Министерство науки и высшего образования Российской Федерации

федеральное государственное автономное   
образовательное учреждение высшего образования   
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

Отчет по лабораторной работе №6

Дисциплина: «Операционные системы»

Тема «Применение графического интерфейса пользователя  
для клиент-серверного взаимодействия»

Выполнила: Иванова А.А.

Группа: 6206-010302D

Самара, 2024

**Задание на лабораторную работу**

Реализовать графический интерфейс пользователя для функционала межпроцессного взаимодействия ранее сделанной лабораторной работы № 4 или для лабораторной работы № 5. В качестве получения дополнительного бонуса допускается реализация обеих лабораторных работ.

Все происходящие события должны документироваться: должен писаться лог, то есть журнал событий – сколько процессов запущено, когда, что передано каким клиентом/процессом на сервер и т.п. Должно быть предусмотрено сохранение лога в файл, а также возможность просмотра лога предыдущей запущенной сессии (допустимо использование для этого разных файлов).

Ограничений по языку программирования, технологиям, операционной системе нет.

**Текст программы**

Сервер .cpp

#include <iostream>

#include <fstream>

#include <string>

#include <vector>

#include <sstream>

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

#include<thread>

#include<winsock2.h>

#include <locale>

#pragma warning(disable: 4996)//для inet\_addr

using namespace std;

int n;

ofstream outputFile("log.txt", ios::app);

vector <SOCKET> clientSocket(n);

HANDLE hMutex;

class Car {

public:

string brand;

string model;

string year;

string country;

string price;

string color;

Car(string brand, string model, string year, string country, string price, string color) {

this->brand = brand;

this->model = model;

this->year = year;

this->country = country;

this->price = price;

this->color = color;

}

};

vector<Car> Database = {

Car("Tayota","Camry","2019","Japan","1000000","Blue"),

Car("Porsche","956","1984","Germany","1100000","Black"),

Car("BMW","M8","2019","Germany","1200000","Green"),

Car("Audi","A4","1994","Germany","1200000","Grey"),

Car("Hyundai","Solaris","2001","Korea","1200000","White"),

Car("Kia","Rio","2005","Korea","1300000","Black"),

Car("Audi","Q5","2008","Germany","1400000","Black"),

Car("Lada","Vesta","2015","Russia","1450000","White"),

};

string getData(string name\_characteristic, string characteristic) {

string msg\_foundCar = "";

if (name\_characteristic == "year") {

for (Car car : Database) {

if (characteristic == car.year) {

msg\_foundCar += "Автомобиль найден: " + car.brand + " " + car.model + " " + car.year + " " + car.country + " " + car.price + " " + car.color + "\n";

}

}

}

else if (name\_characteristic == "price") {

for (Car car : Database) {

if (characteristic == car.price) {

msg\_foundCar += "Автомобиль найден: " + car.brand + " " + car.model + " " + car.year + " " + car.country + " " + car.price + " " + car.color + "\n";

}

}

}

else if (name\_characteristic == "brand") {

for (Car car : Database) {

if (characteristic == car.brand) {

msg\_foundCar += "Автомобиль найден: " + car.brand + " " + car.model + " " + car.year + " " + car.country + " " + car.price + " " + car.color + "\n";

}

}

}

else if (name\_characteristic == "model") {

for (Car car : Database) {

if (characteristic == car.model)

{

msg\_foundCar += "Автомобиль найден: " + car.brand + " " + car.model + " " + car.year + " " + car.country + " " + car.price + " " + car.color + "\n";

}

}

}

else if (name\_characteristic == "country") {

for (Car car : Database) {

if (characteristic == car.country) {

msg\_foundCar += "Автомобиль найден: " + car.brand + " " + car.model + " " + car.year + " " + car.country + " " + car.price + " " + car.color + "\n";

}

}

}

else if (name\_characteristic == "color") {

for (Car car : Database) {

if (characteristic == car.color)

{

msg\_foundCar += "Автомобиль найден: " + car.brand + " " + car.model + " " + car.year + " " + car.country + " " + car.price + " " + car.color + '\n';

}

}

}

else {

msg\_foundCar += "Автомобиль не найден!";

}

return msg\_foundCar;

}

void changeData(string brand, string model, string year, string country, string price, string color) {

Database.push\_back(Car(brand, model, year, country, price, color));

}

void function(int i) {

bool flag = true;

do {

char buffer[1024];

int bytesReceived = recv(clientSocket[i], buffer, sizeof(buffer), 0);

if (bytesReceived > 0) {

string data(buffer, bytesReceived);

// Получаем текущее время

std::time\_t currentTime = std::time(nullptr);

// Преобразуем в строку

char\* timeString = std::ctime(&currentTime);

outputFile << "Request client " << i << ": " << data << " [" << timeString << "]" << endl;

stringstream ss(data);

vector<string> words;

string word;

while (ss >> word) {

words.push\_back(word);

}

while (!ss.eof() && isspace(ss.peek())) {

ss.get();

}

WaitForSingleObject(hMutex, INFINITE);

if (words[0] == "getData") {

string buf = getData(words[1], words[2]);

if (buf == "") {

buf = "Автомобиль не найден!\n";

}

char msg\_buffer[1024];

strcpy(msg\_buffer, buf.c\_str());

int bytesSent = send(clientSocket[i], msg\_buffer, sizeof(msg\_buffer), 0);

// Получаем текущее время

std::time\_t currentTime = std::time(nullptr);

// Преобразуем в строку

char\* timeString = std::ctime(&currentTime);

outputFile << "Answer server " << buf << " [" << timeString << "]" << endl;

if (bytesSent == -1) {

cerr << "Ошибка отправки данных\n";

}

}

else if (words[0] == "changeData") {

changeData(words[1], words[2], words[3], words[4], words[5], words[6]);

string buf = "Database updated!";

char msg\_buffer[1024];

strcpy(msg\_buffer, buf.c\_str());

for (int j = 0; j < clientSocket.size(); ++j) {

int bytesSent = send(clientSocket[j], msg\_buffer, sizeof(msg\_buffer), 0);

}

// Получаем текущее время

std::time\_t currentTime = std::time(nullptr);

// Преобразуем в строку

char\* timeString = std::ctime(&currentTime);

outputFile << "Answer server " << buf << " [" << timeString << "]" << endl;

}

else if (words[0] == "finish") {

flag = false;

string msg\_buffer = "Bye";

cout << msg\_buffer;

closesocket(clientSocket[i]);

// Получаем текущее время

std::time\_t currentTime = std::time(nullptr);

// Преобразуем в строку

char\* timeString = std::ctime(&currentTime);

outputFile << "Client " << i << " disconnect" << " [" << timeString << "]" << endl;

}

else {

char msg\_buffer[1024];

string buf = "Сервер ожидает запроса или завершите работу(finish)!";

strcpy(msg\_buffer, buf.c\_str());

int bytesSent = send(clientSocket[i], msg\_buffer, sizeof(msg\_buffer), 0);

if (bytesSent == -1) {

cerr << "Ошибка отправки данных\n";

}

}

}

else {

cout << "Ошибка передачи сообщения" << endl;

if (closesocket(clientSocket[i] == SOCKET\_ERROR))

cerr << "Не удалось прервать соединение.\n Код ошибки: " << WSAGetLastError();

}

} while (flag);

ReleaseMutex(hMutex);

}

int main()

{

setlocale(LC\_ALL, "Russian");

outputFile << "-----------------------------------------------------------------" << endl;

outputFile << "New session: " << endl;

hMutex = CreateMutexW(NULL, FALSE, L"MyMutex");

//Загрузка библиотеки

WSAData wsaData; //создаем структуру для загрузки

WORD DLLVersion = MAKEWORD(2, 1); // Версия библиотеки winsock

if (WSAStartup(DLLVersion, &wsaData) != 0) { // проверка подключения

cerr << "Error: failed to link library.\n";

return 1;

}

//Заполняем информацию об адресе сокета

SOCKADDR\_IN addr;

int sizeOfAddr = sizeof(addr);

addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

addr.sin\_port = htons(1111);

addr.sin\_family = AF\_INET;

SOCKET sListen = socket(AF\_INET, SOCK\_STREAM, 0); //сокет для прослушивания порта,семейство интернет протоколов,протокол

if (bind(sListen, (SOCKADDR\*)&addr, sizeOfAddr) == SOCKET\_ERROR) {

//привязка адреса сокету

printf("Error bind %d\n", WSAGetLastError());

if (closesocket(sListen) == SOCKET\_ERROR)

cerr << "Failed to terminate connection.\n Error code: " << WSAGetLastError();

WSACleanup();

return 1;

}

//подключение прослушивания с максимальной очередью

if (listen(sListen, SOMAXCONN) == SOCKET\_ERROR) {

cout << "Listen failed;\n";

if (closesocket(sListen) == SOCKET\_ERROR)

cerr << "Failed to terminate connection.\n Error code: " << WSAGetLastError();

WSACleanup();

return 1;

}

//задаем информацию для окна открытия

STARTUPINFOW si;//структура

PROCESS\_INFORMATION pi;// структура с информацией о процессе

ZeroMemory(&si, sizeof(si));

si.cb = sizeof(si);// указываем размер

ZeroMemory(&pi, sizeof(pi));

DWORD dwThreadId = 0;

//создаём новые окна для клиентов

cout << "Input n: ";

cin >> n;

for (int i = 0; i < n; i++)

{

if (!CreateProcessW(

L"C:\\Users\\Asus\\Desktop\\24 инф\\6\_5\_client\\x64\\Debug\\6\_5\_client.exe", // module name

NULL, // Command line (пустая строка)

NULL, // Process handle not inheritable

NULL, // Thread handle not inheritable

FALSE, // Set handle inheritance to FALSE

CREATE\_NEW\_CONSOLE, //creation flags

NULL, // Use parent's environment block

NULL, // Use parent's starting directory

&si, // Pointer to STARTUPINFO structure

&pi // Pointer to PROCESS\_INFORMATION structure

)) {

cout << GetLastError();

}

Sleep(10);

outputFile << "Create process-client: " << i << endl;

clientSocket.push\_back(accept(sListen, (SOCKADDR\*)&addr, &sizeOfAddr));

CreateThread(NULL, NULL, (LPTHREAD\_START\_ROUTINE)function, (LPVOID)(i), NULL, NULL);

}

system("pause");

if (closesocket(sListen) == SOCKET\_ERROR) {

cerr << "Не удалось прервать соединение.\n Код ошибки: " << WSAGetLastError();

}

WSACleanup();

return 0;

}

Клиент

.cpp

#include "client.h"

using namespace System; //подключение директив

using namespace System::Windows::Forms;

[STAThreadAttribute] // атрибут указывает, что основной поток приложения должен быть

//однопоточным (STA - Single-Threaded Apartment), что является требованием для использования Windows Forms

void main(array<String^>^ args)

{

Application::EnableVisualStyles(); // подключаем визуальные стили

Application::SetCompatibleTextRenderingDefault(false); // для корректной обработки текста

My65client::client form; //Создается экземпляр класса client из пространства имен My65client

Application::Run(% form); //Запускает цикл обработки сообщений для указанной формы

}

.h

#include <iostream>

#include <thread>

#include <string>

#include <winsock2.h>

#include <msclr/marshal\_cppstd.h>

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

#pragma warning(disable: 4996) // для inet\_addr

namespace My65client {

using namespace System;

using namespace System::ComponentModel;

using namespace System::Collections;

using namespace System::Windows::Forms;

using namespace System::Data;

using namespace System::Drawing;

using namespace System::Threading;

public ref class client : public System::Windows::Forms::Form

{

public:

client(void)

{

InitializeComponent();

InitializeSocket();

// Запуск потока для получения сообщений от сервера

receiveThread = gcnew Thread(gcnew ThreadStart(this, &client::ReceiveMessages));

receiveThread->IsBackground = true;

receiveThread->Start();

}

protected:

/// <summary>

/// Освободить все используемые ресурсы.

/// </summary>

~client()

{

if (components)

{

delete components;

}

// Закрытие сокета и очистка Winsock

closesocket(Connection);

WSACleanup();

receiveThread->Abort();

}

private:

HANDLE hMutex;

System::Windows::Forms::ComboBox^ comboBox;

System::Windows::Forms::TextBox^ valueTextBox;

private: System::Windows::Forms::TextBox^ resTextBox1;

System::Windows::Forms::Button^ getCarButton;

System::Windows::Forms::TextBox^ newCarTextBox;

System::Windows::Forms::TextBox^ addCarResponseTextBox;

System::Windows::Forms::Button^ addCarButton;

System::Windows::Forms::Button^ finishButton;

private: System::ComponentModel::IContainer^ components;

/// <summary>

/// Обязательная переменная конструктора.

/// </summary>

SOCKET Connection;

private: System::Windows::Forms::BindingSource^ bindingSource1;

Thread^ receiveThread;

/// <summary>

/// Требуемый метод для поддержки конструктора — не изменяйте

/// содержимое этого метода с помощью редактора кода.

/// </summary>

void InitializeComponent(void)

{

this->components = (gcnew System::ComponentModel::Container());

this->comboBox = (gcnew System::Windows::Forms::ComboBox());

this->valueTextBox = (gcnew System::Windows::Forms::TextBox());

this->resTextBox1 = (gcnew System::Windows::Forms::TextBox());

this->getCarButton = (gcnew System::Windows::Forms::Button());

this->newCarTextBox = (gcnew System::Windows::Forms::TextBox());

this->addCarResponseTextBox = (gcnew System::Windows::Forms::TextBox());

this->addCarButton = (gcnew System::Windows::Forms::Button());

this->finishButton = (gcnew System::Windows::Forms::Button());

this->bindingSource1 = (gcnew System::Windows::Forms::BindingSource(this->components));

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->bindingSource1))->BeginInit();

this->SuspendLayout();

//

// comboBox

//

this->comboBox->BackColor = System::Drawing::Color::LightGray;

this->comboBox->DropDownStyle = System::Windows::Forms::ComboBoxStyle::DropDownList;

this->comboBox->Items->AddRange(gcnew cli::array< System::Object^ >(6) {

L"brand", L"model", L"year", L"country", L"price",

L"color"

});

this->comboBox->Location = System::Drawing::Point(16, 15);

this->comboBox->Margin = System::Windows::Forms::Padding(4);

this->comboBox->Name = L"comboBox";

this->comboBox->Size = System::Drawing::Size(345, 24);

this->comboBox->TabIndex = 0;

//

// valueTextBox

//

this->valueTextBox->BackColor = System::Drawing::Color::Gray;

this->valueTextBox->BorderStyle = System::Windows::Forms::BorderStyle::None;

this->valueTextBox->Location = System::Drawing::Point(16, 48);

this->valueTextBox->Margin = System::Windows::Forms::Padding(4);

this->valueTextBox->Name = L"valueTextBox";

this->valueTextBox->Size = System::Drawing::Size(345, 15);

this->valueTextBox->TabIndex = 1;

//

// resTextBox1

//

this->resTextBox1->BackColor = System::Drawing::Color::Linen;

this->resTextBox1->BorderStyle = System::Windows::Forms::BorderStyle::None;

this->resTextBox1->Location = System::Drawing::Point(16, 80);

this->resTextBox1->Margin = System::Windows::Forms::Padding(4);

this->resTextBox1->Multiline = true;

this->resTextBox1->Name = L"resTextBox1";

this->resTextBox1->Size = System::Drawing::Size(345, 122);

this->resTextBox1->TabIndex = 2;

//

// getCarButton

//

this->getCarButton->BackColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(0)), static\_cast<System::Int32>(static\_cast<System::Byte>(192)),

static\_cast<System::Int32>(static\_cast<System::Byte>(192)));

this->getCarButton->ForeColor = System::Drawing::SystemColors::ControlText;

this->getCarButton->Location = System::Drawing::Point(16, 210);

this->getCarButton->Margin = System::Windows::Forms::Padding(4);

this->getCarButton->Name = L"getCarButton";

this->getCarButton->Size = System::Drawing::Size(347, 28);

this->getCarButton->TabIndex = 3;

this->getCarButton->Text = L"Получить машину";

this->getCarButton->UseVisualStyleBackColor = false;

this->getCarButton->Click += gcnew System::EventHandler(this, &client::getCarButton\_Click);

//

// newCarTextBox

//

this->newCarTextBox->BackColor = System::Drawing::Color::Gray;

this->newCarTextBox->BorderStyle = System::Windows::Forms::BorderStyle::None;

this->newCarTextBox->Location = System::Drawing::Point(16, 246);

this->newCarTextBox->Margin = System::Windows::Forms::Padding(4);

this->newCarTextBox->Name = L"newCarTextBox";

this->newCarTextBox->Size = System::Drawing::Size(345, 15);

this->newCarTextBox->TabIndex = 4;

//

// addCarResponseTextBox

//

this->addCarResponseTextBox->BackColor = System::Drawing::Color::Linen;

this->addCarResponseTextBox->BorderStyle = System::Windows::Forms::BorderStyle::None;

this->addCarResponseTextBox->Location = System::Drawing::Point(16, 278);

this->addCarResponseTextBox->Margin = System::Windows::Forms::Padding(4);

this->addCarResponseTextBox->Multiline = true;

this->addCarResponseTextBox->Name = L"addCarResponseTextBox";

this->addCarResponseTextBox->Size = System::Drawing::Size(345, 122);

this->addCarResponseTextBox->TabIndex = 5;

//

// addCarButton

//

this->addCarButton->BackColor = System::Drawing::Color::FromArgb(static\_cast<System::Int32>(static\_cast<System::Byte>(0)), static\_cast<System::Int32>(static\_cast<System::Byte>(192)),

static\_cast<System::Int32>(static\_cast<System::Byte>(192)));

this->addCarButton->Location = System::Drawing::Point(16, 409);

this->addCarButton->Margin = System::Windows::Forms::Padding(4);

this->addCarButton->Name = L"addCarButton";

this->addCarButton->Size = System::Drawing::Size(347, 28);

this->addCarButton->TabIndex = 6;

this->addCarButton->Text = L"Добавить в БД";

this->addCarButton->UseVisualStyleBackColor = false;

this->addCarButton->Click += gcnew System::EventHandler(this, &client::addCarButton\_Click);

//

// finishButton

//

this->finishButton->BackColor = System::Drawing::Color::Red;

this->finishButton->FlatStyle = System::Windows::Forms::FlatStyle::Flat;

this->finishButton->Font = (gcnew System::Drawing::Font(L"Microsoft Sans Serif", 10.2F, System::Drawing::FontStyle::Bold, System::Drawing::GraphicsUnit::Point,

static\_cast<System::Byte>(204)));

this->finishButton->Location = System::Drawing::Point(16, 475);

this->finishButton->Margin = System::Windows::Forms::Padding(4);

this->finishButton->Name = L"finishButton";

this->finishButton->Size = System::Drawing::Size(347, 28);

this->finishButton->TabIndex = 8;

this->finishButton->Text = L"Завершить";

this->finishButton->TextAlign = System::Drawing::ContentAlignment::TopCenter;

this->finishButton->TextImageRelation = System::Windows::Forms::TextImageRelation::TextBeforeImage;

this->finishButton->UseVisualStyleBackColor = false;

this->finishButton->Click += gcnew System::EventHandler(this, &client::finishButton\_Click);

//

// client

//

this->AutoScaleDimensions = System::Drawing::SizeF(8, 16);

this->AutoScaleMode = System::Windows::Forms::AutoScaleMode::Font;

this->BackColor = System::Drawing::Color::Black;

this->ClientSize = System::Drawing::Size(368, 518);

this->Controls->Add(this->finishButton);

this->Controls->Add(this->addCarButton);

this->Controls->Add(this->addCarResponseTextBox);

this->Controls->Add(this->newCarTextBox);

this->Controls->Add(this->getCarButton);

this->Controls->Add(this->resTextBox1);

this->Controls->Add(this->valueTextBox);

this->Controls->Add(this->comboBox);

this->FormBorderStyle = System::Windows::Forms::FormBorderStyle::FixedToolWindow;

this->Margin = System::Windows::Forms::Padding(4);

this->Name = L"client";

this->Text = L"Client";

(cli::safe\_cast<System::ComponentModel::ISupportInitialize^>(this->bindingSource1))->EndInit();

this->ResumeLayout(false);

this->PerformLayout();

}

void InitializeSocket(void)

{

//Загрузка библиотеки

WSAData wsaData; //создаем структуру для загрузки

WORD DLLVersion = MAKEWORD(2, 1); // Версия библиотеки winsock

if (WSAStartup(DLLVersion, &wsaData) != 0) { // проверка подключения

MessageBox::Show("Ошибка инициализации Winsock", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

exit(1);

}

// Настройка адреса сокета

SOCKADDR\_IN addr;

addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

addr.sin\_port = htons(1111);

addr.sin\_family = AF\_INET;

// Создание сокета

Connection = socket(AF\_INET, SOCK\_STREAM, 0);//сокет для прослушивания порта,семейство интернет протоколов,протокол

//проверка на подключение к серверу

if (connect(Connection, (SOCKADDR\*)&addr, sizeof(addr)) != 0) {

int error\_code = WSAGetLastError();

String^ error\_msg = "Не удалось подключиться к серверу. Код ошибки: " + error\_code;

MessageBox::Show(error\_msg, "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

exit(1);

}

}

void ReceiveMessages()

{

while (true)

{

char buffer[1024] = { 0 };

int bytesReceived = recv(Connection, buffer, sizeof(buffer), 0);

if (bytesReceived > 0)

{

std::string str = std::string(buffer, 0, bytesReceived);

if (str == "Database updated!") {

System::String^ newMessage = gcnew System::String(str.c\_str());

this->addCarResponseTextBox->AppendText(newMessage + Environment::NewLine);

}

else this->resTextBox1->Text = gcnew System::String(str.c\_str());

}

else break;

}

}

private:

//кнопка найти машину по характеристике

System::Void getCarButton\_Click(System::Object^ sender, System::EventArgs^ e)

{

String^ parameter = this->comboBox->SelectedItem->ToString();

String^ value = this->valueTextBox->Text;

std::string parameterStr = msclr::interop::marshal\_as<std::string>(parameter);

std::string valueStr = msclr::interop::marshal\_as<std::string>(value);

std::string message = "getData" + std::string(" ") + parameterStr + std::string(" ") + valueStr;

// Отправка сообщения на сервер

int bytesSent = send(Connection, message.c\_str(), (int)strlen(message.c\_str()), 0);

if (bytesSent == -1)

{

this->resTextBox1->Text = "Ошибка отправки данных на сервер";

return;

}

}

//кнопка добавить машину в БД

System::Void addCarButton\_Click(System::Object^ sender, System::EventArgs^ e)

{

String^ newCarData = this->newCarTextBox->Text;

std::string newCarDataStr = msclr::interop::marshal\_as<std::string>(newCarData);

std::string message = "changeData " + newCarDataStr;

// Отправка сообщения на сервер

int bytesSent = send(Connection, message.c\_str(), (int)strlen(message.c\_str()), 0);

if (bytesSent == -1)

{

this->addCarResponseTextBox->Text = "Ошибка отправки данных на сервер";

return;

}

}

//кнопка завершить работу клиента

System::Void finishButton\_Click(System::Object^ sender, System::EventArgs^ e)

{

// Отправка сообщения "finish" на сервер

std::string finishMessage = "finish";

int bytesSent = send(Connection, finishMessage.c\_str(), finishMessage.length(), 0);

if (bytesSent == SOCKET\_ERROR)

{

MessageBox::Show("Ошибка отправки сообщения 'finish' на сервер", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

closesocket(Connection);

WSACleanup();

this->Close(); // Закрытие окна клиента

}

};

}