МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №4

по дисциплине: **«**Программирование сетевых приложений**»**

на тему: «Разработка клиент-серверных приложений на базе

стека протоколов *TCP/IP* в .*Net*»

Выполнил: студент гр. ИТП-41

Дорошко Д. А.

Принял: преподаватель

Гуменников Е. Д.

Гомель 2022

**Цель:** изучить организацию параллельной обработки *htpp*-запросов клиентов с использованием пула потоков.

**Задание:**

Для задания представленном ниже для коммуникации использовать *HTTP* протокол (вся работа должна быть выполнена с использованием стандартных сокетов, все *HTTP* заголовки формируются вручную). Разработать *http*­сервер (использовать пул потоков для параллельной обработки запросов клиентов). В качестве клиентского приложения используется веб­браузер. 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.

Задание согласно варианту: *MegreSort*

**Ход работы:**

Главная страница представлена на рисунке 1.

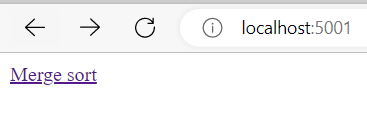


Рисунок 1 – Главная страница

На рисунке 2 представлена страница ввода массива для сортирования

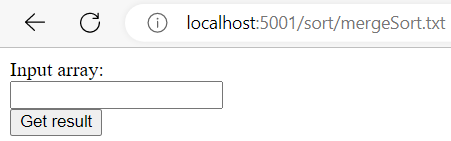


Рисунок 2 – Страница ввода массива для сортирования

Результат возвращаемый сервером после сортирования представлен на рисунке 3.

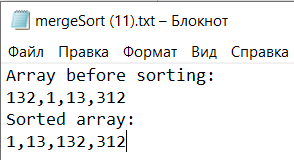


Рисунок 3 – Результат возвращаемый сервером после сортирования

**Вывод:** в ходе выполнения лабораторной работы получены навыки реализации *http*-сервера, изучен и применен пул потоков для параллельной обработки запросов клиентов.

**ПРИЛОЖЕНИЕ А**

**Листинг программы**

using Server.Sorting;

using System;

using System.Net;

using System.Net.Security;

using System.Net.Sockets;

using System.Security;

using System.Security.Authentication;

using System.Security.Cryptography.X509Certificates;

using System.Text;

using System.Text.RegularExpressions;

using System.Threading;

namespace Server

{

public class HttpServer

{

private readonly Regex \_requestTitleRegex = new(@"(?<httpMethod>\S+)\s+(?<httpPath>\S+)\s+(?<httpVersion>\S+)", RegexOptions.Compiled);

private const int BufferSize = 2048;

private int \_port;

public HttpServer(int port = 5001)

{

\_port = port;

}

public void Start()

{

var listener = new TcpListener(IPAddress.Any, \_port);

listener.Start();

while (true)

{

Console.WriteLine($"Server started listening on {listener.LocalEndpoint}");

var client = listener.AcceptTcpClient();

ThreadPool.QueueUserWorkItem(

state =>

{

ProcessClient(state as TcpClient);

}, client);

}

}

private void ProcessClient(TcpClient client)

{

var stream = client.GetStream();

try

{

stream.ReadTimeout = 10000;

stream.WriteTimeout = 10000;

var requestMessage = ReadMessage(stream);

Console.WriteLine($"Received request: {requestMessage}");

string responseMessage;

try

{

responseMessage = GetResponseMessage(requestMessage);

}

catch (Exception ex)

{

responseMessage = GetBadPage(ex.Message);

}

Console.WriteLine($"Sent response: {responseMessage}");

var message = Encoding.UTF8.GetBytes(responseMessage);

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

stream.Flush();

}

finally

{

stream.Close();

client.Close();

}

}

private string GetResponseMessage(string messageData)

{

Regex requestRegex = new Regex("\r\n\r\n");

string[] requestData = requestRegex.Split(messageData, 2);

if (requestData.Length != 2)

{

return GetBadPage();

}

var requestTitle = requestData[0];

var requestBody = requestData[1];

var titleMatch = \_requestTitleRegex.Match(requestTitle.Split("\r\n")[0]);

var httpMethod = titleMatch.Groups["httpMethod"].Value;

var httpPath = titleMatch.Groups["httpPath"].Value;

if (httpPath.Equals("/"))

{

return httpMethod switch

{

"GET" => GetMainPage(),

\_ => GetNotFoundPage(),

};

}

else if (httpPath.Equals("/sort/mergeSort.txt", StringComparison.InvariantCultureIgnoreCase))

{

return httpMethod switch

{

"GET" => GetMergeSortPage(),

"POST" => GetMergeSortResponsePage(requestBody),

\_ => GetNotFoundPage(),

};

}

return GetNotFoundPage(); ;

}

private string ReadMessage(NetworkStream stream)

{

try

{

var buffer = new byte[BufferSize];

var messageData = new StringBuilder();

int bytes;

do

{

bytes = stream.Read(buffer, 0, buffer.Length);

messageData.Append(Encoding.UTF8.GetString(buffer, 0, bytes));

}

while (bytes == BufferSize);

return messageData.ToString();

}

catch

{

return String.Empty;

}

}

private string GetMainPage()

{

StringBuilder bodyBuilder = new StringBuilder();

bodyBuilder.Append("<a href='/sort/mergeSort.txt'>Merge sort</a>");

string body = bodyBuilder.ToString();

return string.Concat(ResponseHeaders(body.Length), body);

}

private string GetNotFoundPage()

{

StringBuilder bodyBuilder = new StringBuilder();

bodyBuilder.Append("Page not found");

bodyBuilder.Append("<br/>");

bodyBuilder.Append("<a href='/'>Go to main page</a>");

return string.Concat(ResponseHeaders(bodyBuilder.Length, responseStatus: "404 NotFound"), bodyBuilder.ToString());

}

private static string GetBadPage(string message = "Something went wrong")

{

StringBuilder bodyBuilder = new StringBuilder();

bodyBuilder.Append(message);

bodyBuilder.Append("<br/>");

bodyBuilder.Append("<a href='/'>Go to main page</a>");

string body = bodyBuilder.ToString();

return string.Concat(ResponseHeaders(body.Length), body);

}

private string GetMergeSortPage()

{

StringBuilder bodyBuilder = new StringBuilder();

bodyBuilder.Append("<form method='post'>");

bodyBuilder.Append("Input array:");

bodyBuilder.Append("<br/>");

bodyBuilder.Append("<input type='text' name='array'>");

bodyBuilder.Append("<br/>");

bodyBuilder.Append("<input type='submit' value='Get result'>");

bodyBuilder.Append("</form>");

string body = bodyBuilder.ToString();

return string.Concat(ResponseHeaders(body.Length), body);

}

private string GetMergeSortResponsePage(string body)

{

int[] array;

var stringArray = body.Split('=')[1].Split("+", StringSplitOptions.RemoveEmptyEntries);

StringBuilder bodyBuilder = new StringBuilder();

array = stringArray.Select(item => int.Parse(item)).ToArray();

var sorter = new MergeSortSolver();

var sortedArray = array.ToArray();

sorter.Sort(sortedArray);

bodyBuilder.Append("Array before sorting:");

bodyBuilder.Append("\n");

bodyBuilder.Append(string.Join(",", array));

bodyBuilder.Append("\n");

bodyBuilder.Append("Sorted array:");

bodyBuilder.Append("\n");

bodyBuilder.Append(string.Join(",", sortedArray));

return string.Concat(ResponseHeaders(bodyBuilder.Length, "application/octet-stream"), bodyBuilder.ToString());

}

public static string ResponseHeaders(int contentLength, string contentType = "text/html;charset=UTF-8", string responseStatus = "200 OK")

{

StringBuilder builder = new StringBuilder();

builder.Append($"HTTP/1.1 {responseStatus}").Append("\r\n");

builder.Append("Date:").Append(DateTime.Now).Append("\r\n");

builder.Append("Server:lab5-server").Append("\r\n");

builder.Append($"Content-Type:{contentType}").Append("\r\n");

builder.Append("Content-Length:").Append(contentLength).Append("\r\n");

builder.Append("Connection:close").Append("\r\n");

builder.Append("\r\n");

return builder.ToString();

}

}

}