



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01 Информатика и вычислительная техника

О Т Ч Е Т

по домашней работе № 2

Название: HTTP-сервер на языке C#

Дисциплина: Разработка приложений на языке C#

Студент

ИУ6-73Б

(Группа)

(Подпись, дата)

К.А. Логачев

(И.О. Фамилия)

Преподаватель

(Подпись, дата)

А.М. Минитаева

(И.О. Фамилия)

Москва, 2024

Цель работы: научиться писать Http-сервера на C#.

Задание: реализовать HTTP сервер на языке C# с использованием классов TcpListener и TcpClient, который будет отдавать по определенному порту статичную html страницу, расположенную на жестком диске по определенному пути. В качестве страницы, которую должен будет вывести браузер при обращении по адресу localhost:<номер порта по варианту>, следует создать html файл. Сам файл может быть любым. Желательно, чтобы в этом html файле была форма ввода и кнопка, это понадобится для выполнения 3 домашнего задания.

По варианту нужно реализовать сервер на Pool Thread и запустить на порту 49314

Ход работы

Описание работы программы:

Данная программа реализует многопоточный HTTP-сервер на языке C#, который обслуживает запросы клиентов, отправляя им HTML-файл. Сервер работает на порту 49314 и использует пул потоков для обработки входящих подключений.

Основные компоненты программы:

1. Класс HttpServer:

- Это основной класс программы, отвечающий за запуск, работу и остановку сервера.
- Он использует HttpListener для прослушивания входящих HTTP-запросов на порту 49314.
- Когда сервер принимает подключение от клиента, запрос передаётся для обработки с использованием пула потоков через метод ThreadPool.QueueUserWorkItem.

Основные методы класса HttpServer:

1. Start():

Этот метод запускает сервер и начинает прослушивание на указанном порту.

Сервер входит в бесконечный цикл, принимая запросы от клиентов через `HttpListener.GetContext()` и передавая их на обработку в пул потоков.

Каждый запрос обрабатывается в отдельном потоке для повышения производительности и параллельной обработки большого числа клиентов.

2. `HandleRequest(HttpListenerContext context)`:

Метод принимает контекст запроса, обрабатывает его и возвращает ответ клиенту.

Если HTML-файл `report.html` существует, его содержимое считывается и отправляется клиенту. Если файл не найден, сервер отправляет текстовое сообщение об ошибке.

Метод использует потоки для чтения данных и отправки их клиенту через объект `HttpListenerResponse`.

3. `Stop()`:

Этот метод останавливает сервер и завершает прослушивание входящих запросов.

Он завершает цикл приёма запросов и останавливает `HttpListener`.

4. Деструктор `~HttpServer()`:

Деструктор автоматически вызывает метод `Stop()`, если сервер всё ещё запущен на момент завершения программы или удаления объекта.

Это гарантирует корректную остановку сервера даже при неожиданных завершениях.

5. Обработка завершения процесса:

В методе `Main()` установлена подписка на событие завершения программы `AppDomain.CurrentDomain.ProcessExit`. Это позволяет гарантировать вызов метода `Stop()` при завершении процесса, чтобы сервер корректно остановился и освободил все ресурсы.

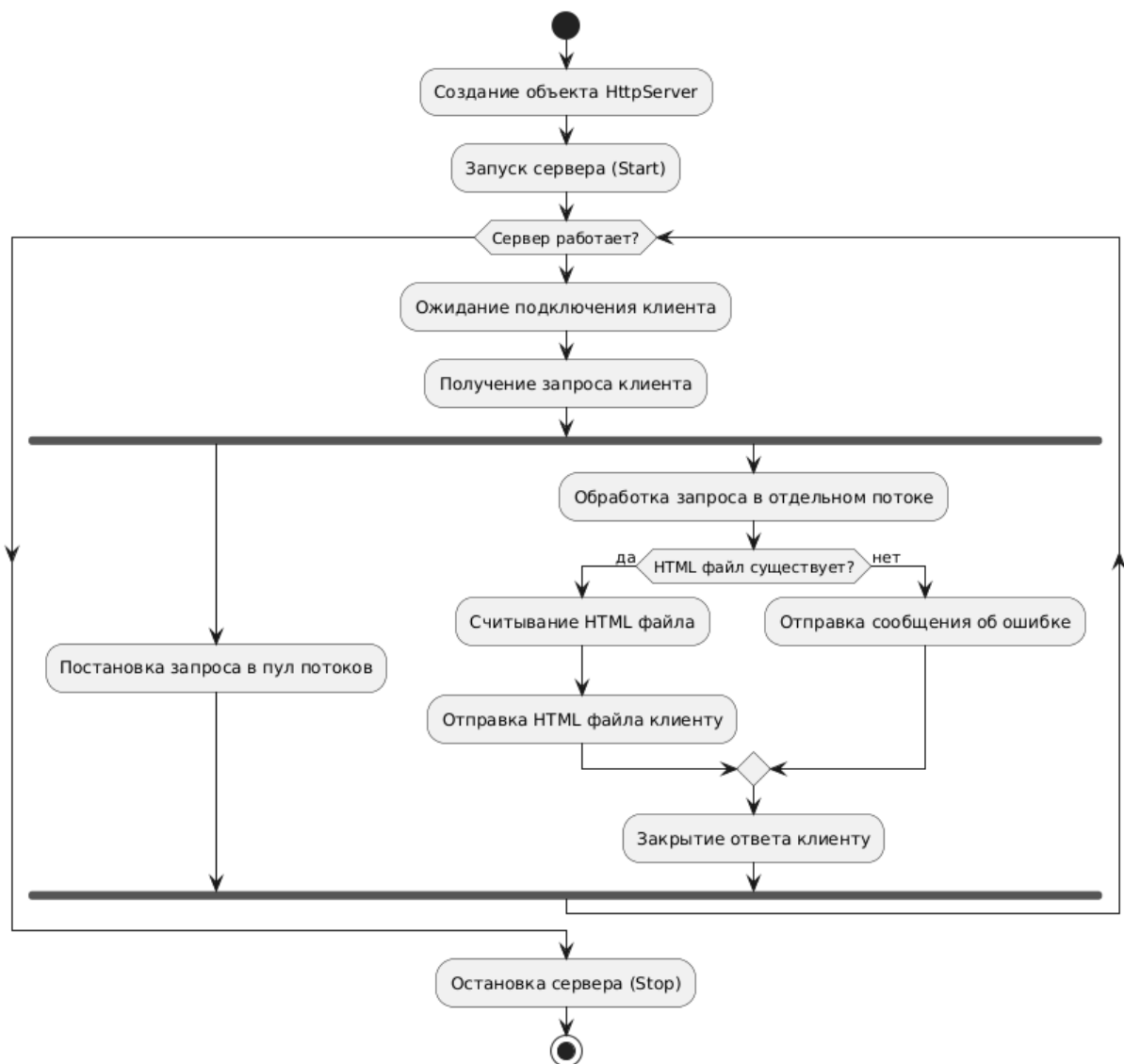


Рисунок 1 – Схема алгоритма

Код программы приведен ниже:

```

using System;
using System.IO;
using System.Net;
using System.Text;
using System.Threading;

namespace HttpServerWithThreadPool {
    class HttpServer {
        private HttpListener _listener;
        private bool _isRunning = false;
        private const int PORT = 49314;
        private string htmlFilePath = "report.html"; // Путь к HTML файлу

        public HttpServer() {
            _listener = new HttpListener();
            _listener.Prefixes.Add($"http://localhost:{PORT}/");
        }

        public void Start() {
            _listener.Start();
            _isRunning = true;
            Console.WriteLine(
                $"Сервер запущен на порту {PORT}. Ожидание запросов...");
        }
    }
}

```

```

        // Основной цикл ожидания подключений
        while (_isRunning) {
            var context = _listener.GetContext();
            ThreadPool.QueueUserWorkItem(o => HandleRequest(context));
        }
    }

    public void Stop() {
        _isRunning = false;
        _listener.Stop();
        Console.WriteLine("Сервер остановлен.");
    }

    // Обработка запроса клиента
    private void HandleRequest(HttpListenerContext context) {
        HttpListenerResponse response = context.Response;

        try {
            // Проверяем существование HTML файла
            if (File.Exists(htmlFilePath)) {
                // Отправляем HTML файл клиенту
                response.ContentType = "text/html";
                byte[] htmlData = File.ReadAllBytes(htmlFilePath);
                response.ContentLength64 = htmlData.Length;

                using (Stream output = response.OutputStream) {
                    output.Write(htmlData, 0, htmlData.Length);
                }
                Console.WriteLine("HTML файл успешно отправлен клиенту.");
            } else {
                // Если файл не найден, отправляем сообщение об ошибке
                string errorMessage = "Ошибка: HTML файл не найден.";
                byte[] errorData = Encoding.UTF8.GetBytes(errorMessage);
                response.ContentType = "text/plain";
                response.ContentLength64 = errorData.Length;

                using (Stream output = response.OutputStream) {
                    output.Write(errorData, 0, errorData.Length);
                }
                Console.WriteLine("Ошибка: HTML файл не найден.");
            }
        } catch (Exception ex) {
            Console.WriteLine($"Ошибка при обработке запроса: {ex.Message}");
        } finally {
            // Закрываем ответ
            response.Close();
        }
    }

    // Деструктор для остановки сервера
    ~HttpServer() {
        if (_isRunning) {
            Stop();
        }
    }

    class Program {
        static void Main(string[] args) {
            HttpServer server = new HttpServer();

            // Обработка остановки сервера при завершении программы
            AppDomain.CurrentDomain.ProcessExit += (s, e) => server.Stop();

            server.Start();
        }
    }
}

```

Код html страницы приведен ниже:

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
<meta charset="UTF-8">
```

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
<title>
```

Document

```
</title>
```

```
</head>
```

```
<body>
```

```
<h1>
```

Hello, world!

```
</h1>
```

```
<form>
```

```
<input type="text" name="inputField" placeholder="Введите текст">
```

```
<button type="submit">
```

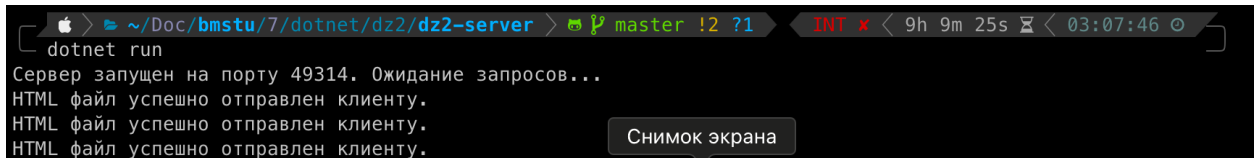
Отправить

```
</button>
```

```
</form>
```

```
</body>
```

</html>



```
dotnet run
Сервер запущен на порту 49314. Ожидание запросов...
HTML файл успешно отправлен клиенту.
HTML файл успешно отправлен клиенту.
HTML файл успешно отправлен клиенту.
```

Снимок экрана

Рисунок 2 – Результат работы сервера (консоль)

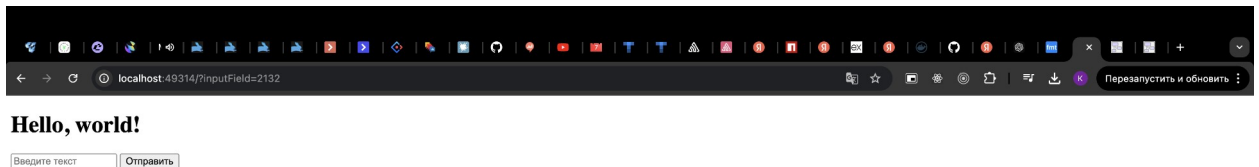



Рисунок 3 – Результат работы программы (браузер)



```
curl http://localhost:49314/
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Document</title>
</head>
<body>
  <h1>Hello, world!</h1>
  <form>
    <input type="text" name="inputField" placeholder="Введите текст">
    <button type="submit">Отправить</button>
  </form>
</body>
</html>
```

Рисунок 4 – Результат работы программы (curl)

Вывод: в ходе данной лабораторной работы были получены навыки написания http-серверов на языке C#.