Лабораторная работа №5:
Клиент-серверные приложения

Цель:

Целью данной работы является получение навыков работы с клиент-серверными приложениями на языке высокого уровня C# в среде программирования Microsoft Visual Studio 2022 Community

Задание:

Разработать программное средство, состоящее и двух приложений:

1) Сервер.
2) Клиент.

Приложения должны иметь WPF интерфейс. Сервер должен поддерживать подключение более двух клиентов.

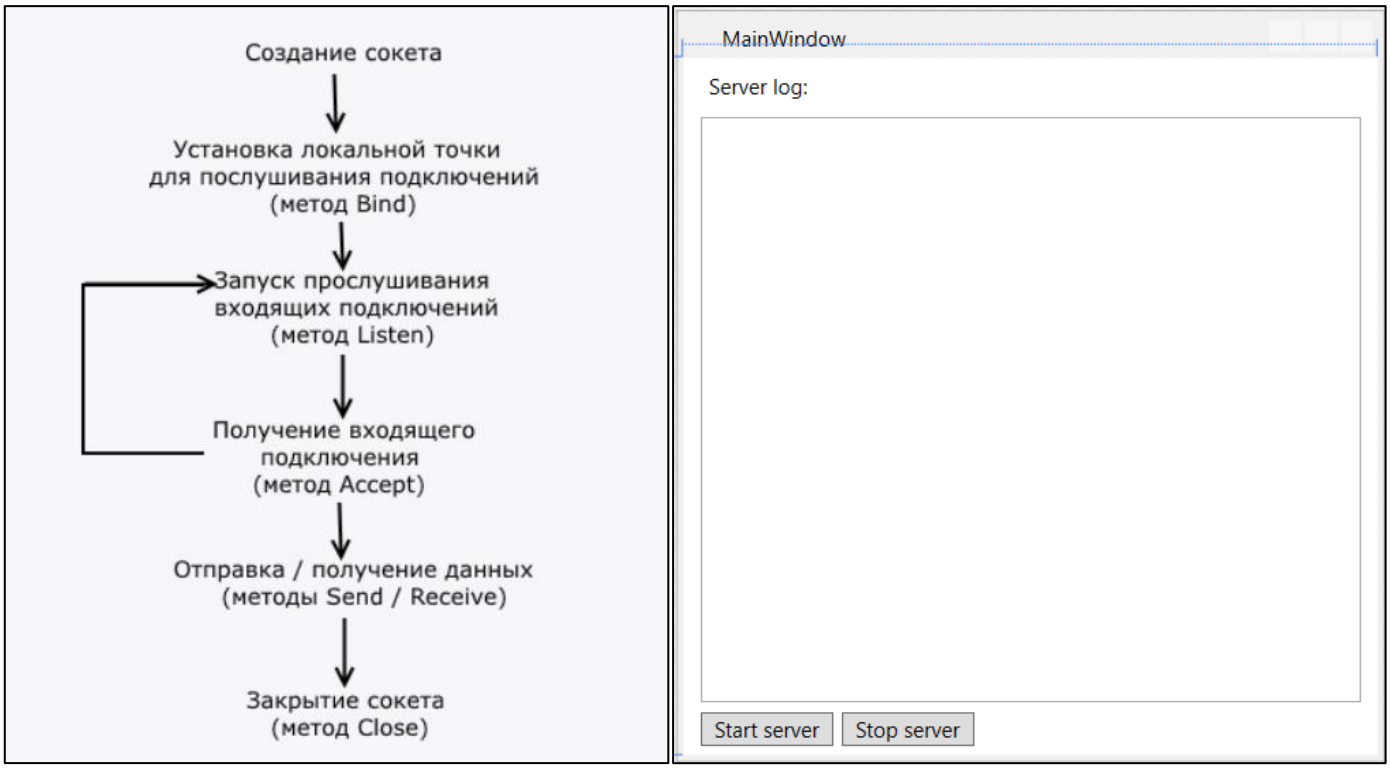
Алгоритм работы программного средства:

1) Запуск сервера.
2) Запуск клиента.
3) Указание имени пользователя.
4) Подключение к серверу.
5) Отправка сообщения на сервер.
6) Сервер получает сообщение, переписывает задом наперёд и отправляет обратно.
7) Клиент получает инвертированное сообщение от сервера.
8) Повторение шага 5 пока не надоест.
9) Отключение от сервера.

Дополнительное задание: реализовать обмен сообщениями между клиентами, подключёнными к серверу.

Справочная информация:

Общая схема работы TCP сервера и его интерфейс выглядят следующим образом:



Где: Server log – объект типа ListBox который будет использоваться для вывода информации и сообщений, а кнопки Start и Stop для запуска и остановки работы сервера.

Для работы с TCP сервером, вам понадобится подключить библиотеки для работы с сетью:

```
//подключение библиотек для работы с сетью и потоками
using System.Net;
using System.Net.Sockets;
using System.Threading;
```

после чего объявить глобальные переменные:

```
//прослушиваемый порт
int port = 8888;
//объект, прослушивающий порт
static TcpListener listener;
```

Само по себе создание объекта, прослушивающего порт, выглядит следующим образом:

```
//создание объекта для отслеживания сообщений переданных с ip адреса через порт
listener = new TcpListener(IPAddress.Parse("127.0.0.1"), port);
//начало прослушивания
listener.Start();
```

Создание такого объекта, логичнее всего привязать на кнопку “Start server”.

После того, как объект, прослушивающий порт создан, необходимо начать устанавливать подключения, запрос на которые будет отлавливать этот объект. Вынести этот процесс лучше всего в отдельную функцию вида:

```
//функция ожидания и приёма запросов на подключение
void listen()
{
    //цикл подключения клиентов
    while (true)
    {
        //принятие запроса на подключение
```

```
        TcpClient client = listener.AcceptTcpClient();

        log.Items.Add("Новый клиент подключен.");
    }
}
```

Обратите внимание, что функция содержит бесконечный цикл, а значит, при вызове, программа перестанет отвечать на любые другие действия пользователя. Что бы этого избежать, можно создать отдельный поток для выполнения этой функции и добавить его в обработчик Start server:

```
//создание нового потока для ожидания и подключения клиентов
Thread listenThread = new Thread(() => listen());
listenThread.Start();
```

В результате, обработка функции прослушивания будет выполняться в отдельном потоке и не будет влиять на работу основной программы. Однако, это приведёт к другой проблеме. Попытка вывести сообщение в ListBox, приведёт к ошибке. Это происходит потому, что одновременно может быть запущенно множество потоков и все они могут попытаться получить доступ к ListBox одновременно, что приведёт к попытке одновременной записи множества данных. Для того, чтобы решить эту проблему, можно воспользоваться стандартным механизмом добавления в очередь выполнения операций:

```
Dispatcher.BeginInvoke(new Action(() => log.Items.Add("Новый клиент подключен.")));
```

Таким образом, даже если множество потоков попытаются одновременно что-то записать в компонент, из запросов на запись будет сформирована очередь и все они будут выполнены последовательно.

Следующим этапом реализации сервера, будет описание функции обработки сообщений от клиента:

```
//обработка сообщений от клиента
public void Process(TcpClient tcpClient)
{
    TcpClient client = tcpClient;
    NetworkStream stream = null;    //получение канала связи с клиентом

    try    //означает что в случае возникновении ошибки, управление перейдёт к блоку catch
    {
        //получение потока для обмена сообщениями
        stream = client.GetStream();    //получение канала связи с клиентом

        // буфер для получаемых данных
        byte[] data = new byte[64];

        //цикл ожидания и отправки сообщений
        while (true)
        {
            //==========================получение сообщения============================
            //объект, для формирования строк
            StringBuilder builder = new StringBuilder();
            int bytes = 0;
            //до тех пор, пока в потоке есть данные
            do
            {
                //из потока считываются 64 байта и записываются в data начиная с 0
                bytes = stream.Read(data, 0, data.Length);
                //из считанных данных формируется строка
                builder.Append(Encoding.Unicode.GetString(data, 0, bytes));
            }
            while (stream.DataAvailable);
            //преобразование сообщения
            string message = builder.ToString();
            //вывод сообщения в лог сервера
            Dispatcher.BeginInvoke(new Action(() => log.Items.Add(message)));
```

```
            //=========================отправка сообщения============================
            //преобразование сообщения в набор байт
            data = Encoding.Unicode.GetBytes(message);
            //отправка сообщения обратно клиенту
            stream.Write(data, 0, data.Length);
        }
    }
    catch (Exception ex)    //если возникла ошибка, вывести сообщение об ошибке
    {
        Dispatcher.BeginInvoke(new Action(() => log.Items.Add(ex.Message)));
    }
    finally //после выхода из бесконечного цикла
    {
        //освобождение ресурсов при завершении сеанса
        if (stream != null)
            stream.Close();
        if (client != null)
            client.Close();
    }
}
```

Блок try/catch – используется для обработки неожиданного разрыва подключения.

В завершении, функцию обработки сообщений от клиентов, необходимо вызвать для каждого подключившегося клиента:

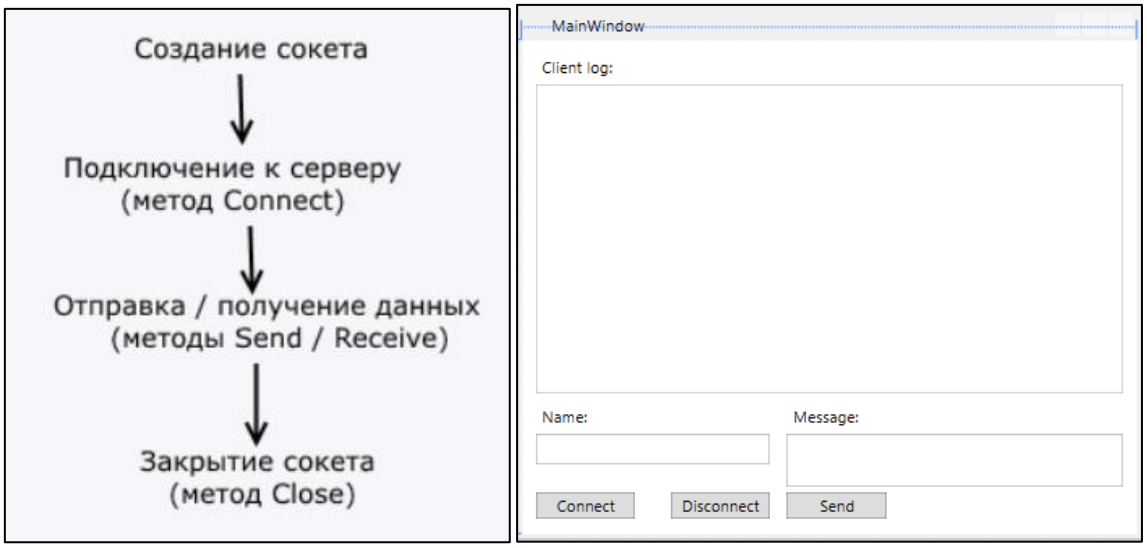
```
void listen()
{
    //цикл подключения клиентов
    while (true)
    {
        //принятие запроса на подключение
        TcpClient client = listener.AcceptTcpClient();

        Dispatcher.BeginInvoke(new Action(() => log.Items.Add("Новый клиент подключен.")));

        //создание нового потока для обслуживания нового клиента
        Thread clientThread = new Thread(() => Process(client));
        clientThread.Start();
    }
}
```

Обработчик кнопки Stop server будет содержать корректное завершение работы объекта прослушивания и, желательно, корректно закрывать каналы соединения с клиентами.

Реализацию метода предлагается описать самостоятельно.

Если всё было сделано правильно, то в результате должен получиться TCP сервер, принимающий сообщения от клиентов и отправляющий им их обратно.

Общая схема работы интерфейс TCP клиента выглядят следующим образом:



Где Client log – объект типа ListBox для вывода сообщений, Name – поле типа TextBox для ввода имени пользователя, Message - поле типа TextBox для ввода сообщений, Connect – кнопка подключения к серверу, Disconnect – кнопка отключения от сервера и Send – кнопка отправки сообщения.

Для работы TCP клиента вам понадобится тот же набор библиотек:

```
//подключение библиотек для работы с сетью и потоками
using System.Net;
using System.Net.Sockets;
using System.Threading;
```

И глобальные переменные:

```
//номер порта для обмена сообщениями
int port = 8888;
//ip адрес сервера
string address = "127.0.0.1";
//объявление TCP клиента
TcpClient client = null;
//объявление канала соединения с сервером
NetworkStream stream = null;
//имя пользователя
string username = “”;
```

В обработчике нажатия кнопки Connect, будет получение имени пользователя и попытка установки соединения с сервером:

```
//получение имени пользователя
userName = name.Text;
try     //если возникнет ошибка - переход в catch
{
    //создание клиента
    client = new TcpClient(address, port);
    //получение канала для обмена сообщениями
    stream = client.GetStream();
}
catch (Exception ex)
{
    log.Items.Add(ex.Message);
}
```

Блок try/catch – используется на случай возникновении ошибки при попытке подключения к серверу.

Функция ожидания сообщений от сервера реализуется по тому же принципу что и функция обработки клиентов сервера:

```
//функция ожидания сообщений от сервера
void listen()
{
    try     //в случае возникновения ошибки - переход к catch
    {
        //цикл ожидания сообщениями
        while (true)
        {
            //буфер для получаемых данных
            byte[] data = new byte[64];
            //объект для построения смтрок
            StringBuilder builder = new StringBuilder();
            int bytes = 0;
            //до тех пор, пока есть данные в потоке
            do
            {
                //получение 64 байт
                bytes = stream.Read(data, 0, data.Length);
                //формирование строки
                builder.Append(Encoding.Unicode.GetString(data, 0, bytes));
            }
            while (stream.DataAvailable);
            //получить строку
            string message = builder.ToString();
            //вывод сообщения в лог клиента
            Dispatcher.BeginInvoke(new Action(() => log.Items.Add("Сервер: " + message)));
        }
    }
    catch (Exception ex)
    {
        //вывести сообщение об ошибке
        log.Items.Add(ex.Message);
    }
    finally
    {
        //закрыть канал связи и завершить работу клиента
        stream.Close();
        client.Close();
    }
}
```

После чего, её можно вызвать в обработчике Connect отдельным потоком:

```
//создание нового потока для ожидания сообщения от сервера
Thread listenThread = new Thread(() => listen());
listenThread.Start();
```

Отправка сообщений будет находится в обработчике Send и может выглядеть следующим образом:

```
//получение сообщения
string message = msg.Text;
//добавление имени пользователя к сообщению
message = String.Format("{0}: {1}", userName, message);
//преобразование сообщение в массив байтов
byte[] data = Encoding.Unicode.GetBytes(message);
//отправка сообщения
stream.Write(data, 0, data.Length);
```

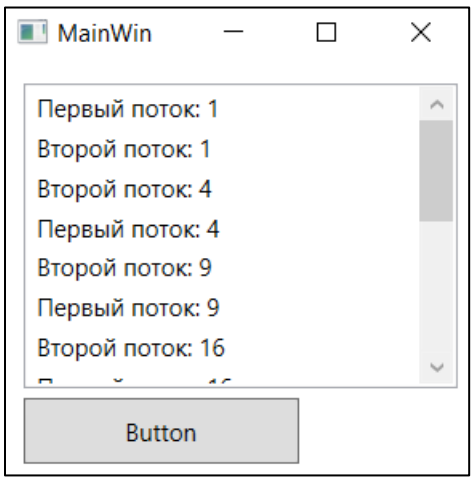
Обработчик Disconnect будет содержать корректное закрытие подключения к серверу. Предлагается реализовать его самостоятельно.

Пример работы с потоками:

Пример простой WPF программы использующей параллельный процесс (поток):

```
using System.Threading;

namespace WpfThreads
{
    public partial class MainWindow : Window
    {
        public MainWindow()
        {
            InitializeComponent();
        }

        private void Button_Click(object sender, RoutedEventArgs e)
        {
            //создание нового потока из функции Count1
            Thread myThread1 = new Thread(new ThreadStart(Count1));
            myThread1.Start(); //запуск потока
            //создание нового потока из функции Count2
            Thread myThread2 = new Thread(new ThreadStart(Count2));
            myThread2.Start(); //запуск потока
        }

        //функция, вызываемая в потоке
        public void Count2()
        {
            for (int j = 1; j < 9; j++)
            {
                //запрос на добавление строки в listBox
                Dispatcher.BeginInvoke(new Action(
                                        () => listBox.Items.Add("Второй поток: " + j * j)));
                Thread.Sleep(100);
            }
        }
        public void Count1()
        {
            for (int j = 1; j < 9; j++)
            {
                //запрос на добавление строки в listBox
                Dispatcher.BeginInvoke(new Action(
                                        () => listBox.Items.Add("Первый поток: " + j * j)));
                Thread.Sleep(100);
            }
        }
    }
}
```

Результат:



Приложение1: листинг консольного TCP сервера

```
//подключение библиотек для работы с сетью и потоками
using System.Net;
using System.Net.Sockets;
using System.Threading;

namespace ConsoleServer
{
    class Program
    {
        //прослушиваемый порт
        const int port = 8888;
        //объект, прослушивающий порт
        static TcpListener listener;

        static void Main(string[] args)
        {
            try
            {
                //создание объекта для отслеживания сообщений
                //переданных с ip адреса через порт
                listener = new TcpListener(IPAddress.Parse("127.0.0.1"), port);
                //начало прослушивания
                listener.Start();
                Console.WriteLine("Ожидание подключений...");
                //цикл подключения клиентов
                while (true)
                {
                    //принятие запроса на подключение
                    TcpClient client = listener.AcceptTcpClient();

                    //создание нового потока для обслуживания нового клиента
                    Thread clientThread = new Thread(() => Process(client));
                    clientThread.Start();
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            //при завершении программы, прекращение отслеживания сообщений
            finally
            {
                if (listener != null)
                    listener.Stop();
            }
        }

        //функция обработки сообщений от клиента
        public static void Process(TcpClient tcpClient)
        {
            TcpClient client = tcpClient;
            NetworkStream stream = null;
            try
            {
                //получение потока для обмена сообщениями
                stream = client.GetStream();

                // буфер для получаемых данных
                byte[] data = new byte[64];

                //цикл обработки сообщений
                while (true)
                {
                    //объект, для формирования строк
```

```
                    StringBuilder builder = new StringBuilder();
                    int bytes = 0;
                    //до тех пор, пока в потоке есть данные
                    do
                    {
                        //из потока считываются 64 байта и записываются в data
                        bytes = stream.Read(data, 0, data.Length);
                        //из считанных данных формируется строка
                        builder.Append(Encoding.Unicode.GetString(data, 0, bytes));
                    }
                    while (stream.DataAvailable);

                    //преобразование сообщения
                    string message = builder.ToString();
                    //вывод сообщения в консоль сервера
                    Console.WriteLine(message);
                    //преобразование сообщения в набор байт
                    data = Encoding.Unicode.GetBytes(message);
                    //отправка сообщения обратно клиенту
                    stream.Write(data, 0, data.Length);
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
                //освобождение ресурсов при завершении сеанса
                if (stream != null)
                    stream.Close();
                if (client != null)
                    client.Close();
            }
        }
    }
}
```

Приложение2: листинг консольного TCP клиента

```
using System.Net.Sockets;

namespace ConsoleClient
{
    class Program
    {
        //номер порта для обмена сообщениями
        const int port = 8888;
        //ip адрес сервера
        const string address = "127.0.0.1";

        static void Main(string[] args)
        {
            Console.Write("Введите свое имя:");
            string userName = Console.ReadLine();

            //объявление TCP клиента
            TcpClient client = null;
            try
            {
                //создание клиента
                client = new TcpClient(address, port);
                //получение потока для обмена сообщениями
                NetworkStream stream = client.GetStream();
                //цикл обмена сообщениями
                while (true)
```

```
                {
                    Console.Write(userName + ": ");
                    //ввод сообщения
                    string message = Console.ReadLine();
                    message = String.Format("{0}: {1}", userName, message);
                    //преобразование сообщение в массив байтов
                    byte[] data = Encoding.Unicode.GetBytes(message);
                    //отправка сообщения
                    stream.Write(data, 0, data.Length);
                    //буфер для получаемых данных
                    data = new byte[64];
                    StringBuilder builder = new StringBuilder();
                    int bytes = 0;
                    //до тех пор пока есть данные в потоке
                    do
                    {
                        //получение 64 байт
                        bytes = stream.Read(data, 0, data.Length);
                        //формирование строки
                        builder.Append(Encoding.Unicode.GetString(data, 0, bytes));
                    }
                    while (stream.DataAvailable);

                    message = builder.ToString();
                    Console.WriteLine("Сервер: {0}", message);
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.Message);
            }
            finally
            {
                client.Close();
            }
        }
    }
}
```

Список литературы:

Класс TcpListener:

https://msdn.microsoft.com/ru-ru/library/system.net.sockets.tcplistener(v=VS.71).aspx

Класс TcpClient:

https://msdn.microsoft.com/ru-ru/library/system.net.sockets.tcpclient(v=VS.71).aspx

Способы асинхронного выполнения задач:

Класс Thread:

https://msdn.microsoft.com/ru-ru/library/system.threading.thread(v=vs.110).aspx

Класс BackgroundWorker:

https://msdn.microsoft.com/ru-ru/library/system.componentmodel.backgroundworker.aspx

Класс Task:

https://msdn.microsoft.com/ru-ru/library/system.threading.tasks.task(v=vs.110).aspx