

if (requiredBufferSize > 0) {

WCHAR\* utf16Text = new WCHAR[requiredBufferSize];

MultiByteToWideChar(CP\_UTF8, 0, (LPCCH)g\_pFileData, -1, utf16Text, requiredBufferSize);

LPCWSTR text = HexToAsciiString(utf16Text);

SetWindowText(hEdit, text);

delete[] utf16Text;

}

lstrcpy(currentFileName, filePath);

}

break;

}

case IDM\_SAVE\_BINARY: {

if (g\_hFileMapping != NULL) {

OPENFILENAME ofn = { 0 };

ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = hWnd;

ofn.lpstrFile = currentFileName;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_OVERWRITEPROMPT;

ofn.lpstrFilter = L"Binary Files\0\*.bin\0All Files\0\*.\*\0";

ofn.lpstrDefExt = L"bin";

if (GetSaveFileName(&ofn)) {

int textLength = GetWindowTextLength(hEdit);

std::wstring text;

if (textLength > 0) {

text.resize(textLength);

GetWindowText(hEdit, &text[0], textLength + 1);

}

std::wstring hexString = AsciiToHex(text.c\_str());

SaveHexToFile(currentFileName, hexString);

}

}

break;

}

case IDM\_SEARCH:

OpenSearchWindow(hWnd);

break;

case IDC\_START\_COUNT: {

hThread = CreateThread(nullptr, 0, WordCountThread, nullptr, 0, nullptr);

if (hThread == nullptr) {

MessageBox(hWnd, L"Не удалось запустить поток.", L"Ошибка", MB\_OK | MB\_ICONERROR);

}

break;

}

case IDC\_STOP\_COUNT: {

StopWordCountThread();

break;

}

case WM\_CLOSE: {

if (hThread != nullptr) {

TerminateThread(hThread, 0);

CloseHandle(hThread);

}

DestroyWindow(hWnd);

break;

}

case IDM\_EXIT:

PostQuitMessage(0);

break;

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}