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

к лабораторной работе №8

на тему

ИНТЕРФЕЙС СОКЕТОВ И ОСНОВЫ СЕТЕВОГО ПРОГРАММИРОВАНИЯ (WINDOWS).

ПРОГРАММИРОВАНИЕ ВЗАИМОДЕЙСТВИЯ ЧЕРЕЗ СЕТЬ С ИСПОЛЬЗОВАНИЕМ ИНТЕРФЕЙСА

СОКЕТОВ. РЕАЛИЗАЦИЯ СЕТЕВЫХ ПРОТОКОЛОВ: СОБСТВЕННЫХ ИЛИ СТАНДАРТНЫХ.

Студент: гр.153502 Матвеев Н.С.

Руководитель: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

1 Формулировка задачи 3

2 Теоретические сведения 4

3 Описание функций программы 5

Список использованных источников 6

Приложение А (обязательное) Исходный код программы 7

**1 ФОРМУЛИРОВКА ЗАДАЧИ**

Целью выполнения лабораторной работы является разработка клиент-серверного приложение для обмена текстовыми сообщениями с использованием *TCP* сокетов.

**2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Клиент-серверная архитектура — наиболее распространенная структура приложений в Интернете. В этой архитектуре клиенты (т.е. персональные компьютеры, устройства Интернета вещей и т. д.) сначала запрашивают ресурсы с сервера. Затем сервер отправляет обратно соответствующие ответы на запросы клиентов [1].

TCP - это надежная служба доставки потока, гарантирующая доставку потока данных из одного хоста в другой без дублирования и потерь данных. Так как передача пакетов не является надежной, то применяется метод подтверждения приема с повторной передачей, гарантирующий надежную доставку пакетов. Для этого базового метода требуется, чтобы приемник отправлял сообщение с подтверждением при приеме данных.

Отправитель хранит запись каждого отправленного пакета и ожидает подтверждение перед отправкой следующего пакета. Также отправитель хранит таймер для каждого отправленного пакета и отправляет пакет повторно по истечении заданного времени. Таймер необходим в случае потери или повреждения пакета [2].

**3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ**

Для обмена сообщениями между клиентами и сервером используются *TCP*-сокеты. Для использования протокола *TCP* необходимо передать в метод *getaddrinfo* объект *addrinfo*, поле *ai*\_*protocol* которого содержит значение *IPPROTO*\_*TCP*.

Клиент посылает сообщение на сервер, после чего оно отправляется всем клиентам, подключенным на данный момент к серверу. Все полученные сообщения отображаются с помощью статического элемента управления для текста.

Графический интерфейс программы представлен на рисунке 1.

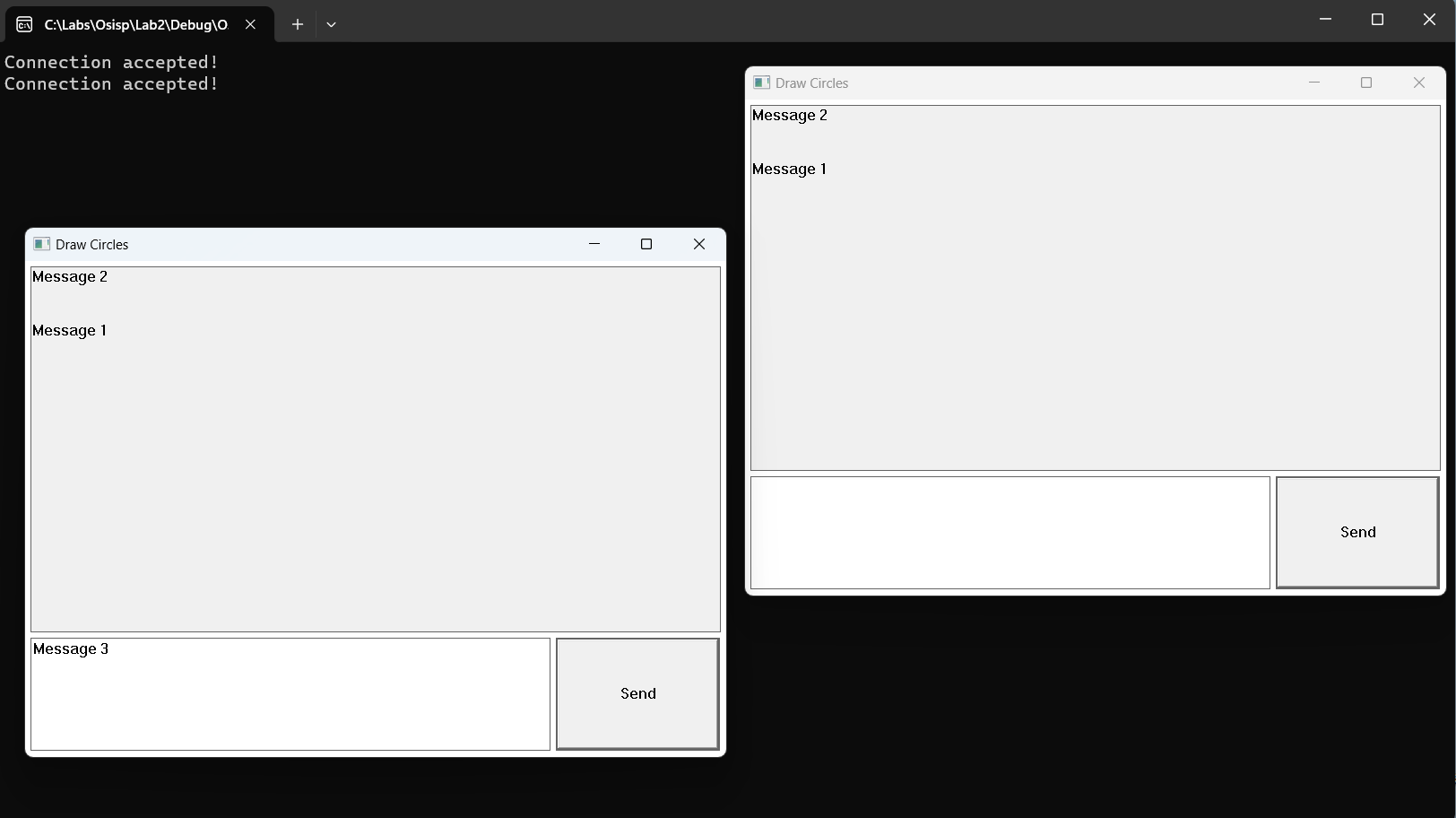


Рисунок 1 – Графический интерфейс программы

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Основы программирования TCP-Сокетов на Java [Электронный ресурс]. – Режим доступа: https://medium.com/nuances-of-programming/основы-программирования-tcp-сокетов-на-java-594b6519d494.

[2] Обзор сокетов – IBM Documentation [Электронный ресурс]. – https://www.ibm.com/docs/ru/rtw/9.0.0?topic=transports-sockets-overview.

**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Исходный код программы**

**Файл BaseWindow.h**

*#pragma once*

*template <class DERIVED\_TYPE>*

*class BaseWindow*

*{*

*public:*

*const PCWSTR CLASS\_NAME;*

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

*{*

*DERIVED\_TYPE \*pThis = NULL;*

*if (uMsg == WM\_NCCREATE)*

*{*

*CREATESTRUCT\* pCreate = (CREATESTRUCT\*)lParam;*

*pThis = (DERIVED\_TYPE\*)pCreate->lpCreateParams;*

*SetWindowLongPtr(hwnd, GWLP\_USERDATA, (LONG\_PTR)pThis);*

*pThis->m\_hwnd = hwnd;*

*}*

*else*

*{*

*pThis = (DERIVED\_TYPE\*)GetWindowLongPtr(hwnd, GWLP\_USERDATA);*

*}*

*if (pThis)*

*{*

*return pThis->HandleMessage(uMsg, wParam, lParam);*

*}*

*else*

*{*

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

*}*

*}*

*BaseWindow(PCWSTR CLASS\_NAME) : CLASS\_NAME(CLASS\_NAME), m\_hwnd(NULL) { }*

*BOOL Create(*

*PCWSTR lpWindowName,*

*DWORD dwStyle,*

*HWND hWndParent = 0,*

*DWORD dwExStyle = 0,*

*int x = CW\_USEDEFAULT,*

*int y = CW\_USEDEFAULT,*

*int nWidth = CW\_USEDEFAULT,*

*int nHeight = CW\_USEDEFAULT,*

*HMENU hMenu = 0*

*)*

*{*

*WNDCLASS wc = {};*

*wc.lpfnWndProc = DERIVED\_TYPE::WindowProc;*

*wc.hInstance = GetModuleHandle(NULL);*

*wc.lpszClassName = CLASS\_NAME;*

*RegisterClass(&wc);*

*m\_hwnd = CreateWindowEx(*

*dwExStyle, CLASS\_NAME, lpWindowName, dwStyle, x, y,*

*nWidth, nHeight, hWndParent, hMenu, GetModuleHandle(NULL), this*

*);*

*return (m\_hwnd ? TRUE : FALSE);*

*}*

*HWND Window() const { return m\_hwnd; }*

*protected:*

*virtual LRESULT HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam) = 0;*

*HWND m\_hwnd;*

*};*

**Файл DPIScale.h**

*#pragma once*

*#include <d2d1.h>*

*class DPIScale*

*{*

*static float scaleX;*

*static float scaleY;*

*public:*

*static void Initialize()*

*{*

*FLOAT dpi = GetDpiForSystem();*

*scaleX = dpi / 96.0f;*

*scaleY = dpi / 96.0f;*

*}*

*template <typename T>*

*static float PixelsToDipsX(T x)*

*{*

*return static\_cast<float>(x) / scaleX;*

*}*

*template <typename T>*

*static float PixelsToDipsY(T y)*

*{*

*return static\_cast<float>(y) / scaleY;*

*}*

*template <typename T>*

*static T DipXToPixels(float x)*

*{*

*return static\_cast<T>(x) \* scaleX;*

*}*

*template <typename T>*

*static T DipYToPixels(float y)*

*{*

*return static\_cast<T>(y) \* scaleY;*

*}*

*};*

**Файл DPIScale.cpp**

*#include "DPIScale.h"*

*float DPIScale::scaleX = 1;*

*float DPIScale::scaleY = 1;*

**Файл main.cpp (клиент)**

*#include <windows.h>*

*#include <Windowsx.h>*

*#include <d2d1.h>*

*#include <Winuser.h>*

*#include <list>*

*#include <memory>*

*using namespace std;*

*#pragma comment(lib, "d2d1")*

*#pragma comment(lib, "Ntdll")*

*#pragma comment(lib, "Comctl32")*

*#pragma comment(lib, "Ws2\_32.lib")*

*#include "BaseWindow.h"*

*#include "MainWindow.h"*

*#include <objidl.h>*

*#include <gdiplus.h>*

*#include <winternl.h>*

*#include <CommCtrl.h>*

*int WINAPI wWinMain(HINSTANCE hInstance, HINSTANCE, PWSTR, int nCmdShow)*

*{*

*//Gdiplus::GdiplusStartupInput gdiplusStartupInput;*

*//ULONG\_PTR gdiplusToken;*

*//Gdiplus::GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);*

*MainWindow win = MainWindow();*

*//Mode mode = Mode::SelectMode;*

*//Figure figure = Figure::Ellipse;*

*//D2D1\_COLOR\_F color = D2D1::ColorF(D2D1::ColorF::Black);*

*//SceneControl win = SceneControl(&mode, &figure, &color);*

*//GraphicsScene win = GraphicsScene(&mode, &figure, &color);*

*if (!win.Create(L"Draw Circles",*

*WS\_OVERLAPPEDWINDOW,*

*NULL,*

*NULL,*

*CW\_USEDEFAULT,*

*CW\_USEDEFAULT,*

*800,*

*600))*

*{*

*return 0;*

*}*

*//SetWindowsHookEx(WH\_GETMESSAGE, GetMsgProcCustom, NULL, GetCurrentThreadId());*

*//HACCEL hAccel1 = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDR\_ACCEL1));*

*//HACCEL hAccel2 = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDR\_ACCEL2));*

*ShowWindow(win.Window(), nCmdShow);*

*MSG msg;*

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

*{*

*//if (!TranslateAccelerator(GetFocus(), hAccel1, &msg) && !TranslateAccelerator(GetFocus(), hAccel2, &msg))*

*//{*

*TranslateMessage(&msg);*

*DispatchMessage(&msg);*

*//}*

*}*

*return 0;*

*}*

**Файл main.cpp (сервер)**

*#ifndef WIN32\_LEAN\_AND\_MEAN*

*#define WIN32\_LEAN\_AND\_MEAN*

*#endif*

*#include <windows.h>*

*#include <winsock2.h>*

*#include <ws2tcpip.h>*

*#include <iphlpapi.h>*

*#include <stdio.h>*

*#include <set>*

*#include <iostream>*

*#pragma comment(lib, "Ws2\_32.lib")*

*const LPCWSTR PORT = L"27015";*

*std::set<SOCKET> clients;*

*const size\_t BUFF\_SIZE = 16384;*

*HANDLE ghMutex;*

*DWORD WINAPI ClientThread(LPVOID lpParameter)*

*{*

*SOCKET\* clientSocket = (SOCKET\*)lpParameter;*

*wchar\_t\* buff = new wchar\_t[BUFF\_SIZE];*

*int recvStatus = 0;*

*while((recvStatus = recv(\*clientSocket, (char\*)buff, BUFF\_SIZE \* sizeof(\*buff), 0)) > 0)*

*{*

*WaitForSingleObject(ghMutex, INFINITE);*

*auto it = clients.begin();*

*while (it != clients.end())*

*{*

*if (send(\*it, (char\*)buff, recvStatus, 0) == SOCKET\_ERROR)*

*{*

*closesocket(\*it);*

*it = clients.erase(it);*

*}*

*else*

*{*

*it++;*

*}*

*}*

*ReleaseMutex(ghMutex);*

*}*

*closesocket(\*clientSocket);*

*delete clientSocket;*

*delete[] buff;*

*return 0;*

*}*

*int main() {*

*wchar\_t\* buff = new wchar\_t[BUFF\_SIZE];*

*ghMutex = CreateMutex(NULL, FALSE, NULL);*

*WSADATA wsaData;*

*if (!WSAStartup(MAKEWORD(2, 2), &wsaData))*

*{*

*ADDRINFOW\* result = NULL;*

*ADDRINFOW pHints = ADDRINFOW();*

*pHints.ai\_family = AF\_INET;*

*pHints.ai\_socktype = SOCK\_STREAM;*

*pHints.ai\_protocol = IPPROTO\_TCP;*

*pHints.ai\_flags = AI\_PASSIVE;*

*if (!GetAddrInfoW(NULL, PORT, &pHints, &result))*

*{*

*SOCKET listenSocket = socket(result->ai\_family, result->ai\_socktype, result->ai\_protocol);*

*if (listenSocket != INVALID\_SOCKET)*

*{*

*if (bind(listenSocket, result->ai\_addr, result->ai\_addrlen) != SOCKET\_ERROR)*

*{*

*while (listen(listenSocket, SOMAXCONN) != SOCKET\_ERROR)*

*{*

*SOCKET\* clientSocket = new SOCKET;*

*\*clientSocket = accept(listenSocket, NULL, NULL);*

*if (\*clientSocket != INVALID\_SOCKET)*

*{*

*std::cout << "Connection accepted!" << std::endl;*

*clients.insert(\*clientSocket);*

*CreateThread(NULL, NULL, &ClientThread, (void\*)clientSocket, NULL, NULL);*

*}*

*}*

*}*

*closesocket(listenSocket);*

*}*

*FreeAddrInfoW(result);*

*}*

*WSACleanup();*

*}*

*CloseHandle(ghMutex);*

*return 0;*

*}*

**Файл MainWindow.h**

*#include <string>*

*#include <map>*

*#include "BaseWindow.h"*

*#include "DPIScale.h"*

*class MainWindow : public BaseWindow<MainWindow>*

*{*

*private:*

*static const PCWSTR DEFAULT\_CLASS\_NAME;*

*public:*

*MainWindow(PCWSTR CLASS\_NAME = DEFAULT\_CLASS\_NAME);*

*~MainWindow();*

*virtual LRESULT HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam) override;*

*protected:*

*void CreateLayout();*

*void SetLayout();*

*};*

**Файл MainWindow.cpp**

*#ifndef WIN32\_LEAN\_AND\_MEAN*

*#define WIN32\_LEAN\_AND\_MEAN*

*#endif*

*#include <windows.h>*

*#include <winsock2.h>*

*#include <ws2tcpip.h>*

*#include <iphlpapi.h>*

*#include "MainWindow.h"*

*#include <stdexcept>*

*#include <CommCtrl.h>*

*const PCWSTR MainWindow::DEFAULT\_CLASS\_NAME = L"Graphics";*

*const float MARGIN\_X = 6.0F;*

*const float MARGIN\_Y = 6.0F;*

*const int BUFF\_SIZE = 16384;*

*wchar\_t BUFF[BUFF\_SIZE];*

*SOCKET connectSocket;*

*LPCWSTR PORT = L"27015";*

*LPCWSTR ADDRESS = L"localhost";*

*HANDLE thread;*

*std::wstring currentChat;*

*HWND chat;*

*HWND sendButton;*

*HWND input;*

*MainWindow::MainWindow(PCWSTR CLASS\_NAME) :*

*BaseWindow<MainWindow>(CLASS\_NAME)*

*{*

*}*

*MainWindow::~MainWindow()*

*{*

*}*

*DWORD WINAPI ServerThread(LPVOID lpParameter)*

*{*

*SOCKET\* clientSocket = (SOCKET\*)lpParameter;*

*wchar\_t\* buff = new wchar\_t[BUFF\_SIZE];*

*int recvStatus = 0;*

*while((recvStatus = recv(connectSocket, (char\*)buff, BUFF\_SIZE \* sizeof(\*buff), 0)) > 0)*

*{*

*currentChat = std::wstring(buff) + L"\n\n\n" + currentChat;*

*currentChat.resize(BUFF\_SIZE);*

*SetWindowTextW(chat, currentChat.c\_str());*

*}*

*closesocket(connectSocket);*

*delete[] buff;*

*return 0;*

*}*

*DWORD SetupConnection()*

*{*

*WSADATA wsaData;*

*if (!WSAStartup(MAKEWORD(2, 2), &wsaData))*

*{*

*ADDRINFOW\* result = NULL;*

*ADDRINFOW pHints = ADDRINFOW();*

*pHints.ai\_family = AF\_INET;*

*pHints.ai\_socktype = SOCK\_STREAM;*

*pHints.ai\_protocol = IPPROTO\_TCP;*

*if (!GetAddrInfoW(NULL, PORT, &pHints, &result))*

*{*

*connectSocket = socket(result->ai\_family, result->ai\_socktype, result->ai\_protocol);*

*if (connectSocket != INVALID\_SOCKET)*

*{*

*if (connect(connectSocket, result->ai\_addr, result->ai\_addrlen) != SOCKET\_ERROR)*

*{*

*FreeAddrInfoW(result);*

*thread = CreateThread(NULL, NULL, &ServerThread, NULL, NULL, NULL);*

*return 0;*

*}*

*closesocket(connectSocket);*

*}*

*FreeAddrInfoW(result);*

*}*

*WSACleanup();*

*}*

*return 1;*

*}*

*void ReleaseConnection()*

*{*

*closesocket(connectSocket);*

*WSACleanup();*

*}*

*LRESULT MainWindow::HandleMessage(UINT uMsg, WPARAM wParam, LPARAM lParam)*

*{*

*switch (uMsg)*

*{*

*case WM\_CREATE:*

*CreateLayout();*

*while(SetupConnection());*

*return 0;*

*case WM\_PAINT:*

*{*

*PAINTSTRUCT ps;*

*HDC hdc = BeginPaint(m\_hwnd, &ps);*

*FillRect(hdc, &ps.rcPaint, (HBRUSH)(COLOR\_WINDOW + 1));*

*EndPaint(m\_hwnd, &ps);*

*return 0;*

*}*

*case WM\_INITDIALOG:*

*InitCommonControls();*

*case WM\_SIZE:*

*SetLayout();*

*return 0;*

*case WM\_CLOSE:*

*ReleaseConnection();*

*DestroyWindow(m\_hwnd);*

*return 0;*

*case WM\_DESTROY:*

*PostQuitMessage(0);*

*return 0;*

*/\*case WM\_SIZING:*

*{*

*RECT\* dragRc = (RECT\*)lParam;*

*RECT rcWindow;*

*GetWindowRect(m\_hwnd, &rcWindow);*

*int WINDOW\_HEIGHT\_PIX = rcWindow.bottom - rcWindow.top;*

*int WINDOW\_WIDTH\_PIX = rcWindow.right - rcWindow.left;*

*switch (wParam)*

*{*

*case WMSZ\_BOTTOM:*

*dragRc->bottom = dragRc->top + WINDOW\_HEIGHT\_PIX;*

*break;*

*case WMSZ\_BOTTOMLEFT:*

*dragRc->bottom = dragRc->top + WINDOW\_HEIGHT\_PIX;*

*dragRc->left = dragRc->right - WINDOW\_WIDTH\_PIX;*

*break;*

*case WMSZ\_LEFT:*

*dragRc->left = dragRc->right - WINDOW\_WIDTH\_PIX;*

*break;*

*case WMSZ\_TOPLEFT:*

*dragRc->left = dragRc->right - WINDOW\_WIDTH\_PIX;*

*dragRc->top = dragRc->bottom - WINDOW\_HEIGHT\_PIX;*

*break;*

*case WMSZ\_TOP:*

*dragRc->top = dragRc->bottom - WINDOW\_HEIGHT\_PIX;*

*break;*

*case WMSZ\_TOPRIGHT:*

*dragRc->top = dragRc->bottom - WINDOW\_HEIGHT\_PIX;*

*dragRc->right = dragRc->left + WINDOW\_WIDTH\_PIX;*

*break;*

*case WMSZ\_RIGHT:*

*dragRc->right = dragRc->left + WINDOW\_WIDTH\_PIX;*

*break;*

*case WMSZ\_BOTTOMRIGHT:*

*dragRc->right = dragRc->left + WINDOW\_WIDTH\_PIX;*

*dragRc->bottom = dragRc->top + WINDOW\_HEIGHT\_PIX;*

*break;*

*}*

*return TRUE;*

*}\*/*

*/\*case WM\_NOTIFY:*

*{*

*NMHDR\* notification = (NMHDR\*)lParam;*

*switch (((NMHDR\*)lParam)->code)*

*{*

*case TVN\_SELCHANGED:*

*selected = (BaseNode\*)((LPNMTREEVIEW)lParam)->itemNew.lParam;*

*update();*

*return 0;*

*case TVN\_ITEMEXPANDED:*

*{*

*NMTREEVIEW\* item = (NMTREEVIEW\*)(lParam);*

*KeyNode\* key = (KeyNode\*)(item)->itemNew.lParam;*

*if (KeyNode::PREDEFINED\_NAMES.find(key->getName()) == KeyNode::PREDEFINED\_NAMES.end())*

*key->initialize(1);*

*break;*

*}*

*}*

*break;*

*}\*/*

*case WM\_COMMAND:*

*if (HIWORD(wParam) == BN\_CLICKED)*

*{*

*if ((HWND)lParam == sendButton)*

*{*

*GetWindowTextW(input, BUFF, BUFF\_SIZE);*

*send(connectSocket, (char\*)BUFF, BUFF\_SIZE \* sizeof(\*BUFF), NULL);*

*SetWindowTextW(input, NULL);*

*}*

*return 0;*

*}*

*break;*

*/\*case WM\_CTLCOLORSTATIC:*

*{*

*HDC hdcStatic = (HDC)wParam;*

*SetTextColor(hdcStatic, RGB(0, 0, 0));*

*SetBkColor(hdcStatic, RGB(230, 2, 2));*

*return (INT\_PTR)hBrush;*

*}\*/*

*}*

*return DefWindowProc(m\_hwnd, uMsg, wParam, lParam);*

*}*

*void MainWindow::CreateLayout()*

*{*

*input = CreateWindow(L"EDIT",*

*NULL,*

*WS\_VISIBLE | WS\_CHILD | WS\_BORDER,*

*0,*

*0,*

*0,*

*0,*

*m\_hwnd,*

*NULL,*

*GetModuleHandle(NULL),*

*NULL);*

*sendButton = CreateWindow(L"BUTTON",*

*L"Send",*

*WS\_TABSTOP | WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON | WS\_BORDER,*

*0,*

*0,*

*0,*

*0,*

*m\_hwnd,*

*NULL,*

*GetModuleHandle(NULL),*

*NULL);*

*chat = CreateWindow(L"STATIC",*

*NULL,*

*WS\_VISIBLE | WS\_CHILD | SS\_LEFT | WS\_BORDER,*

*0,*

*0,*

*0,*

*0,*

*m\_hwnd,*

*NULL,*

*GetModuleHandle(NULL),*

*NULL);*

*}*

*void MainWindow::SetLayout()*

*{*

*RECT rcClient;*

*GetClientRect(m\_hwnd, &rcClient);*

*int MARGIN\_XPix = DPIScale::DipXToPixels<int>(MARGIN\_X);*

*int MARGIN\_YPix = DPIScale::DipYToPixels<int>(MARGIN\_Y);*

*int WINDOW\_HEIGHT\_PIX = rcClient.bottom - rcClient.top;*

*int WINDOW\_WIDTH\_PIX = rcClient.right - rcClient.left;*

*MoveWindow(chat,*

*MARGIN\_XPix,*

*MARGIN\_YPix,*

*WINDOW\_WIDTH\_PIX - 2 \* MARGIN\_XPix,*

*WINDOW\_HEIGHT\_PIX \* 3 / 4 - MARGIN\_YPix,*

*FALSE);*

*MoveWindow(input,*

*MARGIN\_XPix,*

*MARGIN\_YPix + WINDOW\_HEIGHT\_PIX \* 3 / 4,*

*WINDOW\_WIDTH\_PIX \* 3 / 4 - MARGIN\_XPix,*

*WINDOW\_HEIGHT\_PIX \* 1 / 4 - 2 \* MARGIN\_YPix,*

*FALSE);*

*MoveWindow(sendButton,*

*MARGIN\_XPix + WINDOW\_WIDTH\_PIX \* 3 / 4,*

*MARGIN\_YPix + WINDOW\_HEIGHT\_PIX \* 3 / 4,*

*WINDOW\_WIDTH\_PIX \* 1 / 4 - 2 \* MARGIN\_XPix,*

*WINDOW\_HEIGHT\_PIX \* 1 / 4 - 2 \* MARGIN\_YPix,*

*FALSE);*

*InvalidateRect(m\_hwnd, NULL, FALSE);*

*}*