|  |
| --- |
| НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  МОСКОВСКИЙ ЭНЕРГЕТИЧЕСКИЙ ИНСТИТУТ  **КАФЕДРА МАТЕМАТИЧЕСКОГО И КОМПЬЮТЕРНОГО МОДЕЛИРОВАНИЯ** |

**ЛАБОРАТОРНАЯ РАБОТА №1**

по предмету

«Сетевые технологии»

Работа выполнена:

Симаков А.М.

А-16-20

Преподаватель:

Князев А.В.

**Постановка задачи**

Разработать клиент-серверную систему, реализующую заданный протокол

обмена данными.

Программа должна быть разработана в среде Visual Studio на платформе

Win32.

Сервер и клиент должны иметь удобный интерфейс (меню, окна и т.д.).

Сервер должен вести подсчёт числа соединений.

Настройки сервера должны включать задание прослушиваемого порта.

Настройки клиента должны включать задание удалённого адреса и порта.

Клиент и сервер должны реализовать дополнительные настройки, обеспечи-

вающие гибкость выполнения соответствующего задания.

Индивидуальное:

Работа с таблицей.

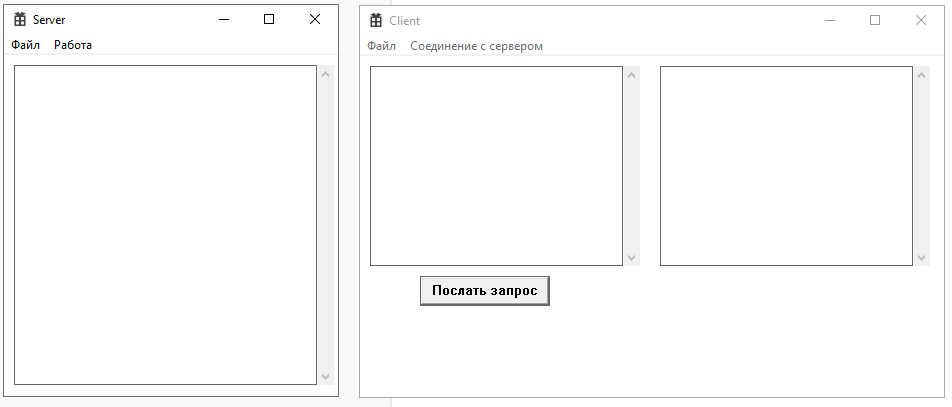
Клиент посылает запрос на создание таблицы заданных размеров. Сервер воз-

вращает подтверждение. Затем клиент посылает информацию для заполнения

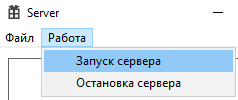
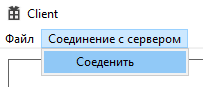
таблицы. Север возвращает образ таблицы.

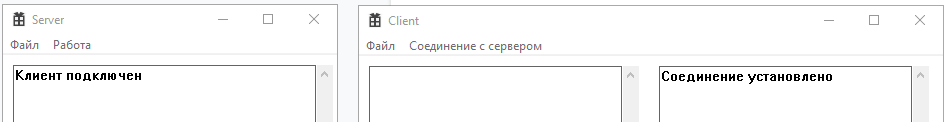
Описание работы программы

Запускаем оба приложения

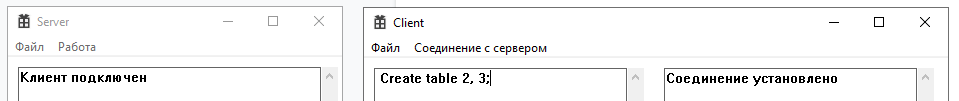


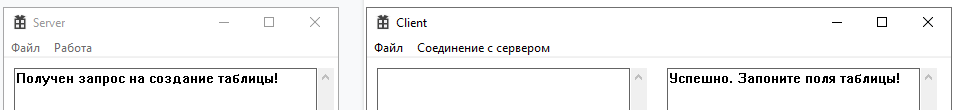
Запускаем сервер и подключаем клиента.

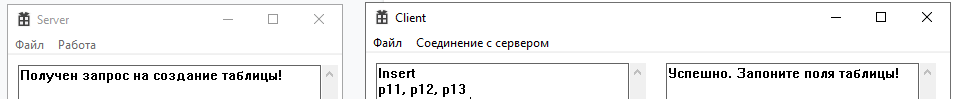


Пишем запрос на создание таблицы желаемых размеров.

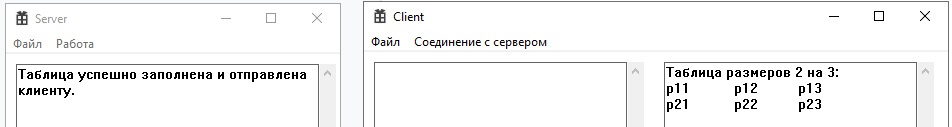
и отправляем



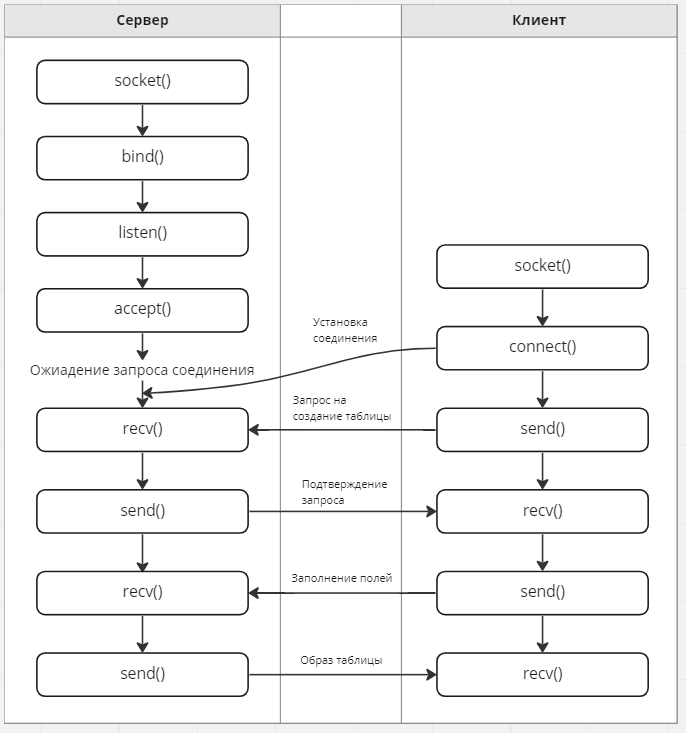
затем заполняем таблицу.



отправляем и получаем образ таблицы.

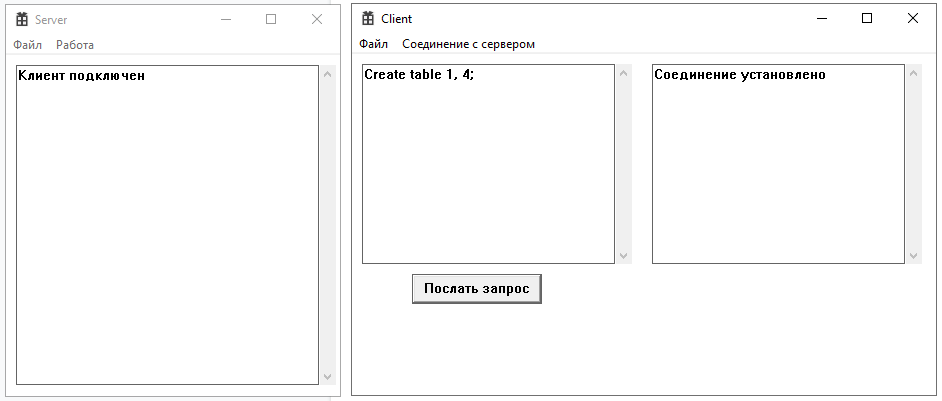


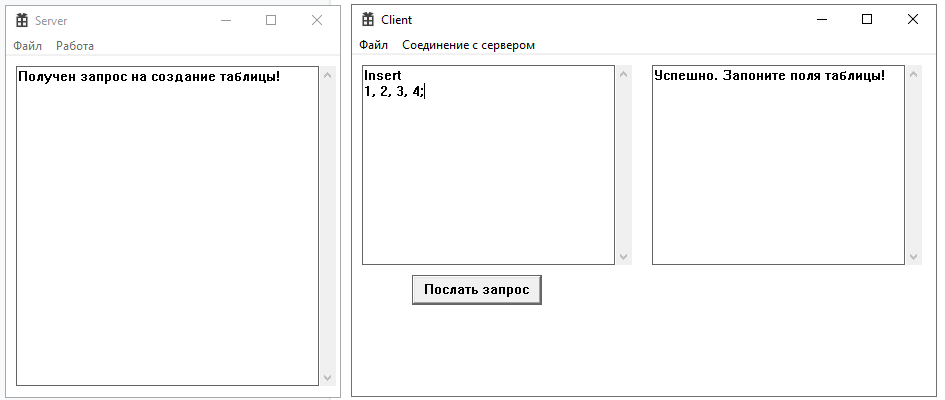
**Схема работы клиента и сервера**

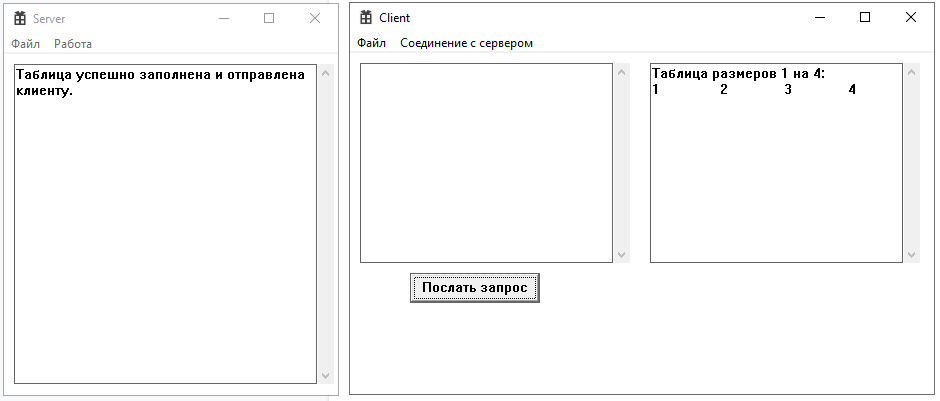


**Тесты**

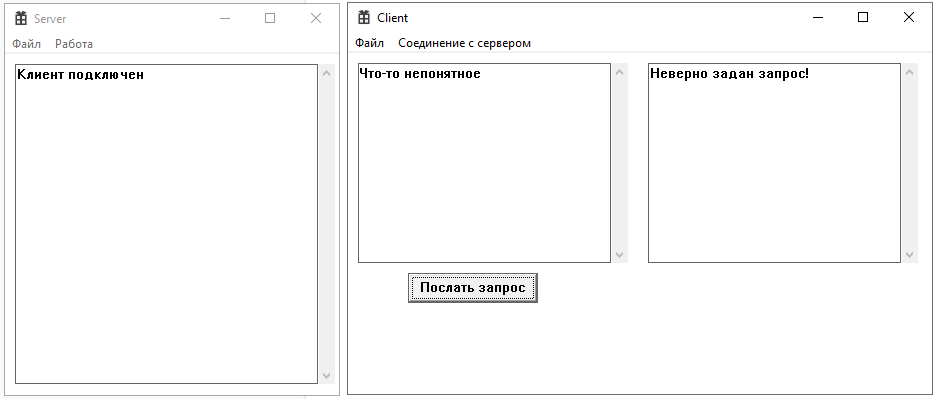
Обычная работа программы





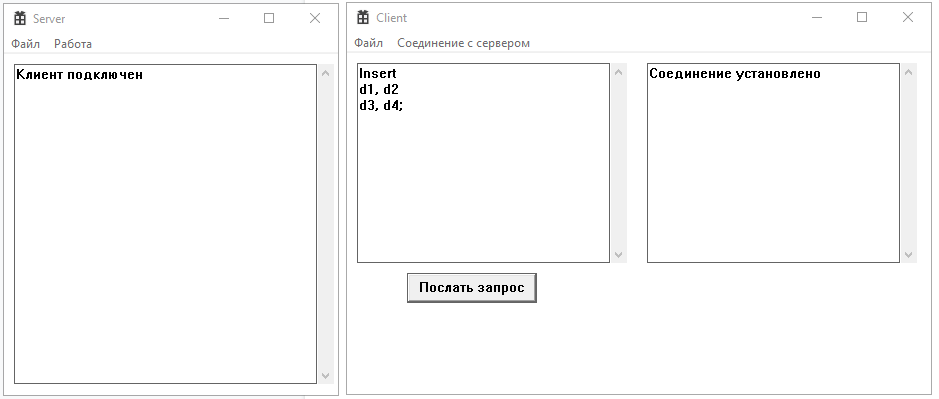


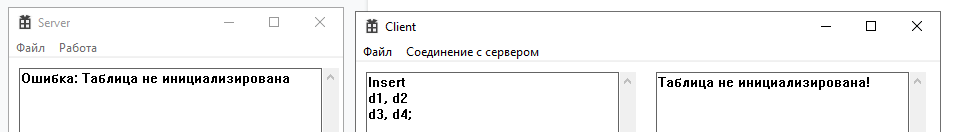
Неверный запрос



либо

попытка вставить в таблицу значения без ее объявления





**Листинг программы**

Клиент

#include "framework.h"

#include "Client.h"

#include <winsock2.h>

#include "stdio.h"

#include <regex>

#include <string>

#pragma comment(lib, "ws2\_32.lib")

#pragma warning(disable: 4996)

#define MAX\_LOADSTRING 100

#define newline L"\r\n"

#define SERV\_PORT 5000

HWND ClientTextEditBox;

HWND ServerTextEditBox;

HWND ButtonSendSequest;

SOCKET server\_socket; // Сокет клиента

SOCKADDR\_IN local\_sin; // Локальный сокет

SOCKADDR\_IN dest\_sin; // Адрес сервера

. . .

BOOL OnCreate();

void SetConnection(HWND hWnd);

BOOL SendRequest();

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{. . .}

ATOM MyRegisterClass(HINSTANCE hInstance)

{ . . . }

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

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

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

550, 150, 600, 400, nullptr, nullptr, hInstance, nullptr);

ClientTextEditBox = CreateWindow(L"Edit", L"", WS\_VISIBLE | WS\_CHILD | WS\_VSCROLL | WS\_BORDER | ES\_MULTILINE | ES\_AUTOVSCROLL,

10, 10, 270, 200, hWnd, NULL, (HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE), NULL);

ServerTextEditBox = CreateWindow(L"Edit", L"", WS\_VISIBLE | WS\_CHILD | WS\_VSCROLL | WS\_BORDER | ES\_MULTILINE | ES\_AUTOVSCROLL,

300, 10, 270, 200, hWnd, NULL, (HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE), NULL);

ButtonSendSequest = CreateWindowA("BUTTON", "Послать запрос", WS\_VISIBLE | WS\_CHILD | BS\_DEFPUSHBUTTON,

60, 220, 130, 30, hWnd, (HMENU)IDB\_BUTTON\_SEND\_REQUEST, (HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE), NULL);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

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

{

switch (message)

{

case WM\_CREATE:

OnCreate();

return 0;

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

switch (wmId)

{

case IDM\_CONNECT:

SetConnection(hWnd);

break;

case IDB\_BUTTON\_SEND\_REQUEST:

SendRequest();

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

default:

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

}

}

break;

case WM\_PAINT:

{ . . . }

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

BOOL OnCreate()

{

int rc;

WSADATA WSAData;

rc = WSAStartup(MAKEWORD(1, 1), &WSAData);

if (rc != 0)

{

MessageBox(NULL, L"WSAStartup Error", L"Error", MB\_OK);

return FALSE;

}

return TRUE;

}

void SetConnection(HWND hWnd)

{

PHOSTENT phe;

/\* Создание сокета \*/

server\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (server\_socket == INVALID\_SOCKET)

{

MessageBox(NULL, L"Socket Error", L"Error", MB\_OK);

return;

}

/\* Установка адреса IP и номера порта \*/

dest\_sin.sin\_family = AF\_INET;

phe = gethostbyname("localhost");

if (phe == NULL)

{

closesocket(server\_socket);

MessageBox(NULL, L"Gethostbyname Error", L"Error", MB\_OK);

return;

}

memcpy((char FAR\*) & (dest\_sin.sin\_addr), phe->h\_addr, phe->h\_length);

dest\_sin.sin\_port = htons(SERV\_PORT);

/\* Установка соединения \*/

if (connect(server\_socket, (PSOCKADDR)&dest\_sin, sizeof(dest\_sin)) < 0)

{

closesocket(server\_socket);

MessageBox(NULL, L"Connect Error", L"Error", MB\_OK);

return;

}

SetWindowText(ServerTextEditBox, L"Соединение установлено");

}

BOOL SendRequest()

{

int rows\_num;

std::string text;

int columns\_num;

std::wstring output;

auto buffer\_size = GetWindowTextLength(ClientTextEditBox);

wchar\_t\* buffer = new wchar\_t[buffer\_size];

GetWindowText(ClientTextEditBox, buffer, buffer\_size + 1);

char\* send\_buffer = new char[buffer\_size];

wcstombs(send\_buffer, buffer, buffer\_size);

send(server\_socket, send\_buffer, buffer\_size, 0);

std::string t;

char buf[100];

recv(server\_socket, buf, 100, 0);

switch (buf[0])

{

case '0':

SetWindowText(ServerTextEditBox, L"Успешно. Запоните поля таблицы!");

SetWindowText(ClientTextEditBox, L"");

break;

case '1':

SetWindowText(ServerTextEditBox, L"Таблица не инициализирована!");

break;

case '2':

text = buf;

text.erase(0, 2);

rows\_num = stoi(text.substr(0, text.find(" ")));

text.erase(0, text.find(" ") + 1);

columns\_num = stoi(text.substr(0, text.find(" ")));

text.erase(0, text.find(" ") + 1);

output = L"Таблица размеров " + std::to\_wstring(rows\_num) + L" на " + std::to\_wstring(columns\_num) + L":" + newline;

for (int i = 0; i < rows\_num; i++)

{

for (int j = 0; j < columns\_num; j++)

{

t = text.substr(0, text.find(" "));

std::wstring wStr(t.begin(), t.end());

output = output + wStr + L" \t ";

text.erase(0, text.find(" ") + 1);

}

output += newline;

}

SetWindowText(ServerTextEditBox, output.data());

SetWindowText(ClientTextEditBox, L"");

break;

default:

SetWindowText(ServerTextEditBox, L"Неверно задан запрос!");

break;

}

return TRUE;

}

Сервер

#include "framework.h"

#include "Server.h"

#include <winsock2.h>

#include "stdio.h"

#include <String>

#pragma comment(lib, "ws2\_32.lib")

#pragma warning(disable: 4996)

using namespace std;

#define MAX\_LOADSTRING 100

#define WSA\_ACCEPT (WM\_USER+1)

#define WSA\_NETEVENT (WM\_USER+2)

#define SERV\_PORT 5000

HWND TextEditBox;

SOCKET server\_socket; // Сокет клиента

SOCKET rab\_socket; // Рабочий сокет сервера

SOCKADDR\_IN local\_sin; // Локальный сокет

SOCKADDR\_IN acc\_sin; // Адрес сервера

. . .

BOOL OnCreate();

void ServerStart(HWND hWnd);

void ServerStop(HWND hWnd);

void OnWSAAccept(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam);

void OnWSANetEvent(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam);

void ReadRequest(SOCKET wParam);

string\* Split(string str, string delimiter);

int APIENTRY wWinMain(\_In\_ HINSTANCE hInstance,

\_In\_opt\_ HINSTANCE hPrevInstance,

\_In\_ LPWSTR lpCmdLine,

\_In\_ int nCmdShow)

{ . . . }

ATOM MyRegisterClass(HINSTANCE hInstance)

{ . . . }

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

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

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

150, 150, 350, 400, nullptr, nullptr, hInstance, nullptr);

TextEditBox = CreateWindow(L"Edit", L"", WS\_VISIBLE | WS\_CHILD | WS\_VSCROLL | WS\_BORDER | ES\_MULTILINE | ES\_AUTOVSCROLL,

10, 10, 320, 320, hWnd, NULL, (HINSTANCE)GetWindowLongPtr(hWnd, GWLP\_HINSTANCE), NULL);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

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

{

switch (message)

{

case WM\_CREATE:

OnCreate();

return 0;

case WSA\_ACCEPT:

OnWSAAccept(hWnd, message, wParam, lParam);

return 0;

case WSA\_NETEVENT:

{

switch (WSAGETSELECTEVENT(lParam))

{

case FD\_READ:

ReadRequest((SOCKET)wParam);

WSAAsyncSelect((SOCKET)wParam, hWnd, WSA\_NETEVENT, FD\_READ | FD\_CLOSE);

break;

case FD\_CLOSE:

SetWindowText(TextEditBox, L"Клиент отключился");

break;

}

}

break;

case WM\_COMMAND:

{

int wmId = LOWORD(wParam);

switch (wmId)

{

case IDM\_START:

ServerStart(hWnd);

break;

case IDM\_STOP:

ServerStop(hWnd);

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

default:

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

}

}

break;

case WM\_PAINT:

{ . . . }

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

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

}

return 0;

}

BOOL OnCreate()

{

int rc;

WSADATA WSAData;

TCHAR szTemp[128] = L"Библиотека инициализирована"; rc = WSAStartup(MAKEWORD(1, 1), &WSAData);

if (rc != 0)

{

MessageBox(NULL, L"WSAStartup Error", L"Error", MB\_OK);

return FALSE;

}

return TRUE;

}

/\* Процедура запуска сервера \*/

void ServerStart(HWND hWnd)

{

struct sockaddr\_in srv\_address;

int rc;

server\_socket = socket(AF\_INET, SOCK\_STREAM, 0);

if (server\_socket == INVALID\_SOCKET)

{

MessageBox(NULL, L"socket Error", L"Error", MB\_OK);

return;

}

/\* Устанавливаем адрес и порт \*/

srv\_address.sin\_family = AF\_INET;

srv\_address.sin\_addr.s\_addr = INADDR\_ANY;

srv\_address.sin\_port = htons(SERV\_PORT);

/\* Связываем адрес IP с сокетом \*/

if (bind(server\_socket, (LPSOCKADDR)&srv\_address,

sizeof(srv\_address)) == SOCKET\_ERROR)

{

closesocket(server\_socket);

MessageBox(NULL, L"Bind Error", L"Error", MB\_OK);

return;

}

/\* Переводим сокет в режим ожидания соединения с клиентом \*/

if (listen(server\_socket, 1) == SOCKET\_ERROR)

{

closesocket(server\_socket);

MessageBox(NULL, L"Listen Error", L"Error", MB\_OK);

return;

}

/\* При попытке установления соединения посылаем

сообщение WSA\_ACCEPT в главное окно \*/

rc = WSAAsyncSelect(server\_socket, hWnd, WSA\_ACCEPT, FD\_ACCEPT);

if (rc > 0)

{

closesocket(server\_socket);

MessageBox(NULL, L"WSAAsyncSelect Error", L"Error", MB\_OK);

return;

}

/\* Вывод сообщения о запуске сервера \*/

SetWindowText(TextEditBox, L"Сервер запущен");

} /\* Конец функции ServerStart \*/

void ServerStop(HWND hWnd)

{

/\* Отмена прихода сообщений \*/

WSAAsyncSelect(server\_socket, hWnd, 0, 0);

/\* Закрытие сокета \*/

if (server\_socket != INVALID\_SOCKET)

{

closesocket(server\_socket);

}

/\* Вывод сообщения об остановке сервера \*/

SetWindowText(TextEditBox, L"Сервер остановлен");

} /\* Конец функции ServerStop \*/

void OnWSAAccept(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)

{

int rc;

/\* При ошибке - отмена сообщений \*/

if (WSAGETSELECTERROR(lParam) != 0)

{

MessageBox(NULL, L"Accept Error", L"Error", MB\_OK);

WSAAsyncSelect(server\_socket, hWnd, 0, 0);

return;

}

/\* Определение размера адреса сокета \*/

int acc\_sin\_len = sizeof(acc\_sin);

/\* Установка соединения \*/

rab\_socket = accept(server\_socket, (LPSOCKADDR)&acc\_sin, (int FAR\*) & acc\_sin\_len);

if (rab\_socket == INVALID\_SOCKET)

{

MessageBox(NULL, L"Accept Error, invalid socket", L"Error", MB\_OK);

return;

}

SetWindowText(TextEditBox, L"Клиент подключен");

/\* Подготовка к посылке сообщение о передаче данных

или разрыве соединения \*/

rc = WSAAsyncSelect(rab\_socket, hWnd, WSA\_NETEVENT, FD\_READ | FD\_CLOSE);

if (rc > 0)

{

closesocket(rab\_socket);

MessageBox(NULL, L"WSAAsyncSelect Error", L"Error", MB\_OK);

return;

}

}

bool table\_init = false;

int rows\_num;

int columns\_num;

void ReadRequest(SOCKET wParam)

{

int rc;

char request[100];

SOCKET sock;

std::string text;

/\* Обработка события - приход данных \*/

sock = (SOCKET)wParam;

rc = recv(sock, request, 100, 0);

if (rc)

{

text = request;

int a = -1;

a = text.find("Create table ");

if (a == 0)

{

SetWindowText(TextEditBox, L"Получен запрос на создание таблицы!");

int comma = text.find(',');

rows\_num = stoi(text.substr(13, comma - 12));

columns\_num = stoi(text.substr(comma + 2, text.find(';') - comma));

send(sock, "0", 1, NULL);

table\_init = true;

return;

}

else

{

a = text.find("Insert");

if (a == 0)

{

if (!table\_init)

{

SetWindowText(TextEditBox, L"Ошибка: Таблица не инициализирована");

send(sock, "1", 1, NULL);

return;

}

text.erase(0, 8);

text.erase(std::remove(text.begin(), text.end(), '\r'), text.end());

string str = text.substr(0, text.find(";"));

string\*\* table = new std::string \* [rows\_num];

string\* rows = Split(str, "\n");

for (int i = 0; i < rows\_num; i++)

{

string\* words = Split(rows[i], ",");

table[i] = words;

}

string request\_to\_client = "2 ";

request\_to\_client += to\_string(rows\_num) + " ";

request\_to\_client += to\_string(columns\_num) + " ";

for (int i = 0; i < rows\_num; i++)

{

for (int j = 0; j < columns\_num; j++)

request\_to\_client += table[i][j];

request\_to\_client += " ";

}

send(sock, request\_to\_client.c\_str(), 100, NULL);

table\_init = false;

SetWindowText(TextEditBox, L"Таблица успешно заполнена и отправлена клиенту.");

return;

}

else

send(sock, "-1", 1, NULL);

}

}

}

string\* Split(string str, string delimiter)

{

string\* mass = new string[20];

int i = 0;

size\_t pos = 0;

while ((pos = str.find(delimiter)) != str.npos) {

mass[i] = str.substr(0, pos);

str.erase(0, pos + delimiter.length());

i++;

}

mass[i] = str;

return mass;

}