**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

по дисциплине: «Программирование сетевых приложений»

на тему: **«**Программирование безопасного клиент-серверного

взаимодействия с использованием *HTTPS***»**

Выполнил: студент гр. ИТИ-41

Ковшаров Г. Ю.

Принял: преподаватель

Гуменников Е. Д.

Гомель 2025

**Цель работы:** изучить технологию *WebSocket*. Научиться использовать эту технологию при разработке интерактивных веб-приложений.

**Задание:**

Для задания представленном ниже для коммуникации использовать *HTTPS* протокол (генерировать тестовый сертификат с использованием *OpenSSL*, можно использовать сертификаты, генерируемые инструментами  
*Windows*). (Вся работа должна быть выполнена с использованием  
стандартных сокетов, все *HTTPS* заголовки формируются вручную)

Разработать *https* сервер (использовать пул потоков для параллельной  
обработки запросов клиентов). В качестве клиентского приложения  
используется веб- браузер. Все заголовки формируются вручную. Браузер  
должен правильно отображать получаемую информацию. Рекомендуется  
на *GET* запрос возвращать *html* страницу с формой ввода, и выполнять  
*POST* запрос, на который сервер присылает решение.

Таблица 1 – Вариант задания

|  |  |
| --- | --- |
| **Вариант** | **Задание** |
| 13 | Разбор математического выражения и вычисление с учетом скобок и приоритетов операций (+ \* / ) . Использовать обратную польскую запись |

**Ход работы**

Первым и ключевым этапом стала подготовка инфраструктуры для защищенного соединения. С помощью утилиты *OpenSSL* был сгенерирован самоподписанный *SSL*-сертификат, включающий в себя приватный ключ (.*key*) и публичный сертификат (.*crt*). Для удобного использования в *.NET*-приложении эти файлы были объединены в один контейнер формата *.pfx*. Чтобы браузер доверял нашему серверу и не блокировал соединение, сгенерированный сертификат был добавлен в хранилище «Доверенные корневые центры сертификации» операционной системы.

На рисунке 1 представлен вывод консоли после генерации *SSL*-сертификата.

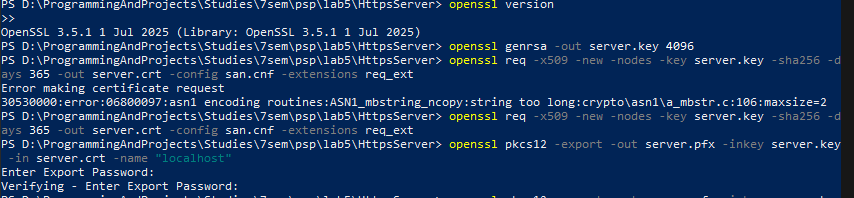


Рисунок 1 – Вывод консоли после генерации *SSL*-сертификата.

Архитектура сервера была построена на основе класса *TcpListener*, который прослушивал указанный сетевой порт на предмет входящих *TCP*-подключений. При поступлении нового клиентского подключения его обработка делегировалась свободному потоку из пула, что позволило серверу оставаться отзывчивым и не блокироваться на время обработки одного запроса.

Для установления защищенного канала связи поверх стандартного *TCP*-соединения использовался класс *SslStream*. После принятия клиента его сетевой поток *NetworkStream* «оборачивался» в *SslStream*. Затем вызывался метод *AuthenticateAsServer*, который выполнял процедуру *TLS*-рукопожатия (*handshake*) с клиентом, используя ранее сгенерированный *.pfx*-сертификат. В случае успеха дальнейший обмен данными между клиентом и сервером происходил по зашифрованному каналу.

Одной из основных задач было ручное формирование и разбор *HTTP*-запросов и ответов. Сервер считывал входящие данные из *SslStream* как текстовый поток. В первую очередь анализировалась стартовая строка запроса для определения метода (*GET* или *POST*) и запрашиваемого ресурса (*URI*). Далее в цикле считывались все заголовки до пустой строки, особое внимание уделялось заголовку Content-Length при обработке *POST*-запросов, чтобы корректно определить размер и прочитать тело запроса, содержащее данные из формы.

Логика работы сервера была разделена на два сценария. При получении *GET*-запроса на корневой ресурс (/) сервер формировал и отправлял *HTML*-страницу с веб-формой для ввода числовых данных. В случае поступления *POST*-запроса сервер извлекал данные из тела запроса, выполнял их парсинг и передавал в функцию, реализующую алгоритм сортировки слиянием (*MergeSort*). После выполнения сортировки сервер генерировал новую *HTML*-страницу, содержащую как исходный, так и отсортированный массив чисел.

Ответ клиенту также формировался вручную. Программно создавалась строка, содержащая статус ответа (например, *HTTP*/1.1 200 OK), необходимые заголовки, такие как Content-Type для указания типа контента и *Content-Length* для указания его размера, а также пустую строку-разделитель. После заголовков в поток записывалось тело ответа — сгенерированная *HTML*-страница с результатами.

На рисунке 3-4 представлен результат работы приложения.

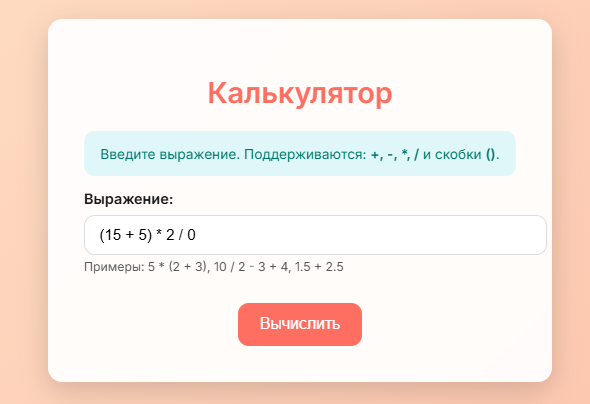


Рисунок 3 – Ввод параметров и формулы

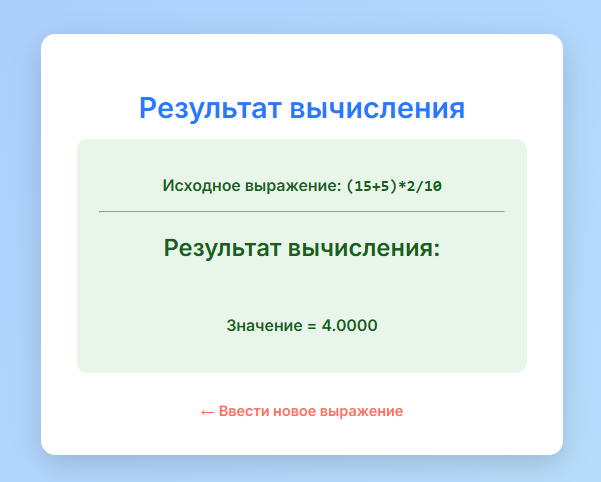


Рисунок 4 – Результат вычисления

В результате был создан полнофункциональный *HTTPS*-сервер, способный принимать и обрабатывать запросы от веб-браузера. При переходе по адресу *https://localhost: 443* браузер корректно отображал страницу с формой, отправлял данные на сервер, который, в свою очередь, выполнял их сортировку и возвращал страницу с результатом. Соединение успешно устанавливалось по защищенному протоколу, что подтверждалось иконкой замка в адресной строке браузера. Сервер продемонстрировал способность одновременно обрабатывать несколько клиентских сессий благодаря использованию пула потоков.

**Вывод:** в ходе выполнения данной работы были получены практические навыки низкоуровневого сетевого программирования с использованием сокетов. Были изучены основы протокола *HTTPS*, включая процесс *TLS*-рукопожатия и использование *SSL*-сертификатов. Основное внимание было уделено ручной работе с протоколом *HTTP*, что позволило глубже понять его структуру и взаимодействие клиента с сервером. Также был успешно применен механизм пула потоков для создания масштабируемого многопоточного приложения.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Листинг программы**

Program.cs  
using System.Net.Sockets;

namespace HttpsServer;

internal class Program

{

public static void Main(string[] args)

{

string certificatePath = "server.pfx";

string certificatePassword = "password";

if (!File.Exists(certificatePath))

{

Console.WriteLine($"Ошибка: Сертификат не найден по пути '{certificatePath}'.");

return;

}

HttpsServer server = new ("127.0.0.1", 8443, certificatePath, certificatePassword);

try

{

server.Start();

}

catch (SocketException e)

{

Console.WriteLine($"Ошибка сокета: {e.Message}.");

}

}

}

HttpsServer.cs

using System.Net;

using System.Net.Security;

using System.Net.Sockets;

using System.Security.Authentication;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Web;

namespace HttpsServer;

public class HttpsServer

{

private readonly TcpListener \_listener;

private readonly X509Certificate2 \_serverCertificate;

public HttpsServer(string ipAddress, int port, string certificatePath, string certificatePassword)

{

\_serverCertificate = new X509Certificate2(certificatePath, certificatePassword);

IPAddress localAddr = IPAddress.Parse(ipAddress);

\_listener = new TcpListener(localAddr, port);

}

public void Start()

{

\_listener.Start();

Console.WriteLine($"Сервер запущен и слушает на https://localhost:{((IPEndPoint)\_listener.LocalEndpoint).Port}/");

Console.WriteLine("Ожидание подключений...");

while (true)

{

TcpClient client = \_listener.AcceptTcpClient();

ThreadPool.QueueUserWorkItem(HandleClient, client);

}

}

private void HandleClient(object clientObj)

{

TcpClient client = (TcpClient)clientObj;

using SslStream sslStream = new (client.GetStream(), false);

try

{

sslStream.AuthenticateAsServer(\_serverCertificate, clientCertificateRequired: false, enabledSslProtocols: SslProtocols.Tls12, checkCertificateRevocation: false);

Console.WriteLine("Клиент подключен по HTTPS.");

using (var reader = new StreamReader(sslStream, Encoding.UTF8, true, 1024, true))

{

string requestLine = reader.ReadLine();

if (string.IsNullOrEmpty(requestLine))

{

return;

}

string headerLine;

var headers = new Dictionary<string, string>(StringComparer.OrdinalIgnoreCase);

while (!string.IsNullOrEmpty(headerLine = reader.ReadLine()))

{

var parts = headerLine.Split([':'], 2);

if (parts.Length == 2)

{

headers[parts[0].Trim()] = parts[1].Trim();

}

}

Console.WriteLine("\n--- Получен запрос ---");

Console.WriteLine(requestLine);

foreach (var header in headers)

{

Console.WriteLine($"{header.Key}: {header.Value}");

}

string[] requestLineParts = requestLine.Split(' ');

string method = requestLineParts[0];

string path = requestLineParts[1];

if (method == "POST")

{

if (headers.TryGetValue("Content-Length", out var contentLengthStr) &&

int.TryParse(contentLengthStr, out int contentLength) &&

contentLength > 0)

{

char[] buffer = new char[contentLength];

reader.Read(buffer, 0, contentLength);

string body = new (buffer);

Console.WriteLine("\n--- Тело запроса ---");

Console.WriteLine(body);

Console.WriteLine("--- Конец запроса ---\n");

HandlePostRequest(sslStream, body);

}

else

{

SendHttpResponse(sslStream, "400 Bad Request", "text/html", "<h1>400 Bad Request</h1>");

}

}

else if (method == "GET" && path == "/")

{

Console.WriteLine("--- Конец запроса ---\n");

HandleGetRequest(sslStream);

}

else

{

SendNotFoundResponse(sslStream);

}

}

}

catch (AuthenticationException e)

{

Console.WriteLine($"Ошибка аутентификации: {e.Message}");

}

catch (Exception e)

{

Console.WriteLine($"Произошла ошибка: {e.Message}");

}

finally

{

sslStream.Close();

client.Close();

Console.WriteLine("Соединение с клиентом закрыто.");

}

}

private void HandleGetRequest(SslStream sslStream)

{

string htmlContent = @"

<!DOCTYPE html>

<html lang='ru'>

<head>

<meta charset='UTF-8'>

<title>Сортировка слиянием (MergeSort)</title>

<style>

body { font-family: Arial, sans-serif; background-color: #f4f4f9; color: #333; margin: 2em; }

.container { max-width: 600px; margin: auto; background: white; padding: 2em; border-radius: 8px; box-shadow: 0 2px 10px rgba(0,0,0,0.1); }

h1 { color: #444; }

input[type='text'] { width: 100%; padding: 10px; margin-bottom: 1em; border: 1px solid #ccc; border-radius: 4px; }

input[type='submit'] { background-color: #007bff; color: white; padding: 10px 15px; border: none; border-radius: 4px; cursor: pointer; }

input[type='submit']:hover { background-color: #0056b3; }

</style>

</head>

<body>

<div class='container'>

<h1>Сортировка слиянием (MergeSort)</h1>

<form action='/' method='post'>

<label for='numbers'>Введите числа через запятую или пробел:</label><br>

<input type='text' id='numbers' name='numbers' value='5, 2, 8, 1, 9, 4' required><br>

<input type='submit' value='Отсортировать'>

</form>

</div>

</body>

</html>";

SendHttpResponse(sslStream, "200 OK", "text/html; charset=utf-8", htmlContent);

}

private void HandlePostRequest(SslStream sslStream, string rawRequest)

{

string body = rawRequest.Substring(rawRequest.IndexOf("\r\n\r\n") + 1);

var formData = HttpUtility.ParseQueryString(body);

string numbersStr = formData["numbers"];

string originalNumbers = "Неверный ввод";

string sortedNumbers = "Не удалось отсортировать";

try

{

List<int> numbers = [.. numbersStr

.Split([',', ' '], StringSplitOptions.RemoveEmptyEntries)

.Select(int.Parse)];

originalNumbers = string.Join(", ", numbers);

var sortedList = numbers.MergeSort();

sortedNumbers = string.Join(", ", sortedList);

}

catch (FormatException)

{

sortedNumbers = "Ошибка: Введены некорректные данные. Пожалуйста, вводите только числа.";

}

string htmlContent = $@"

<!DOCTYPE html>

<html lang='ru'>

<head>

<meta charset='UTF-8'>

<title>Результат сортировки</title>

<style>

body {{ font-family: Arial, sans-serif; background-color: #f4f4f9; color: #333; margin: 2em; }}

.container {{ max-width: 600px; margin: auto; background: white; padding: 2em; border-radius: 8px; box-shadow: 0 2px 10px rgba(0,0,0,0.1); }}

h1 {{ color: #444; }}

p {{ font-size: 1.1em; }}

.result {{ background-color: #e9ecef; padding: 1em; border-radius: 4px; word-wrap: break-word; }}

a {{ color: #007bff; text-decoration: none; }}

a:hover {{ text-decoration: underline; }}

</style>

</head>

<body>

<div class='container'>

<h1>Результат сортировки</h1>

<p><strong>Исходный массив:</strong></p>

<p class='result'>{originalNumbers}</p>

<p><strong>Отсортированный массив (MergeSort):</strong></p>

<p class='result'>{sortedNumbers}</p>

<br>

<a href='/'>Попробовать еще раз</a>

</div>

</body>

</html>";

SendHttpResponse(sslStream, "200 OK", "text/html; charset=utf-8", htmlContent);

}

private void SendHttpResponse(SslStream stream, string statusCode, string contentType, string content)

{

byte[] contentBytes = Encoding.UTF8.GetBytes(content);

var responseBuilder = new StringBuilder();

responseBuilder.AppendLine($"HTTP/1.1 {statusCode}");

responseBuilder.AppendLine($"Content-Type: {contentType}");

responseBuilder.AppendLine($"Content-Length: {contentBytes.Length}");

responseBuilder.AppendLine("Connection: close");

responseBuilder.AppendLine();

string headers = responseBuilder.ToString();

byte[] headerBytes = Encoding.UTF8.GetBytes(headers);

stream.Write(headerBytes, 0, headerBytes.Length);

stream.Write(contentBytes, 0, contentBytes.Length);

stream.Flush();

}

private void SendNotFoundResponse(SslStream sslStream)

{

string content = "<h1>404 Not Found</h1><p>The requested resource was not found on this server.</p>";

SendHttpResponse(sslStream, "404 Not Found", "text/html; charset=utf-8", content);

}

}

MergeSorter.cs

namespace HttpsServer;

public static class MergeSorter

{

public static List<int> MergeSort(this List<int> unsorted)

{

if (unsorted.Count <= 1)

return unsorted;

List<int> left = [];

List<int> right = [];

int middle = unsorted.Count / 2;

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

{

left.Add(unsorted[i]);

}

for (int i = middle; i < unsorted.Count; i++)

{

right.Add(unsorted[i]);

}

left = MergeSort(left);

right = MergeSort(right);

return left.Merge(right);

}

private static List<int> Merge(this List<int> left, List<int> right)

{

List<int> result = [];

while (left.Count > 0 || right.Count > 0)

{

if (left.Count > 0 && right.Count > 0)

{

if (left.First() <= right.First())

{

result.Add(left.First());

left.Remove(left.First());

}

else

{

result.Add(right.First());

right.Remove(right.First());

}

}

else if (left.Count > 0)

{

result.Add(left.First());

left.Remove(left.First());

}

else if (right.Count > 0)

{

result.Add(right.First());

right.Remove(right.First());

}

}

return result;

}

}