Sistemas Distribuídos

TP1 - Cobertura em Wholesalers -

Serviço de cobertura de WholeSaler

Hugo Paredes | José Cunha | Dennis Paulino

Trabalho realizado por:

Daniel Filipe Morais Oliveira, al74575

José Eduardo Monteiro Cosme, al74139

Pedro Jorge Cunha Oliveira, al73346

1 – Protocolo

Descrição do protocolo de comunicação cliente/servidor: O protocolo de comunicação utilizado para efetuar a comunicação entre cliente e servidor neste projeto é o protocolo TCP (*Transmission Control Protocol*). A conexão pretendida estabelece-se através de um processo deste protocolo, conhecido como '*TCP 3-Way Handshake Process*'. Este processo é composto como o nome indica por 3 etapas:

1ª etapa: Nesta etapa inicial o cliente pretende conectar-se com o servidor, desta forma envia para o servidor um segmento de pacote SYN (*Synchronize*) informando que pretende iniciar um processo de comunicação.

2ª etapa: Realizado o primeiro processo para o estabelecimento de uma conexão, o servidor verifica o pedido do cliente, e se estiver disponível, envia um segmento SYN&ACK (*Synchronize* & *Acknowledgement*), confirmando assim a disponibilidade para dar resposta ao pedido do cliente, indicando que o segmento ACK é destinado especificamente para o SYN do pedido do cliente.

3ª etapa: Na última fase para a confirmação da conexão, o cliente envia um último segmento ACK no sentido de corresponder ao estabelecimento de contacto com o servidor, concluindo assim o *handshake* de três vias, e podendo dar início à transmissão de informação.

A implementação deste protocolo em C# recorre às classes TcpClient e TcpListener, das bibliotecas da *framework* .NET, para criar um cliente e um servidor TCP, respetivamente. O TcpClient é responsável pela conexão do cliente com um host remoto criando um *socket* para a efetivação da conexão, e permitindo desta forma a troca de dados com o servidor. O TcpListener por sua vez responsabiliza-se pela monitorização de uma determinada porta TCP onde irão decorrer as solicitações do cliente, criando um *socket* para cada conexão, permitindo a troca de dados com o cliente através do mesmo.

Referências

[1] - GeeksforGeeks. (2021, Outubro 26). *TCP 3-way handshake process*. Acessado a Abril 9, 2023. https://www.geeksforgeeks.org/tcp-3-way-handshake-process/

[2] - Microsoft. (2023, Março 02). *Three-way handshake via TCP/IP*. Acessado a Abril 9, 2023. https://learn.microsoft.com/pt-br/troubleshoot/windows-server/networking/three-way-handshake-via-tcpip

[3] - Microsoft. (2022, Dezembro 06). *Visão geral do TCP*. Acessado a Abril 10, 2023. https://learn.microsoft.com/pt-br/dotnet/fundamentals/networking/sockets/tcp-classes

[4] - Microsoft. (2023, MarÇO 07). *Usar soquetes para enviar e receber dados por TCP*. Acessado a Abril 9, 2023. https://learn.microsoft.com/pt-br/dotnet/fundamentals/networking/sockets/socket-services

2 – Implementação

Atendimento dos clientes/Comunicação com cada cliente:

Na classe Servidor, encontra-se uma secção de código que inicializa uma lista do tipo TcpClient com o nome ‘clientes’ para armazenar todos os clientes que se conectem ao servidor.

```
static List<TcpClient> clientes = new List<TcpClient>();
```

Na função principal, define-se o IP e o número da porta do servidor, criando-se em seguida um objeto TcpListener para aguardar por conexões para o endereço IP e porta que definimos anteriormente.

```
string localIP = "127.0.0.1";
int localPort = 8888;

TcpListener ServerSocket = new TcpListener(IPAddress.Parse(localIP), localPort);
ServerSocket.Start();
Console.WriteLine("Server waiting...");
```

Para aguardar e conectar-se com clientes, é implementado um ciclo while que aguarda a chegada de um cliente. Neste ciclo, o objeto TcpListener associa-se ao método ‘AcceptTcpClient’, que bloqueia a execução até existir comunicação com um cliente, retornando um objeto TcpClient ‘client’ que representa a conexão com o mesmo.

```
while (true)
{
    TcpClient client = ServerSocket.AcceptTcpClient();
    Console.WriteLine("100 OK");
    Console.WriteLine("Client {0} connected", client.Client.RemoteEndPoint);
```

A criação de uma thread permite lidar com o cliente conectado, passando o método ‘HandleClient’, responsável pelas comunicações com o cliente, para a thread.

```
Thread clientThread = new Thread(() => HandleClient(client));
clientThread.Start();
```

Garantia de processamento dos ficheiros de cobertura:

O código que lida com o processamento dos ficheiros de cobertura é chamado quando a mensagem do cliente começa com a string ‘OWNER’.

```
if (message.StartsWith("OWNER"))
```

Para a interpretação do ficheiro de cobertura, que é introduzido através da filepath do ficheiro, é necessário código para ler o ficheiro e fazer o seu armazenamento, através de uma lista de ficheiros que permite guardar informações de diferentes ficheiros.

```
string[] lines = message.Split(new[] { "\r\n", "\r", "\n" }, StringSplitOptions.None);

string cabecalho = lines[0];

List<Ficheiro> ficheiros = new List<Ficheiro>();

foreach (string line in lines)
{
    if (line == cabecalho) continue;

    string[] fields = line.Split(';');

    if (fields.Length == 4)
    {
        Ficheiro ficheiro = new Ficheiro
        {
            OWNER = fields[0].Trim(),
            Municipio = fields[1].Trim(),
            Operadora = fields[2].Trim(),
            Domicilio = fields[3].Trim(),
        };
        ficheiros.Add(ficheiro);
    }
}
```

O ficheiro CSV é processado através de uma thread que ordena o ficheiro pelo município, definindo um arquivo para onde serão enviados todos os dados.

```
ficheiros = ficheiros.Skip(0).OrderBy(f => f.Municipio).ToList();
string outputFilePath = "FicheiroCSV.csv";

using (StreamWriter writer = new StreamWriter(outputFilePath, true))
{
    foreach (Ficheiro ficheiro in ficheiros)
    {
        writer.WriteLine($"{ficheiro.OWNER};{ficheiro.Municipio};{ficheiro.Operadora};{ficheiro.Domicilio}");
    }
}
```

Atendimento simultâneo de múltiplos clientes:

Na classe Servidor é implementado o objeto Mutex de forma a controlar o acesso à lista TcpClient. Desta forma, apenas uma thread de cada vez pode alterar a lista, mantendo assim a sua integridade.

```
static Mutex mutex = new Mutex();
```

Os métodos associados ao Mutex, e invocados na função main, permitem a implementação correta do mesmo e uma abordagem multi-thread eficaz.

```
mutex.WaitOne();
clientes.Add(client);
mutex.ReleaseMutex();
```

Neste código, como referido anteriormente, é criada uma thread para cada cliente que se conecte ao servidor, permitindo a conexão com múltiplos clientes.

```
Thread clientThread = new Thread(() => HandleClient(client));
clientThread.Start();
```

3 – Código Fonte

GitLab: https://gitlab.com/df01mo/sistemas_distribuidos_grupo7_pl3/-/tree/main

Servidor:

```
using Microsoft.VisualBasic.Devices;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using System.Xml.Linq;
using System.Windows.Forms;
using System.Security.Cryptography;
using System.Security.Policy;
using System.Runtime.InteropServices.ComTypes;

namespace Servidor_Projeto
{
    class Ficheiro
    {
        public string OWNER { get; set; }
        public string Municipio { get; set; }
        public string Operadora { get; set; }
        public string Domicilio { get; set; }
        // public string Responsavel { get; set; }
        // public DateTime DataProcessamento { get; set; }
    }

    class Servidor
    {
        static List<TcpClient> clientes = new List<TcpClient>();
```

```
static Mutex mutex = new Mutex();

static void Main(string[] args)
{
    // Define o IP local e port number
    string localIP = "127.0.0.1";
    int localPort = 8888;

    // Cria o objeto listener TCP e start a listening thread
    TcpListener ServerSocket = new TcpListener(IPAddress.Parse(localIP), localPort);
    ServerSocket.Start();
    Console.WriteLine("Server waiting...");

    while (true)
    {
        // Aceita cliente e cria um objeto TCP
        TcpClient client = ServerSocket.AcceptTcpClient();
        Console.WriteLine("100 OK");
        Console.WriteLine("Client {0} connected", client.Client.RemoteEndPoint);

        mutex.WaitOne();
        clientes.Add(client);
        mutex.ReleaseMutex();

        // Cria uma thread separada para lidar com o cliente
        Thread clientThread = new Thread(() => HandleClient(client));
        clientThread.Start();
    }
}

public static void HandleClient(TcpClient client)
{
    try
    {
```

```
NetworkStream stream = client.GetStream();


while (true)
{

   // Lê os bytes enviados pelo cliente e converte em uma string
   byte[] buffer = new byte[1024];
   int byteCount = stream.Read(buffer, 0, buffer.Length);
   string message = Encoding.ASCII.GetString(buffer, 0, byteCount);

   if (message.ToUpper() == "QUIT")
   {
      // Se o cliente enviar "QUIT", desconecta do servidor
      Console.WriteLine("Client {0} disconnected.", client.Client.RemoteEndPoint);
      Console.WriteLine("400 BYE");

      // Remove o cliente da lista de clientes conectados
      mutex.WaitOne();
      clientes.Remove(client);
      mutex.ReleaseMutex();

      // Fecha a conexão com o cliente e encerra o loop
      client.Close();
      break;
   }
   if (message.StartsWith("OWNER"))
   {
      Console.Write("OPEN");
      Thread.Sleep(500);
      Console.Write("\nIN_PROGRESS");
      Thread.Sleep(2000);
      Console.Write("\nCOMPLETED\n");
      Thread.Sleep(500);
```

```
// Separa as linhas da mensagem em um array de strings
string[] lines = message.Split(new[] { "\r\n", "\r", "\n" }, StringSplitOptions.None);

// Armazena a primeira linha como cabeçalho
string cabecalho = lines[0];

// Cria uma lista de objetos Ficheiro para armazenar as informações
List<Ficheiro> ficheiros = new List<Ficheiro>();

foreach (string line in lines)
{
    // Ignora a primeira linha (cabeçalho)
    if (line == cabecalho) continue;

    // Separa os campos da linha em um array de strings
    string[] fields = line.Split(';');

    // Se a linha tiver 4 campos, cria um objeto Ficheiro e adiciona à lista
    if (fields.Length == 4)
    {
        Ficheiro ficheiro = new Ficheiro
        {
            OWNER = fields[0].Trim(),
            Municipio = fields[1].Trim(),
            Operadora = fields[2].Trim(),
            Domicilio = fields[3].Trim(),
        };
        ficheiros.Add(ficheiro);
    }
}

// Cria uma nova thread para processar o arquivo do cliente
Thread thread = new Thread(() =>
```

```
                {
                    // Espera a exclusão mútua antes de acessar recursos compartilhados
                    mutex.WaitOne();

                    try
                    {

                        // Ordena todos os arquivos pelo município
                        ficheiros = ficheiros.Skip(0).OrderBy(f => f.Municipio).ToList();

                        // Mostra ao cliente o número de domicílios presentes no arquivo atual
                        int numDomicilios = ficheiros.Count;
                        string numDomiciliosMessage = $"Number of households received:
{numDomicilios}\n";
                        byte[] numDomiciliosBytes = Encoding.ASCII.GetBytes(numDomiciliosMessage);
                        stream.Write(numDomiciliosBytes, 0, numDomiciliosBytes.Length);
                        stream.Flush();



                        // Define o caminho do arquivo de saída
                        string outputFilePath = "FicheiroCSV.csv";

                        // Abre o arquivo de saída para escrita
                        using (StreamWriter writer = new StreamWriter(outputFilePath, true))
                        {

                            foreach (Ficheiro ficheiro in ficheiros)
                            {
                                // Escreve a linha no arquivo de saída

writer.WriteLine($"{ficheiro.OWNER};{ficheiro.Municipio};{ficheiro.Operadora};{ficheiro.Domicilio}
");
                            }
                        }
```

```
                // Ordena o arquivo final por município
                OrdenarArquivoFinal(outputFilePath);

                stream.Flush();
                Console.WriteLine($"Informations saved in {outputFilePath}");
            }
            finally
            {
                // Liberta a exclusão mútua após o acesso aos recursos compartilhados
                mutex.ReleaseMutex();
            }
        });
        // Inicia a thread para processar o arquivo do cliente
        thread.Start();
    }
    else if (message.ToUpper() == "SEND")
    {
        // Define o caminho do arquivo de saída
        string outputFilePath = "FicheiroCSV.csv";

        // Cria um stream para ler o arquivo de saída
        using (FileStream fileStream = File.OpenRead(outputFilePath))
        {
            // Envia o arquivo para o cliente
            byte[] fileBytes = new byte[fileStream.Length];
            fileStream.Read(fileBytes, 0, fileBytes.Length);
            stream.Write(fileBytes, 0, fileBytes.Length);
        }

        string confirmacao =($"Ficheiro enviado\n");
        byte[] confirmacaoBytes = Encoding.ASCII.GetBytes(confirmacao);
        stream.Write(confirmacaoBytes, 0, confirmacaoBytes.Length);
        stream.Flush();
```

```
                Console.WriteLine($"Arquivo {outputFilePath} enviado para
{client.Client.RemoteEndPoint}");
            }
            //caso seja necessario enviar outras mensagens sem ser o ficheiro
            else
            {
                // Converte mensagem para bytes e envia de volta ao cliente
                byte[] messageBytes = Encoding.ASCII.GetBytes(message);
                stream.Write(messageBytes, 0, messageBytes.Length);
                stream.Flush();

                Console.WriteLine("\nMessage received from client {0}: " + message,
client.Client.RemoteEndPoint);
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error handling client {client.Client.RemoteEndPoint}: {ex.Message}");

        // Remove o cliente da lista de clientes conectados
        mutex.WaitOne();
        clientes.Remove(client);
        mutex.ReleaseMutex();

        // Fecha a conexão com o cliente
        client.Close();
    }
}

public static void OrdenarArquivoFinal(string pathArquivoFinal)
{
    // Lê todas as linhas do arquivo de saída
    List<string> linhas = File.ReadAllLines(pathArquivoFinal).ToList();
```

```
// Criar um dicionário para armazenar as linhas únicas
Dictionary<string, string> linhasUnicas = new Dictionary<string, string>();


// Armazena a primeira linha como cabeçalho
string cabecalho = linhas[0];

// Remove a primeira linha da lista (cabeçalho)
linhas.RemoveAt(0);

// Cria uma lista de objetos Ficheiro para armazenar as informações
List<Ficheiro> ficheiros = new List<Ficheiro>();

// Adiciona cada linha do arquivo de saída como um objeto Ficheiro à lista
foreach (string linha in linhas)
{
    string[] campos = linha.Split(';');

    Ficheiro ficheiro = new Ficheiro
    {
        OWNER = campos[0].Trim(),
        Municipio = campos[1].Trim(),
        Operadora = campos[2].Trim(),
        Domicilio = campos[3].Trim(),
        //DataProcessamento = DateTime.Now,
    };

    // Verifica se a linha já existe no dicionário linhasUnicas. Se sim, pula para a próxima linha.
    if (linhasUnicas.ContainsKey(linha))
    {
        continue;
    }

    // Adiciona a linha ao dicionário linhasUnicas.
```

```
            linhasUnicas.Add(linha, linha);
            ficheiros.Add(ficheiro);
        }

        //string responsavel = ficheiros.First(f => f.OWNER == "TRUE").Operadora;

        // Classifica a lista de arquivos pelo município
        ficheiros = ficheiros.OrderBy(f => f.Municipio).ToList();

        var domiciliosPorMunicipio = ficheiros.GroupBy(f => f.Municipio);



        // Reescreve o arquivo de saída com a nova ordem
        using (StreamWriter writer = new StreamWriter(pathArquivoFinal))
        {
            // Escreve o cabeçalho no arquivo de saída
            writer.WriteLine(cabecalho);

            // Escreve cada objeto Ficheiro ordenado por município no arquivo de saída
            foreach (Ficheiro ficheiro in ficheiros)
            {
                //Preenche com a operadora responsável

                //ficheiro.Responsavel = responsavel;

writer.WriteLine($"{ficheiro.OWNER};{ficheiro.Municipio};{ficheiro.Operadora};{ficheiro.Domicilio}
");
            }

            // Mostra o número de domicílios por município no console
            foreach (var grupo in domiciliosPorMunicipio)
            {
                int numDomicilios = grupo.Count();
                Console.WriteLine($"Município: {grupo.Key} - Número de domicílios: {numDomicilios}");
```

```
            }
        }
    }


    //Utilizado de inicio mas posteriormente substitudo
    public static void Broadcast(string message)
    {

        foreach (TcpClient client in clientes)
        {
            if (client.Connected)
            {
                //using (NetworkStream stream = client.GetStream())
                //{
                //    byte[] messageBytes = Encoding.ASCII.GetBytes(message);
                //    stream.Write(messageBytes, 0, messageBytes.Length);
                //    stream.Flush();
                //}
            }
            else
            {
                // Remove o cliente da lista se ele não estiver mais conectado
                mutex.WaitOne();
                clientes.Remove(client);
                mutex.ReleaseMutex();
            }
        }
    }
  }
}
```

Cliente:

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

class Cliente
{
   static void Main(string[] args)
   {
      // Define o ip do servidor e o número da porta
      string serverIP = "127.0.0.1";
      int serverPort = 8888;

      // Cria o objeto cliente TCP e conecta ao servidor
      TcpClient client = new TcpClient(serverIP, serverPort);

      // Recebe a network stream para enviar e receber dados
      NetworkStream stream = client.GetStream();

      while (true)
      {
         // Lê a mensagem de input da consola
         Console.Write("Write a message to the server or 'CSV' file path to send a file \nor 'send' to recive the file or 'quit' to close connection:\n");
         string message = Console.ReadLine();

         if (message.ToUpper() == "QUIT")
         {
            // Converte a mensagem para bytes e envia ao servidor
            byte[] quitMessageBytes = Encoding.ASCII.GetBytes(message);
            stream.Write(quitMessageBytes, 0, quitMessageBytes.Length);
            Console.WriteLine("400 BYE");
            // Sai do loop se a mensagem for 'quit'
```

```
    break;
}

if (message.EndsWith(".csv") && !File.Exists(message))
{
    Console.WriteLine("ERROR: File not found. Please try again");
    continue;

}

if (message.EndsWith(".csv"))
{
    //Read file content and send to server
    string fileContent = File.ReadAllText(message);

    // Envia uma mensagem de confirmação ao cliente
    Console.WriteLine("File saved successfully!");

    byte[] fileBytes = Encoding.ASCII.GetBytes(fileContent);
    stream.Write(fileBytes, 0, fileBytes.Length);

    //Mostra a mensagem de confirmação do servidor
    byte[] clientBuffer = new byte[1024];
    int clientByteCount = stream.Read(clientBuffer, 0, clientBuffer.Length);
    string clientMessage = Encoding.ASCII.GetString(clientBuffer, 0, clientByteCount);
    Console.WriteLine(clientMessage);
}

if (message.ToUpper() == "SEND")
    {
        // Converte a mensagem para bytes e envia ao servidor
        byte[] messageBytes = Encoding.ASCII.GetBytes(message);
        stream.Write(messageBytes, 0, messageBytes.Length);
```

```
            // Cria um buffer para armazenar os bytes recebidos
            byte[] buffer = new byte[1024];

            // Cria um FileStream para armazenar o arquivo recebido
            using (FileStream fileStream = new FileStream("FicheiroCSV.csv", FileMode.Create))
            {
                int bytesRead;
                while ((bytesRead = stream.Read(buffer, 0, buffer.Length)) > 0)
                {
                    fileStream.Write(buffer, 0, bytesRead);
                    fileStream.Flush();
                }
            }
        // Recebe a mensagem de confirmação do servidor
        byte[] confirmacaobuffer = new byte[1024];
        int confirmacaobytesRead = stream.Read(buffer, 0, buffer.Length);
        string confirmacaoMessage = Encoding.ASCII.GetString(buffer, 0, confirmacaobytesRead);
        Console.WriteLine(confirmacaoMessage);
    }

    //Para envios de mensagens sem ser ficheiros
    else
    {
        // Converte a mensagem para bytes e envia ao servidor
        byte[] messageBytes = Encoding.ASCII.GetBytes(message);
        stream.Write(messageBytes, 0, messageBytes.Length);

        // Recebe a mensagem de confirmação do servidor
        byte[] buffer = new byte[1024];
        int bytesRead = stream.Read(buffer, 0, buffer.Length);
        string serverMessage = Encoding.ASCII.GetString(buffer, 0, bytesRead);

        // Mostra a mensagem de confirmação do servidor
        Console.WriteLine("-Server acknowledge-");
```

```
            }
        }

        // Fecha a conexão e o cliente
        stream.Close();
        client.Close();
    }
}
```