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 в Win 32 API. Оконное приложение Win 32 с минимальной достаточной функциональностью. Обработка основных оконных сообщений»

Выполнил:

студент гр. 153504

Мамченко К.А.

Проверил:

Гриценко Н.Ю.

Минск 2023

СОДЕРЖАНИЕ

1 Цели работы 3

2 Краткие теоретические сведения 4

3 Полученные результаты 7

Выводы 8

**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. Создать текстовый редактор с поддержкой множества открытых документов. Реализовать функциональность сохранения и загрузки файлов.

**2 Краткие теоретические сведения**

WinAPI (Windows API) - это набор функций и интерфейсов, предоставляемых операционной системой Windows для создания приложений под эту платформу. Он позволяет взаимодействовать с различными компонентами операционной системы, такими как окна, ресурсы, файлы, сеть и другие.

При создании простого приложения на основе WinAPI с минимальной достаточной функциональностью обычно используются следующие инструменты и концепции:

1. Заголовочные файлы: Включение заголовочного файла <windows.h> позволяет использовать функции и типы данных WinAPI. Он содержит объявления функций, констант и структур, необходимых для работы с операционной системой Windows.
2. Идентификатор окна (HWND): HWND используется для идентификации и манипулирования окнами в WinAPI. Он представляет собой уникальный идентификатор окна и используется для выполнения различных операций, таких как создание окна, изменение его размера, отображение и обновление.
3. Оконная процедура (Window Procedure): Оконная процедура (WindowProc) является обработчиком сообщений для окна. Она принимает сообщения, такие как нажатия клавиш, изменение размера окна, закрытие окна и другие, и обрабатывает их соответствующим образом.
4. Регистрация класса окна (RegisterClass): Для создания окна необходимо зарегистрировать класс окна с помощью функции RegisterClass. Эта функция определяет различные атрибуты окна, включая оконную процедуру, идентификатор класса и другие параметры.
5. Создание окна (CreateWindowEx): После регистрации класса окна можно создать экземпляр окна с помощью функции CreateWindowEx. Эта функция создает новое окно с заданными атрибутами, такими как размер, положение, стиль и родительское окно.
6. Обработка сообщений (Message Loop): В основе работы приложения на WinAPI лежит цикл обработки сообщений (Message Loop). Цикл GetMessage получает сообщения из очереди сообщений операционной системы и передает их на обработку в оконную процедуру с помощью функций TranslateMessage и DispatchMessage.
7. Сообщения и их обработка: В оконной процедуре обрабатываются различные сообщения, такие как WM\_CREATE (при создании окна), WM\_SIZE (при изменении размера окна), WM\_COMMAND (при выборе пункта меню или нажатии кнопки) и другие. В обработчиках сообщений выполняются соответствующие действия, например, создание элементов интерфейса, обработка ввода пользователя и обновление содержимого окна.
8. Меню (Menu): Для добавления меню в приложение используются функции CreateMenu и AppendMenu. Они позволяют создавать пункты меню и связывать их с командами, которые будут выполняться при выборе пользователем.
9. Отображение и обновление окна: Функции ShowWindow и UpdateWindow используются для отображения и обновления содержимого окна на экране. ShowWindow устанавливает видимость окна, а UpdateWindow обновляет содержимое окна, чтобы отобразить сделанные изменения.
10. Освобождение ресурсов: При завершении работы приложения необходимо освободить ресурсы, выделенные под окно и другие объекты.

Листинг 1 **–** Код исходной программы:

#include <windows.h>

// Идентификаторы элементов управления

#define ID\_EDIT 1001

#define ID\_FILE\_NEW 2001

#define ID\_FILE\_OPEN 2002

#define ID\_FILE\_SAVE 2003

#define ID\_NEW\_WINDOW 2004

// Глобальные переменные

HWND g\_hMainWindow; // Основное окно приложения

HWND g\_hEdit; // Редактор текста

int g\_nDocCount = 1; // Счетчик документов

TCHAR g\_szCurrentFile[MAX\_PATH] = TEXT(""); // Текущий открытый файл

// Прототипы функций

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

void CreateNewDocument();

void OpenDocument();

void SaveDocument();

void UpdateWindowTitle();

void CreateNeWindow();

// Точка входа в программу

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPWSTR lpCmdLine, int nCmdShow)

{

// Регистрация класса окна

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

WNDCLASS wc = {};

wc.lpfnWndProc = WindowProc;

wc.hInstance = hInstance;

wc.lpszClassName = CLASS\_NAME;

RegisterClass(&wc);

// Создание главного окна

g\_hMainWindow = CreateWindowEx(

0,

CLASS\_NAME,

L"Text Editor",

WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT, CW\_USEDEFAULT,

NULL,

NULL,

hInstance,

NULL

);

if (g\_hMainWindow == NULL)

{

return 0;

}

// Создание меню

HMENU hMenu = CreateMenu();

HMENU hFileMenu = CreateMenu();

AppendMenu(hFileMenu, MF\_STRING, ID\_FILE\_NEW, L"&New");

AppendMenu(hFileMenu, MF\_STRING, ID\_FILE\_OPEN, L"&Open");

AppendMenu(hFileMenu, MF\_STRING, ID\_FILE\_SAVE, L"&Save");

AppendMenu(hFileMenu, MF\_STRING, ID\_NEW\_WINDOW, L"&New WIndow");

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

SetMenu(g\_hMainWindow, hMenu);

ShowWindow(g\_hMainWindow, nCmdShow);

// Цикл обработки сообщений

MSG msg;

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

// Обработчик сообщений окна

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

{

switch (uMsg)

{

case WM\_CREATE:

{

// Создание редактора текста

g\_hEdit = CreateWindowEx(

WS\_EX\_CLIENTEDGE,

L"EDIT",

NULL,

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

0, 0, 0, 0,

hwnd,

(HMENU)ID\_EDIT,

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

NULL

);

break;

}

case WM\_SIZE:

{

// Перерасчет размеров редактора текста при изменении размеров окна

int nWidth = LOWORD(lParam);

int nHeight = HIWORD(lParam);

MoveWindow(g\_hEdit, 0, 0, nWidth, nHeight, TRUE);

break;

}

case WM\_COMMAND:

{

// Обработка командных сообщений

int wmId = LOWORD(wParam);

switch (wmId)

{

case ID\_FILE\_NEW:

CreateNewDocument();

break;

case ID\_FILE\_OPEN:

OpenDocument();

break;

case ID\_FILE\_SAVE:

SaveDocument();

break;

case ID\_NEW\_WINDOW:

CreateNeWindow();

break;

}

break;

}

case WM\_CLOSE:

{

// Если пользователь соглашается сохранить изменения, закройте текущее окно

if (MessageBox(hwnd, L"Do you want to save the changes?", L"Save Changes", MB\_YESNO | MB\_ICONQUESTION) == IDYES)

{

SaveDocument();

}

DestroyWindow(hwnd);

return 0;

}

case WM\_DESTROY:

{

// Обработка сообщения разрушения окна

PostQuitMessage(0);

break;

}

}

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

}

void CreateNewDocument()

{

g\_nDocCount++;

g\_szCurrentFile[0] = TEXT('\0');

SetWindowText(g\_hMainWindow, L"Text Editor - New Document");

SetWindowText(g\_hEdit, L"");

}

// Открытие документа

void OpenDocument()

{

OPENFILENAME ofn = {};

TCHAR szFileName[MAX\_PATH] = TEXT("");

ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = g\_hMainWindow;

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

ofn.lpstrFile = szFileName;

ofn.nMaxFile = MAX\_PATH;

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

if (GetOpenFileName(&ofn))

{

// Открытие файла в текущем окне

g\_nDocCount++;

lstrcpy(g\_szCurrentFile, szFileName);

SetWindowText(g\_hMainWindow, g\_szCurrentFile);

HANDLE hFile = CreateFile(g\_szCurrentFile, GENERIC\_READ, 0, NULL, OPEN\_EXISTING, FILE\_ATTRIBUTE\_NORMAL, NULL);

if (hFile != INVALID\_HANDLE\_VALUE)

{

DWORD dwFileSize = GetFileSize(hFile, NULL);

if (dwFileSize != INVALID\_FILE\_SIZE)

{

LPSTR lpFileData = (LPSTR)GlobalAlloc(GPTR, dwFileSize + 1);

if (lpFileData != NULL)

{

DWORD dwBytesRead;

if (ReadFile(hFile, lpFileData, dwFileSize, &dwBytesRead, NULL))

{

lpFileData[dwBytesRead] = '\0';

SetWindowTextA(g\_hEdit, lpFileData);

}

GlobalFree(lpFileData);

}

}

CloseHandle(hFile);

}

}

}

// Сохранение документа

void SaveDocument()

{

if (g\_szCurrentFile[0] == TEXT('\0'))

{

// Если текущий файл пуст, вызываем диалог сохранения файла

OPENFILENAME ofn = {};

TCHAR szFileName[MAX\_PATH] = TEXT("");

ofn.lStructSize = sizeof(OPENFILENAME);

ofn.hwndOwner = g\_hMainWindow;

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

ofn.lpstrFile = szFileName;

ofn.nMaxFile = MAX\_PATH;

ofn.Flags = OFN\_OVERWRITEPROMPT;

if (GetSaveFileName(&ofn))

{

lstrcpy(g\_szCurrentFile, szFileName);

SetWindowText(g\_hMainWindow, g\_szCurrentFile);

}

else

{

return;

}

}

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

if (hFile != INVALID\_HANDLE\_VALUE)

{

int nLength = GetWindowTextLength(g\_hEdit);

LPSTR lpFileData = (LPSTR)GlobalAlloc(GPTR, nLength + 1);

if (lpFileData != NULL)

{

GetWindowTextA(g\_hEdit, lpFileData, nLength + 1);

DWORD dwBytesWritten;

WriteFile(hFile, lpFileData, nLength, &dwBytesWritten, NULL);

GlobalFree(lpFileData);

}

CloseHandle(hFile);

}

}

// Обновление заголовка окна

void UpdateWindowTitle()

{

TCHAR szTitle[MAX\_PATH];

if (g\_szCurrentFile[0] == TEXT('\0'))

{

wsprintf(szTitle, TEXT("Text Editor - New Document %d"), g\_nDocCount);

}

else

{

wsprintf(szTitle, TEXT("Text Editor - %s"), g\_szCurrentFile);

}

SetWindowText(g\_hMainWindow, szTitle);

}

// Создание нового окна

void CreateNeWindow() {

STARTUPINFO si = {};

PROCESS\_INFORMATION pi = {};

// Создание нового процесса с текущим исполняемым файлом

if (CreateProcess(NULL, GetCommandLine(), NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))

{

CloseHandle(pi.hThread);

CloseHandle(pi.hProcess);

}

}

**3 Полученные результаты**

Функция открытия файл продемонстрирована на рисунке 3.1

Изображение выглядит как программное обеспечение, текст, Мультимедийное программное обеспечение, мультимедиа

Автоматически созданное описание

Рисунок 3.1 – Открытие файла

Функция работы в нескольких файлах одновременно продемонстрирована на рисунке 3.2

Изображение выглядит как снимок экрана, текст, Прямоугольник, линия

Автоматически созданное описание

Рисунок 3.2 – Открытие нескольких окон

**Выводы**

В ходе выполнения лабораторной работы были изучены основы программирования в Win 32 API. Как результат, создано оконное приложение с минимальной достаточной функциональностью и реализована обработка основных оконных сообщений. Разработанная программа, позволяющая работу с несколькими документами, реализует функции открытия, создания и сохранения текстовых файлов.