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

Студент Н. С. Шмидт
Преподаватель Н. Ю. Гриценко

Минск 2023

СОДЕРЖАНИЕ

1 Цель работы..... 3
2 Теоретические сведения..... 4
3 Результат выполнения..... 5
Заключение..... 7
Список использованных источников..... 8
Приложение А (обязательное) Листинг кода..... 9

1 ЦЕЛЬ РАБОТЫ

Цель работы изучить и применить на практике знания о расширенном использовании Win32 и GDI для создания приложений на Windows, научиться формировать сложные изображения, обрабатывать различные сообщения, а также изучить механизм перехвата сообщений с использованием winhook. Для достижения цели будет создано приложение, которое позволяет пользователю рисовать и редактировать графические фигуры (круги, прямоугольники, треугольники, ромбы) с помощью мыши и клавиш клавиатуры с возможностью изменения цвета последней нарисованной фигуры.

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Интерфейс графических устройств GDI операционной системы Microsoft Windows, как это можно предположить из названия, предназначен для взаимодействия приложений Windows с графическими устройствами, такими как видеомонитор, принтер или плоттер.

С точки зрения приложения GDI – это контекст отображения и инструменты для рисования. Контекст отображения можно сравнить с листом бумаги, на котором приложение рисует то или иное графическое изображение, а также пишет текст. Инструменты для рисования – это перья, кисти (а также шрифты и даже целые графические изображения), с помощью которых создается изображение. Кроме контекста отображения и инструментов для рисования, приложениям доступны десятки функций программного интерфейса GDI, предназначенные для работы с контекстом отображения и инструментами.

Если говорить более точно, контекст отображения является структурой данных, описывающей устройство отображения. В этой структуре хранятся различные характеристики устройства и набор инструментов для рисования, выбранный по умолчанию. Приложение может выбирать в контекст отображения различные инструменты (например, перья различной толщины и цвета). Поэтому если нужно нарисовать линию красного или зеленого цвета, перед выполнением операции следует выбрать в контекст отображения соответствующее перо.

Функции рисования не имеют параметров, указывающих цвет или толщину линии. Такие параметры хранятся в контексте отображения.

Приложение может создать контекст отображения не только для экрана монитора или окна, но и для любого другого графического устройства вывода, например, для принтера.

Операционная система Windows предоставляет программисту функции, позволяющие установить в приложении требуемое количество программных таймеров. С помощью таймеров приложение, например, может обеспечить временную синхронизацию и задание временных интервалов. Функция SetTimer() создает или видоизменяет системный таймер, который формирует сообщение WM_TIMER.

Winhook – это механизм, позволяющий перехватывать и обрабатывать различные события в операционной системе Windows. Он может использоваться для мониторинга активности пользователя, анализа данных, защиты от вредоносного ПО и других целей.

3 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ

Было создано приложение позволяющее рисовать фигуры(круги, прямоугольники, треугольники, ромбы). Для изменения типа фигуры используются клавиши numpad: numpad1 - прямоугольник, numpad2 - элипс, numpad3 - треугольник, numpad4 - ромб (рисунок 1).

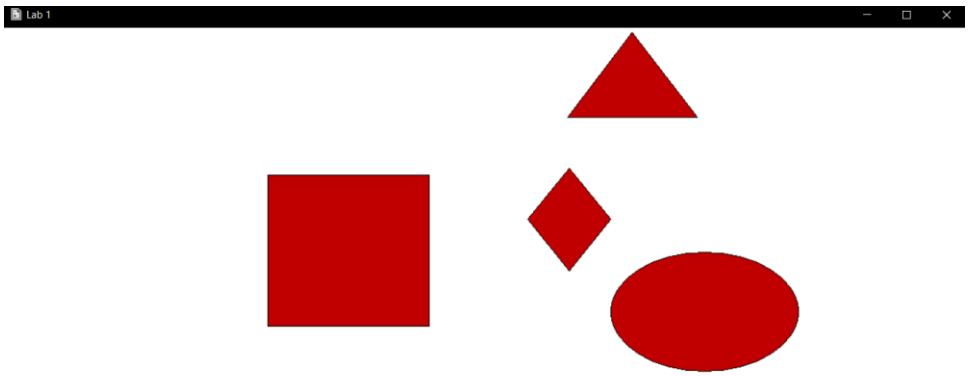

Рисунок 1 – Окно приложения

Для перехода в режим редактирования фигур используется клавиша *VK_SHIFT* и затем требуется выбрать нужную фигуру с помощью ЛКМ. Для изменения положения фигур используется обработка нажатия клавиш *VK_LEFT*, *VK_RIGHT*, *VK_UP* и *VK_DOWN* (рисунок 2).

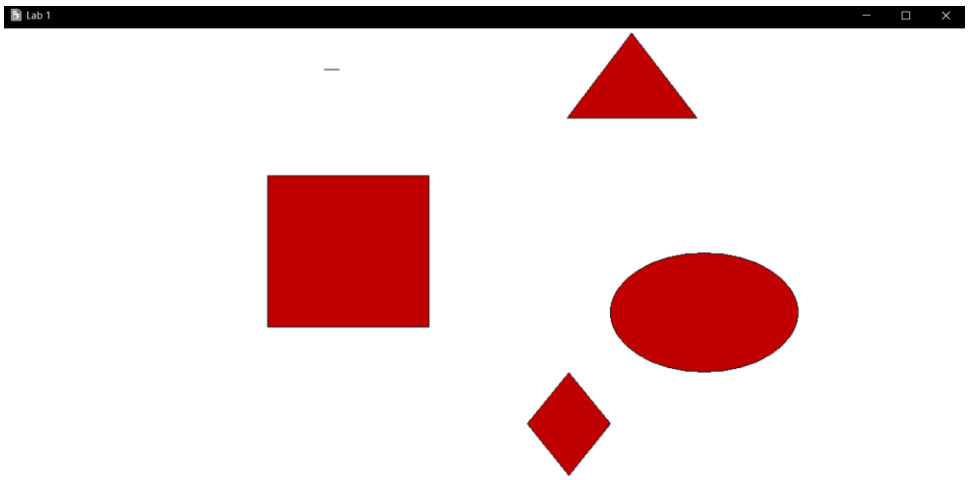

Рисунок 2 – Изменения положения ромба с помощью клавиш

Так же в этот момент появляется возможность изменить размер фигуры. Для этого необходимо нажать кнопки плюса или минуса на верхней панели клавиатуры (рисунок 3).

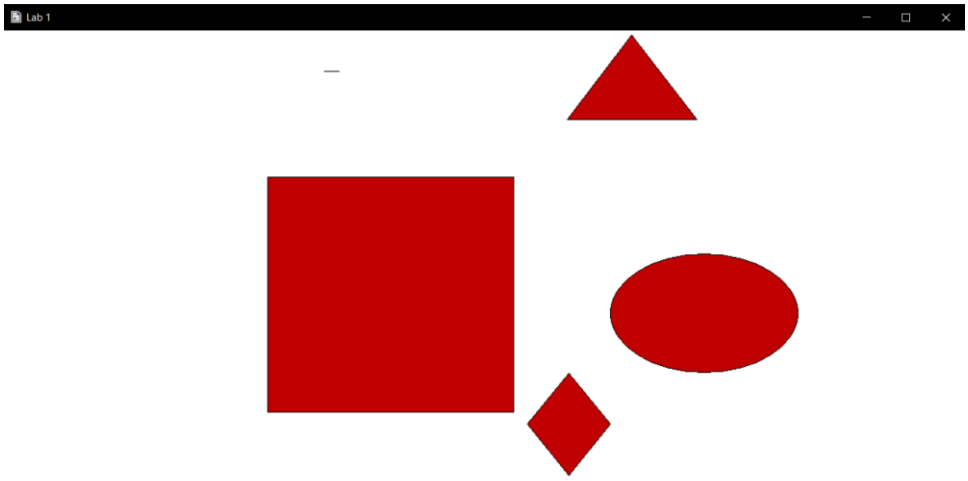

Рисунок 3 – Изменения размера прямоугольника с помощью клавиш

Для изменения цвета последней фигуры был реализован набор кнопок, каждая из которых отвечает за свой цвет. При нажатии на кнопку цвет последней нарисованной фигуры будет изменен на выбранный (рисунок 4).

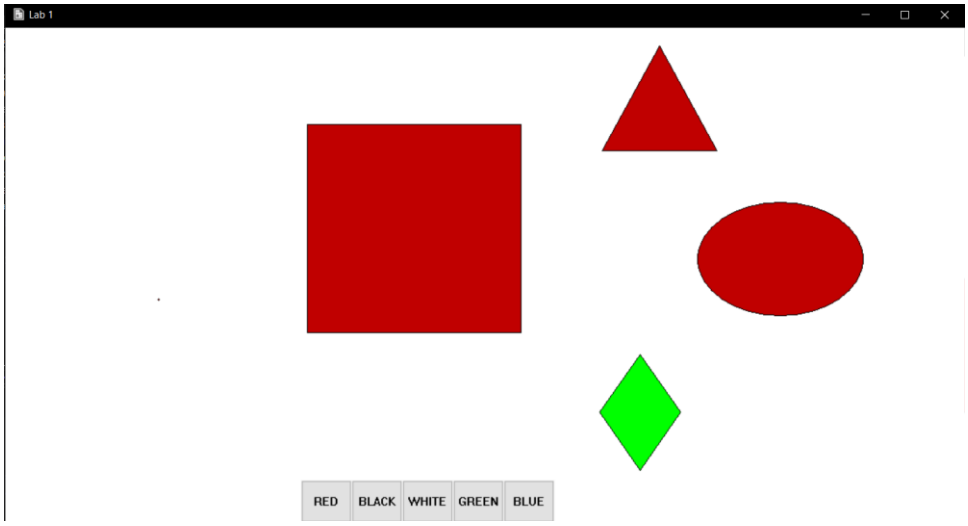

Рисунок 4 – Изменения цвета ромба с помощью кнопки Green

ЗАКЛЮЧЕНИЕ

В результате лабораторной работы были изучены основные принципы работы с GDI: работа с контекстом отображения, использование кистей и цветов, рисование геометрических фигур. Было создано приложение, которое позволяет пользователю рисовать и редактировать графические фигуры (круги, прямоугольники, треугольники, ромбы) с помощью мыши и клавиш клавиатуры с возможностью изменения цвета последней нарисованной фигуры.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Основы программирования для Win32 API [Электронный ресурс]. – Режим доступа: https://dims.karelia.ru/win32/.

[2] Основные сообщения ОС Windows (Win32 API). Программирование в ОС Windows. Лекция 1. – Электронные данные. – Режим доступа: https://www.youtube.com/watch?v=wTArIolxch0

[3] КАК рисовать в Win32 API? [Электронный ресурс]. – Электронные данные. – Режим доступа: https://radiofront.narod.ru/htm/prog/htm/winda/api/paint.html#s

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода

Листинг 1 – Файл main.cpp

```
#ifndef UNICODE
#define UNICODE
#endif

#include <windows.h>
#include <tchar.h>
#include <vector>
#include "resource.h"

#pragma comment(linker,"\"/manifestdependency:type='win32' \
name='Microsoft.Windows.Common-Controls' version='6.0.0.0' \
processorArchitecture='*' publicKeyToken='6595b64144ccf1df' language='*'\"")
      // updated styles

RECT rc = { 0 };
int WindowPosX = 0;
int WindowPosY = 0;

int lastShapeId = 0;

class Shape {
public:
int shapeId;
int x1, y1, x2, y2;
int shapeType;
COLORREF color;

Shape() {}

Shape(int _shapeType)
{
shapeId = ++lastShapeId;
shapeType = _shapeType;
}

void checkCoord()
{
if (this->shapeType != 2) {
if (this->x1 > this->x2)
{
int buff = this->x1;
this->x1 = this->x2;
this->x2 = buff;
}
if (this->y1 > this->y2)
{
int buff = this->y1;
this->y1 = this->y2;
this->y2 = buff;
}
}
}
};
```

```
int choosenShapeType = 0;
std::vector<Shape*> shapes;
Shape* currentShape = nullptr;
bool isDrawing = false;
bool isEditing = false;

int selectedShapeIndex = -1;
HBRUSH brush = CreateSolidBrush(RGB(192, 0, 0));
HBRUSH original = CreateSolidBrush(RGB(192, 0, 0));
COLORREF red = RGB(255, 0, 0);
COLORREF white = RGB(255, 255, 255);
COLORREF black = RGB(0, 0, 0);
COLORREF green = RGB(0, 255, 0);
COLORREF blue = RGB(0, 0, 255);
LOGBRUSH br = { 0 };


void DrawShape(HDC hdc, int shapeType, int x1, int y1, int x2, int y2)
{
if (shapeType == 0)
{
SelectObject(hdc, brush);
Rectangle(hdc, x1, y1, x2, y2);
}
else if (shapeType == 1)
{
SelectObject(hdc, brush);
Ellipse(hdc, x1, y1, x2, y2);
}
else if (shapeType == 2)
{
SelectObject(hdc, brush);
POINT vertices[] = { {x1, y1},  {x2, y1}, {(x2 - x1) / 2 + x1, y2} };
Polygon(hdc, vertices, sizeof(vertices) / sizeof(vertices[0]));
}
else
{
SelectObject(hdc, brush);
POINT vertices[] = { {(x2 - x1) / 2 + x1, y1},  {x2, (y2 - y1) /2 + y1}, (x2
- x1) / 2 + x1, y2, { x1, (y2 - y1) / 2 + y1}};
Polygon(hdc, vertices, sizeof(vertices) / sizeof(vertices[0]));
}

}

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

int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, PWSTR
pCmdLine, int nCmdShow)
{
const wchar_t CLASS_NAME[] = L"Sample Window Class";

WNDCLASS wc = { };

HICON hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_ICON1));

wc.lpfnWndProc = WindowProc;
wc.hInstance = hInstance;
```

```
wc.lpszClassName = CLASS_NAME;
wc.style = CS_HREDRAW | CS_VREDRAW;
wc.hIcon = hIcon;

RegisterClass(&wc);


HWND hwnd = CreateWindowEx(
0,
CLASS_NAME,
L"Lab 1",
WS_OVERLAPPEDWINDOW,


CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT, CW_USEDEFAULT,

NULL,
NULL,
hInstance,
NULL
);

if (hwnd == NULL)
{
return 0;
}

ShowWindow(hwnd, nCmdShow);



MSG msg = { };
while (GetMessage(&msg, NULL, 0, 0) > 0)
{
TranslateMessage(&msg);
DispatchMessage(&msg);
}

return 0;
}

LRESULT CALLBACK WindowProc(HWND hwnd, UINT uMsg, WPARAM wParam, LPARAM
lParam)
{
switch (uMsg)
{
case WM_CREATE:
{
HWND hBtn1 = CreateWindow(
L"BUTTON",
L"RED",
WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
350, 535, 60, 50,
hwnd, reinterpret_cast<HMENU>(1), NULL, NULL
);
HWND hBtn2 = CreateWindow(
L"BUTTON",
L"BLACK",
WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
410, 535, 60, 50,
```

```
hwnd, reinterpret_cast<HMENU>(2), NULL, NULL
);
HWND hBtn3 = CreateWindow(
L"BUTTON",
L"WHITE",
WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
470, 535, 60, 50,
hwnd, reinterpret_cast<HMENU>(3), NULL, NULL
);
HWND hBtn4 = CreateWindow(
L"BUTTON",
L"GREEN",
WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
530, 535, 60, 50,
hwnd, reinterpret_cast<HMENU>(4), NULL, NULL
);
HWND hBtn5 = CreateWindow(
L"BUTTON",
L"BLUE",
WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON,
590, 535, 60, 50,
hwnd, reinterpret_cast<HMENU>(5), NULL, NULL
);
break;
}
case WM_COMMAND:
{
switch (LOWORD(wParam))
{
case 1:
currentShape->color = red;
SetFocus(hwnd);
return 0;

case 2:
currentShape->color = black;
SetFocus(hwnd);
return 0;

case 3:
currentShape->color = white;
SetFocus(hwnd);
return 0;

case 4:
currentShape->color = green;
SetFocus(hwnd);
return 0;

case 5:
currentShape->color = blue;
SetFocus(hwnd);
return 0;
}

break;
}
case WM_DESTROY:
shapes.clear();
PostQuitMessage(0);
return 0;
```

```
case WM_PAINT:
{
PAINTSTRUCT ps;
HDC hdc = BeginPaint(hwnd, &ps);

FillRect(hdc, &ps.rcPaint, (HBRUSH)(COLOR_WINDOW + 1));

for (int i = 0; i < shapes.size(); i++)
{
Shape shape = *shapes[i];
brush = CreateSolidBrush(shape.color);
DrawShape(hdc, shape.shapeType, shape.x1, shape.y1, shape.x2, shape.y2);
}
brush = original;
if (currentShape != nullptr)
DrawShape(hdc, currentShape->shapeType, currentShape->x1, currentShape->y1,
currentShape->x2, currentShape->y2);

EndPaint(hwnd, &ps);
}
return 0;

case WM_LBUTTONDOWN:
if (!isEditing)
{
isDrawing = true;
currentShape = new Shape(choosenShapeType);
currentShape->x1 = currentShape->x2 = LOWORD(lParam);
currentShape->y1 = currentShape->y2 = HIWORD(lParam);
}
else if (isEditing)
{
if (selectedShapeIndex == -1)
{
for (int i = shapes.size() - 1; i >= 0; i--)
{
Shape shape = *shapes[i];
int x = LOWORD(lParam);
int y = HIWORD(lParam);
if (shape.x1 <= x && shape.x2 >= x && shape.y1 <= y && shape.y2 >= y)
{
selectedShapeIndex = i;
break;
}
}
}
}
break;

case WM_MOUSEMOVE:
if (isDrawing)
{
currentShape->x2 = LOWORD(lParam);
currentShape->y2 = HIWORD(lParam);
InvalidateRect(hwnd, 0, TRUE);
}
break;

case WM_LBUTTONUP:
if (isDrawing)
{
```

```
isDrawing = false;
GetObject(brush, sizeof(br), &br);
currentShape->color = br.lbColor;
currentShape->checkCoord();
shapes.push_back(currentShape);
}

case WM_SIZE:
rc.right = LOWORD(lParam);
rc.bottom = HIWORD(lParam);
break;

case WM_MOVE:
WindowPosX = (int)(short)LOWORD(lParam);   // horizontal position
WindowPosY = (int)(short)HIWORD(lParam);   // vertical position
InvalidateRect(hwnd, 0, TRUE);                          // update window after
moving
break;

case WM_KEYDOWN:
if (wParam == VK_ESCAPE)
PostMessage(hwnd, WM_DESTROY, 0, 0);
else if (wParam == VK_OEM_PLUS) {
if (isEditing && selectedShapeIndex != -1) {
shapes[selectedShapeIndex]->x2 += 20;
shapes[selectedShapeIndex]->y2 += 20;
}
InvalidateRect(hwnd, 0, TRUE);
}
else if (wParam == VK_OEM_MINUS) {
if (isEditing && selectedShapeIndex != -1) {
shapes[selectedShapeIndex]->x2 -= 20;
shapes[selectedShapeIndex]->y2 -= 20;
}
InvalidateRect(hwnd, 0, TRUE);
}
else if (wParam == VK_NUMPAD1)
{
choosenShapeType = 0;
}
else if (wParam == VK_NUMPAD2)
{
choosenShapeType = 1;
}
else if (wParam == VK_NUMPAD3)
{
choosenShapeType = 2;
}
else if (wParam == VK_NUMPAD4)
{
choosenShapeType = 3;
}
else if (wParam == VK_SHIFT)
{
if (!isEditing)
{
if (isDrawing)
{
isDrawing = false;
shapes.push_back(currentShape);
}
isEditing = true;
```

```
}
else
{
isEditing = false;
selectedShapeIndex = -1;
}
}
else if (wParam == VK_LEFT)
{
if (isEditing && selectedShapeIndex != -1) {
shapes[selectedShapeIndex]->x1 -= 5;
shapes[selectedShapeIndex]->x2 -= 5;
}
InvalidateRect(hwnd, 0, TRUE);
}
else if (wParam == VK_RIGHT)
{
if (isEditing && selectedShapeIndex != -1) {
shapes[selectedShapeIndex]->x1 += 5;
shapes[selectedShapeIndex]->x2 += 5;
}
InvalidateRect(hwnd, 0, TRUE);
}
else if (wParam == VK_UP)
{
if (isEditing && selectedShapeIndex != -1)
if (isEditing && selectedShapeIndex != -1) {
shapes[selectedShapeIndex]->y1 -= 5;
shapes[selectedShapeIndex]->y2 -= 5;
}
InvalidateRect(hwnd, 0, TRUE);
}
else if (wParam == VK_DOWN)
{
if (isEditing && selectedShapeIndex != -1) {
shapes[selectedShapeIndex]->y1 += 5;
shapes[selectedShapeIndex]->y2 += 5;
}
InvalidateRect(hwnd, 0, TRUE);
}
break;

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