Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет информационных технологий и управления

Кафедра вычислительных методов и программирования

ОТЧЕТ

по лабораторной работе

**« Создание параллельного многопоточного сервера с установлением логического соединения TCP»**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 124401 |  | У.А. ГАВРИЛОВА |
| Проверил |  | А.А. БУТОВ |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Минск 2023

**Цель работы**

Изучить методы создания серверных приложений на основе установления логического соединения *TCP*, используя алгоритм многопоточной обработки запросов.

**Вариант 5**

Разработать приложение, реализующее архитектуру «клиент-сервер». Для этого необходимо реализовать параллельный многопоточный сервер с установлением логического соединения (TCP). Логику взаимодействия клиента и сервера реализовать так, как указано в варианте индивидуального задания. Предусмотреть возможность просмотра, добавления, редактирования, удаления информации клиентом на сервере.

На сервере хранится информация (список) о комплектующих деталях. Каждая запись списка содержит следующую информацию о комплектующем:

* завод-поставщик;
* стоимость;
* дата поставки;

Таких записей должно быть не менее семи.

Клиент посылает на сервер дату поставки. Назад он получает список комплектующих, поставленных именно на эту дату, и стоимость каждого возвращаемого комплектующего должна превосходить минимальную во всем списке.

**Листинг**

Сервер

Файл Program.cs

using System.Net.Sockets;

using System.Net;

using server;

TcpListener tcpListener = new(IPAddress.Any, 8888);

Console.WriteLine("Ожидание подключения...\n");

Repository r = new();

int clientsNumber = 0;

try

{

tcpListener.Start();

while (true)

{

var tcpClient = await tcpListener.AcceptTcpClientAsync();

\_ = Task.Run(async () => await ProcessClientAsync(tcpClient));

}

}

catch (Exception ex)

{

Console.WriteLine(ex.ToString());

}

async Task ProcessClientAsync(TcpClient tcpClient)

{

NetworkStream stream = tcpClient.GetStream();

ClientSession client = new(++clientsNumber, stream, r);

Console.WriteLine($"Установлено подключение с новым клиентом!\n" +

$"Число активных клиентов: {clientsNumber}\n" +

$"IP: {((IPEndPoint)tcpClient.Client.RemoteEndPoint).Address}\n" +

$"Номер порта: {((IPEndPoint)tcpClient.Client.RemoteEndPoint).Port}\n" +

$"Дескриптор сокета: {tcpClient.Client.Handle}\n");

await client.ConnectAsync();

tcpClient.Close();

clientsNumber--;

Console.WriteLine($"Соединение с клиентом разорвано!\n" +

$"Число активных клиентов: {clientsNumber}\n");

}

Файл ClientSession.cs

using System.Net.Sockets;

using System.Text;

namespace server

{

internal class ClientSession

{

private readonly int number;

private readonly NetworkStream stream;

private readonly Repository r;

public ClientSession(int number, NetworkStream stream, Repository r)

{

this.number = number;

this.stream = stream;

this.r = r;

}

public async Task ConnectAsync()

{

string menuText = " Меню\n" +

"1 - информацию обо всех деталях\n" +

"2 - добавить деталь\n" +

"3 - редактировать информацию о детали\n" +

"4 - удалить деталь\n" +

"5 - получить список деталей по дате поставки\n" +

"0 - завершить работу";

while (true)

{

await SendMessageAsync(menuText);

int choice = int.Parse(await ReceiveMessageAsync(1));

switch (choice)

{

case 1:

await GetAllComponentsAsync();

break;

case 2:

await AddNewComponentAsync();

break;

case 3:

await EditComponentAsync();

break;

case 4:

await RemoveComponentAsync();

break;

case 5:

await GetComponentsByDateAsync();

break;

case 0:

return;

}

}

}

private async Task GetAllComponentsAsync()

{

await SendMessageAsync(r.GetAllComponents()

+ "Отправьте любой символ чтобы продолжить...");

await ReceiveMessageAsync(10);

}

private async Task AddNewComponentAsync()

{

Component newComponent = await GetComponentFromConsoleAsync();

r.AddComponent(newComponent);

}

private async Task EditComponentAsync()

{

await SendMessageAsync("Введите название детали: ");

string name = await ReceiveMessageAsync(24);

Component newComponent = await GetComponentFromConsoleAsync();

r.EditComponent(name, newComponent);

}

private async Task RemoveComponentAsync()

{

await SendMessageAsync("Введите название детали: ");

string name = await ReceiveMessageAsync(24);

r.DeleteComponent(r[name]);

}

private async Task GetComponentsByDateAsync()

{

await SendMessageAsync("Введите дату поставки:");

string stringResponse = await ReceiveMessageAsync(10);

await SendMessageAsync(r.GetComponents(DateOnly.Parse(stringResponse))

+ "Отправьте любой символ чтобы продолжить...");

await ReceiveMessageAsync(10);

}

private async Task SendMessageAsync(string message)

{

await stream.WriteAsync(Encoding.UTF8.GetBytes(message));

Console.WriteLine($"Клиенту с номером №{number} отправлено сообщение в {DateTime.Now}\n" +

$"==========Сообщение==========\n\n" +

$"{message}\n\n" +

$"==========Конец сообщения==========\n");

}

private async Task<string> ReceiveMessageAsync(int maxByteNumber)

{

byte[] response = new byte[maxByteNumber];

await stream.ReadAsync(response, 0, maxByteNumber);

string stringResponse = Encoding.UTF8.GetString(response).TrimEnd('\0');

Console.WriteLine($"Клиент №{number} прислал сообщение {DateTime.Now}\n" +

$"==========Сообщение==========\n\n" +

$"{stringResponse}\n\n" +

$"==========Конец сообщения==========\n");

return stringResponse;

}

private async Task<Component> GetComponentFromConsoleAsync()

{

await SendMessageAsync("Введите информацию о детали\n\nНазвание:");

string name = await ReceiveMessageAsync(24);

await SendMessageAsync("Завод:");

string factoryName = await ReceiveMessageAsync(24);

await SendMessageAsync("Цена:");

double price = double.Parse(await ReceiveMessageAsync(24));

await SendMessageAsync("Дата поставки:");

DateOnly deliveryDate = DateOnly.Parse(await ReceiveMessageAsync(24));

return new Component(name, factoryName, price, deliveryDate);

}

}

}

Файл Repository.cs

using Newtonsoft.Json;

namespace server

{

internal class Repository

{

private readonly string fileName;

private double minPrice;

List<Component> componentList;

public Repository()

{

fileName = "components.json";

componentList = new List<Component>();

OutOfFile();

GetMinPrice();

}

public Component this[string name]

{

get

{

foreach (var component in componentList)

{

if (component.Name == name)

return component;

}

return new Component();

}

}

private void GetMinPrice()

{

minPrice = int.MaxValue;

foreach (Component component in componentList)

{

if (minPrice > component.Price)

{

minPrice = component.Price;

}

}

}

private void OutOfFile()

{

if (File.Exists(fileName))

{

using StreamReader stream = new(fileName, true);

string json = stream.ReadToEnd();

if (json.Length != 0)

{

componentList = JsonConvert.DeserializeObject<List<Component>>(json);

}

}

}

public string GetComponents(DateOnly userDate)

{

string reply = String.Empty;

foreach (Component component in componentList)

{

if (component.DeliveryDate == userDate &&

component.Price != minPrice)

{

reply += component.ToString() + "\n";

}

}

return reply;

}

private void InFile()

{

using StreamWriter stream = new(fileName);

string json = JsonConvert.SerializeObject(componentList);

stream.Write(json);

}

public void AddComponent(Component component)

{

componentList.Add(component);

InFile();

GetMinPrice();

}

public void DeleteComponent(Component component)

{

componentList.Remove(component);

InFile();

GetMinPrice();

}

public void EditComponent(string name, Component newComponent)

{

int index = componentList.FindIndex(c => c.Name == name);

componentList[index] = newComponent;

InFile();

GetMinPrice();

}

public string GetAllComponents()

{

string allDetails = String.Empty;

foreach(Component component in componentList)

{

allDetails += component.ToString() + "\n";

}

return allDetails;

}

}

}

Файл Component.cs

using System.Text.Json.Serialization;

namespace server

{

internal class Component

{

public string Name { get; set; }

public string FactoryName { get; set; }

public double Price { get; set; }

public DateOnly DeliveryDate { get; set; }

public Component()

{

Name = String.Empty;

FactoryName = String.Empty;

Price = 0;

DeliveryDate = DateOnly.MinValue;

}

[JsonConstructor]

public Component(string name, string factoryName, double price, DateOnly deliveryDate)

{

Name = name;

FactoryName = factoryName;

Price = price;

DeliveryDate = deliveryDate;

}

public override string ToString()

{

return "Название: " + Name + "\n" +

"Завод: " + FactoryName + "\n" +

"Цена: " + Price + "\n" +

"Дата поставки: " + DeliveryDate + "\n";

}

}

}

Клиент

Файл Program.cs

using System.Net.Sockets;

using System.Text;

using TcpClient tcpClient = new TcpClient();

await tcpClient.ConnectAsync("127.0.0.1", 8888);

var stream = tcpClient.GetStream();

try

{

while (true)

{

byte[] response = new byte[3000];

stream.Read(response, 0, 3000);

string stringResponse = Encoding.UTF8.GetString(response).TrimEnd('\0');

Console.WriteLine(stringResponse);

string message = Console.ReadLine();

await stream.WriteAsync(Encoding.UTF8.GetBytes(message));

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

**Пример работы программы**

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 1. Пример работы клиента

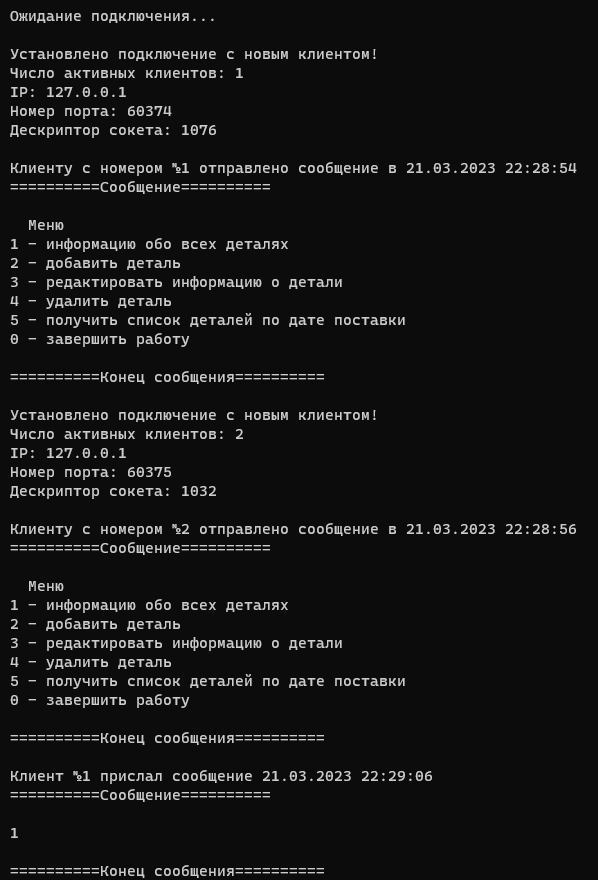


Рисунок 2. Пример работы сервера

# Выводы

Были изучены методы создания серверных приложений на основе установления логического соединения *TCP*, используя алгоритм многопоточной обработки запросов.