Міністерство освіти і науки України Харківський національний університет радіоелектроніки

Кафедра Інформатики

Звіт

з Лабораторної роботи №4

з дисципліни

«Операційні системи»

Виконав: Перевірив:

ITIHФ-20-1 Професор

Самченко С.О. Сінельнікова Т.Ф.

Харків – 2022

**Графіка**

Необхідно розробити програму client, яка дозволятиме відправляти команди на сервер по локальній мережі по протоколу TCP / IP. У сервері для обробки кожної команди створюється власний потік. Клієнтська програма повинна дозволяти відправити команду малювання кола, прямокутника, текстового повідомлення.Кожен об'єкт в клієнтському додатку задається набором параметрів, в тому числі колір і градієнтна заливка. Відправка команди виконується по натисканні на певну клавішу. Малювання об'єктів у серверному додатку виконується на певній робочій області розміром 640х480 пікселів. Повинні відбуватися перевірки коректності заданих параметрів в командах, щоб можна було забезпечити виведення об'єктів. Малювання об'єктів повинно виконуватися за допомогою GDI +. Дані про кількість намальованих об'єктів кожного типу необхідно зберегти в реєстрі. Після першого запуску перша програма повинна поміщати іконку в Windows Tray і при натисканні комбінації клавіш Ctrl + P має виводитися вікно з інформацією про кількість намальованих об'єктів кожного типу, а при натисканні на комбінацію клавіш Alt + T ці дані повинні видалятися з реєстру.

**framework.h:**

#pragma once

#include "targetver.h"

#define WIN32\_LEAN\_AND\_

#include <windows.h>

#include <stdlib.h>

#include <malloc.h>

#include <memory.h>

#include <tchar.h>

**Hook.h:**

#pragma once

#include "resource.h"

**Resource.h:**

//{{NO\_DEPENDENCIES}}

// Включаемый файл, созданный в Microsoft Visual C++.

// Используется Hook.rc

#define IDS\_APP\_TITLE 103

#define IDR\_MAINFRAME 128

#define IDD\_HOOK\_DIALOG 102

#define IDD\_ABOUTBOX 103

#define IDM\_ABOUT 104

#define IDM\_EXIT 105

#define IDI\_HOOK 107

#define IDI\_SMALL 108

#define IDC\_HOOK 109

#define IDC\_MYICON 2

#ifndef IDC\_STATIC

#define IDC\_STATIC -1

#endif

// Следующие стандартные значения для новых объектов

//

#ifdef APSTUDIO\_INVOKED

#ifndef APSTUDIO\_READONLY\_SYMBOLS

#define \_APS\_NO\_MFC 130

#define \_APS\_NEXT\_RESOURCE\_VALUE 129

#define \_APS\_NEXT\_COMMAND\_VALUE 32771

#define \_APS\_NEXT\_CONTROL\_VALUE 1000

#define \_APS\_NEXT\_SYMED\_VALUE 110

#endif

#endif

**targetver.h:**

#pragma once

#include <SDKDDKVer.h>

**Graphics.cpp:**

// Graphics.cpp : Определяет точку входа для приложения.

//

#include "framework.h"

#include "Graphics.h"

#include <gdiplus.h>

#include <string>

#define MAX\_LOADSTRING 100

#pragma comment(lib, "gdiplus.lib")

using namespace Gdiplus;

// Глобальные переменные:

HINSTANCE hInst; // текущий экземпляр

WCHAR szTitle[MAX\_LOADSTRING]; // Текст строки заголовка

WCHAR szWindowClass[MAX\_LOADSTRING]; // имя класса главного окна

HHOOK mouse\_hook;

HWND myHwnd;

Point click\_pointL(0, 0);

Point click\_pointR(0, 0);

// Отправить объявления функций, включенных в этот модуль кода:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

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

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

void InitHooks();

LRESULT CALLBACK Mouse\_Hook(int code, WPARAM wParam, LPARAM lParam);

void draw(HDC hdc);

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// TODO: Разместите код здесь.

GdiplusStartupInput gdiplusStartupToken;

ULONG\_PTR gdiplusToken;

Status st = GdiplusStartup(&gdiplusToken, &gdiplusStartupToken, nullptr);

if (st != Status::Ok)

MessageBoxEx(NULL, L"Can't Load GDI+", L"Error", MB\_OK | MB\_ICONERROR, LANG\_ENGLISH);

InitHooks();

// Инициализация глобальных строк

LoadStringW(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadStringW(hInstance, IDC\_GRAPHICS, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Выполнить инициализацию приложения:

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_GRAPHICS));

MSG msg;

// Цикл основного сообщения:

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

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int) msg.wParam;

}

//

// ФУНКЦИЯ: MyRegisterClass()

//

// ЦЕЛЬ: Регистрирует класс окна.

//

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEXW wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

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

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_GRAPHICS));

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

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+1);

wcex.lpszMenuName = MAKEINTRESOURCEW(IDC\_GRAPHICS);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));

return RegisterClassExW(&wcex);

}

//

// ФУНКЦИЯ: InitInstance(HINSTANCE, int)

//

// ЦЕЛЬ: Сохраняет маркер экземпляра и создает главное окно

//

// КОММЕНТАРИИ:

//

// В этой функции маркер экземпляра сохраняется в глобальной переменной, а также

// создается и выводится главное окно программы.

//

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

hInst = hInstance; // Сохранить маркер экземпляра в глобальной переменной

HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

//

// ФУНКЦИЯ: WndProc(HWND, UINT, WPARAM, LPARAM)

//

// ЦЕЛЬ: Обрабатывает сообщения в главном окне.

//

// WM\_COMMAND - обработать меню приложения

// WM\_PAINT - Отрисовка главного окна

// WM\_DESTROY - отправить сообщение о выходе и вернуться

//

//

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

{

int xPos, yPos;

switch (message)

{

case WM\_LBUTTONDOWN:

{

// Сохраняем координаты курсора мыши

xPos = LOWORD(lParam);

yPos = HIWORD(lParam);

click\_pointL.X = xPos;

click\_pointL.Y = yPos;

}

break;

case WM\_RBUTTONDOWN:

{

// Сохраняем координаты курсора мыши

xPos = LOWORD(lParam);

yPos = HIWORD(lParam);

click\_pointR.X = xPos;

click\_pointR.Y = yPos;

}

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

// Разобрать выбор в меню:

switch (wmId)

{

case IDM\_ABOUT:

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

default:

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

}

}

break;

case WM\_PAINT:

{

PAINTSTRUCT ps;

HDC hdc = BeginPaint(hWnd, &ps);

// TODO: Добавьте сюда любой код прорисовки, использующий HDC...

draw(hdc);

EndPaint(hWnd, &ps);

}

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

// Обработчик сообщений для окна "О программе".

INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

UNREFERENCED\_PARAMETER(lParam);

switch (message)

{

case WM\_INITDIALOG:

return (INT\_PTR)TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)

{

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

break;

}

return (INT\_PTR)FALSE;

}

void InitHooks()

{

mouse\_hook = SetWindowsHookEx(WH\_MOUSE\_LL, Mouse\_Hook, GetModuleHandle(NULL), 0);

if (!mouse\_hook)

{

MessageBox(NULL, L"Ошибка при создании хука клавиатуры.", L"Title", NULL);

}

}

LRESULT CALLBACK Mouse\_Hook(int code, WPARAM wParam, LPARAM lParam)

{

LPMSLLHOOKSTRUCT m = (LPMSLLHOOKSTRUCT)lParam;

switch (wParam)

{

case WM\_LBUTTONUP:

{

InvalidateRect(myHwnd, NULL, TRUE);

}

break;

case WM\_RBUTTONUP:

{

InvalidateRect(myHwnd, NULL, TRUE);

}

break;

}

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

}

void draw(HDC hdc)

{

Graphics gf(hdc);

SolidBrush brushL(Color(255, 200, 200, 200));

SolidBrush brushR(Color(255, 200, 120, 80));

gf.FillEllipse(&brushL, click\_pointL.X - 10, click\_pointL.Y - 10, 20, 20);

gf.FillRectangle(&brushR, click\_pointR.X - 10, click\_pointR.Y - 10, 20, 20);

std::string str = std::to\_string(click\_pointL.X) + " " + std::to\_string(click\_pointL.Y);

std::wstring wide\_string = std::wstring(str.begin(), str.end());

const wchar\_t\* result = wide\_string.c\_str();

Font font(&FontFamily(L"Arial"), 12);

gf.DrawString(result, -1, &font, PointF(20, 20), &brushL);

str = std::to\_string(click\_pointR.X) + " " + std::to\_string(click\_pointR.Y);

wide\_string = std::wstring(str.begin(), str.end());

result = wide\_string.c\_str();

gf.DrawString(result, -1, &font, PointF(20, 40), &brushR);

}