**TASK 2**

**CLIENT**

using System;

using System.Net;

using System.Net.Sockets;

using System.Text;

namespace SSE.Client

{

/// <summary>

/// Simple implementation of a TCP client.

/// </summary>

public class TcpClient

{

/// <summary>

/// Sends a string to a TCP endpoint, waits for an answer and returns it.

/// </summary>

/// <param name="ip">The IP address to send the request to.</param>

/// <param name="port">The target port to send the request to.</param>

/// <param name="message">The data to be sent as a string.</param>

/// <returns>The data received as an answer.</returns>

public string Request(string ip, int port, string message)

{

// create socket

var socket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);

Console.WriteLine("TCP: Created socket");

// connect

var ipe = new IPEndPoint(IPAddress.Parse(ip), port);

try

{

socket.Connect(ipe);

}

catch (SocketException e)

{ // connection failed

Console.WriteLine("TCP: The connection failed ({0}).", e.Message);

return "";

}

Console.WriteLine("TCP: connection established between {0} and {1}.",

socket.LocalEndPoint, socket.RemoteEndPoint);

// send request

var sendData = Encoding.ASCII.GetBytes(message);

socket.Send(sendData, sendData.Length, SocketFlags.None);

Console.WriteLine("TCP: Sent {0} bytes.", sendData.Length);

// receive everything

var receiveBuffer = new byte[10000];

var result = "";

while (true)

{

var receivedBytes = socket.Receive(receiveBuffer, receiveBuffer.Length, SocketFlags.None);

if (receivedBytes == 0) // connection closed by remote host and

break; // all available data has been received

result += Encoding.ASCII.GetString(receiveBuffer, 0, receivedBytes);

Console.WriteLine("TCP: Received {0} bytes.", receivedBytes);

}

// shutdown connection and close socket

Console.WriteLine("TCP: Shutting down connection with {0} in both directions.",

socket.RemoteEndPoint);

socket.Shutdown(SocketShutdown.Both);

return result;

}

/// <summary>

/// Program entry point for testing purposes.

/// </summary>

public static void Main(string[] args)

{

// create client

var client = new TcpClient();

// send a request to a locally running server program

var answer = client.Request("127.0.0.1", 13000, "ask NUMBER\_OF\_PROCESSORS\n");

// display answer and wait for user

Console.WriteLine("Answer: " + answer);

Console.ReadLine();

}

}

}

**SERVER**

**using System;**

**using System.Net;**

**using System.Net.Sockets;**

**using System.Text;**

**namespace SSE.Server**

**{**

**public class TcpServer**

**{**

**/// <summary>**

**/// The socket used for handling an established connection.**

**/// </summary>**

**private Socket \_connectionSocket;**

**/// <summary>**

**/// Sends the specified data back via the connectionSocket.**

**/// </summary>**

**private void SendString(string message)**

**{**

**var sendBuffer = Encoding.UTF8.GetBytes(message);**

**\_connectionSocket.Send(sendBuffer, sendBuffer.Length, SocketFlags.None);**

**Console.WriteLine("TCP: Sent answer message, {0} bytes to {1}.", sendBuffer.Length,**

**\_connectionSocket.RemoteEndPoint);**

**}**

**/// <summary>**

**/// Shuts down the connectionSocket.**

**/// </summary>**

**private void CloseCurrentConnection()**

**{**

**Console.WriteLine("TCP: Shutting down connection with {0} in both directions.",**

**\_connectionSocket.RemoteEndPoint);**

**// shut down**

**\_connectionSocket.Shutdown(SocketShutdown.Both);**

**}**

**/// <summary>**

**/// Waits for bytes arriving on connectionSocket and handles them.**

**/// </summary>**

**private void ReceiveBytes()**

**{**

**Console.WriteLine("TCP: Waiting for bytes from {0}.",**

**\_connectionSocket.RemoteEndPoint);**

**var request = "";**

**while (true)**

**{**

**// receive chunk of bytes**

**var receiveBuffer = new byte[10000];**

**var receivedBytes = \_connectionSocket.Receive(receiveBuffer, receiveBuffer.Length, SocketFlags.None);**

**Console.WriteLine("TCP: Received {0} bytes from {1}.",**

**receivedBytes, \_connectionSocket.RemoteEndPoint);**

**// add received bytes to input buffer**

**request += Encoding.ASCII.GetString(receiveBuffer, 0, receivedBytes);**

**// see if there are any new lines**

**while (request.IndexOf('\n') != -1)**

**{**

**// separate line from buffer**

**var line = request.Substring(0, request.IndexOf('\n'));**

**request = request.Substring(request.IndexOf('\n') + 1);**

**// trim all \r at end of line**

**line = line.TrimEnd('\r');**

**// handle line**

**var answer = ReceiveLine(line);**

**// if the line caused the server to generate an answer, send it back**

**if (answer != null)**

**{**

**SendString(answer);**

**CloseCurrentConnection();**

**return;**

**}**

**}**

**}**

**}**

**/// <summary>**

**/// Starts the HTTP server, which keeps on running until the program is aborted.**

**/// </summary>**

**/// <param name="ip">The IP address to listen tp.</param>**

**/// <param name="port">The port to listen to.</param>**

**public void Run(string ip, int port)**

**{**

**Console.WriteLine("TCP: Starting server.");**

**// create listening socket**

**var listeningSocket = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);**

**Console.WriteLine("TCP: Created listening socket.");**

**// bind**

**var endpoint = new IPEndPoint(IPAddress.Parse(ip), port);**

**listeningSocket.Bind(endpoint);**

**Console.WriteLine("TCP: Bound to {0}.", endpoint.ToString());**

**// listen**

**listeningSocket.Listen(10);**

**Console.WriteLine("TCP: Start listening.");**

**Console.WriteLine();**

**// accept loop**

**while (true)**

**{**

**// accept new connection**

**var newSocket = listeningSocket.Accept();**

**Console.WriteLine("TCP: Connection established with {0} over socket {1}.",**

**newSocket.RemoteEndPoint.ToString(), newSocket.LocalEndPoint.ToString());**

**// receive bytes and handle them**

**\_connectionSocket = newSocket;**

**// create a new instance of the current server class**

**ReceiveBytes();**

**}**

**}**

**/// <summary>**

**/// Handles an incoming line of text.**

**/// </summary>**

**/// <param name="line">Incoming data.</param>**

**/// <returns>The answer to be sent back as a reaction to the received line or null.</returns>**

**protected virtual string ReceiveLine(string line)**

**{**

**// for testing purposes: wait for 10s and echo back received line**

**Console.WriteLine("TCP: Simulate 10s processing time.");**

**// simulate 10s processing time**

**var start = DateTime.Now.Ticks;**

**while (DateTime.Now.Ticks - start < 100000000) { };**

**// example application: provide environment information**

**if (line.StartsWith("ask "))**

**return Environment.GetEnvironmentVariable(line.Substring(4));**

**return "What?";**

**}**

**/// <summary>**

**/// Program entry point for testing purposes.**

**/// </summary>**

**/// <param name="args"></param>**

**public static void Main(string[] args)**

**{**

**var server = new TcpServer();**

**server.Run("127.0.0.1", 13000);**

**Console.WriteLine("Press any key to close...");**

**Console.ReadKey();**

**}**

**}**

**}**

**TASK 3**

// Course Material "Protokolle Verteilte Systeme"

// (c) 2008 by Professur Verteilte und Selbstorganisierende Rechnersysteme, TUC

using System;

using System.Threading;

namespace Vsr.Teaching.Pvs.Sample

{

/// <summary>

/// A sample program that demonstrates the use of threads.

/// </summary>

class ThreadExample

{

static void Method()

{

for (int i = 0; i < 10; i++)

{

Console.Write("b");

// wait for 0.1s

Thread.Sleep(100);

}

}

/// <summary>

/// The main program routine.

/// </summary>

static void Main(string[] args)

{

// start a parallel thread

Thread t = new Thread(Method);

t.Start();

for (int i = 0; i < 10; i++)

{

Console.Write("a");

// wait for 0.1s

Thread.Sleep(100);

}

// wait for thread to end

t.Join();

Console.ReadLine();

}

}

}