МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ОРЛОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИМЕНИ И.С. ТУРГЕНЕВА»

Кафедра программной инженерии

Отчет   
по лабораторной работе №2  
на тему: «Разработка сервера для клиент-серверного приложения с использованием платформы .NET»  
по дисциплине «Проектирование и архитектура программных систем»

Выполнили: Марочкин М.А. Шифр: 170584   
 Шорин В.Д. Шифр: 171406  
 Щекотихин С.Е. Шифр: 170590  
ИПАИТ  
Направление: 09.03.04 «Программная инженерия»  
Группа: 71-ПГ  
Проверил:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
Отметка о зачете:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата «\_\_\_\_» \_\_\_\_\_\_ 2019 г.

Орел, 2019 г.

## Задание на лабораторную работу

Спроектировать клиент-серверную архитектуру для приложения многопользовательского чата. Реализовать серверную часть приложения чата. Задачами сервера является подключение клиентов к чату, получение и рассылка сообщений клиентов между всеми подключёнными клиентами. Сервер должен организовать прослушивание заданного порта по которому к нему может подключаться несколько клиентов. Для обслуживания одновременно подключения от нескольких клиентов сервер должен использовать многопоточную архитектуру. Внутреннюю реализацию сервера рекомендуется сделать с использованием архитектурного шаблона MVC.

**Описание структуры программы**

**Model-View-Controller** (*MVC,* *«Модель-Представление-Контроллер», «Модель-Вид-Контроллер»*) — схема разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

* **Модель** (*Model*) предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.
* **Представление** (*View*) отвечает за отображение данных модели пользователю, реагируя на изменения модели.
* **Контроллер** (*Controller*) интерпретирует действия пользователя, оповещая модель о необходимости изменений.

Компонент «*Model*» реализован в двух файлах: «Client.cs» и «ClientArray.cs», которые хранят описание данных и методы, взаимодействующие с ними для клиента и массива клиентов соответственно.

Компонент «*View*» реализован в файле «View.cs», в котором описаны методы для отображения данных и любой другой информации.

Компонент «*Controller*» реализован в файле «Program.cs», где описана основная логика работы программы.

## Контрольные вопросы

1. **Перечислите основные разновидности клиент-серверной архитектуры.**

**Толстый клиент** (англ. fat client; rich client; heavy client; thick client; также, Rich-клиент) — в архитектуре клиент — сервер — это приложение, обеспечивающее расширенную функциональность независимо от центрального сервера. Часто сервер в этом случае является лишь хранилищем данных, а вся работа по обработке и представлению этих данных переносится на машину клиента.

**Однора́нговая, децентрализо́ванная,**или**пи́ринговая** (англ. peer-to-peer, P2P — равный к равному) сеть — это оверлейная компьютерная сеть, основанная на равноправии участников. Часто в такой сети отсутствуют выделенные серверы, а каждый узел (peer) является как клиентом, так и выполняет функции сервера. В отличие от архитектуры клиент-сервера, такая организация позволяет сохранять работоспособность сети при любом количестве и любом сочетании доступных узлов. Участниками сети являются все пиры.

**Сервер приложений** (англ. application server) — это программная платформа (фреймворк), предназначенная для эффективного исполнения процедур (программ, скриптов), на которых построены приложения. Сервер приложений действует как набор компонентов, доступных разработчику программного обеспечения через API (интерфейс прикладного программирования), определённый самой платформой.

1. **Опишите стек протоколов используемых при организации клиент-серверной архитектуры.**

**TCP/IP** — сетевая модель передачи данных, представленных в цифровом виде. Модель описывает способ передачи данных от источника информации к получателю. В модели предполагается прохождение информации через четыре уровня, каждый из которых описывается правилом (протоколом передачи). Наборы правил, решающих задачу по передаче данных, составляют стек протоколов передачи данных, на которых базируется Интернет

**HTTP** (англ. HyperText Transfer Protocol — «протокол передачи гипертекста») — протокол прикладного уровня передачи данных изначально — в виде гипертекстовых документов в формате «HTML», в настоящий момент используется для передачи произвольных данных. Основой HTTP является технология «клиент-сервер», то есть предполагается существование:

* Потребителей (клиентов), которые инициируют соединение и посылают запрос;
* Поставщиков (серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом.

**FTP** (англ. File Transfer Protocol) — протокол передачи файлов по сети. В отличие от TFTP, гарантирует передачу (либо выдачу ошибки) за счёт применения квитируемого протокола TCP. Стандартный порт управления FTP-соединением — 21. Типичное применение FTP-протокола — загрузка сайтов и других документов с частного устройства разработки на общедоступные сервера хостинга.

**DNS** (англ. Domain Name System «система доменных имён») — компьютерная распределённая система для получения информации о доменах. Чаще всего используется для получения IP-адреса по имени хоста (компьютера или устройства), получения информации о маршрутизации почты, обслуживающих узлах для протоколов в домене (SRV-запись).

1. **Какие компоненты платформы .NET используются для прослушивания входящих подключений на определённом порте?**

Пространство имен System.Net содержит сетевые классы для поиска IP-адресов, сетевой аутентификации, разрешений, отправки и получения данных. Рассмотрим эти классы, рассортировав их по группам.

**Поиск имен**

Чтобы получить IP-адрес из DNS-имени хоста или получить имя хоста из IP-адреса, можно использовать класс Dns. Класс DnsPermission представляет разрешение, необходимое для поиска имени. DnsPermissionAttribute — это класс атрибута, позволяющий отмечать сборки, классы и методы, нуждающиеся в этих полномочиях.

**IP-адреса**

IP-адреса обрабатываются в классе IPAddress. У одного хоста может быть несколько IP-адресов и алиасов. Вся эта информация содержится в классе IPHostEntry. Когда мы ищем имя, класс Dns возвращает объект типа IPHostEntry.

В основе межсетевых взаимодействий по протоколам TCP и UDP лежат сокеты. В .NET сокеты представлены классом System.NET.Sockets.Socket, который предоставляет низкоуровневый интерфейс для приема и отправки сообщений по сети.

Рассмотрим основные свойства данного класса:

* AddressFamily: возвращает все адреса, используемые сокетом. Данное свойство представляет одно из значений, определенных в одноименном перечислении AddressFamily. Перечисление содержит 18 различных значений, наиболее используемые:
* InterNetwork: адрес по протоколу IPv4
* InterNetworkV6: адрес по протоколу IPv6
* Available: возвращает объем данных, которые доступны для чтения
* Connected: возвращает true, если сокет подключен к удаленному хосту
* LocalEndPoint: возвращает локальную точку, по которой запущен сокет и по которой он принимает данные
* ProtocolType: возвращает одно из значений перечисления ProtocolType, представляющее используемый сокетом протокол. Есть следующие возможные значения:
  + IPv4
  + IPv6
  + Tcp
  + Udp

И другие.

Каждое значение представляет соответствующий протокол, но наиболее используемыми являются Tcp и Udp.

* RemoteEndPoint: возвращает адрес удаленного хоста, к которому подключен сокет

При работе с сокетами вне зависимости от выбранных протоколов мы будем опираться на методы класса Socket:

* Accept(): создает новый объект Socket для обработки входящего подключения
* Bind(): связывает объект Socket с локальной конечной точкой
* Close(): закрывает сокет
* Connect(): устанавливает соединение с удаленным хостом
* Listen(): начинает прослушивание входящих запросов
* Poll(): определяет состояние сокета
* Receive(): получает данные
* Send(): отправляет данные
* Shutdown(): блокирует на сокете прием и отправку данных

1. **Что такое потоки в программировании?**

**Поток данных** (англ. stream) в программировании — абстракция, используемая для чтения или записи файлов, сокетов и т. п. в единой манере.

Потоки являются удобным унифицированным программным интерфейсом для чтения или записи файлов (в том числе специальных и, в частности, связанных с устройствами), сокетов и передачи данных между процессами. При запуске процесса ему предоставляются предопределённые стандартные потоки. Возможность перенаправления потоков позволяет связывать различные программы, и придаёт системе гибкость.

1. **Как реализуются потоки в языке C#?**

Основной функционал для использования потоков в приложении сосредоточен в пространстве имен System.Threading. В нем определен класс, представляющий отдельный поток - класс Thread.

Класс Thread определяет ряд методов и свойств, которые позволяют управлять потоком и получать информацию о нем. Основные свойства класса:

* Статическое свойство CurrentContext позволяет получить контекст, в котором выполняется поток
* Статическое свойство CurrentThread возвращает ссылку на выполняемый поток
* Свойство IsAlive указывает, работает ли поток в текущий момент
* Свойство Name содержит имя потока
* Свойство Priority хранит приоритет потока - значение перечисления ThreadPriority
* Свойство ThreadState возвращает состояние потока - одно из значений перечисления ThreadState

Некоторые методы класса Thread:

* Статический метод GetDomain возвращает ссылку на домен приложения
* Статический метод GetDomainId возвращает id домена приложения, в котором выполняется текущий поток
* Статический метод Sleep останавливает поток на определенное количество миллисекунд
* Метод Abort уведомляет среду CLR о том, что надо прекратить поток, однако прекращение работы потока происходит не сразу, а только тогда, когда это становится возможно. Для проверки завершенности потока следует опрашивать его свойство ThreadState
* Метод Interrupt прерывает поток на некоторое время
* Метод Resume возобновляет работу ранее приостановленного потока
* Метод Start запускает поток
* Метод Suspend приостанавливает поток

**Листинг программы**

«Program.cs»

using System;using System.Collections.Generic;using System.Text;using System.Net;using System.Net.Sockets;

using System.Threading;using Server.Models;

namespace Server {

internal class Program {

private static void SendAll(string clientMessage, string clientName) {

byte[] data = new byte[1024];

data = Encoding.UTF8.GetBytes(clientName + ": " + clientMessage);

View.LogClientSendMessage(clientMessage, clientName);

foreach (var client in ClientArray.GetClientsArray()) {

Socket clientSocket = client.GetClientSocket();

clientSocket.Send(data);

}

}

private static void ListenToClient() {

Client client = ClientArray.GetClient(ClientArray.CountOfClients() - 1);

byte[] data = new byte[1024];

string message;

string clientName;

string clientMessage;

Socket clientSocket = client.GetClientSocket();

int dataSize;

try {

clientSocket.Send(Encoding.UTF8.GetBytes("Nickname"));

dataSize = clientSocket.Receive(data);

message = Encoding.UTF8.GetString(data, 0, dataSize);

client.SetClientName(message);

clientName = message;

View.LogClientConnected(clientName);

while (true) {

dataSize = clientSocket.Receive(data);

clientMessage = Encoding.UTF8.GetString(data, 0 ,dataSize);

SendAll(clientMessage, clientName);

}

}

catch (Exception){View.LogClientDisconnected(client.GetClientName());

ClientArray.RemoveClient(client);

}

}

public static void Main(string[] args) {

try {

IPAddress address = Dns.GetHostEntry("127.0.0.1").AddressList[0];

Socket listener = new Socket(address.AddressFamily,SocketType.Stream,ProtocolType.Tcp);

IPEndPoint endPoint = new IPEndPoint(address, 7777);

listener.Bind(endPoint); listener.Listen(1); View.LogStart();

while (true) {

Socket client = listener.Accept(); View.LogNewClientConnected();

ClientArray.AddClient(client); Thread newThread = new Thread(ListenToClient);

newThread.Start();

}

}

catch (Exception e) { View.LogException(e); throw; }

}

}

}

«View.cs»

using System;using System.Collections.Generic;using System.Linq;using System.Text;

namespace Server{

class View {

public static void LogClientSendMessage(string clientMessage, string clientName)

{ Console.WriteLine(clientName + ": " + clientMessage); }

public static void LogClientConnected(string clientName)

{ Console.WriteLine(clientName + " подключен!"); }

public static void LogClientDisconnected(string clientName)

{ Console.WriteLine("Client " + clientName + " disconnected!"); }

public static void LogStart() { Console.WriteLine("Start listening"); }

public static void LogNewClientConnected() { Console.WriteLine("New client"); }

public static void LogException(Exception e) { Console.WriteLine(e); }

}

}

«Client.cs»

using System.Net.Sockets;

namespace Server {

public class Client {

private Socket client; private string \_clientName;

public Client(Socket client) { this.client = client; }

public void SetClientName(string clientName) { \_clientName = clientName; }

public Socket GetClientSocket() { return client; }

public string GetClientName() { return \_clientName; }

}

}

«ClientArray.cs»

using System;using System.Collections.Generic;using System.Linq;using System.Text;using System.Net.Sockets;

namespace Server.Models{

class ClientArray {

private static List<Client> clients = new List<Client>();

public static Client GetClient(int numClient) { return clients[numClient]; }

public static void AddClient(Socket client) { clients.Add(new Client(client)); }

public static void RemoveClient(Client client) { clients.Remove(client); }

public static int CountOfClients() { return clients.Count; }

public static List<Client> GetClientsArray() { return clients; }

}

}