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 № 2

на тему «Расширенное использование оконного интерфейса Win 32 и GDI. Формирование сложных изображений, создание и использование элементов управления, обработка различных сообщений, механизм перехвата сообщений (winhook)»

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. Расширенное использование оконного интерфейса Win 32 и GDI.
2. Научиться создавать и использовать элементы управления, обрабатывать различные сообщения.
3. Изучить механизм перехвата сообщений (winhook).
4. Разработать текстовый редактор с поддержкой настраиваемых тем оформления (стили текста, цвета фона).

**2 Краткие теоретические сведения**

Оконный интерфейс Win32 является основой для создания графических пользовательских интерфейсов в операционных системах Windows.

Он предоставляет множество функций и возможностей для создания и управления окнами, кнопками, меню, диалоговыми окнами и другими элементами пользовательского интерфейса.

GDI (Graphics Device Interface) - это компонент Win32 API, который предоставляет функции для рисования графики, такие как рисование линий, прямоугольников, текста, изображений и т.д.

Используя Win32 и GDI, разработчики могут создавать пользовательские интерфейсы с различными элементами управления и реализовывать сложную графику.

Создание и использование элементов управления, обработка сообщений:

Элементы управления (controls) - это объекты интерфейса, с помощью которых пользователь взаимодействует с приложением (например, кнопки, текстовые поля, списки и т.д.).

Для обработки различных событий и действий, связанных с элементами управления, приложению необходимо обрабатывать сообщения, которые генерируются при взаимодействии с элементами управления.

Сообщения передаются окну приложения в виде целочисленных кодов, которые могут быть обработаны с помощью функции обработки сообщений (message handler).

В обработчике сообщений можно реагировать на определенные события (например, нажатие кнопки) и выполнять соответствующие действия.

Механизм перехвата сообщений (winhook):

Механизм перехвата сообщений (winhook) позволяет перехватывать и обрабатывать сообщения, отправляемые окнам в системе.

Winhook предоставляет возможность установить глобальный или локальный перехватчик сообщений, который будет получать и обрабатывать сообщения до того, как они достигнут целевого окна.

Это позволяет разработчикам расширять функциональность системы или приложения, модифицируя или анализируя сообщения перед их обработкой целевым окном.

Примеры использования winhook включают реализацию горячих клавиш, логирование действий пользователя, фильтрацию сообщений и другие задачи, требующие мониторинга и влияния на сообщения в системе.

Листинг 1 – код исходной программы

#include <windows.h>

#include <string>

// 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

#define ID\_STYLE\_BOLD 3001

#define ID\_STYLE\_ITALIC 3002

#define ID\_STYLE\_UNDERLINE 3003

#define ID\_BACKGROUND\_COLOR 4000

#define ID\_TEXT\_COLOR 4001

#define ID\_FONT\_FACE\_TNR 5001

#define ID\_FONT\_FACE\_CALIBRI 5002

#define ID\_FONT\_FACE\_ARIAL 5003

#define ID\_FONT\_SIZE\_10 6010

#define ID\_FONT\_SIZE\_12 6012

#define ID\_FONT\_SIZE\_14 6014

#define ID\_FONT\_SIZE\_16 6016

#define ID\_FONT\_SIZE\_18 6018

#define ID\_FONT\_SIZE\_20 6020

bool g\_bBold = false;

bool g\_bItalic = false;

bool g\_bUnderline = false;

// 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

HBRUSH hEditBgBrush = NULL;

HBRUSH hEditTextColorBrush = NULL;

static COLORREF bgColor = RGB(255, 255, 255); // Цвет фона по умолчанию

static COLORREF textColor = RGB(0, 0, 0); // Цвет фона по умолчанию

const wchar\_t\* g\_FontFace = L"Arial";

int g\_fontSize = 16;

// Прототипы функций

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

void CreateNewDocument();

void OpenDocument();

void SaveDocument();

void UpdateWindowTitle();

void CreateNeWindow();

void UpdateTextStyle(HWND hWnd);

void ChangeBgc(HWND hwnd);

void ChangeTextColor(HWND hwnd);

// Прототип функции-перехватчика

LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam, LPARAM lParam);

// Глобальная переменная для хранения хука

HHOOK g\_hHook = NULL;

// Точка входа в программу

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

{

g\_hHook = SetWindowsHookEx(WH\_KEYBOARD\_LL, KeyboardProc, NULL, 0);

if (g\_hHook == NULL)

{

// Обработка ошибки

return 1;

}

// 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();

HMENU hStyleMenu = CreateMenu();

HMENU hFontMenu = CreateMenu();

HMENU hFontSize = 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(hStyleMenu, MF\_STRING, ID\_STYLE\_BOLD, L"&Bold");

AppendMenu(hStyleMenu, MF\_STRING, ID\_STYLE\_ITALIC, L"&Italic");

AppendMenu(hStyleMenu, MF\_STRING, ID\_STYLE\_UNDERLINE, L"&Underline");

AppendMenu(hFontMenu, MF\_STRING, ID\_FONT\_FACE\_TNR, L"&Times New Roman");

AppendMenu(hFontMenu, MF\_STRING, ID\_FONT\_FACE\_ARIAL, L"&Arial");

AppendMenu(hFontMenu, MF\_STRING, ID\_FONT\_FACE\_CALIBRI, L"&Calibri");

AppendMenu(hFontSize, MF\_STRING, ID\_FONT\_SIZE\_10, L"&10");

AppendMenu(hFontSize, MF\_STRING, ID\_FONT\_SIZE\_12, L"&12");

AppendMenu(hFontSize, MF\_STRING, ID\_FONT\_SIZE\_14, L"&14");

AppendMenu(hFontSize, MF\_STRING, ID\_FONT\_SIZE\_16, L"&16");

AppendMenu(hFontSize, MF\_STRING, ID\_FONT\_SIZE\_18, L"&18");

AppendMenu(hFontSize, MF\_STRING, ID\_FONT\_SIZE\_20, L"&20");

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

AppendMenu(hMenu, MF\_POPUP, (UINT\_PTR)hStyleMenu, L"&Style");

AppendMenu(hMenu, MF\_POPUP, (UINT\_PTR)hFontMenu, L"&Font Face");

AppendMenu(hMenu, MF\_POPUP, (UINT\_PTR)hFontSize, L"&Font Size");

AppendMenu(hMenu, MF\_STRING, ID\_BACKGROUND\_COLOR, L"&Background color");

AppendMenu(hMenu, MF\_STRING, ID\_TEXT\_COLOR, L"&Text color");

SetMenu(g\_hMainWindow, hMenu);

ShowWindow(g\_hMainWindow, nCmdShow);

// Цикл обработки сообщений

MSG msg;

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

// Удаление хука

UnhookWindowsHookEx(g\_hHook);

return 0;

}

// Обработчик сообщений окна

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

{

static HFONT hDefaultFont = nullptr; // Хранит дескриптор шрифта по умолчанию

switch (uMsg)

{

case WM\_CREATE:

{

// Создаем шрифт с дефолтными параметрами

LOGFONT lf;

ZeroMemory(&lf, sizeof(LOGFONT));

lf.lfHeight = g\_fontSize;

lf.lfWeight = FW\_NORMAL;

lstrcpy(lf.lfFaceName, g\_FontFace);

hDefaultFont = CreateFontIndirect(&lf);

// 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

);

SendMessage(g\_hEdit, WM\_SETFONT, reinterpret\_cast<WPARAM>(hDefaultFont), MAKELPARAM(TRUE, 0));

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:

{

if (LOWORD(wParam) == ID\_BACKGROUND\_COLOR)

{

ChangeBgc(hwnd);

}

if (LOWORD(wParam) == ID\_TEXT\_COLOR)

{

ChangeTextColor(hwnd);

}

// 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;

case ID\_STYLE\_BOLD:

g\_bBold = !g\_bBold;

UpdateTextStyle(g\_hEdit);

break;

case ID\_STYLE\_ITALIC:

g\_bItalic = !g\_bItalic;

UpdateTextStyle(g\_hEdit);

break;

case ID\_STYLE\_UNDERLINE:

g\_bUnderline = !g\_bUnderline;

UpdateTextStyle(g\_hEdit);

break;

case ID\_FONT\_FACE\_ARIAL:

g\_FontFace = L"Arial";

UpdateTextStyle(g\_hEdit);

break;

case ID\_FONT\_FACE\_CALIBRI:

g\_FontFace = L"Calibri";

UpdateTextStyle(g\_hEdit);

break;

case ID\_FONT\_FACE\_TNR:

g\_FontFace = L"Times New Roman";

UpdateTextStyle(g\_hEdit);

break;

case ID\_FONT\_SIZE\_10:

g\_fontSize = 10;

UpdateTextStyle(g\_hEdit);

break;

case ID\_FONT\_SIZE\_12:

g\_fontSize = 12;

UpdateTextStyle(g\_hEdit);

break;

case ID\_FONT\_SIZE\_14:

g\_fontSize = 14;

UpdateTextStyle(g\_hEdit);

break;

case ID\_FONT\_SIZE\_16:

g\_fontSize = 16;

UpdateTextStyle(g\_hEdit);

break;

case ID\_FONT\_SIZE\_18:

g\_fontSize = 18;

UpdateTextStyle(g\_hEdit);

break;

case ID\_FONT\_SIZE\_20:

g\_fontSize = 20;

UpdateTextStyle(g\_hEdit);

break;

}

break;

}

case WM\_CTLCOLOREDIT:

{

HDC hdcEdit = (HDC)wParam;

SetTextColor(hdcEdit, textColor);

SetBkColor(hdcEdit, bgColor);

return reinterpret\_cast<LRESULT>(hEditBgBrush);

}

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);

}

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);

}

}

void UpdateTextStyle(HWND hWnd)

{

LOGFONT lf;

ZeroMemory(&lf, sizeof(LOGFONT));

lf.lfHeight = g\_fontSize; // высота шрифта

if (g\_bBold){lf.lfWeight = FW\_BOLD;}

if (g\_bItalic) {

lf.lfItalic = TRUE; // курсив

}

if (g\_bUnderline) {

lf.lfUnderline = TRUE; // подчеркивание

}

lstrcpy(lf.lfFaceName, g\_FontFace); // имя шрифта

HFONT hFont = CreateFontIndirect(&lf);

SendMessage(hWnd, WM\_SETFONT, reinterpret\_cast<WPARAM>(hFont), MAKELPARAM(TRUE, 0));

// Обновляем окно

UpdateWindow(hWnd);

}

void ChangeBgc(HWND hwnd) {

// Открываем диалоговое окно цвета

CHOOSECOLOR cc;

ZeroMemory(&cc, sizeof(CHOOSECOLOR));

static COLORREF customColors[16] = { 0 };

cc.lStructSize = sizeof(CHOOSECOLOR);

cc.hwndOwner = hwnd;

cc.rgbResult = bgColor;

cc.lpCustColors = customColors;

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

if (ChooseColor(&cc))

{

bgColor = cc.rgbResult;

hEditBgBrush = CreateSolidBrush(cc.rgbResult);

InvalidateRect(hwnd, NULL, TRUE);

}

}

void ChangeTextColor(HWND hwnd) {

// Открываем диалоговое окно цвета

CHOOSECOLOR cc;

ZeroMemory(&cc, sizeof(CHOOSECOLOR));

static COLORREF customColors[16] = { 0 };

cc.lStructSize = sizeof(CHOOSECOLOR);

cc.hwndOwner = hwnd;

cc.rgbResult = textColor;

cc.lpCustColors = customColors;

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

if (ChooseColor(&cc))

{

textColor = cc.rgbResult;

hEditTextColorBrush = CreateSolidBrush(cc.rgbResult);

InvalidateRect(hwnd, NULL, TRUE);

}

}

// Функция-перехватчик клавиатуры

LRESULT CALLBACK KeyboardProc(int nCode, WPARAM wParam, LPARAM lParam)

{

if (nCode >= 0)

{

// Извлечение информации о событии клавиатуры

KBDLLHOOKSTRUCT\* pKeyboardStruct = (KBDLLHOOKSTRUCT\*)lParam;

// Проверка типа события

if (wParam == WM\_KEYDOWN)

{

// Проверка нажатия комбинации клавиш Ctrl+N

if ((pKeyboardStruct->vkCode == 'N') && (GetAsyncKeyState(VK\_CONTROL) & 0x8000))

{

// Выполнение действия по созданию нового файла

CreateNewDocument();

}

else if ((pKeyboardStruct->vkCode == 'O') && (GetAsyncKeyState(VK\_CONTROL) & 0x8000)) {

OpenDocument();

}

else if ((pKeyboardStruct->vkCode == 'S') && (GetAsyncKeyState(VK\_CONTROL) & 0x8000)) {

SaveDocument();

}

}

}

// Передача события дальше в цепочку хуков

return CallNextHookEx(NULL, nCode, wParam, lParam);

}

**3 Полученные результаты**

На рисунке 3.1 показаны все стили примененные к тексту, включая жирный, курсив, подчеркивание текста и изменение цвета текста.

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Рисунок 3.1 – Примененные стили текста

На рисунке 3.2 продемонстрирована функция изменения фона редактора с помощью rgb инструментов.

Изображение выглядит как текст, снимок экрана, Красочность, фиолетовый

Автоматически созданное описание

Рисунок 3.2 – Изменение фона редактора

**Выводы**

В ходе данной лабораторной работы было изучено создание и использование элементов управления, обработка различных сообщений и механизм перехвата сообщений в Win API. С применением изученного материала был доработан текстовый редактор с предыдущей лабораторной.

Теперь это текстовый редактор с возможностью изменения стиля текста, цвета фона и с возможностью использования основных функций с файлами с помощью горячих клавиш через механизм перехвата сообщений.