Sistemas Distribuídos

Git Lab Project Id: 44090462

Hugo Paredes | José Cunha | Dennis Paulino
Vitor Novo Al74280| Gustavo Santos Al70800| Ricardo Silva Al70672| Vasco Teixeira Al74107

Protocolo Prático 1- Cobertura em Wholesalers -

Protocolo: A comunicação Cliente-Servidor funciona inicialmente com o cliente a inserir o endereço IP do servidor ao qual deseja conectar. Quando o cliente é conectado, o servidor envia uma mensagem de confirmação '100-OK' para o cliente. Após a conexão, é solicitado ao cliente que envie um arquivo CSV. Após o envio deste arquivo, o servidor processará o documento, verificando primeiramente se o arquivo já foi processado anteriormente. Caso positivo, é enviada uma mensagem de erro e o evento é registado nos logs. Ao longo do processamento do arquivo, serão enviadas mensagens de status ao cliente, e o progresso será registado nos logs como OPEN, ERROR, IN_PROGRESS ou COMPLETED.

Após processar todas as linhas do documento, é verificado se já existe uma cobertura com a mesma morada, e caso exista, é enviada uma mensagem de erro para essa linha. Caso contrário, a nova cobertura é adicionada à base de dados. Concluído o processamento de todas as coberturas, os dados são guardados na base de dados, e uma mensagem 'Completed' é enviada ao cliente.

O cliente pode enviar mais arquivos, e caso queira terminar a comunicação, deve enviar uma mensagem com 'quit'. Após isso, o servidor responde com '400-BYE'.

Implementação: A implementação começou com a construção duma comunicação 1-N entre o servidor e vários clientes de forma concorrente, utilizando threads. Em seguida, foi criada uma Base de Dados Relacional usando a Framework .Net.

Posteriormente, foi desenvolvida a comunicação cliente-servidor, que incluiu o envio de mensagens de confirmação, solicitação de ficheiros CSV, em caso de problemas no processamento, as mensagens de erro aparecem no ecrã do servidor e são registadas nos logs .

Os logs são atualizados ao longo do processamento.

Por fim, foi criada a função para receber o ficheiro e processá-lo. Após o processamento, os valores foram guardados na Base de Dados, caso fosse possível.

Conclusão: Este projeto não é excessivamente complicado, mas requer uma abordagem cuidadosa para evitar erros que possam comprometer a eficácia do sistema. Nós nos esforçamos para seguir todas as etapas do projeto com precisão e atenção aos detalhes. Desde a criação da comunicação cliente-servidor usando threads, até o desenvolvimento da função de processamento de ficheiros e a utilização da base de dados relacional.

Acreditamos que o projeto atingiu todos os objetivos propostos pelo professor e estamos satisfeitos com o trabalho realizado. O sistema é capaz de proporcionar uma comunicação eficiente e segura entre o cliente e o servidor, permitindo o processamento de ficheiros e a manutenção de registos precisos na base de dados relacional.

Anexo A-

```
using Aula_2___Sockets;
using Aula_2___Sockets.Models;
using Newtonsoft.Json.Linq;
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;

namespace Aula_2___Sockets___Server
{


    class Program
    {
        public static dataContext dataContext = new dataContext();
        public static Mutex mutex = new Mutex();
        public enum StatusCode
        {
            OK = 100,
            ERROR = 300,
            BYE = 400
        }

        public enum FileStatus
        {
            OPEN,
            ERROR,
            IN_PROGRESS,
            COMPLETED
        }

        static void Main(string[] args)
        {
            //A classe TCPListener implementa os métodos da classe Socket utilizando o protócolo TCP, permitindo
uma maior abstração das etapas tipicamente associadas ao Socket.
            TcpListener ServerSocket = new TcpListener(IPAddress.Any, 1337);
            Console.WriteLine($"Listening on:
{((IPEndPoint)ServerSocket.LocalEndpoint).Address}:{((IPEndPoint)ServerSocket.LocalEndpoint).Port}");

            //A chamada ao método "Start" inicia o Socket para ficar à escuta de novas conexões por parte dos clientes
            ServerSocket.Start();
            Thread thread = new Thread(() =>
            {
                Program.MainThread(ServerSocket);
            });
            thread.Start();
        }

        public static void MainThread(TcpListener ServerSocket)
        {
            while (true)
            {
                //Ciclo infinito para ficar à espera que um cliente Socket/TCP até quando pretender conectar-se

                TcpClient client = ServerSocket.AcceptTcpClient();
                Thread thread = new Thread(() =>
```

```
        {
            Program.MainThread(ServerSocket);
        });
        thread.Start();
        //Só avança para esta parte do código, depois de um cliente ter se conectado ao servidor
        handle_client(client);
    }
}

public static void handle_client(TcpClient client)
{

    string id = Guid.NewGuid().ToString();
    Console.WriteLine($"{id} Connected!");
    Thread.CurrentThread.Name = id;
    // Neste método, é iniciada a gestão da comunicação do servidor com o cliente
    OnClientConnected(client);
    try
    {
        ParseFile(client);
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
    CloseConnection(client);
    Console.WriteLine($"{id} Disconnected!");
}

/// <summary>
/// Função que faz parsing a um ficheiro e que organiza os
/// dados recebidos
/// </summary>
/// <param name="client"></param>
public static void ParseFile(TcpClient client)
{
    List<string> Erros = new List<string>();
    byte[] bRec = new byte[1024];
    int n;
    var sb = new StringBuilder();
    string filename = Guid.NewGuid().ToString();

    //receber o ficheiro
    do
    {
        n = client.GetStream().Read(bRec, 0, bRec.Length);
        sb.Append(Encoding.UTF8.GetString(bRec, 0, n));
    } while (!Encoding.UTF8.GetString(bRec, 0, n).Contains("\0\0\0"));

    sb.Replace("\0\0\0", "");

    File.WriteAllText($"./Coberturas/{filename}.csv", sb.ToString(), Encoding.UTF8);
    var hash = ChecksumUtil.GetChecksum(HashingAlgoTypes.SHA256, $"./Coberturas/{filename}.csv");

    //ficheiro recebido
    bRec = Encoding.UTF8.GetBytes($"{(int)StatusCode.OK} - File Received\0\0\0");
    Console.WriteLine($"{(int)StatusCode.OK} - File Received\0\0\0");
    client.GetStream().Write(bRec, 0, bRec.Length);

    //wait
    Console.WriteLine($"{Thread.CurrentThread.Name} is requesting access");
    mutex.WaitOne();
    Console.WriteLine($"{Thread.CurrentThread.Name} is in the protected area");
```

```
        //se n existir já o ficheiro
        if (!dataContext.Ficheiros.Any(x => x.Hash == hash))
        {

            Ficheiro file = new Ficheiro();
            file.Hash = hash;
            dataContext.Ficheiros.Add(file);
            dataContext.SaveChanges();
            mutex.ReleaseMutex();
            Console.WriteLine($"{Thread.CurrentThread.Name} released the mutex");
            //release


            var lista = CsvParser.CsvToList($"./Coberturas/{filename}.csv", ';');

            //Verifico se a 1 linha não contem todos as componentes
            if (lista[0][0] != "Operador" || lista[0][1] != "Município" || lista[0][2] != "Rua" || lista[0][3] != "Número" ||
lista[0][4] != "Apartamento" || lista[0][5] != "Owner")
            {
                //wait
                Console.WriteLine($"{Thread.CurrentThread.Name} is requesting access");
                mutex.WaitOne();
                Console.WriteLine($"{Thread.CurrentThread.Name} is in the protected area");

                //guardar nos logs que o documento não foi processado pois é invalido
                Console.WriteLine($"{(int)StatusCode.ERROR} - {StatusCode.ERROR}: Invalid File!\0\0\0");
                bRec = Encoding.UTF8.GetBytes($"{(int)StatusCode.ERROR} - {StatusCode.ERROR}: Invalid
File!\0\0\0");

                dataContext.Logs.Add(new Logs() { DataInicio = DateTime.Now, Estado =
FileStatus.ERROR.ToString(), Ficheiro = $"{filename}.csv"});
                dataContext.SaveChanges();

                mutex.ReleaseMutex( );
                Console.WriteLine($"{Thread.CurrentThread.Name} released the mutex");
                //release

                client.GetStream().Write(bRec, 0, bRec.Length);

                //delete no ficheiro
                File.Delete($"./Coberturas/{filename}.csv");
            }
            else
            {

                //Lista de Coberturas que vão estar no Document
                List<Cobertura> coberturas = new List<Cobertura>();

                lista.RemoveAt(0);
                lista = lista.OrderBy(x => x[1]).ToList();
                lista.Insert(0, new List<string>() { "Operador", "Município", "Rua", "Número", "Apartamento",
"Owner" });

                //wait
                Console.WriteLine($"{Thread.CurrentThread.Name} is requesting access");
                mutex.WaitOne();

                Console.WriteLine($"{Thread.CurrentThread.Name} is in the protected area");

                //adicionar a tabela Logs da base de dados o status da situação
                dataContext.Logs.Add(new Logs() { DataInicio = DateTime.Now, Estado =
FileStatus.OPEN.ToString(), Ficheiro = $"{filename}.csv", Operador = lista[1][0] });
                dataContext.Logs.Add(new Logs() { DataInicio = DateTime.Now, Estado =
FileStatus.IN_PROGRESS.ToString(), Ficheiro = $"{filename}.csv", Operador = lista[1][0] });
```

```
            dataContext.SaveChanges();


            mutex.ReleaseMutex();
            Console.WriteLine($"{Thread.CurrentThread.Name} released the mutex");
            //released

            foreach (var item in lista)
            {
                if (item[0] != "Operador")
                {

                    //verificamos se os items são nulos
                    if (String.IsNullOrEmpty(item[0]) || String.IsNullOrEmpty(item[1]) ||
                        String.IsNullOrEmpty(item[2]) || String.IsNullOrEmpty(item[3]))
                    {

                        //erro se algum é null ou empty
                        Console.WriteLine($"{(int)StatusCode.ERROR} - {StatusCode.ERROR}: Parsing error, empty
or null value! '{item[0]};{item[1]};{item[2]};{item[3]};{item[4]};{item[5]} is invalid! File:
{filename}.csv\0\0\0");
                        Erros.Add($"{(int)StatusCode.ERROR} - {StatusCode.ERROR}: Parsing error, empty or null
value! '{item[0]};{item[1]};{item[2]};{item[3]};{item[4]};{item[5]} is invalid!\n");

                        //wait
                        Console.WriteLine($"{Thread.CurrentThread.Name} is requesting access");
                        mutex.WaitOne();
                        Console.WriteLine($"{Thread.CurrentThread.Name} is in the protected area");

                        //adicionar aos logs o erro
                        dataContext.Logs.Add(new Logs() { DataInicio = DateTime.Now, Estado =
FileStatus.ERROR.ToString(), Ficheiro = $"{filename}.csv", Operador = lista[1][0] });
                        dataContext.SaveChanges();

                        mutex.ReleaseMutex();
                        Console.WriteLine($"{Thread.CurrentThread.Name} released the mutex");
                        //release

                    }
                    else
                    {

                        //Classificar se é Owner
                        if (String.IsNullOrEmpty(item[5])) item[5] = "false"; //se for nulo quer dizer que é falso
                        else item[5] = "true";  //qualquer coisa é owner

                        //Criar Objeto Cobertura e adicionar a uma lista
                        Cobertura cobertura = new Cobertura()
                        {
                            Operador = item[0],
                            Municipio = item[1],
                            Rua = item[2],
                            Numero = item[3],
                            Apartamento = item[4],
                            Owner = Boolean.Parse(item[5]),
                        };
                        //adicionar a cobertura a lista de coberturas que estão no documento
                        coberturas.Add(cobertura);
                    }
                }
            }

            //wait
            Console.WriteLine($"{Thread.CurrentThread.Name} is requesting access");
```

```
                mutex.WaitOne();
                Console.WriteLine($"{Thread.CurrentThread.Name} is in the protected area");

                //Status para os Logs
                dataContext.Logs.Add(new Logs() { DataInicio = DateTime.Now, Estado =
FileStatus.COMPLETED.ToString(), Ficheiro = $"{filename}.csv", Operador = lista[1][0] });
                dataContext.SaveChanges();

                //Guardar na Base de Dados as coberturas
                GuardarCoberturasBaseDados(coberturas, client, Erros);
                mutex.ReleaseMutex();
                Console.WriteLine($"{Thread.CurrentThread.Name} released the mutex");
                //release

                string errostobuff = "";
                foreach (var e in Erros)
                {
                   errostobuff += e;
                }
                errostobuff += $"{(int)StatusCode.OK} - {StatusCode.OK}: File processed!\0\0\0";
                bRec = Encoding.UTF8.GetBytes(errostobuff);
                client.GetStream().Write(bRec, 0, bRec.Length);
                Console.WriteLine($"{(int)StatusCode.OK} - {StatusCode.OK}: File processed!\0\0\0");
            }



        }
        else
        {

            //Status para os Logs
            dataContext.Logs.Add(new Logs() { DataInicio = DateTime.Now, Estado =
FileStatus.ERROR.ToString(), Ficheiro = $"{filename}.csv" });
            dataContext.SaveChanges();
            Console.WriteLine($"{(int)StatusCode.ERROR} - {StatusCode.ERROR}: File already
processed!\0\0\0");

            mutex.ReleaseMutex();
            Console.WriteLine($"{Thread.CurrentThread.Name} released the mutex");

            //release

            bRec = Encoding.UTF8.GetBytes(
               $"{(int)StatusCode.ERROR} - {StatusCode.ERROR}: File already processed!\0\0\0");
            client.GetStream().Write(bRec, 0, bRec.Length);
            File.Delete($"./Coberturas/{filename}.csv");
            return;
        }
        Console.WriteLine($"Saved as '{filename}.csv'");
    }

    /// <summary>
    /// Função para guardar uma lista de coberturas na base de dados
    /// </summary>
    /// <param name="coberturas"></param>
    /// <param name="client"></param>
    /// <param name="Erros"></param>
    public static void GuardarCoberturasBaseDados(List<Cobertura> coberturas, TcpClient client, List<string>
Erros)
    {
        byte[] bRec = new byte[1024];
        //wait
        Console.WriteLine($"{Thread.CurrentThread.Name} is requesting access");
```

```
            mutex.WaitOne();
            Console.WriteLine($"{Thread.CurrentThread.Name} is in the protected area");

            foreach (var cobertura in coberturas)
            {
                //Verifico se não existe nenhuma cobertura com a mesma morada
                if (!dataContext.Coberturas.Any(c => c.Municipio == cobertura.Municipio && c.Rua == cobertura.Rua
&& c.Apartamento == cobertura.Apartamento && c.Numero == cobertura.Numero))
                {
                    dataContext.Coberturas.Add(cobertura);

                }
                else
                {
                    Console.WriteLine($"{(int)StatusCode.ERROR} - {StatusCode.ERROR}: The cobertura :
{cobertura.Municipio}, {cobertura.Rua}, {cobertura.Numero}, {cobertura.Apartamento}  wasn't accepted because
it already exists!\0\0\0");
                    Erros.Add($"{(int)StatusCode.ERROR} - {StatusCode.ERROR}: The cobertura :
{cobertura.Municipio}, {cobertura.Rua}, {cobertura.Numero}, {cobertura.Apartamento} wasn't accepted because it
already exists!\n");

                }

            }
            dataContext.SaveChanges();
            mutex.ReleaseMutex();
            Console.WriteLine($"{Thread.CurrentThread.Name} released the mutex");
            //release
        }

        /// <summary>
        /// get dos municipios por abcedario
        /// </summary>
        /// <returns></returns>
        public static List<Cobertura> GetDataModelsMunicipio()
        {
            //wait
            mutex.WaitOne();
            List<Cobertura> data = dataContext.Coberturas.OrderBy(x => x.Municipio).OrderBy(x => x.Rua).ToList();
            mutex.ReleaseMutex();
            //release

            return data;
        }

        /// <summary>
        /// get dos municipios por abcedario com pesquisa
        /// </summary>
        /// <param name="municipio"></param>
        /// <returns></returns>
        public static List<Cobertura> GetDataModelMunicipio(string municipio)
        {
            //wait
            mutex.WaitOne();
            List<Cobertura> data = dataContext.Coberturas.Where(x => x.Municipio.Contains(municipio)).OrderBy(x
=> x.Rua).ToList();
            mutex.ReleaseMutex();
            //release

            return data;
        }

        /// <summary>
        /// On first connection
```

```
        /// </summary>
        /// <param name="client"></param>
        public static void OnClientConnected(TcpClient client)
        {
            var bytes = Encoding.UTF8.GetBytes($"{(int)StatusCode.OK} - {StatusCode.OK}\0\0\0");
            client.GetStream().Write(bytes, 0, bytes.Length);
        }

        /// <summary>
        /// closing the connection
        /// </summary>
        /// <param name="client"></param>
        public static void CloseConnection(TcpClient client)
        {
            byte[] buffer = new byte[1024];
            StringBuilder data = new StringBuilder();
            if (!client.Connected) return;
            do
            {
                int byte_count = client.GetStream().Read(buffer, 0, buffer.Length);
                data.Append(Encoding.UTF8.GetString(buffer, 0, byte_count));
            } while (!data.ToString().Contains("\0\0\0"));

            if (data.ToString().Contains("QUIT"))
            {
                buffer = Encoding.UTF8.GetBytes($"{(int)StatusCode.BYE} - {StatusCode.BYE}\0\0\0");
                client.GetStream().Write(buffer, 0, buffer.Length);
                client.Client.Shutdown(SocketShutdown.Both);
                client.Close();
            }
        }
    }
}
```

Git Lab Project Id: 44090462

Anexo Cliente-

```
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.Threading.Tasks;

namespace Aula_2___Sockets___Client {
    internal class Program {
        public enum StatusCode {
            OK = 100,
            ERROR = 300,
            BYE = 400
        }
        static void Main(string[] args) {

            TcpClient ClientSocket = ConnectServer();

            if (ClientSocket.Connected) {
                Console.WriteLine($"Connected to:
{((IPEndPoint)ClientSocket.Client.RemoteEndPoint).Address}:{((IPEndPoint)ClientSocket.Client.RemoteEndPoin
t).Port}");

                byte[] buffer = new byte[1024];
                string data;

                data = GetDataFromStream(ClientSocket);

                //Se a mensagem for diferente de ok deu erro na transmissão
                if (!data.Contains($"{(int)StatusCode.OK} - {StatusCode.OK}"))
                {
                    Console.WriteLine($"Error: Expected '{(int)StatusCode.OK} - {StatusCode.OK}' \nClosing
connection...");
                    ClientSocket.Client.Shutdown(SocketShutdown.Both);
                    ClientSocket.Close();

                    Console.ReadKey();
                    return;
                }

                //Path do documento
                string path;
                do {
                    Console.Write("Full Path to File or Drag an Drop File: ");
                    path = Console.ReadLine();
                    path = path.Replace("\"", "");
                    path = path.Replace(@"\\", @"\");
                } while (!SendFile(ClientSocket, path));

                data = GetDataFromStream(ClientSocket);

                Console.WriteLine(data);

                data = GetDataFromStream(ClientSocket);

                Console.WriteLine(data);
```

```
        //quit
        buffer = Encoding.UTF8.GetBytes("QUIT\0\0\0");
        ClientSocket.GetStream().Write(buffer, 0, buffer.Length);

        data = GetDataFromStream(ClientSocket);

        Console.WriteLine(data);


        Console.WriteLine("\nPress any key to exit...");
        Console.ReadKey();

    }
    ClientSocket.Close();

}


/// <summary>
/// Receber resposta do server
/// </summary>
/// <param name="client"></param>
/// <returns></returns>
public static string GetDataFromStream(TcpClient client) {
    byte[] buffer = new byte[1024];
    StringBuilder data = new StringBuilder();
    do {
        int byte_count = client.GetStream().Read(buffer, 0, buffer.Length);
        data.Append(Encoding.UTF8.GetString(buffer, 0, byte_count));
    } while (!data.ToString().Contains("\0\0\0"));

    return data.ToString();
}

/// <summary>
/// Conectar a um servidor
/// </summary>
/// <returns></returns>
public static TcpClient ConnectServer() {
    TcpClient ClientSocket = new TcpClient();

    string[] ipBuffer;
    while (true) {
        Console.WriteLine("Write the IP you want to connect to:");
        ipBuffer = Console.ReadLine().Split(':');
        if (ipBuffer.Length == 2) {
            if (!String.IsNullOrEmpty(ipBuffer[0]) && !String.IsNullOrEmpty(ipBuffer[1]))
                break;
        }
    }

    try {
        ClientSocket.Connect(ipBuffer[0], int.Parse(ipBuffer[1]));
    } catch (Exception e) {
        Console.WriteLine($"{e.Message}\n");
        Console.ReadKey();
    }


    return ClientSocket;
}


/// <summary>
```

```
        /// enviar ficheiro para o servidor
        /// </summary>
        /// <param name="ClientSocket"></param>
        /// <param name="path"></param>
        /// <returns></returns>
        public static bool SendFile(TcpClient ClientSocket, string path) {
            try {

                Console.WriteLine($"Sending File: {path}");
                var buff = Encoding.UTF8.GetBytes(File.ReadAllText(path, Encoding.GetEncoding(1252)) + "\0\0\0");
                ClientSocket.GetStream().Write(buff, 0, buff.Length);
                return true;

            } catch (FileNotFoundException e) {

                Console.WriteLine(e.Message);
                return false;
            }
        }

    }
}
```

Git Lab Project Id: 44090462

Anexo Models -

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Aula_2___Sockets.Models
{
    public class Cobertura
    {
        [Key]
        public int Id { get; set; }

        public string Operador { get; set; }

        public string Municipio { get; set; }

        public string Rua { get; set; }

        public string Numero { get; set; }

        public string Apartamento { get; set; }

        public Boolean Owner { get; set; }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Aula_2___Sockets.Models {
    public class Ficheiro {
        [Key]
        public string Hash { get; set; }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Aula_2___Sockets.Models {
    public class Logs {
        [Required]
        public int ID { get; set; }
        [Required]
        public DateTime DataInicio { get; set; }
        [Required]
        public string Ficheiro { get; set; }
        [Required]
        public string Operador { get; set; }
        [Required]
        public string Estado { get; set; }
```

```
    }
}
```