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

по лабораторной работе №1

на тему:

Основы программирования в Win 32 API

Студент Миролюбов И.И

Принимающий Гриценко Н.Ю.

МИНСК 2023

СОДЕРЖАНИЕ

[Теоретические сведения 3](#_Toc146620203)

[Результаты работы 4](#_Toc146620204)

[Листинг кода 7](#_Toc146620205)

[Источники 14](#_Toc146620206)

# Теоретические сведения

# Win 32 API является основой программного интерфейса для написания приложений под операционные системы Microsoft Windows, начиная с Windows 95 и продолжая до современных версий, таких как Windows 10. Win32 API обеспечивает программистам доступ к различным возможностям операционной системы, таким как управление окнами, файлами, реестром, устройствами ввода-вывода, сетевыми функциями, управление памятью и многое другое. Он предоставляет низкоуровневый доступ к функциям операционной системы, что позволяет разработчикам создавать мощные и гибкие приложения с более прямым управлением ресурсами системы. Win32 API используется в различных языках программирования, таких как C++, C#, Visual Basic и других. Разработчики используют Win32 API для создания разнообразных приложений, включая графические интерфейсы пользователя, системные службы, драйверы устройств, игры и другое. Используя Win32 API, разработчики могут создавать приложения для Windows, работающие на широком спектре устройств, включая персональные компьютеры, ноутбуки, планшеты и смартфоны под управлением Windows.

В своей лабараторной работе я создал оконное приложение Win 32 с минимально достаточной функциональностью и обработкой основных оконных сообщений. Мое оконное приложение может выполнять все функции простого калькулятора: сложение, вычитание, умножение, деление.

В моей лабораторной представлен функционал создания окон и обработки основных оконных сообщений. Чтобы зарегистрировать новый класс окна, для начала нужно заполните структуру WNDCLASS. Необходимо создать следующие элементы структуры: *lpfnWndProc* — это указатель на определяемую приложением функцию, называемую процедурой окна или оконным процессом. Процедура окна определяет большую часть поведения окна. На данный момент это значение представляет собой прямое объявление функции. *hInstance* — это дескриптор экземпляра приложения. Получите это значение из параметра hInstance объекта WinMain. Операционная система использует это значение для идентификации исполняемого файла или EXE-файла при загрузке в память. Некоторым функциям Windows требуется дескриптор экземпляра, например для загрузки значков или растровых изображений. *lpszClassName* — это строка, идентифицирующая класс окна.

Затем передаем адрес структуры WNDCLASS в функцию RegisterClass . Эта функция регистрирует класс window в операционной системе. После всех этих манипуляций нам нужно создать окно.

Чтобы создать новый экземпляр окна, вызовите функцию CreateWindowEx. Параметры данной функии:

1. dwExStyle – расширенный список создаваемого окна
2. lpClassName – имя класса окна
3. lpWindowName – имя окна
4. dwStyle – стиль создаваемого окна
5. X – начальное горизонтальное положение окна
6. Y – начальное вертикальное положение окна
7. nWidth – начальная ширина в единицах устройства
8. nHeight – начальная высота в единицах устройства
9. hWndParent – родителський дескриптор файла
10. hMenu – дескриптор меню или задает идентификатор дочернего окна в зависимости от стиля окна.
11. hInst – дескриптор экземпляра модуля, связанного с окном
12. lparam – Указатель на значение, передаваемое в окно через структуру CREATESTRUCT (член lpCreateParams ), на которую указывает параметр lParam сообщения WM\_CREATE . Это сообщение отправляется в созданное окно этой функцией перед возвратом.

CreateWindowEx возвращает дескриптор новому окну или ноль в случае сбоя функции. Чтобы отобразить окно, то есть сделать окно видимым, нужно передать дескриптор окна в функцию ShowWindow.

Далее в своей лабораторной работе я использовал цикл сообщений. Во время выполнения приложение получит тысячи сообщений. Кроме того, приложение может иметь несколько окон, каждое из которых имеет собственную процедуру окна. Как программа получает все эти сообщения и доставляет их в правильную процедуру окна? Приложению требуется цикл для получения сообщений и их отправки в правильные окна.

Для каждого потока, создающего окно, операционная система создает очередь для оконных сообщений. Эта очередь содержит сообщения для всех окон, созданных в этом потоке. Сама очередь скрыта от программы. Вы не можете управлять очередью напрямую. Однако вы можете извлечь сообщение из очереди, вызвав функцию GetMessage. Эта функция удаляет первое сообщение из заголовка очереди. Если очередь пуста, функция блокируется до тех пор, пока в очередь не будет помещено другое сообщение. Тот факт, что блоки GetMessage не сделают вашу программу не отвечающей. Если сообщений нет, программе нечего делать. Первый параметр функции – адресс структуры msg. Если функция завершается успешно, она заполняет структуру MSG сведениями о сообщении. Сюда входит целевое окно и код сообщения. Другие три параметра позволяют фильтровать сообщения, которые вы получаете из очереди. Почти во всех случаях для этих параметров необходимо задать нулевое значение.

Хотя структура MSG содержит сведения о сообщении, вы почти никогда не будете изучать эту структуру напрямую. Вместо этого вы будете передавать его непосредственно в две другие функции (TranslateMessage(&msg) и DispatchMessage(&msg)). Функция TranslateMessage связана с вводом с клавиатуры. Он преобразует нажатия клавиш (клавиши вниз, клавишу вверх) в символы. Функция DispatchMessage сообщает операционной системе, что нужно вызвать процедуру окна, которое является целевым объектом сообщения. Другими словами, операционная система ищет дескриптор окна в таблице окон, находит указатель функции, связанный с окном, и вызывает функцию.

Далее функция DispatchMessage() вызывает процедуру окна, которое является целевым объектом сообщения. Процедура окна имеет следующую сигнатуру. LRESULT CALLBACK WindowProc(HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam).

1. hwnd — это дескриптор окна.
2. uMsg — это код сообщения; например, сообщение WM\_SIZE указывает, что размер окна был изменен.
3. wParam и lParam содержат дополнительные данные, относящиеся к сообщению. Точное значение зависит от кода сообщения.

LRESULT — это целочисленное значение, которое программа возвращает в Windows. Он содержит ответ программы на определенное сообщение. Значение этого значения зависит от кода сообщения. CALLBACK — это соглашение о вызовах для функции. Типичная процедура окна — это просто большой оператор switch, который переключает код сообщения.

# Результаты работы

# 

# Рисунок 1 Рисунок 2 Рисунок 3

На рисунках 1, 2 и 3 проддемонстрирована операция сложения двух чисел.

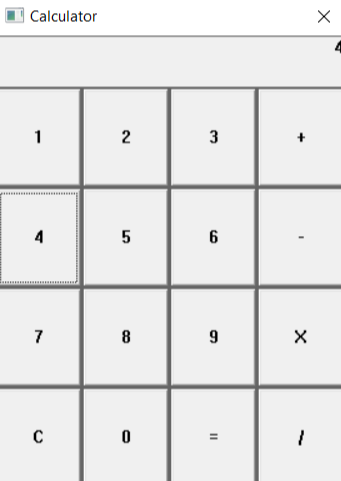
  

Рисунок 4 Рисунок 5 Рисунок 6

На рисунках 4, 5 и 6 продемонстрированна операция умножения двух чисел.

# Листинг кода

#include <Windows.h>

#include <math.h>

#include <stdlib.h>

#include <string.h>

#include <sstream>

using namespace std;

//Main Window Procedure

LRESULT CALLBACK window(HWND, UINT, WPARAM, LPARAM);

//Functions for different windows in the program

void screen(HWND);

void buttons(HWND);

//character to display numbers to the screen

char out[1000];

//Window handlers for windows in the program

HWND one, two, three, four, five, six, seven, eight, nine, zero, add, delenie, sub, mul, scr, clear, eq;

//Macros for buttons to function

#define btn\_one 1

#define btn\_two 2

#define btn\_three 3

#define btn\_four 4

#define btn\_five 5

#define btn\_six 6

#define btn\_seven 7

#define btn\_eight 8

#define btn\_nine 9

#define btn\_zero 10

#define btn\_plus 11

#define btn\_divide 12

#define btn\_times 13

#define btn\_minus 14

#define btn\_clear 15

#define btn\_equal 16

//Variable for checking operator(sign)

char sign;

//Variables for holding numbers

float firstSet = 0, Secondset = 0, Result = 0;

//Program main function where the window procedure is called

int WINAPI WinMain(HINSTANCE hinst, HINSTANCE, LPSTR args, int nCmdShow)

{

const wchar\_t name[] = L"name";

WNDCLASS wc = {};

wc.hInstance = hinst;

wc.lpszClassName = name;

wc.lpfnWndProc = window;

RegisterClass(&wc);

CreateWindowEx(0,

name,

L"Calculator",

WS\_VISIBLE | WS\_SYSMENU, CW\_USEDEFAULT, CW\_USEDEFAULT,

296,

400,

NULL,

NULL,

hinst,

NULL);

MSG msg = {};

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

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

return 0;

}

//The window procedure which is called from the main function

LRESULT CALLBACK window(HWND hwnd, UINT mas, WPARAM wp, LPARAM lp)

{

switch (mas)

{

//case to handle every command (that is every action carried out onButtonClick)

case WM\_COMMAND:

{

switch (wp)

{

//Case to handle when one is clicked

case btn\_one:

{

char one[] = "1";

strcat\_s(out, one);

SetWindowTextA(scr, out);

}

break;

//Case to handle when two is clicked

case btn\_two:

{

char one[] = "2";

strcat\_s(out, one);

SetWindowTextA(scr, out);

}

break;

//Case to handle when three is clicked

case btn\_three:

{

char one[] = "3";

strcat\_s(out, one);

SetWindowTextA(scr, out);

}

break;

//Case to handle when four is clicked

case btn\_four:

{

char one[] = "4";

strcat\_s(out, one);

SetWindowTextA(scr, out);

}

break;

//Case to handle when five is clicked

case btn\_five:

{

char one[] = "5";

strcat\_s(out, one);

SetWindowTextA(scr, out);

}

break;

//Case to handle when six is clicked

case btn\_six:

{

char one[] = "6";

strcat\_s(out, one);

SetWindowTextA(scr, out);

}

break;

//Case to handle when seven is clicked

case btn\_seven:

{

char one[] = "7";

strcat\_s(out, one);

SetWindowTextA(scr, out);

}

break;

//Case to handle when eight is clicked

case btn\_eight:

{

char one[] = "8";

strcat\_s(out, one);

SetWindowTextA(scr, out);

}

break;

//Case to handle when nine is clicked

case btn\_nine:

{

char one[] = "9";

strcat\_s(out, one);

SetWindowTextA(scr, out);

}

break;

//Case to handle when zero is clicked

case btn\_zero:

{

char const\* one = "0";

strcat\_s(out, one);

SetWindowTextA(scr, out);

}

break;

//Case to handle when the clear button(C) is clicked

case btn\_clear:

{

char clear[] = "";

strcpy\_s(out, clear);

SetWindowTextA(scr, out);

}

break;

//Case to handle when the plus button(+) is clicked

case btn\_plus:

{

sign = '+';

firstSet = stoi(out);

char clear[] = "";

strcpy\_s(out, clear);

SetWindowTextA(scr, out);

}

break;

//Case to handle when the minus button(-) is clicked

case btn\_minus:

{

sign = '-';

firstSet = stoi(out);

char clear[] = "";

strcpy\_s(out, clear);

SetWindowTextA(scr, out);

}

break;

//Case to handle when the divide button(/) is clicked

case btn\_divide:

{

sign = '/';

firstSet = stoi(out);

char clear[] = "";

strcpy\_s(out, clear);

SetWindowTextA(scr, out);

}

break;

//Case to handle when the multiplication button(X) is clicked

case btn\_times:

{

sign = '\*';

firstSet = stoi(out);

char clear[] = "";

strcpy\_s(out, clear);

SetWindowTextA(scr, out);

}

break;

//Case to handle when the equal button(=) is clicked

case btn\_equal:

{

Secondset = stoi(out);

char clear[] = "";

//This statement is carried out when the plus button is clicked

if (sign == '+')

{

Result = firstSet + Secondset;

string show = to\_string(Result);

char const\* chars = show.c\_str();

strcpy\_s(out, chars);

SetWindowTextA(scr, out);

}

//This statement is carried out when the minus button is clicked

if (sign == '-')

{

Result = firstSet - Secondset;

string show = to\_string(Result);

char const\* chars = show.c\_str();

strcpy\_s(out, chars);

SetWindowTextA(scr, out);

}

//This statement is carried out when the multiplication button is clicked

if (sign == '\*')

{

Result = firstSet \* Secondset;

string show = to\_string(Result);

char const\* chars = show.c\_str();

strcpy\_s(out, chars);

SetWindowTextA(scr, out);

}

//This statement is carried out when the division button is clicked

if (sign == '/')

{

//This statement is carried out whenever the division by zero is tried

if (Secondset == 0)

{

MessageBox(NULL, L"ERROR: Cannot divide by zero", L"ERROR: ZERO DIVISION", MB\_OK);

strcpy\_s(out, clear);

SetWindowTextA(scr, out);

}

else {

Result = firstSet / Secondset;

Result = ("%.2f", Result);

string show = to\_string(Result);

char const\* chars = show.c\_str();

strcpy\_s(out, chars);

SetWindowTextA(scr, out);

}

}

}

}

break;

}

//Case to handle the creation of the controls on the GUI

case WM\_CREATE:

screen(hwnd);

buttons(hwnd);

break;

//Case to handle the destruction(that is, closing) of the GUI

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hwnd, mas, wp, lp);

}

return 0;

}

//Function called in the window procedure to help create the static control(screen) on the GUI

void screen(HWND srn\_handler)

{

scr = CreateWindowEx(0, L"STATIC", NULL, WS\_CHILDWINDOW | WS\_VISIBLE | WS\_BORDER | SS\_RIGHT, 0, 0, 280, 42, srn\_handler, NULL, NULL, NULL);

}

//Function called in the window procedure to help create the buttons on the GUI

void buttons(HWND btn\_handler)

{

//Buttons for 1, 2, 3, +

one = CreateWindowEx(0, L"BUTTON", L"1", WS\_CHILD | WS\_VISIBLE | WS\_BORDER, 0, 42, 70, 80, btn\_handler, (HMENU)btn\_one, NULL, NULL);

two = CreateWindowEx(0, L"BUTTON", L"2", WS\_CHILD | WS\_VISIBLE | WS\_BORDER, 70, 42, 70, 80, btn\_handler, (HMENU)btn\_two, NULL, NULL);

three = CreateWindowEx(0, L"BUTTON", L"3", WS\_CHILD | WS\_VISIBLE | WS\_BORDER, 140, 42, 70, 80, btn\_handler, (HMENU)btn\_three, NULL, NULL);

add = CreateWindowEx(0, L"BUTTON", L"+", WS\_CHILD | WS\_VISIBLE | WS\_BORDER, 210, 42, 70, 80, btn\_handler, (HMENU)btn\_plus, NULL, NULL);

//Buttons for 4, 5, 6, -

four = CreateWindowEx(0, L"BUTTON", L"4", WS\_CHILD | WS\_VISIBLE | WS\_BORDER, 0, 122, 70, 80, btn\_handler, (HMENU)btn\_four, NULL, NULL);

five = CreateWindowEx(0, L"BUTTON", L"5", WS\_CHILD | WS\_VISIBLE | WS\_BORDER, 70, 122, 70, 80, btn\_handler, (HMENU)btn\_five, NULL, NULL);

six = CreateWindowEx(0, L"BUTTON", L"6", WS\_CHILD | WS\_VISIBLE | WS\_BORDER, 140, 122, 70, 80, btn\_handler, (HMENU)btn\_six, NULL, NULL);

sub = CreateWindowEx(0, L"BUTTON", L"-", WS\_CHILD | WS\_VISIBLE | WS\_BORDER, 210, 122, 70, 80, btn\_handler, (HMENU)btn\_minus, NULL, NULL);

//Buttons for 7, 8, 9, X

seven = CreateWindowEx(0, L"BUTTON", L"7", WS\_CHILD | WS\_VISIBLE | WS\_BORDER, 0, 202, 70, 80, btn\_handler, (HMENU)btn\_seven, NULL, NULL);

eight = CreateWindowEx(0, L"BUTTON", L"8", WS\_CHILD | WS\_VISIBLE | WS\_BORDER, 70, 202, 70, 80, btn\_handler, (HMENU)btn\_eight, NULL, NULL);

nine = CreateWindowEx(0, L"BUTTON", L"9", WS\_CHILD | WS\_VISIBLE | WS\_BORDER, 140, 202, 70, 80, btn\_handler, (HMENU)btn\_nine, NULL, NULL);

mul = CreateWindowEx(0, L"BUTTON", L"X", WS\_CHILD | WS\_VISIBLE | WS\_BORDER, 210, 202, 70, 80, btn\_handler, (HMENU)btn\_times, NULL, NULL);

//Buttons for C, 0, =, /.

clear = CreateWindowEx(0, L"BUTTON", L"C", WS\_CHILD | WS\_VISIBLE | WS\_BORDER, 0, 282, 70, 80, btn\_handler, (HMENU)btn\_clear, NULL, NULL);

zero = CreateWindowEx(0, L"BUTTON", L"0", WS\_CHILD | WS\_VISIBLE | WS\_BORDER, 70, 282, 70, 80, btn\_handler, (HMENU)btn\_zero, NULL, NULL);

eq = CreateWindowEx(0, L"BUTTON", L"=", WS\_CHILD | WS\_VISIBLE | WS\_BORDER, 140, 282, 70, 80, btn\_handler, (HMENU)btn\_equal, NULL, NULL);

delenie = CreateWindowEx(0, L"BUTTON", L"/", WS\_CHILD | WS\_VISIBLE | WS\_BORDER, 210, 282, 70, 80, btn\_handler, (HMENU)btn\_divide, NULL, NULL);

# }

# Источники

1. <https://learn.microsoft.com/en-us/windows/win32/winprog/windows-data-types>
2. <https://learn.microsoft.com/ru-ru/windows/win32/learnwin32/creating-a-window>
3. <https://learn.microsoft.com/ru-ru/windows/win32/api/winuser/nf-winuser-createwindowexa>
4. <https://learn.microsoft.com/ru-ru/windows/win32/winmsg/window-styles>
5. https://learn.microsoft.com/ru-ru/windows/win32/learnwin32/writing-the-window-procedure